Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Sixteen Topics Explored from Quality and Understandability Studies on Unified Modelling Language Notation

Sina Alizadeh Tabrizi

s.alizadehtabrizi@iau.ac.ir

Department of Computer Engineering, Ur.C., Islamic Azad University, Urmia, Iran
https://orcid.org/0000-0001-6785-675X

Damla Topalli

Assistant Professor, Department of Software Engineering, Ankara University, Ankara, Turkey
https://orcid.org/0000-0002-4394-287X

Nergiz Ercil Cagiltay

Professor, Hacettepe University, Institute of Informatics, Ankara, Turkey. https://orcid.org/0000-0003-0875-9276

Systematic Review

Keywords: Quality, Understandability, Unified Modelling Language, Systematic Mapping, Model-driven Software Development

Posted Date: October 22nd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9996788/v3

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Sixteen Topics Explored from Quality and Understandability Studies on Unified Modelling Language Notation

Sina Alizadeh Tabrizi[1,*], Damla Topalli[2], Nergiz Ercil Cagiltay[3]

[1]*Department of Computer Engineering, Ur.C., Islamic Azad University, Urmia, Iran.*
[*]*Corresponding Author:* s.alizadehtabrizi@iau.ac.ir *; ORCID: 0000-0001-6785-675X*
[2]*Assistant Professor, Department of Software Engineering, Ankara University, Ankara, Turkey.*
E-mail: dtopalli@ankara.edu.tr ; ORCID: 0000-0002-4394-287X

[3]*Professor, Hacettepe University, Institute of Informatics, Ankara, Turkey.*
E-mail: nergizcagiltay@hacettepe.edu.tr ; ORCID: 0000-0003-0875-9276

ABSTRACT

The aim of this study was to identify and demonstrate trends in the areas of quality and understandability of unified modeling language notation. It is aimed at giving future directions for the researchers working in this field. A systematic mapping study was conducted on 227 journal and conference articles that were retrieved from the Web of Science, Scopus, IEEE Xplore and ACM. The results indicated that quality is the main aspect, in such a way that other aspects like understandability and maintainability are subsets of this aspect. Moreover, sixteen perspectives were found from which the quality and understandability of the UML diagrams have been studied by the preceding studies. The results obtained from the 227 article mapping provided 16 inductive categories. Quality Assessment category was found to be the most prevalent one, followed by Representations, Design and Modelling, and Testing categories respectively. Guideline, Pedagogical Strategy, Software Engineer categories had the same prevalence, 6.61%. System Structure (0.88%) and Model Transformation (0.44%) are under-explored areas. Output was highest during the period between 2015-2019, whereas Pedagogical Strategy category emerged after 2020 (3.96%) with the help of gamification, LLM feedback, remote learning and process thinking approach. Co-occurrence and theoretical anchoring (SEQUAL, ISO/IEC 25010, Moody's Physics of Notations) provide evidence that quality and understandability are integrative and multi-perspective concepts. This research paper presents a new inductive taxonomy, theory-based synthesis, and future research directions on UML quality and understandability in agile, AI-enabled and low-code/no-code modelling environments.

Keywords: Quality, Understandability, Unified Modelling Language, Systematic Mapping, Model-driven Software Development

1. Introduction

The analysis phase is one of the integral steps of the software development lifecycle, during which an analyst endeavors to model the business process using two prominent notations, namely, the Business Process Model and Notation (BPMN) and the Unified Modeling Language (UML). The developers can discern the system's behavior, structure, and elements by reviewing the artifacts from the modeling phase in order to improve the system's quality. In fact, the models used to communicate with stakeholders, through which the analysis and development teams can document the system's both functional and non-functional requirements. As a result, "software is not only difficult to create, but also difficult to understand" [1]. It is not only confined to the software, but it is also true for the models, especially for the design models. One of the first steps to solving the understandability problem in the software field was to introduce another artifact, called documentation.

Both the software code and the related design models should carry an intent, since the level of intent expressed by the code influences the simplification of error discovery and any deviation from or adherence to the functionality that the application should exhibit. The preservation of a comprehensible intent is significant, and the source of the intent in the models is the organization's people [1]. Hence, people are crucial for generating comprehensible software designs. An earlier study reports that the agile and lean approaches have the advantage of bringing the stakeholders together for face-to-face contact in order to facilitate and expedite communication, which is effective in terms of time, cost, and resources [2]. According to the study conducted by Kabeli and Shoval, in the contemporary era, Unified Modeling Language (UML) is the indispensable modeling language used in the software engineering discipline, as it adopts multiple diagram types to cover all aspects of the modeled system [3].

Firstly, the central status of the UML in software modeling is inevitable, as highlighted in seminal works by [4]. In fact, UML serves as a widely accepted visual representation tool, facilitating effective communication and design within software engineering projects. For Bersini [5], the abstraction level provided by UML is higher than that of object-oriented programming languages, in such a way that the scholars are interested in spending more time on modeling rather than on programming. Indeed, using well-defined and standardized diagrams of UML the system analysts can naturally elaborate and resolve problems based on the high-level concepts which are inherent to formulation of the problem. The graphical representation which is yielded by UML notation enables the engineers to constitute serious progress towards the algorithmic solution of the problem in the field of software engineering.

During software development, UML diagrams facilities the process of identifying the requirements and scopes of the systems and applications by graphical models [6]. UML diagrams are not just user-friendly for non-technical stakeholders, but they also offer a straightforward and comprehensible representation and description of the process as perceived by the end-users [7]. Modeling should be considered as a trajectory for handling the increasing complexity of software development through aiding engineers to cooperate at a higher level of abstraction [8]. Finally, it is impossible to overlook the impact of UML on software development life cycle documentation. By using UML, it was possible to achieve better design quality for implementing changes, which was expected to make future changes easier. However, these claims are only valid in certain situations. In addition to thorough training in UML modeling, benefits might not be realized when working on tasks with low complexity or when software engineers are not proficient in using UML models to assess change impacts [8].

Second, we acknowledge the importance and subtleties of the concepts being studied, taking inspiration from foundational works like Gamma et al. [9] for design patterns and Booch, Rumbaugh and Jacobson [10] for modeling techniques. These seminal works emphasize how crucial a comprehensive analysis is to expand our knowledge of the topic. The current study used a thorough search strategy, strict inclusion criteria, and a methodical synthesis process in

accordance with the guidelines provided by Kitchenham et al. [11]. By adhering to the established practices of systematic mapping, it was aimed to contribute to the field's knowledge base while ensuring the integrity and reliability of our research findings. Generally, the method consists of three phases, namely, planning, execution and reporting, while depending on the research target there may be other steps within the afore-mentioned phases, and the steps, including data retrieval, study selection, data extraction and data synthesis are the integral steps of Systematic Literature Review (SLR) [6].

Aside from its scholarly significance, there is a research gap in the state-of-the-art of UML diagram quality and understandability. The quality of UML diagrams can be defined as the absence of defects and errors in a UML model caused by a conflict between different points of view or the absence of UML norms. Understandability of the UML diagrams can be defined as the correctness and completeness of the design models facilitating communication between the stakeholders to discern the system's functional and non-functional requirements and to correct the reflection of the business process in the real world. This enables users to identify problem domain information and broaden their understanding of problem solving.

The understandability and comprehension of UML diagrams can be varied between individuals. In essence, no specific type of diagram held universal significance over the others in terms of aiding comprehension, implying any disparities in preference among different teams balanced each other out [12].

Understandability is a key sub-characteristic of software quality. Based on ISO/IEC 9126 and its successor ISO/IEC 25010, understandability denotes the degree to which a software product enables the user to comprehend whether it is suitable and how it can be used for particular tasks. In the context of UML diagrams, understandability can be defined as the degree to how easily a stakeholder can interpret and use the visual representation to reason about system behavior, structure, or requirements. By anchoring our study in the established definition of understandability from ISO standards and analyzing how it is concretely assessed in the literature, we provide a solid conceptual foundation for this systematic mapping.

To wrap it up, UML notations possess several features making it the de facto graphical representation tool during software development process, including standardized notation, rich in expressivity and having different diagrams for depicting different views and abstraction level. However, the notion of quality is also applicable to UML diagrams, since even if the model is read vigilantly, the defects remain undetected and such a situation increases the level of risk for different interpretations [13].

Hence, when considering the available literature on the topics, the quality and understandability of UML models are not well specified. As a result, the purpose of this research is to examine the state-of-the-art for UML diagrams in terms of quality and comprehension, as well as to reveal the topics and trends associated with this concept.

In what follows, the background section explores relevant earlier research, providing context for our investigation. In accordance with accepted research design principles, the methodology section describes our methodical research procedures, including the formulation of research question and data collection techniques. Results present both quantitative and qualitative evidence for empirical conclusions drawn from a complete data analysis. We interpret results within theoretical frameworks in discussions and conclusions, providing insightful analysis and directions for future research.

2. Background

The characterization and applications of UML diagrams: In software engineering, UML diagrams serve as visual aids for modeling, designing, and communicating different system components. They include a variety of types, including activity diagrams that show workflows and processes, sequence diagrams that show interactions between objects over time, and class

diagrams that show class structures and relationships. Software development and design decisions are guided by UML diagrams, which also help to clarify complex system structures and promote efficient stakeholder communication [4].

Key problems with model analysis in software engineering: System development presents important model analysis challenges in software engineering. These include model completeness, which addresses possible omissions in a model's representation, model validation, which confirms that a model accurately reflects the intended system behavior and model consistency, which guarantees coherence between various parts of a model. These problems affect the dependability and quality of software designs, highlighting the necessity of exacting model analysis methods to reduce possible risks and uncertainties. [14, 15].

Several studies have been carried out on the understandability, visualization, quality, and current practices for high-quality UML diagrams. Moreover, several empirical studies on UML notation, model-driven development using UML, consistency management, maintainability, and complexity metrics were also conducted by applying a systematic literature review approach or other research methodologies.

Genero et al. [16] presented a detailed study on the quality of UML models, using a systematic approach to review different strategies for making UML models more robust toward adaptability with modern software development database management. The conclusions of their study emphasized tailored UML modeling practices that would allow more effective integration and management of databases. Cabot et al. [17] proposed a novel approach, UMLtoCSP, to use constraint programming for the formal validation of UML/OCL models.

Through rigorous verification, their study produced a significant contribution toward the advancement of software engineering by enhancing the reliability of modeling and ensuring the accuracy of software specifications and designs. Torre et al. [18] carefully carried out an expert survey to disclose a range of consistency management strategies within model-driven software engineering and UML. By providing a broad overview, their research brought varied approaches and their consequences into light, thus offering practitioners a comprehensive catalog of how to effectively master the intricacies of this important area. Brambilla et al. [19] extensively explored practical applications of model-driven software engineering; in that, the key insights obtained in the context of MDE theory were effectively combined into real-world implementation. Their research, based on an analysis of methods and techniques, enriched software professionals with a formidable guide to exploit model-driven approaches to improve software development practices. Egyed [20] pioneering research was devoted to inconsistent correction in UML design models. The research, performed with much eagerness and supported by novel analysis and techniques, has tackled a very important challenge and provided truly valued methods that help to enhance precision and efficiency in UML-based software design. Egyed's results represented a serious step toward more reliable and accurate modeling.

one of the comparable studies to the present research was conducted by [21], who sought to conduct a Systematic Literature Review (SLR) on process models for listing the factors effective on the understandability. They searched for the studies between 1995 and 2015, and they selected 45 articles for full review. However, it should be noted that they only analyzed the empirical studies, while this current study reviewed both theoretical and experimental studies. The other study was also carried out as a SLR, but it focused on software visualization evaluation [22]. A similar study was carried out by performing a SLR on the studies whose main concern was the quality of models to gain insight into the meaning of quality for the models and the methods to enhance it [23].

El-Attar and Miller [24] studied the quality of use case models by applying SLR. In the study conducted by Budgen et al. [25], the authors adopted SLR to reveal how widely the notation of UML is used by addressing its advantages empirically. Haug et al. [26] conducted a SLR on 21 empirical studies on model-driven development (MDD) with UML to gain an

overview of the state-of-the-art in MDD with UML empirical research. A SLR was used in a study by Lucas et al. [27] to discover the consistency conceptions, proposals, problems, and solutions of various models. Fernández-Sáez et al., [28] also conducted a SLR study in which they focused on the maintenance of UML diagrams. As seen in the current literature, there are not many sytemmatic studies conducted to understand the topics and trends related to the quality and understandability of UML diagrams. Accordingly, this study aims to provide some insights into literature from this perspective. [32] [33]. What makes this study add to the existing systematic reviews is not so much the broadening of the time period covered in the literature search, but the convergence of the five contributions that have never been present simultaneously in any of the other systematic reviews. Genero et al. [16] review 266 studies published between 1997 and 2009 according to a predefined five-dimensional classification scheme (quality type, evidence type, research result, diagram type, research objective). Their classification is deductive and treats quality in its general sense as opposed to the quality-and-understandability composite used in our review. Dikici et al. [21] reviewed 45 empirical studies published between 1995 and 2015 but limit the focus of their review to process models and UML behavioral diagrams. Class, use-case and structural diagrams were excluded from the scope of their review. Fernández-Sáez et al. [28] reviewed 38 empirical papers published between 1997 and 2010 but restricted their scope to maintenance, treating understandability only as a prerequisite of maintainability. Whereas Budgen et al. [25] reviewed 49 papers till 2008 using a very broad six category classification framework and did not consider quality and understandability together. Hence, there are five main aspects where the current study is different. First, an inductive bottom-up taxonomy of 16 perspectives is extracted from systematic thematic analysis of 227 primary articles, as opposed to a pre-defined categorization framework used previously. Some of the perspectives highlighted by this taxonomy are those which could not be discovered by the previous research, such as Pedagogical Strategy, Visualization/Interactions, Model Transformation, and System Structure perspectives. Second, the unified framework of quality and understandability is applied using ISO/IEC 25010 as a starting point. Third, we consider both theoretical and empirical articles while [21], [25], and [28] limited themselves to empirical studies only. Fourth, we take into account all types of UML diagrams while [21] considered only process models and UML behavioral diagrams. Fifth, we performed an analysis of co-occurrence and inter-topic network that proves the integrative structure of the research area and reveals critical under-researched topics like System Structure (0.88%) and Model Transformation (0.44%) and emergence of Pedagogical Strategy (3.96%) due to gamification, Large Language Model-supported feedback, and remote-learning tools. Taken together, these results make our study complementary to the existing literature.

3. Methodology

In the present study, systematic mappingstudy was conducted to identify the topics studied under the quality and understandability of UML diagrams. Boland, Cherry, and Dickson [29] define systematic mapping study as a literature review used for identifying, evaluating, and synthesizing the most appropriate contents from which the answer to research questions can be derived.

In framing the research question, population, intervention, comparison, outcome, context criteria (PICOC) were applied, in which we have the following categorization of the key terms of our research question:

Population: software or software modeling or modeling language

Intervention: Unified Modeling Language.

Comparison: N/A Outcomes: understandability or quality or topic trends.

The question that was addressed in the present paper is as below:

From which perspectives are the quality and understandability of UML diagrams studied during 2000 to 2026?

Research Steps

The following research steps were performed to create the corpus of this study. As seen in Figure 1, the search string was created to address the studies conducted on the quality and understandability of UML diagrams. In search string creation, we followed the following approach:

In creating the search string Population, Intervention, Comparison, Outcomes and Context suggested by Kitchenham and Charters [30] was adopted. During the creation of search string, to cover all possible derivation of the words and alternative, synonyms or related terms, and the keywords were considered.

("understand*" OR "Read*" OR "Compreh*" OR "mental*" OR "cognit*" OR "Aesth*" OR "quality*") AND ("Unified Modeling Language" OR "UML")

On August 25, 2026, the search string was used in Web of Science, Scopus, IEEE Xplore and ACM. The search for the term gave us 2247 conference papers and journal articles in the case where the search was conducted using title, abstract and keywords and it is important to mention that in Scopus, we limited the search results only to those areas which included computer science, engineering, mathematics, decision sciences, social sciences, business, management, accounting, and psychology. On the other hand, in WoS, we narrowed down the search to areas, namely, Computer Science Software Engineering, Computer Science Theory & Methods, Computer Science Information Systems, Engineering Electrical & Electronic, Computer Science Artificial Intelligence, Computer Science Interdisciplinary Applications, Telecommunications, Computer Science Hardware & Architecture, Operations Research & Management Science, Engineering Multidisciplinary, Engineering Industrial, Education Scientific Disciplines, Automation & Control Systems, Computer Science Cybernetics, Information Science & Library Science, Management, Business, Engineering Manufacturing, Multidisciplinary Sciences, Education Educational Research, Imaging Science & Photographic Technology, Robotics and Logic.

Afterwards, duplicates studies were removed using conditional formatting, duplicate values function of Excel, which was applied on the title of the articles. Next, the screening was applied between 27 August to 5 September 2026, first based on the articles title, the abstract, the irrelevant studies were eliminated. An excel file was used to keep the extracted information from the articles, especially in answering the research question and holding the meta-data (e.g. authors, publication year, key words and DOI). Finally, the information in the Excel file (the columns related to research question), was analyzed and synthesized to derive general conclusions about the findings.

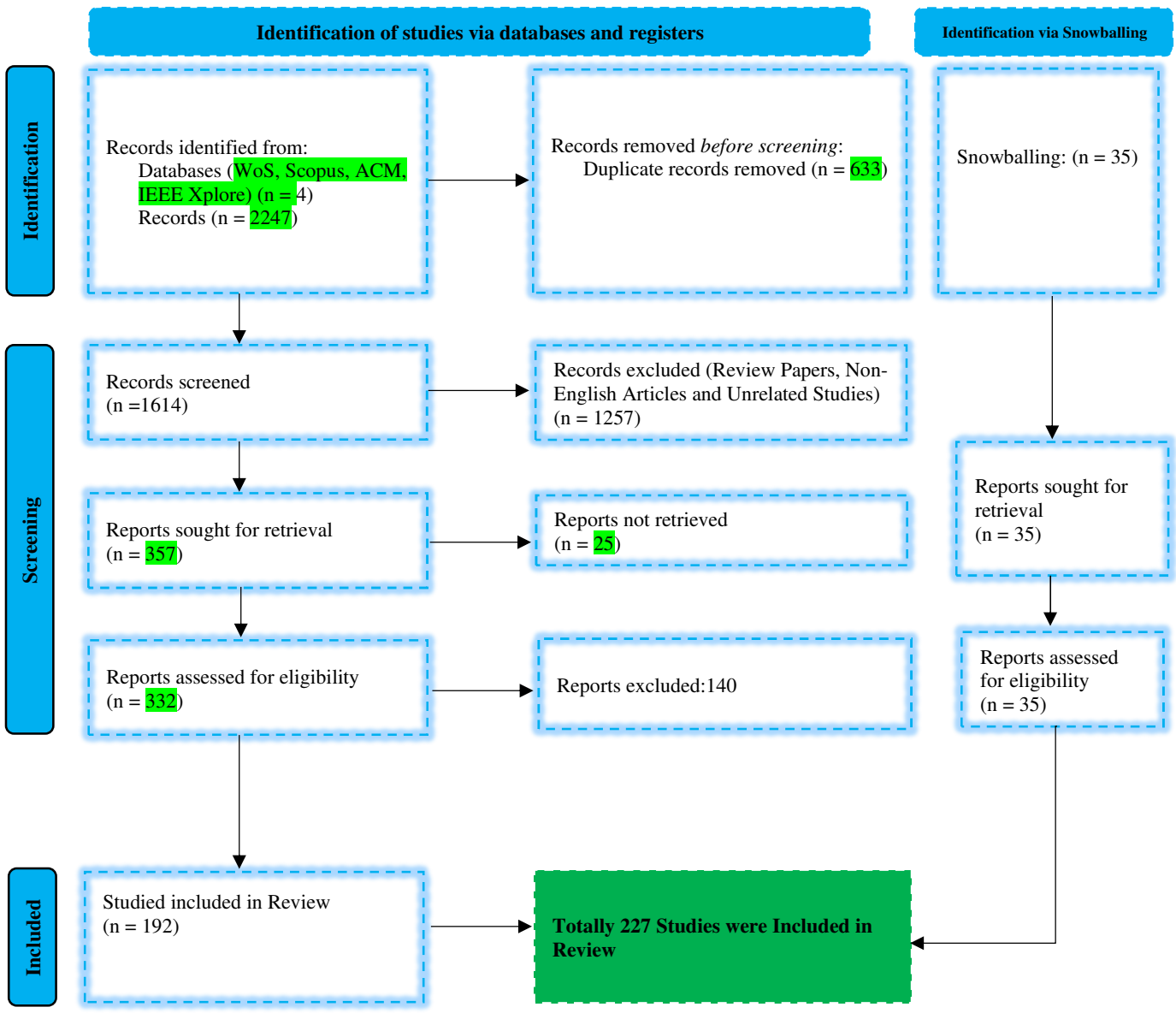


FIGURE 1: Systematic mapping study flowchart diagram

The inclusion of studies was done according to PRISMA 2020 criteria, which is shown in Figure 1. In the first step of identification, a total number of 2,247 studies were found in Web of Science, Scopus, IEEE Xplore, and ACM databases. By excluding 633 duplicates, 1614 studies were included in the next step called screening. In the screening step, title and abstract were read and 1257 studies were excluded (reviews, non-English studies, and those which were not related to UML quality or understandability). The search was continued for the other 357 full-text reports, 25 of which were inaccessible. These 332 full-text reports were evaluated against the inclusion and exclusion criteria, and as a result, 140 reports were excluded from the study. Then, in order to supplement the automated database search and prevent any omission of the potential studies not covered by the search string, we conducted the backward snowballing procedure. Within the backward snowballing procedure, we analyzed the references of all 192 primary studies in order to find more potentially relevant studies. Within the forward snowballing procedure, we utilized Google Scholar to identify studies that cited any of the primary studies. All procedures were performed iteratively, and all newly found studies were

reviewed according to the same inclusion and exclusion criteria stated below. As a result, we got the final set of 227 studies.

Inclusion Criteria:

- Year of publication after 2000
- English-language article
- Publications that investigate the understandability or quality of UML diagrams
- Publications that investigate either conceptual models or process models of UML notation
- Publications that study merely UML diagrams quality and understandability, rather than impact of UML quality and understandability on software or the user satisfaction with UML modeling tools

Exclusion Criteria:

- Articles without journal, issues, etc., information
- Articles do not explicitly state their adopted methods in Abstract section, especially the experimental studies which do not provide details of their controlled experiments in Abstract section.
- Articles focused on refactoring and resolving code smells

For Meline [31], each study has its own research questions and objectives, so the formulated inclusion and exclusion criteria are unique. Generally, the inclusion and exclusion criteria belong to categories, including, exposure of interest, geographical location of study, participants, peer review, study population, type of publication, nature of the intervention, outcome variables, time period, cultural and linguistic range, and methodological quality.

In justifying the present study inclusion and exclusion criteria, the categories specified in Table 1 can be assigned to the applied criteria on the extracted studies

TABLE 1: Categorization of inclusion and exclusion criteria.

Inclusion and exclusion criteria	Category
Year of publication after 2000.	time period
English-language article	cultural and linguistic range
Publications that investigate both conceptual models and process models of UML notation.	peer review
Articles without journal, issue, etc. information.	peer review
Articles do not explicitly state their adopted methods in Abstract section, especially the experimental studies which do not provide details of their controlled experiments in Abstract section.	peer review and methodological quality
Articles published at conferences and in journals were included.	type of publication

Quality Assessment of Primary Studies

Following the guidelines proposed by Kitchenham and Charters [30] and the approach adopted by Dyba and Dingsøyr [34], we conducted a formal quality assessment of the 332 primary studies. Three researchers independently evaluated each study against 12 criteria adapted from Dyba and Dingsøyr [34] (See Table 2). Each criterion was scored dichotomously ("Yes"/"No"). Inter-rater agreement was measured using Cohen's Kappa [35], yielding a value of 0.81, indicating "almost perfect agreement" [36].

TABLE 2: Quality Assessment Criteria for Primary Studies

ID	Quality Criterion	Dimension
	Screening Criteria	
Q1	Is the paper based on empirical research (rather than merely "lessons learned")?	Rigor
Q2	Is there a clear statement of the aims and objectives?	Rigor
Q3	Is there an adequate description of the research context?	Rigor
	Research Design Criteria	
Q4	Was the research design appropriate to address the aims?	Rigor
Q5	Was the recruitment strategy appropriate?	Rigor
Q6	Was there a control group or baseline for comparison?	Rigor
Q7	Were appropriate data collection methods used and described?	Rigor
Q8	Was the data analysis sufficiently rigorous?	Rigor
	Credibility Criteria	
Q9	Was researcher bias or reflexivity considered?	Credibility
Q10	Is there a clear statement of findings with justified conclusions?	Credibility
Q11	Are limitations discussed explicitly?	Credibility
	Relevance Criteria	
Q12	Is the study of value for research or practice?	Relevance

After implementation of these criteria and quality assessment procedure, 227 articles were considered part of the corpus. These 227 articles are downloaded as full text and analyzed as the corpus of this study. When the publication years of these articles are analyzed, as seen in Figure 2, an increasing trend in the number of publications considering 5-year periods can be observed, specifically a peak in the number of the publications can be observed for year range of 2015-2019.

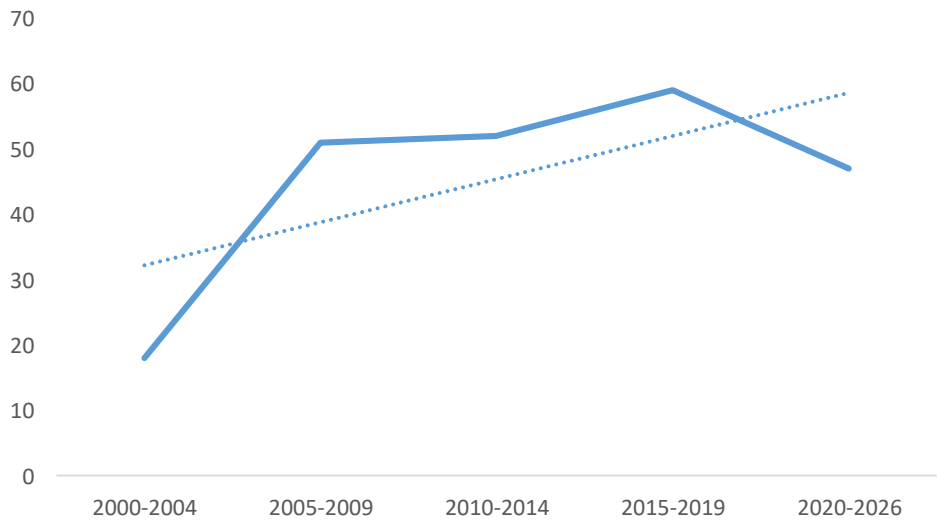


FIGURE 2: Yearly Distribution of Articles.

Study protocol

The study protocol serves as a blueprint for the entire review and is essential for ensuring transparency, replicability, and the minimization of bias. It should be noted that the review protocol was developed prior to conducting the systematic review, following the established guidelines proposed by Kitchenham and Charters [30]. The protocol specified the

research question, search strategy, inclusion and exclusion criteria, quality assessment procedures, and data extraction forms. In the present study, the following components need to be clarified for the adopted protocol throughout the research. As a database to be searched, Web of Science, Scopus, IEEE Xplore and ACM were selected, and in the query designed for the mentioned indexing, the key terms incorporated are as follow: "understand", "compreh", "cognit, "aesth", "read", "mental", "cognit", "quality", "unified modeling language" and "UML". During the screening process, priority was given to study the articles titles and abstracts, to find out the answer to the research question, and the articles missing answer to the research question, were fully studied. Then, for the purpose of providing the replication package and to show the authenticity of the results obtained, all extracted data were included in an Excel file. Finally, the results and findings of the review studies were synthesized and presented.

4. Results

The study's corpus was analyzed to determine the topics studied for the quality and understandability of UML diagrams. The thematic categories used in this study, namely Modelling, Design, Evaluation, Representations, Testing, Guideline, Software Engineer, Constraints and Requirements, Diagram Type, Quality Assessment, Redundancy, Refactoring, System Structure, Pedagogical Strategy, Visualization/Interactions and Model Transformation were derived through a systematic thematic coding process based on the abstracts, keywords, and methodology sections of the primary studies. Moreover, categories such as Pedagogical Strategy and Diagram Type were more specific to our dataset and emerged inductively. Totally sixteen topics, were identified in this scope.

The sixteen perspectives are descriptive categories that are inductively formed and are not theoretical constructs. With regard to Table 3, to make them theoretically base, without losing their inductiveness, we have tried to align each perspective with three existing theoretical frameworks: SEQUAL [39], ISO/IEC 25010 [40], and Physics of Notations of Moody [41]. The alignment results are presented in Table 3. It should be pointed out that the perspectives cannot be aligned one-to-one with any of the frameworks since they are based on a three-level structure (meta-structure): some perspectives express quality goals (Testing, Evaluation, Quality Assessment, Constraints and Requirements, Redundancy, System Structure), some express quality means (Design, Modeling, Representations, Refactoring, Model Transformation, Visualization/Interactions, Guideline), and some express context-related aspects (Software Engineer, Pedagogical Strategy, Diagram Type).

TABLE 3: Theoretical anchoring of the 16 perspectives.

Perspective	SEQUAL dimension	ISO/IEC 25010 characteristic	Moody's principle
Modeling	Semantic (validity)	Functional suitability	-
Design	Semantic (validity)	Functional suitability; Maintainability (modularity)	-
Representations	Physical; Syntactic	-	Semiotic clarity; Semantic transparency; Visual expressiveness
Visualization/Interactions	Physical	Usability (aesthetics, operability)	Perceptual discriminability; Graphic economy; Cognitive integration
Diagram Type	Syntactic; Pragmatic (comprehension)	-	Cognitive fit

Redundancy	Syntactic	-	Semiotic clarity (symbol redundancy/overload)
Constraints and Requirements	Semantic (validity, completeness)	Functional correctness; Completeness	-
Testing	Semantic (validity); Empirical	Functional correctness	-
Evaluation	Empirical	Maintainability (analysability)	-
Quality Assessment	Empirical; Social	Maintainability; Reliability	-
Refactoring	Semantic	Maintainability (modifiability, modularity)	Complexity management
System Structure	Semantic	Maintainability (modularity)	Complexity management
Model Transformation	Semantic	Maintainability; Portability	-
Guideline	Social; Deontic	-	(Prescriptive; all principles)
Software Engineer	Pragmatic (audience)	Usability; Human factors	-
Pedagogical Strategy	Social; Pragmatic	-	-

The definition of the sixteen topics is as below:

Modeling: mathematically generating simplified representations that convey entities, relationships, and behaviors of real-life systems.

Design: design in UML involves the creation of visual renderings that describe the structure, behaviors, and interactions of the various components in a software system. UML diagrams are an important part of transforming abstract requirements into concrete design solutions.

Evaluation: evaluation assesses UML diagram quality and effectiveness, checking whether diagrams accurately reflect the intended system, are comprehensible, and successfully carry the essential information to stakeholders.

Representations: representations relate to how UML diagrams portray various aspects of a software system. In these diagrams standardized symbols are employed to visually illustrate classes, objects, relationships, behaviors, and other system elements.

Testing: The testing of UML diagrams involves the confirmation of diagram accuracy against the actual software implementation. The procedure in testing ensures that diagrams represent the behavior and structure of the system faithfully.

Guideline: guidelines are the best practices for creating coherent, clear UML diagrams. The use of guidelines can allow the enhancement of diagram quality and understandability by availing consistent approaches.

Software Engineer: a software engineer is responsible for designing, developing, testing, and maintenance of software systems. The educational and experience level of the software engineer develops the quality of the UML diagram.

Constraints and requirements: Constraints refer to operational boundaries, and requirements are specifications for the operation of a software system.

Diagram Type: UML notation possesses 14 diagram types, such as class, sequence, and activity, based on which this modeling language is able to provide diverse visualizations of distinct system facets, enabling multiple perspectives for understanding.

Quality Assessment: Quality assessment appraises the effectiveness of UML diagrams by clarity, following the best practices, and how well they communicate to the stakeholders.

Redundancy: in UML diagrams, it denotes the avoidable repetition of non-essential elements during design which are capable of misleading the stakeholders.

Refactoring: Refactoring is a process of restructuring which enhances readability, maintainability, and understandability of UML diagrams.

System Structure: system structure in UML diagrams shows the arrangement of software components Which encompasses class hierarchies, module relationships, and architectural elements.

Pedagogical Strategy: It refers to the structured plan or approach that educators use to teach students how to understand, create, analyze, and apply software models (such as UML diagrams, flowcharts, or domain-specific modeling languages) effectively. It encompasses the methods, techniques, and tools chosen to facilitate learning, skill acquisition, and cognitive engagement with modeling concepts.

Visualization/Interactions: this view examines the impact of diagram layout effectiveness, animations, interactivity, visualization enhancements, and virtual reality/3-D on modeler performance and comprehension. Research indicates that the use of better layouts, dynamic visualization, and interactive navigation increases comprehension accuracy while decreasing time taken to complete the task.

Model Transformation: this critically underexplored perspective examines how quality-driven transformation rules and decisions, particularly the prioritization of association relationships, affect the structural quality, maintainability, and understandability of the resulting UML models.

In order to check the validity of thematic categorization procedure, 227 primary studies were separately categorized into sixteen topics by three reviewers. Every study was assigned to one or more than one of the primary topics depending on its abstract, keywords and methodology part. The degree of interrater reliability was calculated using the Fleiss' Kappa coefficient that can be used to calculate the degree of agreement among three or more raters on categorical variables [38]. The value of Fleiss' Kappa calculated for this particular case is $\kappa = 0.79$, indicating 'substantial agreement' according to Landis and Koch [36] scale.

Considering the research question "Which topics are studied for the quality and understandability of UML diagrams during 2000-2026?", it can be inferred it was sought to categorize the papers within 2000-2026 in terms of the perspectives, from which, the authors have studied the quality and understandability of UML diagrams, implying that those perspectives in the first glance are disconnected, while in order to develop more quality and understandable UML artifacts, we need to consider them during the analysis and design phase of software development. That is, they need to be considered as a whole to contribute to the quality of UML diagrams.

The number of articles in the corpus addressing the related topic (n) and their percentage of the whole corpus (%) are also tabulated in Table 4. Here, the keywords are assigned by reading each article classified under each topic. For this classification purpose, three researchers studied individually, and the data is compiled in a single document by resolving the inconsistencies.

TABLE 4: Studied topics on quality and understandability of UML diagrams.

Topic	Sub-topics	n	%
Quality Assessment	[S10], [S74], [S77], [S90], [S117], [S119], [S125], [S137], [S143], [S144], [S148], [S154], [S157], [S166], [S170], [S172], [S179], [S180], [S181], [S187], [S188], [S195], [S196], [S198], [S204], [S206], [S208], [S212], [S214], [S216], [S218], [S221], [S222], [S223], [S224], [S225]	45	**19.82%**
Representations	[S6], [S7], [S28], [S29], [S46], [S51], [S64], [S69], [S79], [S87], [S109], [S113], [S134], [S139], [S142], [S150], [S155], [S171], [S177], [S178], [S186], [S191], [S197], [S205], [S207], [S209], [S219], [S227]	28	**12.33%**
Design	[S84], [S94], [S102], [S131], [S1], [S11], [S43], [S52], [S77], [S123], [S61], [S72], [S88], [S45], [S54], [S58], [S85], [S104], [S105], [S106], [S110], [S115], [S122], [S136], [S145]	25	**11.01%**
Modeling	[S22], [S96], [S38], [S49], [S57], [S2], [S3], [S19], [S38], [S31], [S35], [S55], [S67], [S78], [S81], [S98], [S100], [S108], [S112], [S121], [S132], [S158], [S202], [S213]	24	**10.57%**
Testing	[S27], [S56], [S65], [S70], [S73], [S91], [S120], [S135], [S21], [S68], [S71], [S42], [S116], [S173]	15	**6.61%**
Software Engineer	[S39], [S20], [S41], [S14], [S124], [S127], [S44], [S133], [S111], [S12], [S15], [S33], [S147], [S160], [S189], [S199]	15	**6.61%**
Guideline	[S34], [S92], [S82], [S37], [S60], [S63], [S99], [S101], [S163], [S164], [S165], [S176], [S200], [S220]	15	**6.61%**
Pedagogical Strategy	[S118], [S128], [S129], [S4], [S44], [S130], [S159], [S162], [S167], [S169], [S184], [S190], [S217]	15	**6.61%**
Refactoring	[S5], [S18], [S24], [S107], [S114], [S175], [S182], [S203], [S210], [S226]	13	**5.73%**
Constraints and Requirements	[S94], [S32], [S17], [S28], [S36], [S126], [S133], [S141], [S156], [S183], [S194], [S201], [S211]	13	**5.73%**
Evaluation	[S25], [S26], [S86], [S4], [S93], [S9], [S53], [S16], [S76], [S102], [S103], [S133]	12	**5.28%**
Visualization/Interaction	[S146], [S149], [S151], [S152], [S153], [S168], [S185], [S192], [S215]	12	**5.28%**
Diagram Type	[S40], [S75], [S83], [S95], [S62], [S138], [S140], [S161]	11	**4.85%**
Redundancy	[S47], [S13], [S89], [S66], [S174]	5	**2.20%**
System Structure	[S50], [S59];	2	**0.88%**
Model Transformation	[S193]	1	**0.44%**

4.1. Inter-topic Relationships and Co-occurrence Evidence

In order not to state the relationships between the sixteen topics without any justification, we considered the topic allocations in the final stage as a bibliometric co-occurrence network. It was possible that one primary study could allocate to several topics if it helped form more than one perspective. For 227 primary studies, there were 242 allocations for the 16 topics. Next, a 16 × 16 co-occurrence matrix was established, with the Jaccard similarity coefficient calculated as:

$$J = \frac{n_{ij}}{n_i + n_j - n_{ij}}$$

where n_{ij} is the number of studies assigned to both topic i and topic j, and n_i and n_j are the total numbers of studies assigned to topics i and j, respectively. The full matrix is provided in Supplementary Data 3. Table 5 reports the top co-occurring topic pairs.

TABLE 5: Top inter-topic co-occurrences based on multiple topic assignments.

Topic pair	Co-occurring study IDs	n_{ij}	Jaccard J
Constraints and Requirements - Evaluation	S133	1	0.0417
Testing - Diagram Type	S75	1	0.0400
Software Engineer - Evaluation	S133	1	0.0385
Pedagogical Strategy - Evaluation	S4	1	0.0385
Software Engineer - Constraints and Requirements	S133	1	0.0370
Software Engineer - Pedagogical Strategy	S44	1	0.0345
Design - Evaluation	S102	1	0.0278
Design - Constraints and Requirements	S94	1	0.0270
Design - Quality Assessment	S77	1	0.0167

For example, [S102] is classified under Design and Evaluation, [S94] under Design, Constraints and Requirements, and Quality Assessment, and [S133] under Software Engineer, Constraints and Requirements, and Evaluation. This is an illustration that the co-occurrence provides empirical evidence for the claim that quality and understandability are not considered separately but rather together.

The yearly distribution of the articles in five-year periods is given in Figure 3 and Supplementary Data 1. As seen from this figure, the topic Quality Assessment is mostly studied and has a higher (**19.82**%) percentage, particularly during 2005-2009 and 2010-2014, which is followed by, Representations, Design, Modelling, Testing, Software Engineering, Guideline, Pedagogical Strategy, Refactoring, Constraints and Requirements, Evaluation, Visualization/Interaction, Diagram Type, Redundancy, System Structure and Model Transformation. In fact, over the last 6 years (2020-2026), issues such as Guideline and System Structure have not been researched at all (zero frequency). At the same time, issues such as Evaluation and Guideline, Constraint and Requirements, Diagram Type, Redundancy, a System Structure, Visualization/Interaction, and Model Transformation are among the least researched because of the appearance of a new need to model the new field , software and systems enabled with machine learning and artificial intelligence. Therefore, bibliometric analysis of 227 papers revealed that the field of software modeling now includes quite a high proportion of research focused on Quality Assessment (19.82% of the papers). It is a sign of the maturity of the research field because it concerns more the assessment of structural integrity, complexity, and maintainability of the model than the process of modeling itself.

The emphasis placed on them is quite valid in view of the consistent research conducted in the areas of Representations (12.33%), Design (11.01%) and Modeling (10.57%). The point is that researchers continue improving the languages and notations which are being examined. The equal weight given to Guideline (6.61%) and Pedagogical Strategy (6.61%) indicates the trend towards practical implementation of knowledge obtained. The attempt is made to translate theoretical ideas into practice, which will be applied both by beginners and professionals. Contrary to that, topics like System Structure (0.88%) and Model Transformation (0.44%) are rather insufficiently explored.

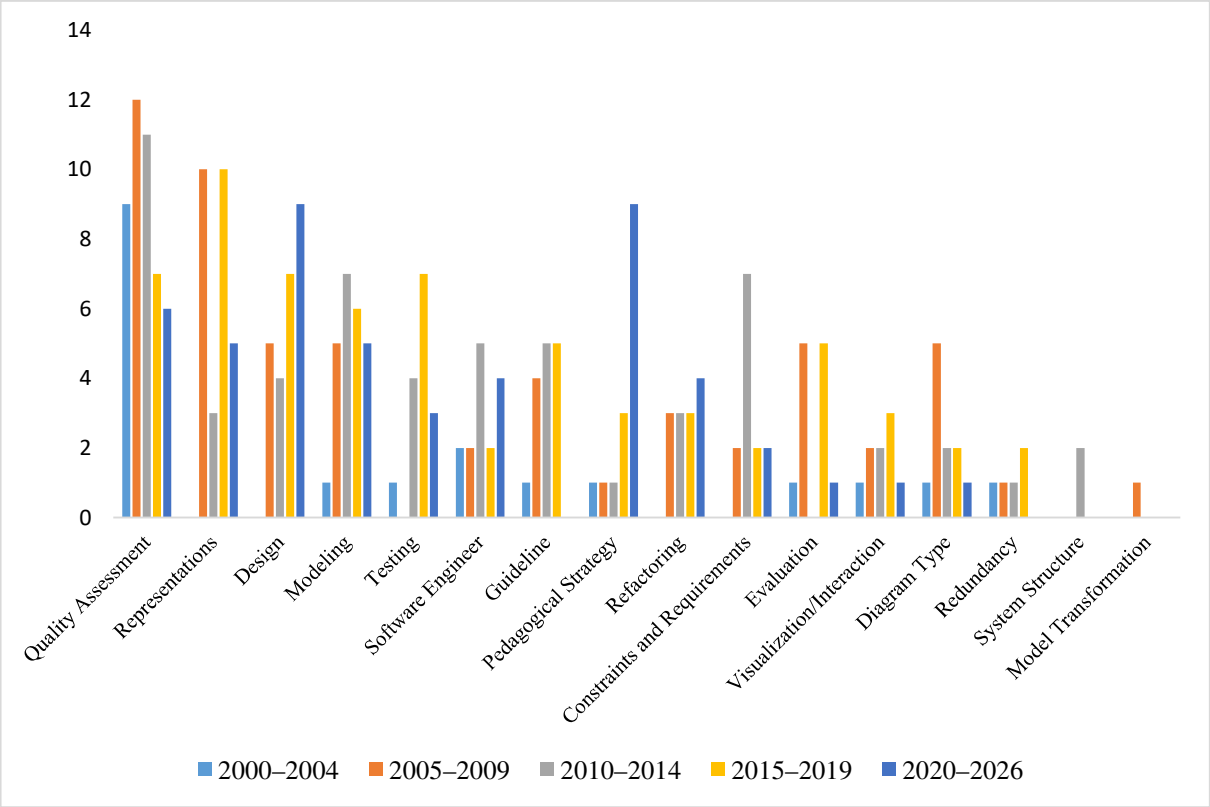
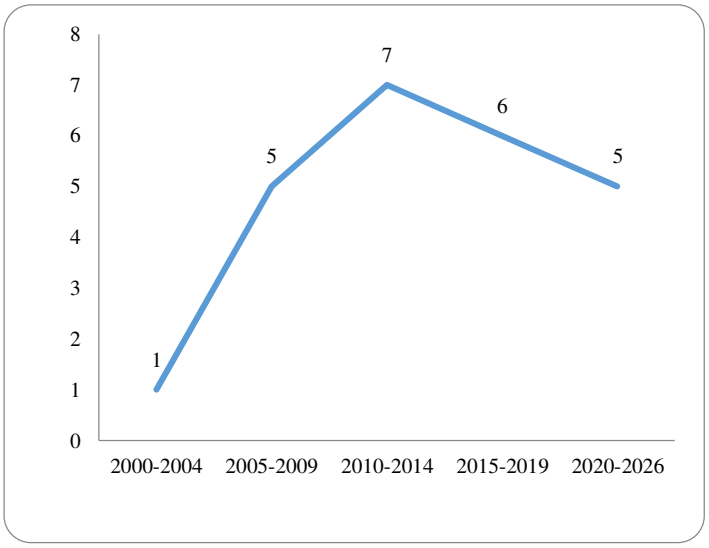


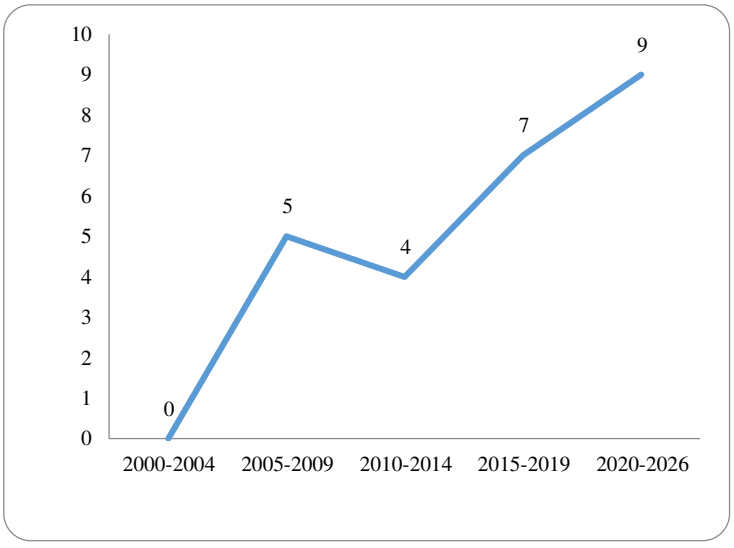
Figure 3: Topic Trends.

According to Supplementary Data 2 and Figure 3, time series analysis on the 227 classifications shows a varying but constant body of literature, with the highest classifications in 2005-2009 (25.55%) and 2015-2019 (28.19%). The last time series, 2020-2026, is an incomplete one and ends in 2026 owing to the search cut-off date of 2026. This time series' share in percentage is 22.02%, while its rate on annual basis is 7.14 papers per year or 6.71 papers per year in case of taking the year 2026 into account as a full year. It is less than the 2015-2019 rate of 11.80 papers per year.

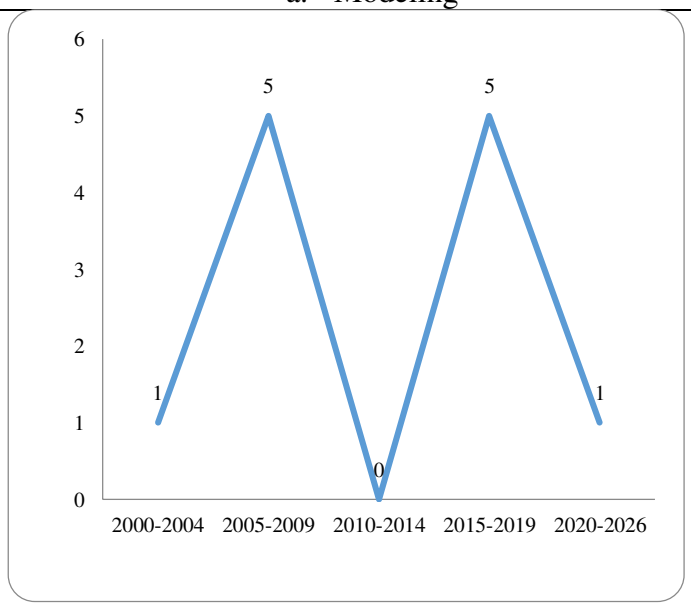
The major development is the inclusion of Pedagogical Strategy, which has risen from a very minimal share in previous time periods to 3.96% in the partial time period of 2020-2026.



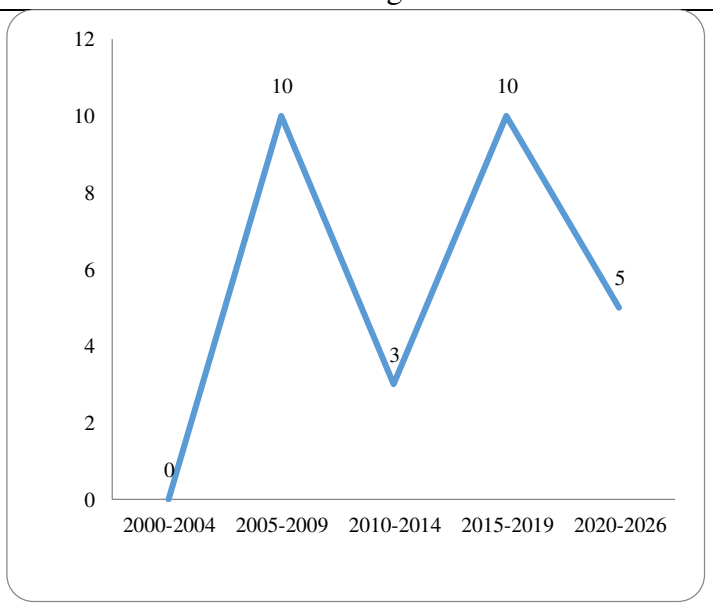
a. Modeling



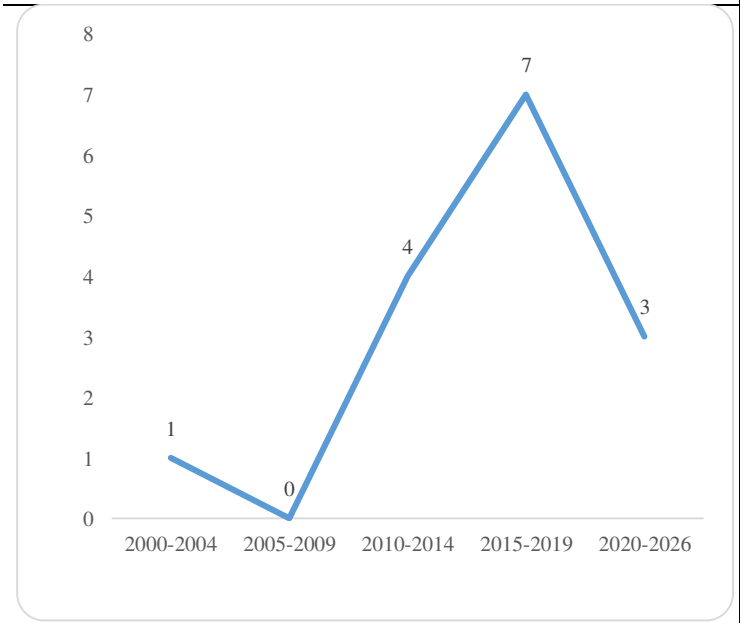
b. Design



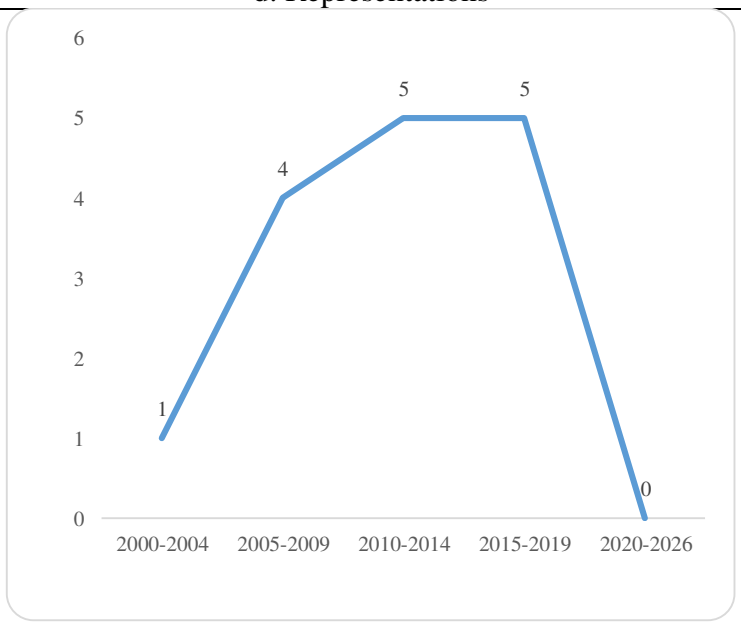
c. Evaluation



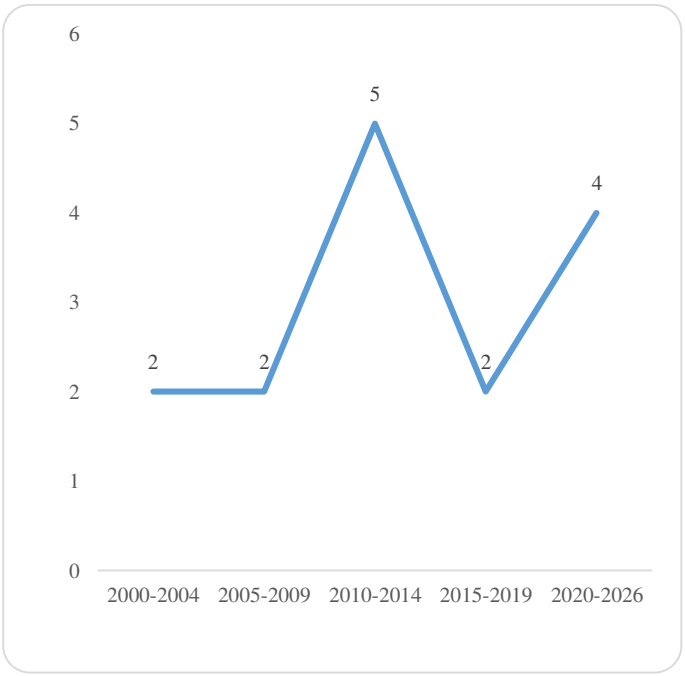
d. Representations



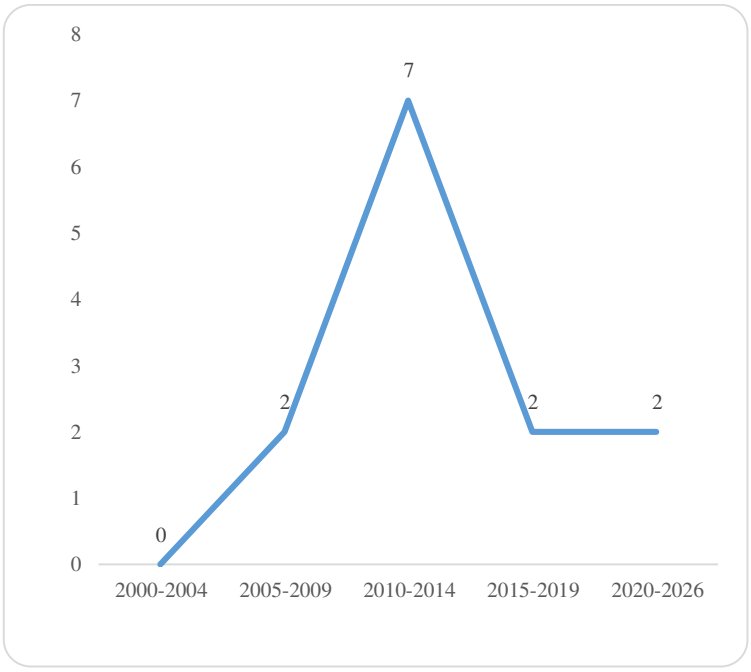
e. Testing



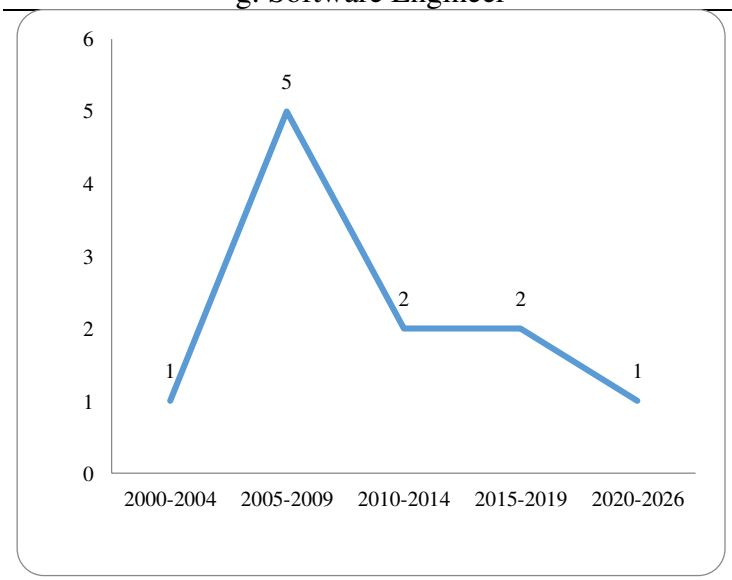
f. Guideline



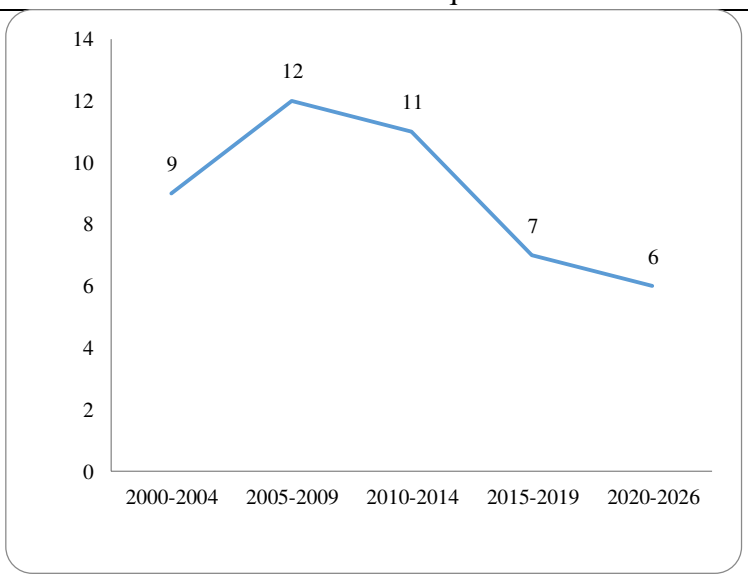
g. Software Engineer



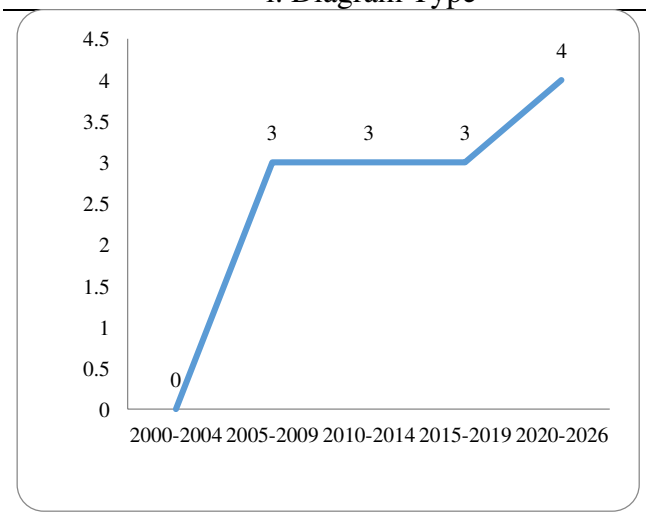
h. Constraints and Requirements



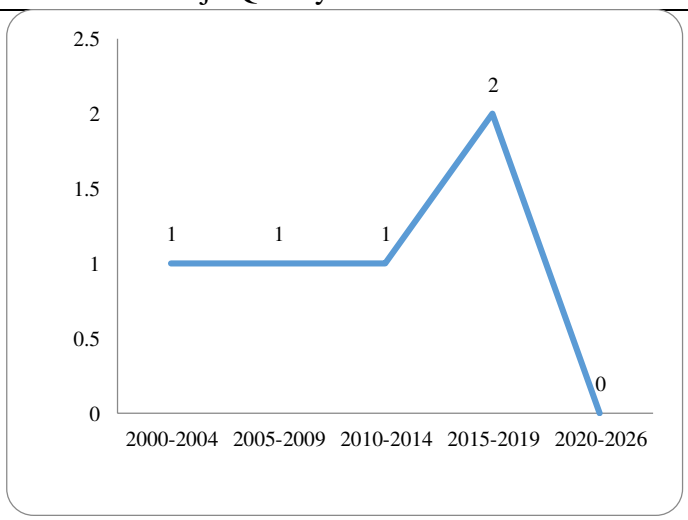
i. Diagram Type



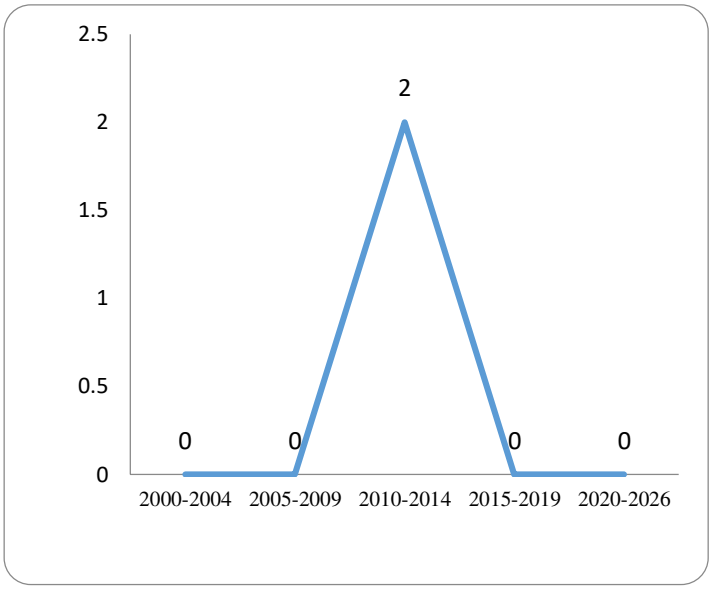
j. Quality Assessment



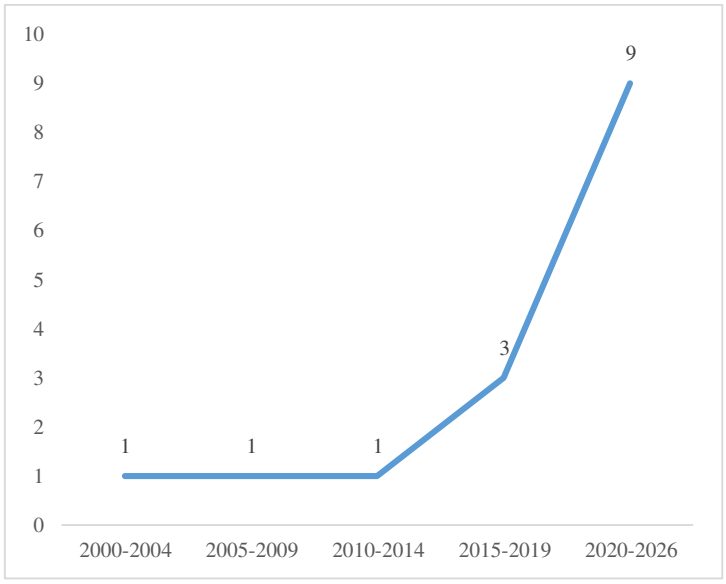
k. Refactoring



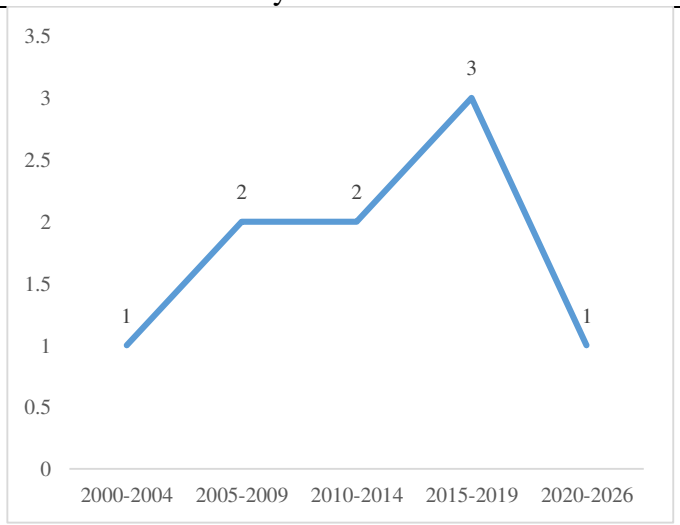
l. Redundancy



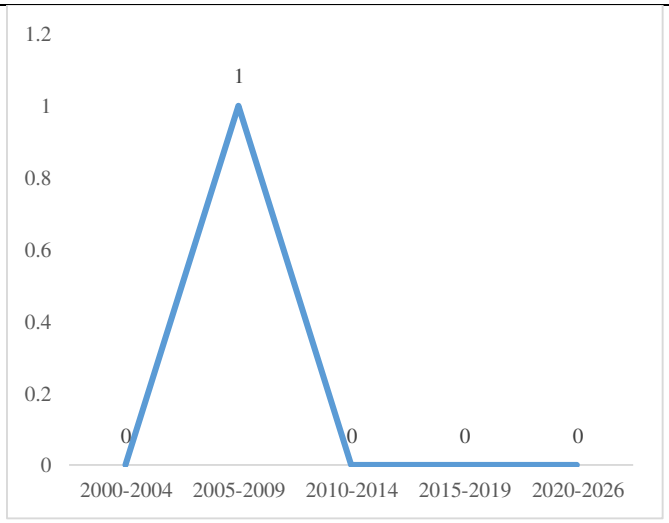
m. System structure



n. Pedagogical Strategy



o. Visualization/Interactions



p. Model Transformation

FIGURE 4: Distribution of topics across periods

To avoid misinterpreting the incomplete last period's artifact as a real decline, all period numbers have been adjusted to annualized article publication rates. The number of articles in the dataset is 227 articles across the five periods, namely 18 articles from 2000 to 2004, 51 articles from 2005 to 2009, 52 articles from 2010 to 2014, 59 articles from 2015 to 2019, and 47 articles from 2020 to 2026. As the period of 2020-2026 includes only a partial year 2026 because of the cut-off date being 25 August 2026, we consider that the length of this period is around 6.58 years, which is six years and seven months. Annualized rates are 3.60, 10.20, 10.40, 11.80, and 7.14 articles per year, respectively. In case the partial year of 2026 is taken as a full year for sensitivity analysis, the rate will be 6.71 articles per year.

Anyhow, the rate for 2020-2026 is provisional and below the peak rate for 2015-2019; thus, the decrease is an artifact. However, the trend from 2000 to 2019 is positive. Furthermore, Figure 4 presents the distribution of 16 topics during the five periods in a descriptive way. As trend analysis is done on the basis of only five time periods and some of the topics lack enough data or zeros in many time periods, trend analysis is not done /has not been done formally using statistics. Thus, trends have to be viewed in a descriptive way rather than statistically significant

ones. It is even more so the case of topics like Redundancy and System Structure that lack enough data for trend analysis.

Next, the results of the articles addressing each topic and sub-topics are summarized and synthetized according to the keywords used for the classification of each topic.

Quality Assessment

Four articles contribute to the Quality Assessment Perspective, presenting insights into UML diagram quality. A method for security model assessment demonstrates efficiency in producing high-quality models, fostering correctness, semantic completeness, modifiability, understandability, and stakeholder communication [S10]. A quality model-based approach gauges meta-model quality across syntax, semantics, pragmatics, evolvability, and capability [S74]. Systematic rule-based checking ensures time consistency in cruise control and guidance software [S77]. Using a hyperlink-induced topic search algorithm for class diagram evaluation yields hub weights, reflecting communication effects and supporting adjacency matrix adoption [S90]. As the new sub-topics of Quality Assessment, the scholars have focus on the role of software reverse engineering [S119] and measuring adaptability [S125]. In [S119], the authors state that quality of the models yielded by software reverse engineering is low, and not well-documented. To this end, they have proposed a knowledge-based ontological system (OntoRESec) for enabling the online quality assessment of converted UML structural design content. The study [125] came up with concept of self-adaptable software for software architecture, and adaptability in software architecture is defined as the degree to which architecture can adapt to changes. They define adaptability as a positive attribute in software architecture. These studies collectively highlight diverse strategies to assess and enhance UML diagram quality. In [S137], structural complexity and size measures (methodology, attributes, generalizations, dependency, hierarchy depth or height) have been recognized as relevant factors impacting the amount of work involved in maintaining UML class diagrams. In addition, [S143] has stated that consistency management approaches and quality-management techniques are highly important, as researches have neglected inter-model and semantic consistency issues while concentrating mainly on class, sequence, and state charts diagrams. [S144] has pointed out that quality-management approaches focusing on consistency, completeness, and defects management are extremely important, because it is easier and cheaper to take the initial actions aimed at improving quality. In turn, [S148] has shown that a more even distribution of cyclomatic complexity positively impacts understandability of executable UML models.

The study [S154] proved with meta-analysis that measures associated with associations and generalizations show a high correlation with cognitive complexity, which has a higher correlation with comprehensibility compared to modifiability. The study [S157] identified weak points for understanding data model associated with multi-value attributes, composite attributes, and weak entities. [S166] proved that most class diagram metrics (NAssoc, NAgg, NaggH, MaxHAgg, NGen, NgenH, and MaxDIT) serve as good maintainability indicators, except NDep, which is uncertain. [S170] focused on clone detection as the quality factor by proving that model clones are negative for the quality of models, just like code clones. [S172] introduced the software stability metrics on the model level as indicators of software stability. Both [S179] and [S180] confirmed the class diagram complexity metrics as maintainability indicators via the controlled experiment and prediction models. [S181] proved further that early structural metrics are applicable as class diagram maintainability indicators. [S187] explored the multimodal Large Language Model (LLM) assessment accuracy and proved that LLMs show promising partial score alignment with experts but lack semantic interpretation and evaluative reasoning.

[S188] validated automated UML diagram generation and multi-modal scoring, demonstrating excellent agreement between automated and manual evaluation approaches with

Pearson correlation r=0.82. [S195] presented the application of OOAD metrics according to the CMMI model for the evaluation of UML diagrams. [S196] provided a verification and validation methodology for UML diagrams to deal with problems of consistency, correctness, and quality of diagrams. [S198] suggested continuous quality assessment methodology for software artifacts. [S204] compared two common class diagram complexity metrics and concluded that statistical metrics have better classification accuracy than entropy and distance metrics. [S206] created domain-specific design metrics and rules for real-time UML diagrams. [S208] proposed an approach to deal with impacts of changes in UML diagrams. [S212] proposed a set of metrics derived from UML class diagrams adjusted for Java to measure complexity. [S214] compared experts and students in evaluating the quality of class diagrams in terms of six criteria: Understandability, Layout, Extensibility, Modifiability, Completeness, and Correctness. [S216] examined types of UML defects, consistency, completeness, and impact of UML defects on interpretation. [S218] suggested metrics methodology for business modeling quality assessment. [S221] indicated that metric counting techniques, attributes, generalizations, dependencies, and hierarchical depth or height influence the effort of maintaining UML class diagrams. [S222] suggested automatic quality assessment of UML-based software architectures by means of combined metrics. [S223] studied the problem of unnoticed defects in UML models and caused misinterpretation. [S224] studied quality dependencies among UML artifacts created by novice systems analysts. [S225] examined aesthetical and notation aspects of conceptual diagram understandability.

In the literature dealing with quality assessment, there are three common major topics covered: (1) measurement-based assessment of quality (n=18), which uses structural complexity measures (the number of methods, attributes, generalizations, associations, depth of hierarchy) as precursors for assessibility and maintainability [S137, S148, S154, S157, S179-S181, S204, S221], (2) automated verification and consistency checking (n=10), which focuses on consistency, completeness and defect detection in various types of UML diagrams, especially class, sequence and statechart diagrams [S143, S144, S166, S196, S208, S223] and (3) empirical evaluation of quality features (n=8), including use of controlled experiments and human evaluations to prove quality attributes, such as understandability, layout, extensibility, modifiability, and correctness [S214, S216, S224, S225]. A contradiction appears regarding the importance of cognitive complexity, whereas [S148] claims that better distribution of cyclomatic complexity is helpful to understandability of executable UML models, [S154] proves through meta-analysis that cognitive complexity correlates with comprehensibility than with modifiability.

Representations

In the Representations perspective, ten studies comprehensively assess UML representations, throwing some insight into various representation issues. Role-based regulations improve the understanding and importance of conceptual models [S6]. Iconic signs are more effective than symbolic signs in UML graphics [S7]. Non-functional requirement representation helps in requirement-to-design translation [S8]. Customization of process models is important for representation [S29]. Visual perception helps in mental model representation [S46]. Representation of physical quantities helps in safe system manipulations [S51]. Semantic name regulations improve representation of elements [S64]. Layout strategies improve vertically aligned class representation [S69]. Use case description checklist helps in implementation of UML adherence and quality [S79]. SORT approach refines UML representations for better understanding of models [S87]. Finally, it is argued that “behavioral issue of the system is of utmost importance to be represented” [S134], whereas existing models of domains tend to neglect behavioral representation of domains depicted through UML Activity and State Machine Diagrams. In overcoming this, a “new domain-specific language named AGL is proposed to formally represent the domain behaviors with a unified domain

model" besides having "a unified modeling technique for domain-driven software development." This work aims to fill the gap between models and code by developing better intellectual models that combine structural and behavioral models of domains by narrowing down the differences between models and code. This work helps move the current state of the art for Domain-Driven Design by developing intellectual models with both structural and behavioral models that tend to narrow down the differences between models and code. These observations improve our understanding of representation techniques for UML representation effectiveness. The authors [S139] have stated that action language notation and the use of UML notation in addition to the programming code made action languages more understandable than object-oriented programming languages. [S142] found that combination of UML and B notations is more comprehensible than just B models, and the role of familiarity with the notation and training is considerable. [S150] proved that Object-Process Methodology (OPM) is better than UML in modeling the dynamic part of web applications, as models created using OPM were of higher quality than those developed using UML. The asymmetry between the notation of on-entry behavior and on-exit behavior in UML statecharts has been mentioned in [S155], and it is very difficult to understand. [S171] has revealed that visualization of method invocations in reverse-engineered sequence diagrams increases the comprehensibility of program interaction. [S177] revealed behavioral regularities of reading UML diagrams by modelers; different reading strategies depend on the level of expertise and kind of diagrams but not their quality. [S178] found out that simulation-executable model notation would need further development to get closer to the actual software engineering process. [S186] proved that there are distinct approaches to diagram understanding used by modelers with experts using another approach than novices. [S191] supported that giving analysts the similarity between system model terminology and high-level artifacts leads to the improvement in system model vocabulary. [S197] listed visual characteristics of UML, notation deficiencies and visual improvements. [S205] showed that integration of semi-formal and formal notations accelerates comprehension tasks with accuracy even in case of minimal training. [S207] introduced a formal semantic model based on rewriting logic of UML to give UML diagrams accurate semantics. [S209] discussed representational clarity, discriminability, immediacy, expressiveness and graphic economy in modeling languages. [S219] proved that integration of semi-formal and formal notations accelerates comprehension tasks with accuracy even in case of minimal training. [S227] proved that UML-B models accelerate comprehension tasks compared to B or Event-B models.

The UML representation research focuses on, (1) notational efficiency, showing that iconic signs surpass symbolic signs [S7], an action-language notation is more comprehensible than traditional object-oriented programming language [S139], and semantic naming principles enhance element representation [S64], (2) modeling notations integration (n=8), in which integration of UML with B notation [S142], Object-Process Methodology [S150], or formal notations [S205, S219] facilitates understanding tasks and (3) modeler behavior and reading techniques (n=8), finding that experts and novices use different reading techniques, where regularities of behavior differ depending on the kind of diagram used rather than layout quality [S177, S186]. While some papers highlight the advantage of semi-formal and formal notations integration in terms of comprehension [S205, S219, S227], other works mention the need for further progress in creating a simulation-executable model notation [S178].

Design

Reviewed articles reveal key insights into the design perspective, in such a way that stereotypes [S84] tailor modeling language for domains, enhancing semantics. Anti-pattern detection ([S94] and [S102]), as well as pre-defined use case patterns [S131] through model transformation and already validated patterns improves use-case models. Architecture-level dependability [S1] evaluates components for reliable systems. Object Constraint Language

(OCL) post-conditions [S11] enhance behavior descriptions. User experience affects object diagrams' effectiveness [S43]. AI-driven design inconsistency detection ([S52] and [S77]) ensures completeness, while [S123] addresses sematic inconsistences. Detail level [S61] shapes comprehensibility. Automated UML change propagation [S72] maintains coherence. Transition techniques [S88] impact developers differently. Color and icons [S54] enhance readability. Novel cohesion metric [S58] aids assessment. Composite states [S85] offer hierarchical structure. Multiplicity [S104], versioning, merging conflicts [S105], and cross-diagram integration [S106] influence comprehension. Cooperative work [S110] impacts quality. Annotative variability [S115] improves comprehensibility. More recently, in the studies [S122] and [S136] scholars have focused on integration and usability, as well as UML profile for class diagram. For [S122], Well-designed UML class diagrams are a valuable resource for software design reuse. Therefore, integrating these diagrams is essential to enhance their modeling capabilities, with redundancy elimination being an additional benefit of such integration. In [S136], they showed how the UML profile integrates with classical UML diagrams to support the design of complex systems. Furthermore, the validation of the feasibility of their conceptual framework through a theoretical quality assessment and its implementation in an agroecology case study focused on monitoring autonomous agricultural robots. Generally, these sub-topics illustrate the complex relationship between design considerations and UML diagram functionality. In [S145], the authors have shown that the problem of structural quality enhancement of UML class diagrams by means of semantically equivalent transformations can be solved by structural improvement based on genetic algorithms and structural quality metrics. From the design point of view, four thematic clusters were revealed: (1) pattern-based design with stereotype [S84], anti-patterns [S94, S102], and predefined use case patterns [S131] contributing to modeling semantics and use case quality, (2) design inconsistencies detection via AI-based [S52, S77] and rule-based techniques [S123] for finding semantic inconsistencies; (3) design quality metrics (n=5) with an emphasis on cohesion [S58] and structural quality improvement by means of genetic algorithms [S145] and (4) cross-diagram integration (n=7) when versioning and conflicts merging [S105], and cross-diagram integration [S106] strongly affect comprehensibility.

Modelling

The Modeling perspective is widely investigated with a frequency of 21, encompassing a range of subtopics that significantly improve UML diagram quality. Aspect-oriented modeling (AOM) [S22], [S96] focuses on matters of concerns, improving issues of modularization. Object-process methodology (OPM) [S38] is better suited for dynamic web application modeling that outperforms UML. Open Modeling Language (OML) [S49] is an alternative for dynamic system behavior representation. Scenario- and goal-based modeling [S57] significantly improve understanding of requirements, where Tropos outperforms. Precise- and precision-style modeling [S2] differently influence understanding, with precise style helping developers. Domain models [S3] support context-specific abstractions with category lists contributing comprehensively to models. Helpful decomposition [S19] is effective for better modularity. Dynamic models [S38] are in line with the behavior of users. Stereotypes [S31] improve understandability, with part-whole relations requiring precision [S35]. Model slicing [S55] and Process Notations [S67] improve understanding. Multi-view refactoring [S78] keeps understanding consistent. Visual models [S81] improve cognitive effectiveness. Conceptual mental models [S98] support capture of requirements. OO-DFD and Usecase methods [S100] provide effective models for functional requirements of OO systems. Passive and Negative sentence analysis [S108] helps generate UML static models programmatically. Hierarchical Agglomerative Dynamic threshold-based Clustering improves

decomposition of classes [S112]. This study prepares the reader for the various models developed by researchers by highlighting the significant application of UML Diagrams. Recently, in the papers [S121] and [S132], the principal emphasis of research for scholars is centered on Automatic Backlogs Refining and Cross Application Domain and Cross Modeling Paradigm Model Reuse. In [S121], the authors showed that after refining their backlog, their approach automatically creates UML class and use case diagrams for each cluster of the backlog. It makes use of learning algorithms of machine learning along with natural language processing techniques. Specifically, they built a tool that automatically identifies the file of the user story, clusters the user story into actors, then applies the k-means unsupervised learning approach for developing clusters. In [S132] Software reuse is a widely recognized practice in software development that aims to minimize costs and efforts while improving quality and productivity. However, it creates challenges related to the recovery and utilization of artifacts developed in other application domains or models, such as data-driven models compared to object-oriented models. In [S158], the researchers found that completeness was achieved to a level of 71%, with 6% of redundancy, due to the weaving operation and modeling of the aspect-state machine, and the participants spent significantly more time for weaving more complicated aspect state machines. In [S202], it was found that the aspect-oriented modeling of the platform-independent behavior UML models by a model weaving technique had significantly increased their understandability and maintainability. [S213] suggests an extended UML metamodel with adequate semantics for creating robust and expressive models for transformation into code.

Modeling Research Trends Towards: (1) Aspect-Oriented and Domain-Specific Modeling, in which aspect-state machine weaving [S158] and aspect-oriented platform independent modeling [S202] greatly increase understandability and maintainability, then (2) Model Reusability and Transformation, covering cross-application domain and cross-modeling paradigm problems [S132] and automatic backlog refinement [S121] and (3) Cognitive Effectiveness of Model Techniques, proving that Object Process Methodology is more effective than UML in developing dynamic web applications [S38], scenario and goal-based modeling aids understanding of requirements [S57], and model slicing [S5].

While precise-style modeling helps developers [S2], Object-Process Methodology was found better suited for dynamic aspects [S38], suggesting that no single modeling approach is universally superior; effectiveness depends on the application domain.

Testing

In this perspective, nine studies present the evaluation and testing of UML diagrams thoroughly. These works brought to light important issues related to several aspects of validation and verification. Testing conceptual models appeared as tools for validation, focusing on finding faults and causing failures [S65]. Once object Constraint Language constraints were formalized, expressive schema verification ensured termination and properties satisfaction [S21]. Validation and verification use cases played a central role, targeting tasks like model consistency, satisfiability of properties, and exploration of constraints [S70]. Lightweight verification provided static analysis and repair feedback to designers to support the examination of models [S71]. A Systematic transformation and verification approach for UML sequence diagrams helped practitioners check the accuracy of models [S73]. A state chart-based validation model illustrated higher performance compared to traditional state chart designs [S91]. A novel approach detected modeling errors and bridged the gaps between domain and modeling expertise using customizable natural language questions [S27]. The error analysis discovered that though activity diagrams were perceived as more comprehensible, they were more error-prone [S56]. Identification of security vulnerability at an early stage took place through the UMLsec model, easing up the job of security checks and re-verification [S68]. Incorporation of formal syntax and semantics for variability in Petri nets extended UML activity diagrams for product line engineering testing techniques [S42]. Added to this is the statement

of study [S120] says that although OCL is a helpful formal language for formulating requirements over UML, it also faces the ambiguity problem during the expression of different Object-Oriented notions such as states, links, attributes of the object. Due to this problem faced by OCL, developers could not find errors in requirements such as conflicts, redundancies, or failures. Transforms OCL contracts and error detection patterns into SMT (Satisfiability Modulo Theories) formulas, Uses an SMT solver to verify the formulas and detect errors, Maps the results back to the original OCL contracts, clearly indicating the type and location of each error and in [S135] authors seems to believe that formal methods are well-known techniques offering a sound mathematical basis to software development. Such techniques enhance the quality of development by enabling precise and explicit modeling, thus allowing the problems like ambiguities and incompleteness to be ascertained very much earlier in the software development life cycle. Taken together, these findings improve the knowledge base on rigorous testing strategies and enhance the reliability and effectiveness of UML diagrams in an array of domains. In [S173] the scholars established that object-oriented UML reading techniques, inspection procedures, and reviewer expertise fit well into incremental development processes and managed to detect defects not found by existing reading techniques. Testing research reveals three strands: (1) validation and verification techniques (n=7), including conceptual model testing for fault detection [S65], SMT-based verification of OCL contracts [S120], and formal verification using Uppaal [S135], (2) test case derivation from UML diagrams (n=5), comparing activity diagrams and state machines [S56, S75] and (3) security and error detection (n=3), using UMLsec for early security vulnerability identification [S68].

Software Engineer

Cognitive factors that influence or are susceptible to understanding, are techniques like concept mapping to solve problems [S12]. Reading techniques, like defect detection, [S20] or perspective or good vs. poor, [S41] affect software engineers' performance. Novice software analysts enhance sequence diagrams using techniques that are related to the cognitive load management theory, [S14] while the instructors or software mentors increase the skills of software analysis [S15]. Moreover, there is evidence about the recent work of the schools that highlight the role of intelligence of the developer in problem-solving or comprehensibility [S124], error due to "analyst-analyst's experience" [S127], or without experience [S133]. Understanding of representation or the familiarity of the application has an impact on performers completion for software developers [S44], while the learning obstacles about software UML representation, like the attribute definition or multiplicity application, [S111] can be solved by techniques like concept mapping [S12]. Semantic distance or transition relation affects the change in logical design due to the object-oriented design performance for software developers [S33].

The cognitive perspective of software engineers is associated with cognitive issues, learning approaches, learning programs, learning barriers, or the experiences of software engineers to improve UML Diagram Quality as suggested in [S39]. As reported in [S147], the scientists stated that intelligence and cognitive factors of developers affect the understandability of domain models, where better intelligence results in enhanced model understandability performance, as well as higher adoption intention. The authors of [S160] found that user experience, competence, and stereotypes are vital aspects, with various behavior demonstrated by users with different competence levels.

[S189] researched open-source CASE tools for software modeling processes using UML, and found that StarUML and UML Designer were involved in the most good practices. [S199] described the integration of graphical and textual editors for a UML profile-based DSL, mentioning that there are certain constraints and requirements for the tooling environment and the user interface. The research regarding the role of software engineers converges on: (1)

cognitive and individual aspects, showing that developer general intelligence and cognitive aspects affect the understandability of the domain model [S147, S124], (2) the effect of experience and expertise, as user experience, skills, and stereotype awareness are significant in this aspect [S160], while novices can use methods related to the management of cognitive load [S14] and (3) tooling and training interventions, such as the analysis of CASE tools for modeling [S189] and the integration of graphical or textual editors [S199].

Guideline

The Guideline perspective encompasses nine diverse studies, each probing various aspects of UML guideline effectiveness. Developers benefit from augmented abstractions and examples, fostering superior diagram understanding and task performance [S92]. Clearing ambiguities, abstraction rules enhance UML diagrams' coherence [S23]. Design exemplars in a repository enhance design quality [S34]. Explicit reuse guidance clarifies domain models and variability [S82]. Theory-guided learning improves conceptual model comprehension [S37]. Visual guidelines standardize diagram presentations for better readability [S60]. Object modeling advisors bridge gaps between analysts and engineers [S63]. Ontological guidelines enhance UML's applicability in complex domains [S99]. Work system snapshots yield quality use case diagrams [S101]. These findings bolster our understanding of effective guideline-based approaches for enhanced UML diagram quality. In [S163], it was found that the precise approach to business-process modeling style with modeling guidelines and constraints was associated with significantly higher levels of comprehension, especially among more experienced participants. In [S164], it was proved by means of meta-analysis that the ontological guidelines for the conceptual models according to Bunge's model can enhance the comprehensibility of the model. In [S165], it was shown that precise approach to modeling style and modeling guidelines led to significantly higher levels of comprehensibility of workflows, especially among more experienced participants. [S176] stated that UML modeling guidelines and practitioner modeling approaches were comparable in clarity, completeness, and validity of information. Model Verificator [S200] was offered as a means for validation of UML models against specific design guidelines and standards. [S220] suggested aesthetic guidelines, layout or design guidelines, and HCI guidelines for UML class diagrams.

Research on guideline shows that there are two contradictory trends: (1) formal guideline efficiency showing that strict business process modeling style with guidelines and constraints greatly increases understandability, with experienced participants having better understanding [S163, S165], and ontological guidelines derived from the Bunge's model increase understandability of model [S164], (2) while there is an insignificant practical influence where modeling guidelines and practitioner practices were not significantly different in clarity, completeness and validity [S176].

Pedagogical Strategy

The most recent trend of the activities in the field of software modeling quality and understandability encompasses the integration of gaming elements, process thinking, education platforms into the pedagogical strategy of the software engineering discipline. For instance, in [S118], the authors have developed a gamified web platform to learn UML, and in [S128] and [S130] the authors have used gamification, i.e., the gaming mechanics to boost the learn engagement. In the other studies, the scholars claimed that using process thinking by the novices, higher-quality process models are developed for information system. [S129]. All of the afore-mentioned points regarding educational perspective of software engineering, are also documented by Cihan and Kalıpsız [32], and they believe that software engineering graduates need to improve their skills in system analysis and design, software engineering principles, and agile software development courses, which is feasible by giving importance to social abilities of the students. In [S159] the researchers found out that large language models help in UML

class-diagram modeling tasks leads to solutions having significantly more semantic mistakes and textual discrepancies in comparison with reference solutions, but there is no significant difference between syntactic and pragmatic qualities. [S162] revealed that assessment of UML diagrams using GPT does not allow completely automating assessment procedures and cannot replace human experts. [S167] demonstrated that gamification in UML modeling education increases the size of diagrams by 8% and their correctness by 11%. [S169] revealed that gamification in UML class-diagram learning tasks significantly increased students' engagement, completeness of exercises, and semantic quality of diagrams generated by them. [S184] proved that usage of examples of software designs increased students' models by 18%, and models become better by 19% after applying update procedure based on examples. [S190] demonstrated that 94.7% of instructors believe that the developed quality model for gamified software modeling learning will increase the quality of UML models. [S217] describes common errors of novice systems analysts in creating domain models.

To wrap it up, there is high consensus in the pedagogical approach towards: (1) gamification's efficacy, which entails that gamification makes for a learning exercise with increased engagement by 8%-11%, improved diagrams' accuracy, and semantic value [S167, S169, S128, S130]. Next, (2) LLM and AI in pedagogy, where GPT-based assessment holds much promise yet lacks the ability to take the place of human experts, and LLMs yield solutions with more semantic errors [S159, S162] and (3) the use of examples and process thinking, which means that the collection of examples enhances students' designs by 18% [S184], and process thinking yields

Refactoring

Refactoring, redistributing classes, attributes, and methods, enhances software systems as a whole [S24]. Software refactoring improves internal structure without affecting external behavior and can already be performed at the modeling level. Refactoring of UML models helps preserve class invariants, which improves the quality of the model [S24]. Multi-view integrated approaches lead to the detection of more instances of bad smells [S5]. Refactoring use diagrams with the assistance of Formal Concept Analysis enhances precision [S18]. A trade-off is noted between smells and quality: smells reflect design problems, whereas refactoring enhances quality [S107]. In addition, PARMOREL introduces Selective Refactoring extension for smells detection and refines model quality by reducing syntactic errors [S107, S114]. Collectively, these sub-topics put forth that refactoring strategies have an impact in raising the quality and understandability of UML diagrams. The scholars proposed that process reverse engineering and class diagram complexity metrics have the potential in the assessment of maintainability by measuring time required and errors produced. [S182] showed that prioritization of design flaws through metric and automatic ranking of flaws helps in identifying and prioritizing important design flaws easily. [S203] studied detection of design smells and anti-patterns in UML and their repair. [S210] used model refactoring, model matching, and duplication detection to study duplications in models of individual products. [S226] presented a prototype that was able to detect undesirable characteristics of the model and enhance its quality using some selected metrics and model check rules. Refactoring studies include: (1) refactoring at model level to enhance quality, where refactoring of UML models has been found to retain class invariants and improve model quality [S24, S226]. (2) smell detection and prioritization, multi-view approaches detect more bad smells [S5], while metric based ranking of flaws facilitates prioritization of flaws [S182]; and (3) tool supported refactoring framework for customizing model repair [S114].

Constraints and Requirements

The Constraints and Requirements Perspective cover the essential aspects affecting UML diagram quality and understandability. Constraints, involving limitations on Class

Diagram elements and data states [S94], are of paramount importance. Specifically, interactions between class-diagram constraints are rich in defects [S94]; hence, extra awareness is required. The use of propositional logic patterns enhances constraint modeling in UML-based conceptual modeling [S32]. Besides, early integration of business rules into models provides adaptability [S17]. An extended notation of security aspects in UML state charts enhances the model's readability [S28]. In addition, the ontology-based reuse of requirements improves reusability, streamlining elicitation and reasoning effort [S36]. The newest studies involve Requirement Engineering and unclear expectations have been addressed by studies [S126] and [S133], respectively. The key contribution of [S126] was aimed at requirement engineering in AI-enabled software, while according to [S133], the expectation from stakeholders should be clearly mentioned in the requirements specification, so that the expectations will be fulfilled properly. Overall, this perspective emphasizes constraint interaction, logic pattern modeling, integration of business rules, security modeling, and requirements reusability for improving the quality of UML diagrams. In [S141], the authors found out that explicit guidelines for reuse in UML-based product line domain models improved comprehension of the model, whereas explicit variability improved model comprehension to a certain extent only. In [S156], the transformation process from use case models to analysis models using restriction rules significantly improved class correctness, completeness of class diagrams, message correctness, completeness of sequence diagrams, and comprehension of use case specifications. In [S183], requirement modeling approaches were compared (OO-DFD and Use Case models), and it was found out that Use Case models were of better quality as they were simpler and had less details, although there was no significant difference in comprehension. It was shown that [S194], when the process of developing a model is guided by knowledge, results in a considerable increase in correct classes, attributes, and relationships discovered. It was noted that [S201] pattern-guided modeling using design patterns is beneficial as it decreases the cognitive distance between end user-developers' mental models and software models. [S211] stated that derivation process of requirements to conceptual models is crucial in relation to the high-competence group.

So, the studies on Constraints and Requirements are focused on three main themes: (1) constraint modeling and validation, involving multiplicity constraint interaction issues [S94] and propositional logic-based patterns for constraint modeling [S32]. (2) requirements to model transformation, where pattern-based approaches have greatly influenced model completeness [S201] and derivation [S211] and (3) requirements engineering for certain domains including AI-based software applications [S126] and product-line domain models [S141].

Evaluation

In the Evaluation perspective, understanding improvement of UML diagram quality is gained through various sub-topics. Metrics were explored, including a quality model with innovative metrics for object-oriented design [S25], and a 3D visualization approach for extracting metrics from class UML diagrams [S26]. Overall UML class diagram assessment was proposed [S86]. In addressing system complexity, approaches such as Page Rank and HITS algorithms reveal critical use cases, improving the clarity of diagrams [S4]. Size was a performer constraint, promoting compact representation [S93]. Validations using ontology were utilized for UML class models [S9]. Acceptance had multi-dimensionality for assurance [S76]. Prototyping enhanced understanding, flourishing in dynamic systems [S53]. Helpful class interactions support testability in design [S102]. Analysis of the role of meta-associations in meta-models was approached [S103]. Recently in [S133], it was demonstrated that students were aware of the benefits of modeling for planning and group communication but encountered issues in understanding assignment tasks, receiving unpredictable or inadequate feedback, as well as little familiarity with the domains of the problems. The findings from evaluation research include: (1) evaluation using metrics that involves 3D visualization for extracting

metrics [S26] and the use of quality models that employ new object-oriented (OO) metrics [S25]. (2) evaluation using algorithms, such as Page Rank and HITS in identifying important use cases [S4] and (3) user-focused evaluation in prototyping approaches and students' views on modeling [S133].

Visualization/Interactions

In [S146] the authors have shown that quality criteria of layout and machine learning-based layout evaluation techniques can be applied for quality assurance of class diagrams in an automated manner, with automated feedback generation being possible. [S149] has confirmed the previously obtained results regarding the effect of diagram size and topographic quality on diagram comprehension and has expanded them to new diagram types. [S151] has demonstrated that the use of animation and dynamic or timescale visualizations of sequence diagrams increases diagram comprehension in terms of control flow in comparison with static visualization. [S152] has discovered strong negative correlation between diagram size and modeler's performance and has provided recommendations about ideal diagram sizes.

[S168] stated that interactive views (MetaView, ContextView, MetricView, UML-City-View) increased comprehension correctness by 4.5% and saved 20% of time needed. [S185] confirmed previous findings about diagram size and layout errors and gave improved definitions of both metrics. [S192] proved that visualizing additional information to the UML diagrams via visual variables (color, brightness, text, border decoration) increased diagram's readability and usability. [S215] suggested applying Virtual Reality (3D appearance, animation, interaction, immersion) to improve the quality of UML modeling. The major trends in visualization studies can be summarized as follows: (1) layout quality and automation and machine-learning-based layout evaluation made automation of quality assessment possible [S146, S149, S153]. (2) dynamic and interactive visualization, where animation enhances comprehension of control flows [S151] and interactive views increase comprehension correctness by 4.5% while saving 20% of time [S168] and (3) new technologies, such as Virtual Reality and 3D visualization [S215].

Diagram Type

The Diagram Type Perspective: This perspective focuses on the type of UML diagrams, based on five selected articles. State diagrams provide effective semantic comprehension for the dynamic aspects in real-time domains whereas sequence diagrams are very effective in management information systems [S40]. Sequence-state diagrams were found effective by combining semantic comprehension for dynamic system modeling synergistically [S40]. UML class diagrams outperform ERD in terms of providing better comprehension and verification support [S83]. Though ERD was found dominating in database conceptual modeling, UML class diagrams also performed well regarding comprehension and adaptability during data modeling [S95]. Activity diagrams outperform state machines regarding understandability but pose the threat of being more error-prone during manual test case derivation [S75]. Collaboration diagrams performed well for real-time systems whereas higher quality collaboration diagrams were found suitable for management information systems compared to sequence diagrams for improving diagram quality and comprehensibility based on this perspective [S62]. The researchers found that composite states and experience with composite states make UML statechart diagrams more understandable, especially when subjects have experience. [S140] found that although conventional wisdom suggests that hierarchical modeling mechanisms can be useful, the use of composite states did not seem to make the UML statechart diagrams more understandable even when practitioners were taken as subjects. It was found that, being an expert in manipulating models is crucial for understanding the behavior models while the need for the clarification of the semantics of logic labelled finite state

machines is crucial for formal verifiability [S161]. Comparison of the types of diagram shows: (1) effectiveness of a diagra is related to the application domain, for instance state diagrams are effective in real-time domains, sequence diagrams in management information systenms [S40], and class diagrams perform better than Entity Relationship Diagram for comprehension and verification [S83]. (2) experience has an impact on the improvement in the understandability due to the use of composite states occurs when the subjects have experience [S138, S140] and (3) in terms of perception versus usability, activity diagrams are perceived to be more comprehensible but they are prone to more errors [S75].

Redundancy

Redundancy refers to the existence of excessive and unnecessary information [S89]. The Redundancy Perspective delves into UML diagram quality with insights from various sub-topics. Redundancy, an overflow of unnecessary information, challenges diagram clarity [S89]. UML class diagrams, though capable of representing complex constraints, risk correctness due to interactions. Algorithmic rule-based methods address multiplicity constraint redundancies [S89]. Model readability is influenced by redundancy strength, with trained readers less affected [S13]. Verification prioritizes integrity and constraint necessity to counteract redundancies [S47]. Referential redundancy diminishes model quality, addressable through language and model-level strategies [S66]. In summary, redundancy's impact on UML diagram quality is multifaceted, addressing constraint, construct, and referential redundancies for enhanced understandability.

System Structure

Two studies have focused on system structure considering its complexity. One study indicates the importance of examining structural complexity through three dimensions: size and control flow complexity, state-related actions, and action sequence. Each of these dimensions significantly impact the comprehensibility of state chart diagrams [S50]. Another study reveals a link between understandability and measures of structural complexity [S59].

Model Transformation

Quality-driven model transformation and transformation decisions influence the quality of the resulting models in [S193], with preference for association relationships driving transformations in case understandability is preferred.

Finally, looking at redundancy, system structure and model transformation from their perspectives, we could say that these three aspects are highly underexplored. The research of redundancy investigates issues like multiplicity constraint redundancy [S89], construct redundancy effect on reader [S13], and referential redundancy [S66]. Studies of system structure explore the dimensionality of structural complexity [S50, S59]. The single study of model transformation [S193] reveals that transformation decisions impact model quality, with a tendency to favor association relationships when understandability is prioritized.

5. Discussion and Conclusion

Analysis of 227 peer-reviewed articles on UML diagrams in the period of 2000-2026 showed the thorough coverage of the research field in terms of the quality and understandability of the diagrams, which is represented by 16 different research topics according to the thematic coding. The topic studied most often is Quality Assessment (**19.82**%) that takes the lead ahead of the Design and Modeling that used to be the main research focus in the preliminary analysis. This indicates the maturing stage of the field when the primary research issue has changed from design and modeling to evaluation, measurement, validation and improvement of the quality of the models and their structural integrity and maintainability. This trend correlates well with the

recent empirical study conducted by OliveiraJr et al. [35] where 80 instructors and 206 practitioners were surveyed in Brazil and Portugal, and they found out that despite being the key element of academic training, only certain types of UML diagrams are used by the professionals such as Class, Use Case, and Sequence diagrams, and light-weight modeling is gaining popularity among the practitioners as a part of agile development practices.

Topics at the foundation, namely, Representations (12.33%), Design (11.01%), and Modeling (10.57%) are solid, which reflects a continuous focus on notations, construction of structure, and visual languages. This is in line with the systematic review conducted by Ciccozzi et al. [36] on the subject of UML model execution, who determined that Class, State Machine, and Activity diagrams are the most frequently needed representations for models capable of being executed, usually used in combination. Analysis of 63 papers and 19 tools showed that translational execution strategies outnumbered interpretive ones and that there were very few model level debugging facilities and execution of partial models possible.

Testing, Software Engineer, Guideline, and Pedagogical Strategy (all accounting for 6.61%) indicate clearly practical and educational focus of the topics on correct models, human factors, best practices, and pedagogical strategies. Analysis of the distribution across five-time intervals showed several specific shifts. Most notably, there is an emergence of Pedagogical Strategy, which increases from a marginal 0.44% in previous time intervals to 3.96% in 2020-2026 due to gamification, LLM-based feedback, remote learning, and increased awareness of the need for new pedagogical strategy to teach modeling. This shift indicates a paradigm shift in the field and is confirmed by the recent results indicating that students consider UML as important for abstract thinking but unconnected with reality [35].

On the other hand, Quality Assessment has significantly dropped from 5.29% (2005-2009) to 2.64% (2020-2026), indicating the shift of this theme from independent frontier issue to mainstream concern that has become an integral part of design, testing, and refactoring researches. Such trend coincides with the results of the systematic literature review performed by Koç et al. [37], which involved analysis of 128 publications of 2000-2019 years and revealed that most of UML diagrams were applied for design purposes (about 67%) with class diagrams being the most frequently applied type (26.3%). At the same time, testing and implementation accounted for 18% and 13%, respectively.

Guideline seem to be completely absorbed (2020-2026), indicating that the concept could have been absorbed by teaching or quality guidelines or the fact that modeling guidelines are already considered common knowledge. This might indicate that UML has matured as a standard, and the use of guidelines for basic modeling tasks is no longer regarded as novel research anymore. The category of refactoring has a mild upward trend (up to 1.76%), corresponding to the increasing interest in maintainability and technical debt in the industry sector, which is also indicated in Wendt et al.'s [38] work on MBCG testing.

Pillars like Representations (peaks at 4.41% through two eras) and Design or Modeling (stays at 2-3%) show consistent core focus. There are some areas that are critically under-researched: System Structure (0.88%) and Model Transformation (0.44%) have almost none of research conducted during many eras. In times of increasingly complex systems, especially those using AI and distributed structures, understanding of structure model representation and automatic transformations becomes crucial. Ciccozzi et al. [36] also noted expressiveness enhancement and tool enhancement as most frequent limitations of model execution approaches, drawing attention to the problem of lack of comprehensive UML coverage and tools. Redundancy (2.20%) and Visualization/Interaction (5.29%) stay marginal in spite of their practical importance, yet the last one might grow due to new VR/3D and AI-powered interaction technology [35].

It appears consistently that the totals in each column always exceed 100% by varying margins (between 8.37% and 28.19%), proving the inherent integrative nature of software

modeling research. Each piece of research incorporates more than one point of view, such as notation, Design, Evaluation, and Pedagogical Approaches, showing the complex connections within modeling where Quality, Representations, Design, Testing, and human aspects are tightly bound together.

Based on this systematic mapping study, we see an advanced and developing field, which is characterized by dominance of Quality Assessment, the most pronounced shift towards Pedagogical Strategy, and clear gaps in terms of System Structure and Model Transformation. The integrative character of this field dictates the necessity to apply comprehensive, multiple-perspective approaches to improving the quality of UML diagrams in light of increasing software system complexity and AI transformation of modeling and teaching. Research in the future should focus on filling the gap between practice and education found in this review, through introducing agile-compatible modeling approaches, promoting selective and goal-oriented usage of diagrams, and practical experience in using popular modeling tools [35]. In addition, low usage of UML in agile environments, as evidenced by this study and [35], calls for addressing modeling techniques in agile development workflows, such as performing modeling tasks in sprint planning sessions.

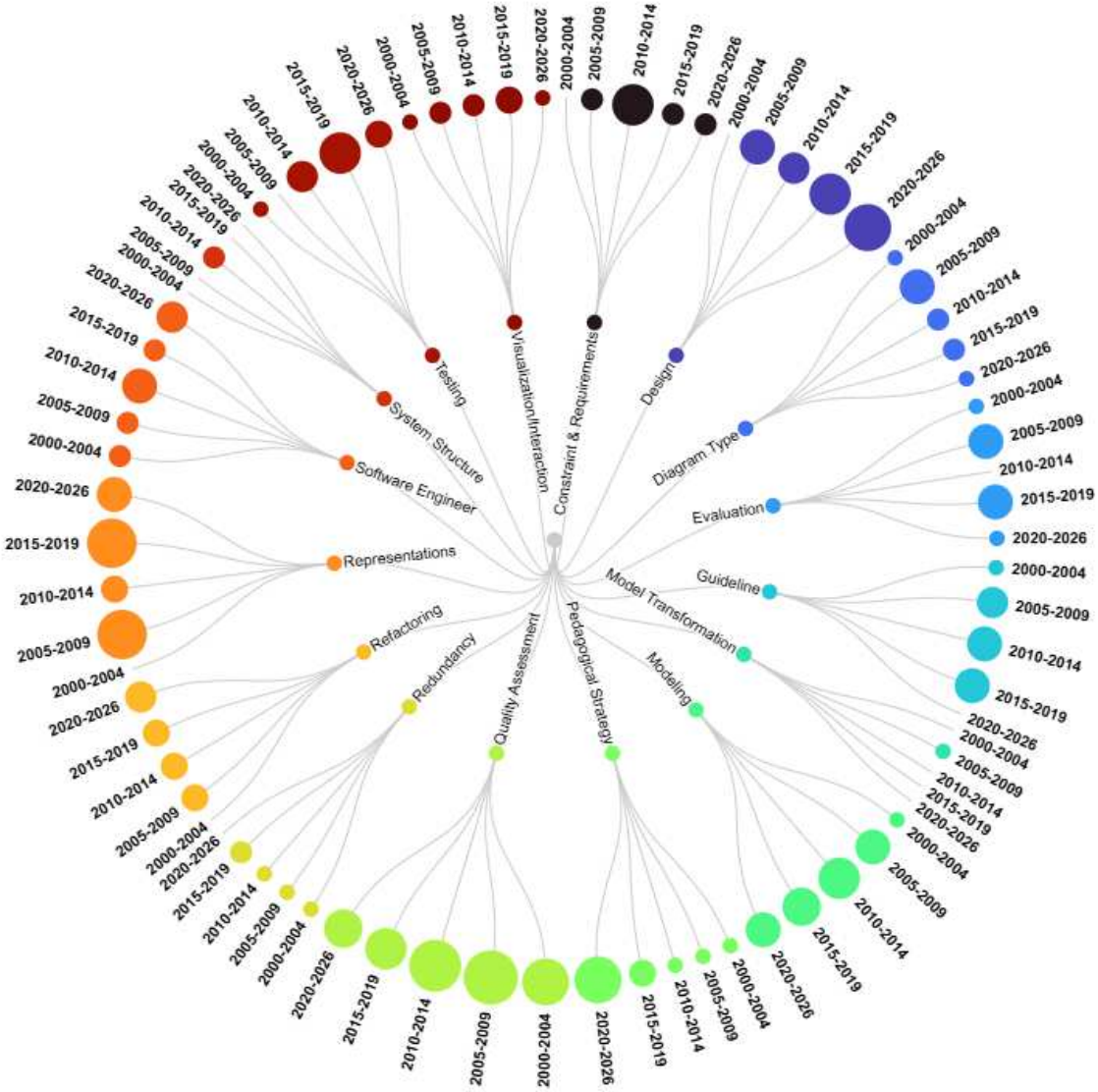


FIGURE 4: Yearly distribution of topics (2000-2026).

In relation to Figure 4, the most important contribution of this paper is to identify the state-of-the-art research on UML diagram quality and understandability from the period of 2000 to 2026 through a systematic mapping study on 227 articles. Through a robust thematic analysis process, we were able to classify the articles into 16 themes: Quality Assessment, Representations, Design, Modeling, Testing, Software Engineer, Guideline, Pedagogical Strategy, Refactoring, Constraint and Requirement, Evaluation, Visualization/Interaction, Diagram Type, Redundancy, System Structure, and Model Transformation. These themes are the factors influencing the quality and therefore the understandability of UML diagrams.

6. Future Research Directions

Regarding the findings, there are a number of research gaps that need to be addressed. First, it is necessary to conduct further research on the subjects of System Structure (0.88%) and Model Transformation (0.44%, n=1), which remain critically understudied in today's increasingly complex AI-based and distributed systems where the issue of automated transformation of models becomes increasingly important. Second, despite the dominance of Quality Assessment in the literature, the majority of studies focus on metrics as opposed to experimental approaches. Third, the rise in interest in Pedagogical Strategy (0.44% to 3.96%) along with the persistent education-practice gap outlined by OliveiraJr et al. [35] calls for research in the development of agile compatible modeling curriculum with gamification and LLM feedback included.

Fourthly, tool enhancement and automation, seen to be among the recurring weaknesses [36], should be studied in relation to the role of AI in assisting with refactoring, redundancy detection, and the integration of UML with low-code or no-code platforms and novel VR/3D environments [S215]. Next, cross-disciplinary empirical research on the usefulness of UML in AI-based software development, social robotics, and other fields will help understand its interpretability and effectiveness across different domains of application. Finally, since the field of software development is shifting away from object-oriented development practices, further research into the use of UML artifacts within the context of low-code and no-code development environments is required.

7. Threats to Validity and Limitations

Taking into consideration the criteria used in systematic literature review [30], we outline the main validity threats based on four dimensions. Construct validity is associated with how well our search strategy has covered all relevant studies. To minimize this threat, we created the search string consisting of terms developed based on research questions and pilot studies, which encompassed different synonyms (such as understand, comprehend, cognit, aesth) and roots. Internal validity can be threatened by bias related to study selection and data extraction. The independent study selection performed by three reviewers has resulted in inter-rater agreement of 0.81 (Cohen's Kappa [34]), which is "almost perfect agreement". Nevertheless, the allocation of subfactors under 16 major perspectives is subjective, and other researchers could categorize them differently. These categories are more like analytical perspectives and not mutually exclusive categories, so it is possible that there will be some overlapping concepts in certain categories (for example between Design and Modeling, between Evaluation and Quality Assessment). External validity is concerned with the generalizability of our study. By limiting our search to only Web of Science, Scopus, IEEE Xplore and ACM databases, we might miss out on some studies that have been published only in other databases, like Google Scholar. However, these three databases are considered the most complete source for software engineering studies [30]. In addition, limiting our search to only English-language articles may

result in some language bias. Quality of primary studies was evaluated using 12 criteria based on proven guidelines [30, 34]. However, due to all these precautions, variability in the methodological quality of the primary studies, specifically, their small number, limited number of empirical studies, as well as prevalence of validation studies over evaluation ones, might have an impact on the validity of our conclusions. In addition to that, the existence of publication bias may also be a problem as we excluded non-published papers, specifically technical reports and white papers that might include important findings, but do not go through the same procedures of quality control as articles published in peer-reviewed journals. Finally, last period, 2020-2026, is not complete as the search was conducted on 25 August 2026. All comparisons in time for this period should be considered as annualized rates and thus should not be taken literally.

Acknowledgment

A preprint has previously been published [33].

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Conflict of Interest

The authors declare no conflicts of interest.

Funding

There is no funding for this research.

References

1. V. Vranić, J. Porubän, M. Bystrický, T. Frťala, I. Polášek, M. Nosáľ, and J. Lang, "Challenges in preserving intent comprehensibility in software," *Acta Polytechnica Hungarica* 12, no. 7 (2015): 57–75. https://doi.org/10.12700/aph.12.7.2015.7.4.
2. J. O. Coplien and G. Bjørnvig, *Lean Architecture: For Agile Software Development* (Chichester: John Wiley and Sons, 2010).
3. J. Kabeli and P. Shoval, "Comprehension and quality of analysis specifications—a comparison of FOOM and OPM methodologies," *Information and Software Technology* 47, no. 4 (2005): 271–290. https://doi.org/10.1016/j.infsof.2004.08.009.
4. M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (Boston: Pearson Paravia Bruno Mondadori, 2004).
5. H. Bersini, "UML for ABM," *Journal of Artificial Societies and Social Simulation* 15, no. 1 (2012): 9. https://doi.org/10.18564/jasss.1897.
6. H. Koc, A. M. Erdoga, Y. Barjakly, and S. Peker, "UML diagrams in software engineering research: A systematic literature review," *Proceedings* 74, no. 13 (2021). https://doi.org/10.3390/proceedings2021074013.
7. C. Vasilakis, D. Lecznarowicz, and C. Lee, "Developing model requirements for patient flow simulation studies using the Unified Modelling Language (UML)," *Journal of Simulation* 3, no. 3 (2009): 141–149. https://doi.org/10.1057/jos.2009.3.
8. E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: An experimental evaluation," *IEEE Transactions on Software Engineering* 32, no. 6 (2006): 365–381. https://doi.org/10.1109/TSE.2006.59.
9. G. Erich, H. Richard, J. Ralph, and V. John. *Design Patterns: Elements of Reusable Object-Oriented Software* (Québec: Braille Jymico Incorporated, 2000).
10. G. Booch, J. Rumbaugh, J. and I. Jacobson. *The Unified Modeling Language Reference Manual* (Boston: Addison-Wesley, 1999).

11. B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and Software Technology* 51, no. 1 (2009): 7–15. https://doi.org/10.1016/j.infsof.2008.09.009.
12. I. Hadar and O. Hazzan, "On the contribution of UML diagrams to software system comprehension," *Journal of Object Technology* 3, no. 1 (2004): 143–156. https://doi.org/10.5381/jot.2004.3.1.a3.
13. C. F. Lange, M. R. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," *IEEE Software* 23, no. 2 (2006): 40–46. https://doi.org/10.1109/MS.2006.50.
14. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. (Boston: Addison-Wesley, 2012).
15. I. Sommerville, *Software Engineering*, 10th ed. (Harlow: Pearson, 2016).
16. M. Genero, A. M. Fernández-Saez, H. J. Nelson, G. Poels, and M. Piattini, "Research review: A systematic literature review on the quality of UML models," *Journal of Database Management (JDM)* 22, no. 3 (2011): 46–70. https://doi.org/10.4018/jdm.2011070103.
17. J. Cabot, R. Clarisó, and D. Riera, "UMLtoCSP: A tool for the formal verification of UML/OCL models using constraint programming," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007, 547–548. https://doi.org/10.1145/1321631.1321737.
18. D. Torre, M. Genero, Y. Labiche, and M. Elaasar, "How consistency is handled in model-driven software engineering and UML: An expert opinion survey," *Software Quality Journal* 31, no. 1 (2023): 1–54. https://doi.org/10.1007/s11219-022-09585-2.
19. M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice* (San Rafael: Morgan and Claypool Publishers, 2017). https://doi.org/10.2200/s00751ed2v01y201701swe004.
20. A. Egyed, "Fixing inconsistencies in UML design models," in *29th International Conference on Software Engineering (ICSE'07)*, 2007, 292–301. https://doi.org/10.1109/ICSE.2007.38.
21. A. Dikici, O. Turetken, and O. Demirors, "Factors influencing the understandability of process models: A systematic literature review," *Information and Software Technology* 93 (2018): 112–129. https://doi.org/10.1016/j.infsof.2017.09.001.
22. L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "A systematic literature review of software visualization evaluation," *Journal of Systems and Software* 144 (2018): 165–180. https://doi.org/10.1016/j.jss.2018.06.027.
23. P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development–a review of literature," *Information and Software Technology* 51, no. 12 (2009): 1646–1669. https://doi.org/10.1016/j.infsof.2009.04.004.
24. M. El-Attar and J. Miller, "Constructing high quality use case models: A systematic review of current practices," *Requirements Engineering* 17 (2012): 187–201. https://doi.org/10.1007/s00766-011-0135-y.
25. D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius, "Empirical evidence about the UML: A systematic literature review," *Software: Practice and Experience* 41, no. 4 (2011): 363–392. https://doi.org/10.1002/spe.1009.
26. T. H. Haug, *A Systematic Review of Empirical Research on Model-Driven Development with UML* (Master's thesis, University of Oslo, 2007).
27. F. J. Lucas, F. Molina, and A. Toval, "A systematic review of UML model consistency management," *Information and Software Technology* 51, no. 12 (2009): 1631–1645. https://doi.org/10.1016/j.infsof.2009.04.009.

28. A. M. Fernández-Sáez, M. Genero, and M. R. Chaudron, "Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study," *Information and Software Technology* 55, no. 7 (2013): 1119–1142. https://doi.org/10.1016/j.infsof.2012.12.006.
29. A. Boland, G. Cherry, and R. Dickson, eds., *Doing a Systematic Review: A Student's Guide* (London: SAGE, 2017).
30. S. Keele, *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, ver. 2.3, EBSE Technical Report, 2007.
31. T. Meline, "Selecting studies for systemic review: Inclusion and exclusion criteria," *Contemporary Issues in Communication Science and Disorders* 33, Spring (2006): 21–27.
32. P. Cihan and O. Kalıpsız, "Evaluation of technical skills in software engineering course projects," *Sigma Journal of Engineering and Natural Sciences* 31, no. 4 (2013): 595–603.
33. S. Alizadeh Tabrizi, D. Topalli and N. Ercil Cagiltay, "Fourteen Topics Explored from Quality and Understandability Studies on Unified Modelling Language Notation," Preprint in *Research Square,* 2026. https://doi.org/10.21203/rs.3.rs-9996788/v1.
34. Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, *50*(9-10), 833-859. https://doi.org/10.1016/j.infsof.2008.01.006.
35. OliveiraJr, E., Canedo, E. D., & Brito, I. S. (2026). Understanding education and professional practice of UML: Insights from Brazil and Portugal. *ACM Transactions on Computing Education*. (In Press). https://doi.org/10.1145/3779309.
36. Ciccozzi, F., Malavolta, I., & Selic, B. (2019). Execution of UML models: A systematic review of research and practice. *Software and Systems Modeling*, 18(3), 2313–2360. https://doi.org/10.1007/s10270-018-0675-4.
37. Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (2021). UML diagrams in software engineering research: A systematic literature review. *Proceedings*, 74(1), 13. https://doi.org/10.3390/proceedings2021074013.
38. Wendt, J., Kizgin, U. V., Clasen, D., & Vietor, T. (2026). Model-based systems engineering for requirements verification and test case generation at the subsystem level. *Systems*, 14(2), 1–15.
39. Krogstie, J. (2015). SEQUAL as a framework for understanding and assessing quality of models and modeling languages. In Encyclopedia of Information Science and Technology, Third Edition (pp. 1611-1620). IGI Global Scientific Publishing.
40. International Organization for Standardization (2011). IEC 25010: 2011, Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models. Japan: Multiple. Distributed through American National Standards Institute (ANSI).
41. Moody, D. (2009). The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on software engineering, 35(6), 756-779. https://doi.org/10.1109/TSE.2009.67.

APPENDIX A. List of Primary Studies Included in the Systematic Mapping Study

[S1] Sedaghatbaf, A., and Abdollahi Azgomi, M. (2018). A method for evaluation of software architecture. Computing, 100(2), 119-150. https://doi.org/10.1007/S00607-017-0568-3

[S2] Reggio, G., Ricca, F., Scanniello, G., Cerbo, F. D., and Dodero, G. (2015). On the comprehension of workflows modeled with a precise style: results from a family of controlled experiments. Software and Systems Modeling, 14(4), 1481-1504., https://doi.org/10.1007/s10270-013-0386-9

[S3] Nisa, H., Imtiaz, S., Khan, M. U., and Imtiaz, S. (2016). Impact of Domain Modeling Techniques on the Quality of Domain Model: An Experiment. Development, 2, 6, https://doi.org/10.14569/ijacsa.2016.071158.

[S4] Sabharwal, S., Kaur, P., and Sibal, R. (2018). Applying Page Rank and HITS Algorithm to Identify Key Use Cases. Journal of Applied Science and Engineering, 21(4), 655-662, https://doi.org/10.6180/jase.201812_21(4).0019.

[S5] [Misbhauddin, M., and Alshayeb, M. (2019). An integrated metamodel-based approach to software model refactoring. Software and Systems Modeling, 18(3), 2013-2050, https://doi.org/10.1007/s10270-017-0628-3.

[S6] Bera, P., Burton-Jones, A., and Wand, Y. (2018). Improving the representation of roles in conceptual modeling: theory, method, and evidence. Requirements Engineering, 23(4), 465-491, https://doi.org/10.1007/s00766-017-0275-9.

[S7] Siau, K., and Tian, Y. (2009). A semiotic analysis of unified modeling language graphical notations. Requirements Engineering, 14(1), 15-26, https://doi.org/10.1007/s00766-008-0071-7.

[S8] Iqbal, S., Al-Azzoni, I., Allen, G., and Khan, H. U. (2020). Extending UML use case diagrams to represent non-interactive functional requirements. E-Informatica Software Engineering Journal, 14(1), https://doi.org/10.37190/E-INF200104.

[S9] Hafeez, A., Musavi, S. H. A., and ur Rehman, A. (2018). Ontology-based verification of UML class/OCL model. Mehran University Research Journal of Engineering and Technology, 37(4), 521-534, https://doi.org/10.22581/muet1982.1804.07.

[S10] Matulevičius, R., Lakk, H., and Lepmets, M. (2011). An approach to assess and compare quality of security models. Computer Science and Information Systems, 8(2), 447-476, https://doi.org/10.2298/CSIS101231014M.

[S11] Desai, N., and Gogolla, M. (2019). Developing Comprehensive Postconditions Through a Model Transformation Chain. J. Object Technol., 18(3), 5-1, https://doi.org/10.5381/jot.2019.18.3.a5.

[S12] Shen, Z., Tan, S., and Siau, K. (2018). Challenges in learning unified modeling language: From the perspective of diagrammatic representation and reasoning. Association for Information Systems, https://doi.org/10.17705/1CAIS.04330.

[S13] Bera, P., and Poels, G. (2017). The Effects of Construct Redundancy on Readers' Understanding of Conceptual Models. Journal of Database Management (JDM), 28(3), 1-25, https://doi.org/10.4018/JDM.2017070101.

[S14] Syn, T., and Batra, D. (2013). Improving sequence diagram modeling performance: a technique based on chunking, ordering, and patterning. Journal of Database Management (JDM), 24(4), 1-25, https://doi.org/10.4018/JDM.2013100101.

[S15] VanderMeer, D. (2009). Applying learner-centered design principles to UML sequence diagrams. Journal of Database Management (JDM), 20(1), 25-47, https://doi.org/10.4018/jdm.2009010102.

[S16] Reinhartz-Berger, I., and Sturm, A. (2008). Enhancing UML models: a domain analysis approach. Journal of Database Management (JDM), 19(1), 74-94, https://doi.org/10.4018/jdm.2008010104.

[S17] Loucopoulos, P., and Kadir, W. M. N. W. (2008). BROOD: business rules-driven object-oriented design. Journal of Database Management (JDM), 19(1), 41-73.
[S18] Dolques, X., Huchard, M., Nebut, C., and Reitz, P. (2012). Fixing generalization defects in UML use case diagrams. Fundamenta Informaticae, 115(4), 327-356, https://doi.org/10.3233/FI-2012-658.
[S19] Burton-Jones, A., and Meso, P. N. (2006). Conceptualizing systems for understanding: An empirical test of decomposition principles in object-oriented analysis. Information Systems Research, 17(1), 38-60, https://doi.org/10.1287/isre.1050.0079.
[S20] Marín, B., Giachetti, G., Pastor, O., Vos, T. E., and Abran, A. (2013). Using a functional size measurement procedure to evaluate the quality of models in MDD environments. ACM Transactions on Software Engineering and Methodology (TOSEM), 22(3), 1-31., https://doi.org/10.1145/2491509.2491520.
[S21] Queralt, A., and Teniente, E. (2012). Verification and validation of UML conceptual schemas with OCL constraints. ACM Transactions on Software Engineering and Methodology (TOSEM), 21(2), 1-41., https://doi.org/10.1145/2089116.2089123.
[S22] Wimmer, M., Schauerhuber, A., Kappel, G., Retschitzegger, W., Schwinger, W., and Kapsammer, E. (2011). A survey on UML-based aspect-oriented design modeling. ACM Computing Surveys (CSUR), 43(4), 1-33, https://doi.org/10.1145/1978802.1978807.
[S23] Egyed, A. (2002). Automated abstraction of class diagrams. ACM Transactions on Software Engineering and Methodology (TOSEM), 11(4), 449-491, https://doi.org/10.1145/606612.606616.
[S24] Dao, T. H., Nguyen, X. T., and Truong, N. T. (2019). Preservation of Class invariants in refactoring UML models. International Journal of Software Engineering and Knowledge Engineering, 29(02), 221-244., https://doi.org/10.1142/S0218194019500104.
[S25] Ramaraj, E., and Duraisamy, S. (2007). Design optimization metrics for uml based object-oriented systems. International Journal of Software Engineering and Knowledge Engineering, 17(03), 423-448., https://doi.org/10.1142/S0218194007003252.
[S26] Lakshminarayana, A., and Newman, T. S. (2000). Three-dimensional visualization of object-oriented software design metrics. Journal of Electronic Imaging, 9(4), 368-374, https://doi.org/10.1117/1.1289353.
[S27] Autili, M., Bertolino, A., De Angelis, G., Di Ruscio, D., and Di Sandro, A. (2015). A tool-supported methodology for validation and refinement of early-stage domain models. IEEE Transactions on Software Engineering, 42(1), 2-25, https://doi.org/10.1109/TSE.2015.2449319.
[S28] El-Attar, M., Luqman, H., Karpati, P., Sindre, G., and Opdahl, A. L. (2015). Extending the UML statecharts notation to model security aspects. IEEE Transactions on Software Engineering, 41(7), 661-690., https://doi.org/10.1109/TSE.2015.2396526.
[S29] Samuel, B. M., Watkins III, L. A., Ehle, A., and Khatri, V. (2014). Customizing the representation capabilities of process models: Understanding the effects of perceived modeling impediments. IEEE Transactions on Software Engineering, 41(1), 19-39, https://doi.org/10.1109/TSE.2014.2354043.
[S30] Abrahao, S., Gravino, C., Insfran, E., Scanniello, G., and Tortora, G. (2012). Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: Results from a family of five experiments. IEEE transactions on software engineering, 39(3), 327-342., https://doi.org/10.1109/TSE.2012.27.
[S31] Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., and Ceccato, M. (2009). How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: A series of four experiments. IEEE Transactions on Software Engineering, 36(1), 96-118, https://doi.org/10.1109/TSE.2009.69.

[S32] Davis, J. P., and Bonnell, R. D. (2007). Propositional logic constraint patterns and their use in UML-based conceptual modeling and analysis. IEEE Transactions on Knowledge and Data Engineering, 19(3), 427-440, https://doi.org/10.1109/TKDE.2007.45.
[S33] Shin, S. S. (2014). A study on the difficulties of learning phase transition in object-oriented analysis and design from the viewpoint of semantic distance. IEEE Transactions on Education, 58(2), 117-123, https://doi.org/10.1109/TE.2014.2334556.
[S34] Karasneh, B., Jolak, R., and Chaudron, M. R. (2015). Using examples for teaching software design: an experiment using a repository of UML class diagrams. In 2015 Asia-Pacific Software Engineering Conference (APSEC) (pp. 261-268). IEEE, https://doi.org/10.1109/APSEC.2015.59.
[S35] Motschnig-Pitrik, R., and Kaasboll, J. (1999). Part-whole relationship categories and their application in object-oriented analysis. IEEE Transactions on Knowledge and Data Engineering, 11(5), 779-797, https://doi.org/10.1109/69.806936.
[S36] Diamantopoulos, T., and Symeonidis, A. (2018). Enhancing requirements reusability through semantic modeling and data mining techniques. Enterprise information systems, 12(8-9), 960-981, https://doi.org/10.1080/17517575.2017.1416177.
[S37] Bera, P., and Poels, G. (2019). How quickly do we learn conceptual models?. European Journal of Information Systems, 28(6), 663-680, https://doi.org/10.1080/0960085X.2019.1673972.
[S38] Reinhartz-Berger, I., and Dori, D. (2005). OPM vs. UML-Experimenting with Comprehension and Construction of Web Application Models. Empirical Software Engineering, 10(1), 57-80, https://doi.org/10.1023/B:EMSE.0000048323.40484.e0.
[S39] Robbins, J. E., and Redmiles, D. F. (2000). Cognitive support, UML adherence, and XMI interchange in Argo/UML. Information and Software Technology, 42(2), 79-89, https://doi.org/10.1016/S0950-5849(99)00083-X.
[S40] Otero, M. C., and Dolado, J. J. (2004). Evaluation of the comprehension of the dynamic modeling in UML. Information and Software Technology, 46(1), 35-53, https://doi.org/10.1016/S0950-5849(03)00108-3.
[S41] Laitenberger, O., Atkinson, C., Schlich, M., and El Emam, K. (2000). An experimental comparison of reading techniques for defect detection in UML design documents. Journal of Systems and Software, 53(2), 183-204, https://doi.org/10.1016/S0164-1212(00)00052-2.
[S42] Heuer, A., Stricker, V., Budnik, C. J., Konrad, S., Lauenroth, K., and Pohl, K. (2013). Defining variability in activity diagrams and Petri nets. Science of Computer Programming, 78(12), 2414-2432, https://doi.org/10.1016/j.scico.2012.06.003.
[S43] Torchiano, M., Scanniello, G., Ricca, F., Reggio, G., and Leotta, M. (2017). Do UML object diagrams affect design comprehensibility? Results from a family of four controlled experiments. Journal of Visual Languages and Computing, 41, 10-21, https://doi.org/10.1016/j.jvlc.2017.06.002.
[S44] Scanniello, G., and Erra, U. (2014). Distributed modeling of use case diagrams with a method based on think-pair-square: Results from two controlled experiments. Journal of Visual Languages and Computing, 25(4), 494-517, https://doi.org/10.1016/j.jvlc.2014.03.002.
[S45] Byelas, H., and Telea, A. (2009). Towards realism in drawing areas of interest on architecture diagrams. Journal of Visual Languages and Computing, 20(2), 110-128, https://doi.org/10.1016/j.jvlc.2008.09.001.
[S46] El-Attar, M. (2019). Evaluating and empirically improving the visual syntax of use case diagrams. Journal of Systems and Software, 156, 136-163, https://doi.org/10.1016/j.jss.2019.06.096.

[S47] Cabot, J., Clarisó, R., and Riera, D. (2014). On the verification of UML/OCL class diagrams using constraint programming. Journal of Systems and Software, 93, 1-23, https://doi.org/10.1016/j.jss.2014.03.023.
[S48] Staron, M., Kuzniarz, L., and Wohlin, C. (2006). Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments. Journal of Systems and Software, 79(5), 727-742, https://doi.org/10.1016/j.jss.2005.09.014.
[S49] Otero, M. C., and Dolado, J. J. (2005). An empirical comparison of the dynamic modeling in OML and UML. Journal of Systems and Software, 77(2), 91-102, https://doi.org/10.1016/j.jss.2004.11.022.
[S50] J Cruz-Lemus, J. A., Maes, A., Genero, M., Poels, G., and Piattini, M. (2010). The impact of structural complexity on the understandability of UML statechart diagrams. Information Sciences, 180(11), 2209-2220, https://doi.org/10.1016/j.ins.2010.01.026.
[S51] Burgueño, L., Mayerhofer, T., Wimmer, M., and Vallecillo, A. (2019). Specifying quantities in software models. Information and Software Technology, 113, 82-97, https://doi.org/10.1016/j.infsof.2019.05.006.
[S52] George, R., and Samuel, P. (2018). Fixing class design inconsistencies using self-regulating particle swarm optimization. Information and Software Technology, 99, 81-92, https://doi.org/10.1016/j.infsof.2018.03.005.
[S53] G Sedrakyan, G., Poelmans, S., and Snoeck, M. (2017). Assessing the influence of feedback-inclusive rapid prototyping on understanding the semantics of parallel UML statecharts by novice modellers. Information and Software Technology, 82, 159-172, https://doi.org/10.1016/j.infsof.2016.11.001.
[S54] Saleh, F., and El-Attar, M. (2015). A scientific evaluation of the misuse case diagrams visual syntax. Information and Software Technology, 66, 73-96, https://doi.org/10.1016/j.infsof.2015.05.002.
[S55] Blouin, A., Moha, N., Baudry, B., Sahraoui, H., and Jézéquel, J. M. (2015). Assessing the use of slicing-based visualizing techniques on the understanding of large metamodels. Information and Software Technology, 62, 124-142, https://doi.org/10.1016/j.infsof.2015.02.007.
[S56] Felderer, M., and Herrmann, A. (2015). Manual test case derivation from UML activity diagrams and state machines: A controlled experiment. Information and Software Technology, 61, 1-15, https://doi.org/10.1016/j.infsof.2014.12.005.
[S57] Hadar, I., Reinhartz-Berger, I., Kuflik, T., Perini, A., Ricca, F., and Susi, A. (2013). Comparing the comprehensibility of requirements models expressed in Use Case and Tropos: Results from a family of experiments. Information and Software Technology, 55(10), 1823-1843, https://doi.org/10.1016/j.infsof.2013.05.003.
[S58] Al Dallal, J., and Briand, L. C. (2010). An object-oriented high-level design-based class cohesion metric. Information and software technology, 52(12), 1346-1361, https://doi.org/10.1016/j.infsof.2010.08.006.
[S59] Muñoz, L., Mazón, J. N., and Trujillo, J. (2010). A family of experiments to validate measures for UML activity diagrams of ETL processes in data warehouses. Information and Software Technology, 52(11), 1188-1203, 10.1016/j.infsof.2010.06.003.
[S60] Eichelberger, H., and Schmid, K. (2009). Guidelines on the aesthetic quality of UML class diagrams. Information and Software Technology, 51(12), 1686-1698, https://doi.org/10.1016/j.infsof.2009.04.008.
[S61] Nugroho, A. (2009). Level of detail in UML models and its impact on model comprehension: A controlled experiment. Information and Software Technology, 51(12), 1670-1685, https://doi.org/10.1016/j.infsof.2009.04.007.

[S62] Glezer, C., Last, M., Nachmany, E., and Shoval, P. (2005). Quality and comprehension of UML interaction diagrams-an experimental comparison. Information and Software Technology, 47(10), 675-692, https://doi.org/10.1016/j.infsof.2005.01.003.
[S63] Bolloju, N., and Sugumaran, V. (2012). A knowledge-based object modeling advisor for developing quality object models. Expert Systems with Applications, 39(3), 2893-2906, https://doi.org/10.1016/j.eswa.2011.08.151.
[S64] Aguilera, D., Gómez, C., and Olivé, A. (2013). A complete set of guidelines for naming UML conceptual schema elements. Data and Knowledge Engineering, 88, 60-74, https://doi.org/10.1016/j.datak.2013.09.001.
[S65] Tort, A., and Olivé, A. (2010). An approach to testing conceptual schemas. Data and Knowledge Engineering, 69(6), 598-618, https://doi.org/10.1016/j.datak.2010.02.002.
[S66] Opdahl, A. L., and Henderson-Sellers, B. (2005). A unified modelling language without referential redundancy. Data and Knowledge Engineering, 55(3), 277-300, https://doi.org/10.1016/j.datak.2004.12.007.
[S67] Jošt, G., Huber, J., Heričko, M., and Polančič, G. (2016). An empirical investigation of intuitive understandability of process diagrams. Computer Standards and Interfaces, 48, 90-111, https://doi.org/10.1016/j.csi.2016.04.006.
[S68] Wenzel, S., Poggenpohl, D., Jürjens, J., and Ochoa, M. (2014). Specifying model changes with UMLchange to support security verification of potential evolution. Computer Standards and Interfaces, 36(4), 776-791, https://doi.org/10.1016/j.csi.2013.12.011.
[S69] Haug, A., Hvam, L., and Mortensen, N. H. (2010). A layout technique for class diagrams to be used in product configuration projects. Computers in Industry, 61(5), 409-418, https://doi.org/10.1016/j.compind.2009.10.002.
[S70] Gogolla, M., Hilken, F., and Doan, K. H. (2018). Achieving model quality through model validation, verification and exploration. Computer Languages, Systems and Structures, 54, 474-511, https://doi.org/10.1016/j.cl.2017.10.001.
[S71] Planas, E., Cabot, J., and Gómez, C. (2016). Lightweight and static verification of UML executable models. Computer Languages, Systems and Structures, 46, 66-90, https://doi.org/10.1016/j.cl.2016.07.002.
[S72] Kchaou, D., Bouassida, N., and Ben-Abdallah, H. (2016). Managing the impact of UML design changes on their consistency and quality. Arabian Journal for Science and Engineering, 41(8), 2863-2881, https://doi.org/10.1007/s13369-016-2040-7.
[S73] Zafar, N. A. (2016). Formal specification and verification of few combined fragments of UML sequence diagram. Arabian Journal for Science and Engineering, 41(8), 2975-2986, https://doi.org/10.1007/s13369-015-1999-9.
[S74] López-Fernández, Guerra, and De Lara. (2014). Assessing the quality of meta-models, CEUR Workshop Proc., vol. 1235, no. 4, pp. 3–12, 2014.
[S75] Felderer, M., and Herrmann, A. (2019). Comprehensibility of system models during test design: a controlled experiment comparing UML activity diagrams and state machines. Software Quality Journal, 27(1), 125-147, https://doi.org/10.1007/s11219-018-9407-9.
[S76] Ganser, A., Lichter, H., Roth, A., and Rumpe, B. (2016). Staged model evolution and proactive quality guidance for model libraries. Software Quality Journal, 24(3), 675-708, https://doi.org/10.1007/s11219-015-9298-y.
[S77] Choi, J., Jee, E., and Bae, D. H. (2016). Timing consistency checking for UML/MARTE behavioral models. Software Quality Journal, 24(3), 835-876, https://doi.org/10.1007/s11219-015-9290-6.
[S78] Mansoor, U., Kessentini, M., Wimmer, M., and Deb, K. (2017). Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. Software Quality Journal, 25(2), 473-501, https://doi.org/10.1007/s11219-015-9284-4.

[S79] Phalp, K. T., Vincent, J., and Cox, K. (2007). Assessing the quality of use case descriptions. Software Quality Journal, 15(1), 69-97, https://doi.org/10.1007/s11219-006-9006-z.
[S80] Khan, Y. A., and El-Attar, M. (2016). Using model transformation to refactor use case models based on antipatterns. Information systems frontiers, 18(1), 171-204, https://doi.org/10.1007/s10796-014-9528-z.
[S81] Amálio, N., Briand, L., and Kelsen, P. (2020). An experimental scrutiny of visual design modelling: VCL up against UML+ OCL. Empirical Software Engineering, 25(2), 1205-1258, https://doi.org/10.1007/s10664-019-09784-9.
[S82] I Reinhartz-Berger, I., and Sturm, A. (2014). Comprehensibility of UML-based software product line specifications. Empirical Software Engineering, 19(3), 678-713.
[S83] De Lucia, A., Gravino, C., Oliveto, R., and Tortora, G. (2010). An experimental comparison of ER and UML class diagrams for data modelling. Empirical Software Engineering, 15(5), 455-492, https://doi.org/10.1007/s10664-009-9127-7.
[S84] Cepeda Porras, G., and Guéhéneuc, Y. G. (2010). An empirical study on the efficiency of different design pattern representations in UML class diagrams. Empirical Software Engineering, 15(5), 493-522, https://doi.org/10.1007/s10664-009-9125-9.
[S85] Cruz-Lemus, J. A., Genero, M., Manso, M. E., Morasca, S., and Piattini, M. (2009). Assessing the understandability of UML statechart diagrams with composite states—A family of empirical studies. Empirical Software Engineering, 14(6), 685-719, https://doi.org/10.1007/s10664-009-9106-z.
[S86] Genero, M., Manso, E., Visaggio, A., Canfora, G., and Piattini, M. (2007). Building measure-based prediction models for UML class diagram maintainability. Empirical Software Engineering, 12(5), 517-549.
[S87] Bunse, C. (2006). Using patterns for the refinement and translation of UML models: A controlled experiment. Empirical Software Engineering, 11(2), 227-267, https://doi.org/10.1007/s10664-006-6403-7.
[S88] Anda, B., and Sjøberg, D. I. (2005). Investigating the role of use cases in the construction of class diagrams. Empirical Software Engineering, 10(3), 285-309, https://doi.org/10.1007/s10664-005-1289-3.
[S89] Balaban, M., and Maraee, A. (2019). Removing redundant multiplicity constraints in UML class models. Software and Systems Modeling, 18(4), 2717-2751, https://doi.org/10.1007/s10270-018-0696-z.
[S90] Chatzigeorgiou, A. (2003). Mathematical assessment of object-oriented design quality. IEEE Transactions on Software Engineering, 29(11), 1050-1053, https://doi.org/10.1109/TSE.2003.1245306.
[S91] Mens, T., Decan, A., and Spanoudakis, N. I. (2019). A method for testing and validating executable statechart models. Software and Systems Modeling, 18(2), 837-863, https://doi.org/10.1007/s10270-018-0676-3.
[S92] D Zayan, D., Sarkar, A., Antkiewicz, M., Maciel, R. S. P., and Czarnecki, K. (2019). Example-driven modeling: on effects of using examples on structural model comprehension, what makes them useful, and how to create them. Software and Systems Modeling, 18(3), 2213-2239, https://doi.org/10.1007/s10270-017-0652-3.
[S93] Störrle, H. (2018). On the impact of size to the understanding of UML diagrams. Software and Systems Modeling, 17(1), 115-134, https://doi.org/10.1007/s10270-016-0529-x.
[S94] Balaban, M., Maraee, A., Sturm, A., and Jelnov, P. (2015). A pattern-based approach for improving model quality. Software and Systems Modeling, 14(4), 1527-1555, https://doi.org/10.1007/s10270-013-0390-0.
[S95] Bavota, G., Gravino, C., Oliveto, R., De Lucia, A., Tortora, G., Genero, M., and Cruz-Lemus, J. A. (2015). A fine-grained analysis of the support provided by UML class

diagrams and ER diagrams during data model maintenance. Software and Systems Modeling, 14(1), 287-306, https://doi.org/10.1007/s10270-012-0312-6.
[S96] Ali, S., Yue, T., and Briand, L. C. (2014). Does aspect-oriented modeling help improve the readability of UML state machines?. Software and Systems Modeling, 13(3), 1189-1221.
[S97] El-Attar, M., and Miller, J. (2010). Improving the quality of use case models using antipatterns. Software and systems modeling, 9(2), 141-160, https://doi.org/10.1007/s10270-009-0112-9.
[S98] Beimel, D., and Kedmi-Shahar, E. (2019). Improving the identification of functional system requirements when novice analysts create use case diagrams: the benefits of applying conceptual mental models. Requirements Engineering, 24(4), 483-502, https://doi.org/10.1007/s00766-018-0296-z.
[S99] Bera, P., and Evermann, J. (2014). Guidelines for using UML association classes and their effect on domain understanding in requirements engineering. Requirements Engineering, 19(1), 63-80, https://doi.org/10.1007/s00766-012-0159-y.
[S100] Dahan, M., Shoval, P., and Sturm, A. (2014). Comparing the impact of the OO-DFD and the Use Case methods for modeling functional requirements on comprehension and quality of models: a controlled experiment. Requirements Engineering, 19(1), 27-43, https://doi.org/10.1007/s00766-012-0155-2.
[S101] Bolloju, N., and Alter, S. (2016). Better use case diagrams by using work system snapshots. International Journal of Information Technologies and Systems Approach (IJITSA), 9(2), 1-22, https://doi.org/10.4018/IJITSA.2016070101.
[S102] Baudry, B., and Le Traon, Y. (2005). Measuring design testability of a UML class diagram. Information and software technology, 47(13), 859-879, https://doi.org/10.1016/j.infsof.2005.01.006.
[S103] Burgués, X., Franch, X., and Ribó, J. M. (2008). Improving the accuracy of UML metamodel extensions by introducing induced associations. Software and Systems Modeling, 7(3), 361-379, https://doi.org/10.1007/s10270-007-0062-z.
[S104] Boot, W. R., Dunn, C. L., Fulmer, B. P., Gerard, G. J., and Grabski, S. V. (2022). An eye tracking experiment investigating synonymy in conceptual model validation. International Journal of Accounting Information Systems, 47, 100578, https://doi.org/10.1016/j.accinf.2022.100578.
[S105] Sharbaf, M., Zamani, B., and Sunyé, G. (2022). Automatic resolution of model merging conflicts using quality-based reinforcement learning. Journal of Computer Languages, 71, 101123, https://doi.org/10.1016/j.cola.2022.101123.
[S106] Al-Fedaghi, S. (2021). UML modeling to TM modeling and back. arXiv preprint arXiv:2102.02982, https://doi.org/10.48550/arXiv.2102.02982.
[S107] Barriga, A., Bettini, L., Iovino, L., Rutle, A., and Heldal, R. (2021). Addressing the trade off between smells and quality when refactoring class diagrams. The Journal of Object Technology, 20(3), 1-1., http://dx.doi.org/10.5381/jot.2021.20.3.a1.
[S108] Sanyal, R. (2020). Impact of passive and negative sentences in automatic generation of static UML diagram using NLP. Journal of Intelligent and Fuzzy Systems, 39(2), 2047-2059, http://dx.doi.org/10.3233/JIFS-179871.
[S109] Albaghajati, A., and Hassine, J. (2022). A use case driven approach to game modeling. Requirements Engineering, 1-34, https://doi.org/ http://dx.doi.org/10.1007/s00766-021-00362-4.
[S110] Akiki, P. A., and Maalouf, H. W. (2021). CHECKSUM: tracking changes and measuring contributions in cooperative systems modeling. Software and Systems Modeling, 20(4), 1079-1122, http://dx.doi.org/10.1007/s10270-020-00840-3.

[S111] Flores, P., Alvarez, M., and Torres, J. (2022). Identifying Difficulties of Software Modeling Through Class Diagrams: A Long-Term Comparative Analysis. IEEE Access, 10, 28895-28910, https://doi.org/10.1109/ACCESS.2022.3157290.
[S112] Priyambadha, B., and Katayama, T. (2022). Design Level Class Decomposition using the Threshold-based Hierarchical Agglomerative Clustering. International Journal of Advanced Computer Science and Applications, 13(3) , https://dx.doi.org/10.14569/IJACSA.2022.0130310.
[S113] Ozcan, E., Topalli, D., Tokdemir, G., and Cagiltay, N. E. (2021). A user task design notation for improved software design. PeerJ Computer Science, 7, e503, https://doi.org/10.7717/peerj-cs.503.
[S114] Barriga, A., Heldal, R., Rutle, A., and Iovino, L. (2022). PARMOREL: a framework for customizable model repair. Software and Systems Modeling, 21(5), 1739-1762, https://doi.org/10.1007/s10270-022-01005-0.
[S115] Mahmood, W., Strüber, D., Anjorin, A., and Berger, T. (2022). Effects of variability in models: a family of experiments. Empirical Software Engineering, 27(3), 72, https://doi.org/10.1007/s10664-021-10112-3.
[S116] Beimel, D., Tsoury, A., and Lev, S. B. (2023). Identifying common and persistent errors made by novice analysts when modeling business processes using UML activity diagram: utilizing a hierarchical error classification. Software Quality Journal, 31(4), 1149-1178, https://doi.org/s11219-023-09628-2.
[S117] Haridy, S., Ismail, R. M., Badr, N., and Hashem, M. (2023). An ontology development methodology based on ontology-driven conceptual modeling and natural language processing: tourism case study. Big Data and Cognitive Computing, 7(2), 101. https://doi.org/10.3390/bdcc7020101.
[S118] Feichas, F. A., and Seabra, R. D. (2023). Evaluation of perception of use of a gamified platform from the student perspective: An approach for studying unified modeling language. *Informatics in Education*, *22*(3), 369-394. https://doi.org/10.15388/infedu.2023.22.
[S119] Chen, C. Y., and Tai, K. Y. (2023). Online ontological quality assessment of converted UML class diagrams in SRE. *Automated Software Engineering*, *30*(2), 18. https://doi.org/10.1007/s10515-023-00385-x.
[S120] Yang, P., Zhang, L., Li, Q., Gao, X., and Yang, Y. (2025). OCLVerifer: Automated verification of OCL contracts in requirements models. *Science of Computer Programming*, *240*, 103197. https://doi.org/10.1016/j.scico.2024.103197.
[S121] NASIRI, S., and LAHMER, M. (2024). A Smart AI Framework for Backlog Refinement and UML Diagram Generation. *International Journal of Advanced Computer Science and Applications*, *15*(4). https://doi.org/10.14569/IJACSA.2024.0150474.
[S122] Yuan, Z., Hu, X., Zhang, G., and Ma, Z. (2024). Integration of UML Class Diagrams Based on Semantics and Structure. *International Journal of Software Engineering and Knowledge Engineering*, *34*(08), 1281-1312. https://doi.org/10.1142/S0218194024500200.
[S123] Elekes, M., Molnár, V., and Micskei, Z. (2023). Assessing the specification of modelling language semantics: a study on UML PSSM. *Software Quality Journal*, *31*(2), 575-617. https://doi.org/10.1007/s11219-023-09617-5.
[S124] Meliá, S., Reyes, R., and Cachero, C. (2023). The Effect of Developers' General Intelligence on the Understandability of Domain Models: An Empirical Study. *IEEE Access*, *11*, 70153-70167. https://doi.org/10.1109/ACCESS.2023.3293199.
[S125] Li, Q., and Zeng, F. (2024). Enhancing Software Architecture Adaptability: A Comprehensive Evaluation Method. *Symmetry*, *16*(7), 894. https://doi.org/10.3390/sym16070894.

[S126] Ahmad, K., Abdelrazek, M., Arora, C., Bano, M., and Grundy, J. (2023). Requirements practices and gaps when engineering human-centered Artificial Intelligence systems. *Applied Soft Computing*, *143*, 110421. https://doi.org/10.1016/j.asoc.2023.110421.

[S127] Beimel, D., Tsoury, A., and Lev, S. B. (2023). Identifying common and persistent errors made by novice analysts when modeling business processes using UML activity diagram: utilizing a hierarchical error classification. *Software Quality Journal*, *31*(4), 1149-1178. https://doi.org/10.1007/s11219-023-09628-2.

[S128] Garaccione, G., Coppola, R., Ardito, L., and Torchiano, M. (2025). Gamification of conceptual modeling education with UML class diagrams: an experimental analysis. *Software and Systems Modeling*, 1-32. https://doi.org/10.1007/s10270-025-01282-5.

[S129] Tsoury, A., Hindy Ling, H., Beimel, D., and Bar-Lev, S. (2025). From omelet to UML: assessing the role of process thinking on process model quality among novice analyst. *Software Quality Journal*, *33*(1), 1-30. https://doi.org/10.1007/s11219-025-09713-8.

[S130] Garaccione, G., Coppola, R., Ardito, L., and Torchiano, M. (2025). Gamification of conceptual modeling education: an analysis of productivity and students' perception. *Software Quality Journal*, *33*(1), 1-19. https://doi.org/10.1007/s11219-024-09701-4.

[S131] Ņikiforova, O., Babris, K., and Guliyeva, A. (2024). Definition of a Set of Use Case Patterns for Application Systems: A Prototype-Supported Development Approach. Applied Computer Systems, 29(1), 59-67. https://doi.org/10.2478/acss-2024-0008.

[S132] Reinhartz-Berger, I. (2024). Challenges in software model reuse: cross application domain vs. cross modeling paradigm. *Empirical Software Engineering*, *29*(1), 16. https://doi.org/10.1007/s10664-023-10386-9.

[S133] Chakraborty, S., and Liebel, G. (2023). We do not understand what it says–studying student perceptions of software modelling. *Empirical Software Engineering*, 28(6), 149. https://doi.org/10.1007/s10664-023-10404-w.

[S134] Dang, D. H., Le, D. M., and Le, V. V. (2023). AGL: Incorporating behavioral aspects into domain-driven design. *Information and Software Technology*, *163*, 107284. https://doi.org/10.1016/j.infsof.2023.107284.

[S135] Arfi, F., Courbis, A. L., Lambolais, T., Bughin, F., and Hayot, M. (2023). Formal verification of a telerehabilitation system through an abstraction and refinement approach using Uppaal. *IET Software*, *17*(4), 582-599. https://doi.org/10.1049/sfw2.12128.

[S136] Bimonte, S., Badir, H., Battistoni, P., Bazza, H., Belhassena, A., Cariou, C., ... and Tricot, N. (2023). Data-centric UML profile for agroecology applications: Agricultural autonomous robots monitoring case study. *Computer Science and Information Systems*, 20(1), 459-489. https://doi.org/10.2298/CSIS220301064B.

[S137] Genero, M., Piatini, M., & Manso, E. (2004). Finding" early" indicators of UML class diagrams understandability and modifiability. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04.* (pp. 207-216). IEEE. https://doi.org/10.1109/ISESE.2004.1334908.

[S138] Cruz-Lemus, J. A., Genero, M., Manso, M. E., & Piattini, M. (2005). Evaluating the effect of composite states on the understandability of UML statechart diagrams. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 113-125). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/11557432_9

[S139] Badreddin, O., Elaasar, M., & Hamou-Lhadj, A. (2016). A controlled experiment for evaluating the comprehensibility of UML action languages. In *2016 4th International*

Conference on Model-Driven Engineering and Software Development (MODELSWARD) (pp. 52-64). IEEE.

[S140] Cruz-Lemus, J. A., Genero, M., Morasca, S., & Piattini, M. (2007). Using practitioners for assessing the understandability of UML statechart diagrams with composite states. In *International Conference on Conceptual Modeling* (pp. 213-222). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-76292-8_25

[S141] Reinhartz-Berger, I., & Sturm, A. (2014). Comprehensibility of UML-based software product line specifications: A controlled experiment. *Empirical Software Engineering*, *19*(3), 678-713. https://doi.org/10.1007/s10664-012-9234-8.

[S142] Razali, R., & Garratt, P. W. (2006). Measuring the Comprehensibility of a UML-B Model and a B Model. *Enformatika*, *16*, 338-343.

[S143] Bashir, R. S., Lee, S. P., Khan, S. U. R., Chang, V., & Farid, S. (2016). UML models consistency management: Guidelines for software quality manager. *International Journal of Information Management*, *36*(6), 883-899. https://doi.org/10.1016/j.ijinfomgt.2016.05.024.

[S144] Lange, C. F., & Chaudron, M. R. (2005). Managing model quality in UML-based software development. In *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)* (pp. 7-16). IEEE. https://doi.org/10.1109/STEP.2005.16.

[S145] Deryugina, O. (2016). Improving the structural quality of UML class diagrams with the genetic algorithm. In *ITM Web of Conferences* (Vol. 6, p. 03003). EDP Sciences. https://doi.org/10.1051/itmconf/20160603003.

[S146] Bergström, G., Hujainah, F., Ho-Quang, T., Jolak, R., Rukmono, S. A., Nurwidyantoro, A., & Chaudron, M. R. (2022). Evaluating the layout quality of UML class diagrams using machine learning. *Journal of Systems and Software*, *192*, 111413. https://doi.org/10.1016/j.jss.2022.111413.

[S147] Melia, S., Reyes, R., & Cachero, C. (2023). The Effect of Developers' General Intelligence on the Understandability of Domain Models: An Empirical Study. *IEEE Access*, *11*, 70153-70167. https://doi.org/10.1109/ACCESS.2023.3293199

[S148] Ukić, N., Maras, J., & Šerić, L. (2018). The influence of cyclomatic complexity distribution on the understandability of xtUML models. *Software quality journal*, *26*(2), 273-319. https://doi.org/10.1007/s11219-016-9351-5.

[S149] Störrle, H. (2016). Diagram size vs. layout flaws: understanding quality factors of uml diagrams. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10). https://doi.org/10.1145/2961111.2962609.

[S150] Reinhartz-Berger, I., & Dori, D. (2005). OPM vs. UML--Experimenting with Comprehension and Construction of Web Application Models. *Empirical Software Engineering*, *10*(1), 57-80. https://doi.org/10.1023/B:EMSE.0000048323.40484.e0.

[S151] Burd, E., Overy, D., & Wheetman, A. (2002). Evaluating using animation to improve understanding of sequence diagrams. In *Proceedings 10th International Workshop on Program Comprehension* (pp. 107-113). IEEE. https://doi.org/10.1109/WPC.2002.1021329.

[S152] Störrle, H. (2014). On the impact of layout quality to understanding UML diagrams: size matters. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 518-534). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-11653-2_32

[S153] Störrle, H. (2012). On the impact of layout quality to understanding UML diagrams: Diagram type and expertise. In *2012 IEEE Symposium on Visual Languages and Human-*

Centric Computing (VL/HCC) (pp. 49-56). IEEE. https://doi.org/10.1109/VLHCC.2011.6070390

[S154] Esperanza Manso, M., Cruz-Lemus, J. A., Genero, M., & Piattini, M. (2008). Empirical validation of measures for UML class diagrams: A meta-analysis study. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 303-313). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-01648-6_32

[S155] Estivill-Castro, V., & Hexel, R. (2019). Resolving the asymmetry of on-exit versus on-entry in executable models of behaviour. In *MODELSWARD 2019 7th International Conference on Model-Driven Engineering and Software Development*. Scitepress. https://doi.org/10.5220/0007323300490061.

[S156] Yue, T., Briand, L. C., & Labiche, Y. (2013). Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *22*(1), 1-38. https://doi.org/10.1145/2430536.2430539.

[S157] Bavota, G., Gravino, C., Oliveto, R., De Lucia, A., Tortora, G., Genero, M., & Cruz-Lemus, J. A. (2011). Identifying the weaknesses of UML class diagrams during data model comprehension. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 168-182). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-24485-8_13

[S158] Ali, S., & Yue, T. (2012). Studying the understandability of aspect state machines through the weaving activity. In *2012 19th Asia-Pacific Software Engineering Conference* (Vol. 1, pp. 593-602). IEEE. https://doi.org/10.1109/APSEC.2012.104.

[S159] De Bari, D., Garaccione, G., Coppola, R., Torchiano, M., & Ardito, L. (2024). Evaluating large language models in exercises of uml class diagram modeling. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 393-399). https://doi.org/10.1145/3674805.3690741.

[S160] Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., & Ceccato, M. (2007). The role of experience and ability in comprehension tasks supported by UML stereotypes. In *29th International Conference on Software Engineering (ICSE'07)* (pp. 375-384). IEEE. https://doi.org/10.1109/ICSE.2007.86

[S161] Estivill-Castro, V., & Hexel, R. (2019). The understandability of models for behaviour. In *International Conference on Model-Driven Engineering and Software Development* (pp. 50-75). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-37873-8_3.

[S162] Wang, C., Wang, B., Liang, P., & Liang, J. (2026). Assessing UML diagrams by GPT: Implications for education. *Journal of Systems and Software*, 112709. https://doi.org/10.1016/j.jss.2025.112709.

[S163] Reggio, G., Ricca, F., Scanniello, G., Di Cerbo, F., & Dodero, G. (2011). A precise style for business process modelling: Results from two controlled experiments. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 138-152). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-24485-8_11

[S164] Saghafi, A., & Wand, Y. (2014). Do ontological guidelines improve understandability of conceptual models? a meta-analysis of empirical work. In *2014 47th Hawaii International Conference on System Sciences* (pp. 4609-4618). IEEE. https://doi.org/10.1109/HICSS.2014.567.

[S165] Reggio, G., Ricca, F., Scanniello, G., Cerbo, F. D., & Dodero, G. (2015). On the comprehension of workflows modeled with a precise style: results from a family of controlled experiments. *Software & Systems Modeling*, *14*(4), 1481-1504. https://doi.org/10.1007/s10270-013-0386-9.

[S166] Genero, M., Piattini, M., & Calero, C. (2002). Empirical validation of class diagram metrics. In *Proceedings International Symposium on Empirical Software Engineering* (pp. 195-203). IEEE. https://doi.org/10.1109/SCCC.2001.972637.

[S167] Garaccione, G., Coppola, R., Ardito, L., & Torchiano, M. (2024). Gamification of conceptual modeling education: an analysis of productivity and students' perception. *Software Quality Journal*, *33*(1), 3. https://doi.org/10.1007/s11219-024-09701-4

[S168] Lange, C. F., & Chaudron, M. R. (2007). Interactive views to improve the comprehension of UML models-an experimental validation. In *15th IEEE International Conference on Program Comprehension (ICPC'07)* (pp. 221-230). IEEE. https://doi.org/10.1109/ICPC.2007.23

[S169] Garaccione, G., Coppola, R., Ardito, L., & Torchiano, M. (2026). Gamification of conceptual modeling education with UML class diagrams: an experimental analysis: G. Garaccione et al. *Software and Systems Modeling*, *25*(1), 239-270. https://doi.org/10.1007/s10270-025-01282-5.

[S170] Störrle, H. (2013). Towards clone detection in UML domain models. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume* (pp. 285-293). https://doi.org/10.1007/s10270-011-0217-9.

[S171] Ghaleb, T. A., Aljasser, K., & Alturki, M. A. (2020). Enhanced visualization of method invocations by extending reverse-engineered sequence diagrams. In *2020 Working Conference on Software Visualization (VISSOFT)* (pp. 49-60). IEEE. https://doi.org/10.1109/VISSOFT51673.2020.00010.

[S172] AbuHassan, A., & Alshayeb, M. (2019). A metrics suite for UML model stability. *Software & Systems Modeling*, *18*(1), 557-583. https://doi.org/10.1007/s10270-016-0573-6.

[S173] Conradi, R., Mohagheghi, P., Arif, T., Hegde, L. C., Bunde, G. A., & Pedersen, A. (2003). Object-oriented reading techniques for inspection of UML models–an industrial experiment. In *European Conference on Object-Oriented Programming* (pp. 483-500). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-45070-2_21

[S174] Manso, M. E., Genero, M., & Piattini, M. (2003). No-redundant metrics for UML class diagram structural complexity. In International Conference on Advanced Information Systems Engineering (pp. 127-142). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-45017-3_11

[S175] Sheldon, F. T., & Chung, H. (2006). Measuring the complexity of class diagrams in reverse engineering. *Journal of Software Maintenance and Evolution: Research and Practice*, *18*(5), 333-350. https://doi.org/10.1002/smr.336.

[S176] Du Bois, B., Lange, C. F., Demeyer, S., & Chaudron, M. R. (2007). A qualitative investigation of UML modeling conventions. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 91-100). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-69489-2_12

[S177] Störrle, H., Baltsen, N., Christoffersen, H., & Maier, A. M. (2018). How do modelers read UML diagrams? preliminary results from an eye-tracking study. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (pp. 396-397). https://doi.org/10.1145/3183440.3195025.

[S178] Lebtag, B. G. A., Teixeira, P. G., Santos, R. P. D., Viana, D., & Neto, V. V. G. (2020). Evaluating the Understandability and Expressiveness of Simulation Executable Models with Professionals: Obtaining perceptions from researchers and practitioners for improving quality of models. In Proceedings of the XIX Brazilian Symposium on Software Quality (pp. 1-10). https://doi.org/10.1145/3439961.3439973.

[S179] Genero, M., Piattini, M., & Jiménez, L. (2001). Empirical validation of class diagram complexity metrics. In *SCCC 2001. 21st International Conference of the Chilean Computer Science Society* (pp. 95-104). IEEE. https://doi.org/10.1109/SCCC.2001.972637.

[S180] Genero, M., Olivas, J., Piattini, M., & Romero, F. (2003). Assessing object-oriented conceptual models maintainability. In *International Conference on Conceptual Modeling* (pp. 288-299). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-45275-1_26

[S181] Genero, M., Piattini, M., Manso, E., & Cantone, G. (2004). Building UML class diagram maintainability prediction models based on early metrics. In *Proceedings. 5th International Workshop on Enterprise Networking and Computing in Healthcare Industry (IEEE Cat. No. 03EX717)* (pp. 263-275). IEEE. https://doi.org/10.1109/METRIC.2003.1232473.

[S182] Chaudron, M. R., Katumba, B., & Ran, X. (2014). Automated prioritization of metrics-based design flaws in UML class diagrams. In *2014 40th EUROMICRO conference on software engineering and advanced applications* (pp. 369-376). IEEE. 10.1109/SEAA.2014.82

[S183] Dahan, M., Shoval, P., & Sturm, A. (2014). Comparing the impact of the OO-DFD and the Use Case methods for modeling functional requirements on comprehension and quality of models: a controlled experiment. *Requirements Engineering*, *19*(1), 27-43. https://doi.org/10.1007/s00766-012-0155-2.

[S184] Karasneh, B., Jolak, R., & Chaudron, M. R. (2015). Using examples for teaching software design: An experiment using a repository of uml class diagrams. In *2015 Asia-Pacific Software Engineering Conference (APSEC)* (pp. 261-268). IEEE. https://doi.org/10.1109/APSEC.2015.59

[S185] Störrle, H. (2016). Diagram size vs. layout flaws: understanding quality factors of uml diagrams. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 1-10). https://doi.org/10.1145/2961111.2962609.

[S186] Störrle, H., Baltsen, N., Christoffersen, H., & Maier, A. M. (2018). How do modelers read UML diagrams? preliminary results from an eye-tracking study. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings* (pp. 396-397). https://doi.org/10.1145/3183440.3195025.

[S187] Ibáñez, M. B., Barrón-Estrada, M. L., & Zatarain-Cabada, R. (2025). Can multimodal large language models grade like an expert? A study on UML class diagram assessment accuracy. *Computer Applications in Engineering Education*, *33*(5), e70080. https://doi.org/10.1002/cae.70080.

[S188] Nguyen, V. V., Nguyen, H. K., Nguyen, K. S., Luong, T. M. H., Vu, D. Q., & Phung, T. N. (2026). A novel unified framework for automated generation and multimodal validation of UML diagrams. *Computer Modeling in Engineering & Sciences*, *146*(1), 33. https://doi.org/10.32604/cmes.2025.075442.

[S189] Freire, E. S. S., Oliveira, G. C., & de Sousa Gomes, M. E. (2018). Analysis of open-source CASE tools for supporting software modeling process with UML. In *Proceedings of the XVII brazilian symposium on software quality* (pp. 51-60). https://doi.org/10.1145/3275245.3275251.

[S190] Júnior, E., & Farias, K. (2021). ModelGame: A quality model for gamified software modeling learning. In *Proceedings Of The 15th Brazilian Symposium On Software Components, Architectures, And Reuse* (pp. 100-109). https://doi.org/10.1145/3483899.3483910.

[S191] Ganapathy, N. B. S., & Alagarsamy, K. (2017). Improving and analyzing the quality of system model lexicons using semantic based information mining. In 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS) (pp. 3827-3835). IEEE. https://doi.org/10.1109/ICECDS.2017.8390180.
[S192] Raissi, R. (2017). Improving readability of software model with adding visualized extra information to uml diagram. In *Proceedings of the International Conference on Compute and Data Analysis* (pp. 28-32). https://doi.org/10.1145/3093241.3093255.
[S193] Insfran, E. (2008). Towards quality-driven model transformations: A replication study.
[S194] Bolloju, N., & Schneider, C. (2011). Facilitating the Creation of Quality Object Models Using a Knowledge Based System. In *Information Systems Development: Business Systems and Services: Modeling and Development* (pp. 137-147). New York, NY: Springer New York. https://doi.org/10.1007/978-1-4419-9790-6.
[S195] Meena, S., & Vishwakarma, R. G. (2012). CMMI based software metrics to evaluate OOAD. In Proceedings of the Second International Conference on Computational Science. https://doi.org/10.1145/2393216.2393220.
[S196] Baruzzo, A., & Comini, M. (2008). A methodology for uml models v&v. In *2008 1st International Conference on Software Testing, Verification, and Validation* (pp. 513-516). IEEE. https://doi.org/10.1109/ICST.2008.66
[S197] Fish, A., & Storrle, H. (2007). Visual qualities of the unified modeling language: deficiencies and improvements. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)* (pp. 41-49). IEEE. https://doi.org/10.1109/VLHCC.2007.62
[S198] Rodríguez, M., Genero, M., Torre, D., Blasco, B., & Piattini, M. (2010). A methodology for continuos quality assessment of software artefacts. In 2010 10th International Conference on Quality Software (pp. 254-261). IEEE. https://doi.org/10.1109/QSIC.2010.24
[S199] Maro, S., Steghöfer, J. P., Anjorin, A., Tichy, M., & Gelin, L. (2015). On integrating graphical and textual editors for a UML profile based domain specific language: an industrial experience. In Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering (pp. 1-12). https://doi.org/10.1145/2814251.2814253.
[S200] Hnatkowska, B. (2007). Verification of Good Design Style of UML Models. In *ISIM*.
[S201] Austrem, P. G. (2011). Using EUREQA for End-User UML Model Development through Design Patterns. J. Softw., 6(4), 690-704. https://doi.org/10.4304/jsw.6.4.690-704.
[S202] Mosconi, M., Charfi, A., Svacina, J., & Wloka, J. (2008). Applying and evaluating AOM for platform independent behavioral UML models. In *Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling* (pp. 19-24). https://doi.org/10.1145/1404920.1404924
[S203] Zamani, B., & Butler, G. (2009). Smell detection in UML designs which utilize pattern languages. Iranian Journal of Electrical and Computer Engineering (IJECE) 8(1), 47-52.
[S204] Yi, T. (2010). Comparison research of two typical UML-class-diagram metrics: Experimental software engineering. In 2010 International Conference on Computer Application and System Modeling (ICCASM 2010) (Vol. 12, pp. V12-86). IEEE. https://doi.org/10.1109/ICCASM.2010.5622152.
[S205] Razali, R., Snook, C. F., & Poppleton, M. R. (2007). Comprehensibility of uml-based formal model: A series of controlled experiments. In Proceedings of the 1st ACM international workshop on empirical assessment of software engineering languages and technologies: Held in conjunction with the 22nd IEEE/ACM international conference on

automated software engineering (ASE) 2007 (pp. 25-30). https://doi.org/10.1145/1353673.1353680

[S206] French L.J.; Zorman L. (2007). Measuring UML design quality in a real-time development environment. *16th International Conference on Software Engineering and* Data Engineering, SEDE 2007

[S207] Boukhelfa, K., Belala, F., Choutri, A., & DOuibi, H. (2010). For more understandable UML diagrams. In ACS/IEEE International Conference on Computer Systems and Applications-AICCSA 2010 (pp. 1-7). IEEE. https://doi.org/10.1109/AICCSA.2010.5586987.

[S208] Ben-Abdallah, H. (2015). CQV-UML Tool: a tool for managing the impact of change on UML models. In Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering. https://doi.org/10.18293/seke2015-40.

[S209] Figl, K., Mendling, J., & Strembeck, M. (2009). Towards a usability assessment of process modeling languages. In 8th GI-Workshop Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK), CEUR-WS (Vol. 554, pp. 138-156).

[S210] Rubin, J., & Chechik, M. (2010). From Products to Product Lines Using Model Matching and Refactoring. In *SPLC Workshops* (pp. 155-162). https://doi.org/10.1007/978-3-642-28872-2_20

[S211] España, S., Ruiz, M., & González, A. (2012). Systematic derivation of conceptual models from requirements models: a controlled experiment. In *2012 Sixth International Conference on Research Challenges in Information Science (RCIS)* (pp. 1-12). IEEE. https://doi.org/10.1109/RCIS.2012.6240428

[S212] Xiao, H., Li, S., & Wang, B. (2009). A tool for the application of software metrics to UML class diagram. In *2009 First International Workshop on Education Technology and Computer Science* (Vol. 1, pp. 181-184). IEEE. https://doi.org/10.1109/ETCS.2009.48

[S213] Cestra, G., Liguori, G., & Clementini, E. (2013). An Extended UML Metamodel for Efficient Application Design and Development. *International Journal of Software Engineering and Its Applications*, *7*(4), 359-378.

[S214] Karasneh, B., Stikkolorum, D. R., Larios, E., Sturm, A., & Clark, T. (2015). Quality Assessment of UML Class Diagrams-A Study Comparing Experts and Students. In *EduSymp@ MoDELS* (pp. 55-67).

[S215] Zhang, B., & Chen, Y. S. (2005). Enhancing UML conceptual modeling through the use of virtual reality. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences* (pp. 11b-11b). IEEE. https://doi.org/10.1109/HICSS.2005.239

[S216] Lange, C. F. (2006). Improving the quality of UML models in practice. In *Proceedings of the 28th international conference on Software engineering* (pp. 993-996). https://doi.org/10.1145/1134285.1134472

[S217] Leung, F., & Bolloju, N. (2005). Analyzing the quality of domain models developed by novice systems analysts. In *Proceedings of the 38th annual Hawaii international conference on system sciences* (pp. 188b-188b). IEEE. https://doi.org/10.1109/HICSS.2005.98

[S218] Sharma, M., & Chandwani, M. (2006). Quality in Business Modeling using UML. In *2006 1st International Conference on Digital Information Management* (pp. 294-299). IEEE. https://doi.org/10.1109/ICDIM.2007.369213

[S219] Razali, R., Snook, C. F., Poppleton, M. R., Garratt, P. W., & Walters, R. (2007). Experimental Comparison of the Comprehensibility of a UML-based Formal Specification versus a Textual One. In *11th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. BCS Learning & Development. https://doi.org/10.14236/ewic/EASE2007.1.

[S220] Eichelberger, H. (2003). Nice class diagrams admit good design?. In *Proceedings of the 2003 ACM symposium on Software visualization* (pp. 159-ff). https://doi.org/10.1145/774833.774857

[S221] Genero, M., Piatini, M., & Manso, E. (2004). Finding" early" indicators of UML class diagrams understandability and modifiability. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04.* (pp. 207-216). IEEE. https://doi.org/10.1109/ISESE.2004.1334908.

[S222] Günzler, A., & von Roennebeck, D. (2004). Towards an automated quality assessment of UML-based software architectures. In *IASTED Conf. on Software Engineering and Applications* (pp. 167-172).

[S223] Lange, C. F., & Chaudron, M. R. (2006). Effects of defects in UML models: an experimental investigation. In *Proceedings of the 28th international conference on Software engineering* (pp. 401-411). https://doi.org/10.1145/1134285.1134341

[S224] Bolloju, N. (2006). Exploring quality dependencies among UML artifacts developed by novice systems analysts. *AMCIS 2006 Proceedings*, 472.

[S225] Amrit, C., & Tax, N. (2014). Towards understanding the understandability of uml models. In *Proceedings of the 6th International Workshop on Modeling in Software Engineering* (pp. 49-54). https://doi.org/10.1145/2593770.2593779

[S226] Enckevort, T. V. (2009). Refactoring UML models: using openarchitectureware to measure uml model quality and perform pattern matching on UML models with OCL queries. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (pp. 635-646). https://doi.org/10.1145/1639950.1639959

[S227] Razali, R., Snook, C. F., & Poppleton, M. R. (2007). Comprehensibility of uml-based formal model: A series of controlled experiments. In *Proceedings of the 1st ACM international workshop on empirical assessment of software engineering languages and technologies: Held in conjunction with the 22nd IEEE/ACM international conference on automated software engineering (ASE) 2007* (pp. 25-30). https://doi.org/10.1145/1353673.1353680

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- APPENDIXA.docx