The Journal of Systems & Software 195 (2023) 111515

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

A decade of code comment quality assessment: A systematic literature review☆

Pooja Rani [a,*], Arianna Blasi [b], Nataliia Stulova [a], Sebastiano Panichella [c], Alessandra Gorla [d], Oscar Nierstrasz [a]

[a] Software Composition Group, University of Bern, Bern, Switzerland
[b] Università della Svizzera italiana, Lugano, Switzerland
[c] Zurich University of Applied Science, Zurich, Switzerland
[d] IMDEA Software Institute, Madrid, Spain

ARTICLE INFO

Article history:
Received 8 October 2021
Received in revised form 3 August 2022
Accepted 12 September 2022
Available online 22 September 2022

Keywords:
Code comments
Documentation quality
Systematic literature review

ABSTRACT

Code comments are important artifacts in software systems and play a paramount role in many software engineering (SE) tasks related to maintenance and program comprehension. However, while it is widely accepted that high quality matters in code comments just as it matters in source code, *assessing* comment quality in practice is still an open problem. First and foremost, there is no unique definition of quality when it comes to evaluating code comments. The few existing studies on this topic rather focus on specific attributes of quality that can be easily quantified and measured. Existing techniques and corresponding tools may also focus on comments bound to a specific programming language, and may only deal with comments with specific scopes and clear goals (*e.g.,* Javadoc comments at the method level, or in-body comments describing TODOs to be addressed).

In this paper, we present a Systematic Literature Review (SLR) of the last decade of research in SE to answer the following research questions: (i) What *types of comments* do researchers focus on when assessing comment quality? (ii) What *quality attributes* (QAs) do they consider? (iii) Which *tools and techniques* do they use to assess comment quality?, and (iv) How do they *evaluate* their studies on comment quality assessment in general?

Our evaluation, based on the analysis of 2353 papers and the actual review of 47 relevant ones, shows that (i) most studies and techniques focus on comments in Java code, thus may not be generalizable to other languages, and (ii) the analyzed studies focus on four main QAs of a total of 21 QAs identified in the literature, with a clear predominance of checking *consistency* between comments and the code. We observe that researchers rely on manual assessment and specific heuristics rather than the automated assessment of the comment quality attributes, with evaluations often involving surveys of students and the authors of the original studies but rarely professional developers.

1. Introduction

Software systems are often written in several programming languages (Abidi and Khomh, 2020), and interact with many hardware devices and software components (Lehman et al., 1997; Törngren and Sellgren, 2018). To deal with such complexity and to ease maintenance tasks, developers tend to document their software with various artifacts, such as design documents and code comments (de Souza et al., 2005). Several studies have demonstrated that *high quality* code comments can support developers in software comprehension, bug detection, and program maintenance activities (Dekel and Herbsleb, 2009; McMillan et al., 2010; Tan et al., 2007). However, code comments are typically written using natural language sentences, and their syntax is neither imposed by a programming language's grammar nor checked by its compiler. Additionally, static analysis tools and linters provide limited syntactic support to check comment quality. Therefore, writing high-quality comments and maintaining them in projects is a responsibility mostly left to developers (Allamanis et al., 2014; Kernighan and Pike, 1999).

The problem of *assessing the quality of code comments* has gained a lot of attention from researchers during the last decade (Khamis et al., 2010; Steidl et al., 2013; Ratol and Robillard, 2017; Pascarella and Bacchelli, 2017; Wen et al., 2019). Despite the research community's interest in this topic, there is

☆ Editor: Shane McIntosh.

* Corresponding author.
E-mail addresses: pooja.rani@unibe.ch (P. Rani), arianna.blasi@usi.ch (A. Blasi), nataliia.stulova@unibe.ch (N. Stulova), panc@zhaw.ch (S. Panichella), alessandra.gorla@imdea.org (A. Gorla), oscar.nierstrasz@unibe.ch (O. Nierstrasz).

https://doi.org/10.1016/j.jss.2022.111515

no clear agreement on what quality means when referring to code comments. Having a general definition of quality when referring to code comments is hard, as comments are diverse in purpose and scope.

Problem Statement. Maintaining high-quality code comments is vital for software evolution activities, however, *assessing the overall quality of comments is not a trivial problem*. As developers use various programming languages, adopt project-specific conventions to write comments, embed different kinds of information in a semi-structured or unstructured form (Padioleau et al., 2009; Pascarella and Bacchelli, 2017), and lack quality assessment tools for comments, ensuring comment quality in practice is a complex task. Even though specific comments follow all language-specific guidelines in terms of syntax, it is still challenging to determine automatically whether they satisfy other quality aspects, such as whether they are consistent or complete with respect to the code or not (Zhou et al., 2017). There are various such aspects, *e.g.,* readability, content relevance, and correctness that should be considered when assessing comments, but tools do not support all of them. Therefore, a comprehensive study of the specific attributes that influence code comment quality and techniques proposed to assess them is essential for further improving comment quality tools.

Previous mapping and literature review studies have collected numerous quality attributes (QAs) that are used to assess the quality of software documentation based on their importance and effect on the documentation quality. Ding et al. (2014) focused specifically on software architecture and requirement documents, while Zhi et al. (2015) analyzed code comments along with other types of documentation, such as requirement and design documents. They identified 16 QAs that influence the quality of software documentation. However, the identified QAs are extracted from a body of literature concerning relatively old studies (*i.e.,* studies conducted prior to the year 2011) and are limited in the context of code comments. For instance, only 10% of the studies considered by Zhi et al. concern code comments. Given the increasing attention that researchers pay to comment quality assessment, it is essential to know which QAs, tools and techniques they propose to assess code comment quality.

To achieve this objective, we perform an SLR on studies published in the last decade, *i.e.,* 2011-2020. We review 2353 studies and find 47 to be relevant to assessing comment quality. From these we extract the programming language, the types of analyzed comments, QAs for comments, techniques to measure them, and the preferred evaluation type to validate their results.

We observe that (i) most of the studies and techniques focus on comments in Java code, (ii) many techniques that are used to assess QAs are based on heuristics and thus may not be generalizable to other languages, (iii) a total of 21 QAs are used across studies, with a clear dominance of *consistency*, *completeness*, *accuracy*, and *readability*, and (iv) several QAs are often assessed manually rather than with the automated approaches. We find that the studies are typically evaluated by measuring performance metrics and surveying students rather than by performing validations with practitioners. This shows that there is much room for improvement in the state of the art of comment quality assessment.

The **contributions** of this paper are:

(i) an SLR of a total of 2353 papers, of which we review the 47 most relevant ones, focusing on QAs mentioned and research solutions proposed to assess code comment quality,
(ii) a catalog of 21 QAs of which four QAs are often investigated, while the majority is rarely considered in the studies, and of which 10 are new with respect to the previous study by Zhi et al. (2015),
(iii) a catalog of methods used to measure these 21 QAs in research studies,
(iv) an overview of the approaches and tools proposed to assess comment quality, taking into account the types of comments and the programming languages they consider,
(v) a discussion of the challenges and limitations of approaches and tools proposed to assess different and complementary comment QAs, and
(vi) a publicly available dataset including all validated data, and steps to reproduce the study in the replication package.[1]

Paper structure. The rest of the paper is organized as follows. In Section 2 we highlight our motivation and rationale behind each research question, and we present our methodology, including the different steps performed to answer our research questions. In Section 3 we report the study results. We discuss the results in Section 4 and their implications and future direction in Section 5. We highlight the possible threats to validity for our study in Section 6. Then Section 7 summarizes the related work, in relation to the formulated research questions. Finally, Section 8 concludes our study, outlining future directions.

2. Study design

The main objective of our study is to present an overview of the state of the art in assessing the quality of code comments. Specifically, we aim to highlight the QAs mentioned in the literature, and the techniques used so far to assess comment quality. To this end, we carry out an SLR, following the widely accepted guidelines of Kitchenham and Charters (2007) and Keele (2007). The first step in this direction is to specify the research questions related to the topic of interest (Kitchenham and Charters, 2007). The following steps focus on finding a set of relevant studies that are related to the research questions based on an unbiased search strategy.

2.1. Research questions

Our *goal* is to foster research that aims at building code comment assessment tools. To achieve this goal, we conduct an SLR, investigating the literature of the last decade to identify comment related QAs and solutions that address related challenges. We formulate the following research questions:

- **RQ_1**: *What types of comments do researchers focus on when assessing comment quality?*
  Motivation: Comments are typically placed at the beginning of a file, usually to report licensing or author information, or placed preceding a class or function to document the overview of a class or function and its implementation details. Depending on the specific type of comment used in source code and the specific programming language, researchers may use different techniques to assess them. These techniques may not be generalizable to other languages. For example, studies analyzing class comments in object-oriented programming languages may need extra effort to generalize the comment assessment approach to functional programming languages. We, therefore, investigate the comment types researchers target.
- **RQ_2**: *What QAs do researchers consider in assessing comment quality?*
  Motivation: QAs may solely concern syntactic aspects of the comments (*e.g.,* syntax of comments), writing style (*e.g.,* grammar), or content aspects (*e.g.,* consistency with the code). Researchers may use different terminology for the

[1] https://doi.org/10.5281/zenodo.4729054

same QA and thus these terms must be mapped across studies to obtain a unifying view of them, for instance, if the *accuracy* QA is defined consistently across studies or another terminology is used for it. We collect all the possible QAs that researchers refer to and map them, if necessary, following the methodology of Zhi et al. Future studies that aim to improve specific aspects of comment quality evaluation can use this information to design their tools and techniques.

- **RQ$_3$**: *Which tools and techniques do researchers use to assess comment QAs?*
  Motivation: Researchers may assess QAs manually, or may use sophisticated tools and techniques based on simple heuristics or complex machine learning (ML) to assess them automatically. We aim to identify if there are clear winning techniques for this domain and collect various metrics and tools used for this purpose.
- **RQ$_4$**: *What kinds of contribution do studies often make?*
  Motivation: Engineering researchers usually motivate their research based on the utility of their results. Auyang clarifies that engineering aims to apply scientific methods to real world problems (Auyang, 2006). However, software engineering currently lacks validation (Zelkowitz and Wallace, 1997). With this question, we want to understand what types of solution researchers contribute to improving automatic comment quality assessment, such as metrics, methods, or tools. This RQ can provide insight into specific kinds of solutions for future work.
- **RQ$_5$**: *How do researchers evaluate their comment quality assessment studies?*
  Motivation: Researchers may evaluate their comment assessment approaches, *e.g.,* by surveying developers, or by using a dataset of case studies. However, how often they involve professional developers and industries in such studies is unknown.

2.2. Search strategy

After formulating the research questions, the next steps focus on finding relevant studies that are related to the research questions. In these steps, we

1. construct search keywords in Section 2.2.1,
2. choose the search timeline in Section 2.2.2,
3. collect sources of information in Section 2.2.3,
4. retrieve studies in Section 2.2.4,
5. select studies based on the inclusion/exclusion criteria in Section 2.2.5, and
6. evaluate the relevant studies to answer the research questions in Section 2.2.6.

2.2.1. Search keywords

Kitchenham et al. recommended formulating individual facets or search units based on the research questions (Kitchenham and Charters, 2007). These search units include abbreviations, synonyms and other spellings, and they are combined using boolean operators. Pettricrew et al. suggested PIO (population, interventions, and outcome) criterion to define such search units (Petticrew and Roberts, 2008).

The *populations* include terms related to the standards. We first examine the definitions of *documentation* and *comment* in *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Standard 610.12-1990) to collect the main keywords. According to the definition, we identify the keywords *comment*, *documentation*, and *specification* and add them to the set K_1. We further add frequently mentioned comment-related keywords, such as *API*, *annotation*, and *summar* to the set K_1.

Table 1
keywords selected according to PIO criterion.

Criteria	keywords
Populations (K_1)	*comment*, *documentation*, *specification*, *API*, *annotation*, and *summar*
Interventions (K_2)	*quality*, *assess*, *metric*, *measure*, *score*, *analy*, *practice*, *structur*, *study*, and *studied*

The *interventions* include terms that are related to software methodology, tools, technology, or procedures. With respect to quality assessment, we define the intervention keywords to be *quality*, *assess*, *metric*, *measure*, *score*, *analy*, *practice*, *structur*, *study*, or *studied* and add them to the set K_2.

Note that we add common variations of the words manually, for example, we add "summar" keyword to the set to cover both "summary" and "summarization". We do not use any NLP libraries to stem words due to two main reasons, (i) to reduce the noisy matches, and (ii) the words from the title and abstract of the papers are not preprocessed (stemmed or lemmatized), therefore stemming the keywords might not find the exact or prefix matches. For example, using the porter stemming approach, the word "study" will be stemmed to "studi" and we might miss the papers with "study" word. To avoid such cases, we add common variations of this word *study* and *studied* to our search keywords.

The *outcomes* include terms that are related to factors of significance to developers (*e.g.,* reduced cost, reduced time to assess quality). Since it is not a required unit to restrict the search scope, and our focus is on all kinds of quality assessment approaches, we exclude the outcomes in our search keywords. However, to narrow down our search and exclude irrelevant papers, such those about code reviews or testing, or non-technical papers, we formulate another set of keywords, K_3. In this set, we include *code review*, *test*, *keynote*, *invited*, and *poster*, to exclude entries of non-technical papers that were not filtered out using the heuristics on the number of pages.

Hence, using the final set of keywords (also given in Table 1), we select a paper if its title and abstract match the keywords from K_1 and K_2 but not from K_3 where the prefix function is used to match the keywords in the paper.

2.2.2. Timeline

We focus our SLR on the last decade (*i.e.,* January 2011-December 2020) since Zhi et al. investigated the works on software documentation quality — including code comments — from 1971 to 2011 Zhi et al. (2015). Our results can thus be used to observe the evolution of comment quality assessment, but, more importantly, they naturally complement the existing body of knowledge on the topic.

We then proceed to the main steps *i.e.,* retrieving the paper data, selecting venues, and identifying the relevant papers for our comment context.

2.2.3. Data collection

Concretely, our data collection approach comprises three main steps, *i.e.,* literature data collection, data selection, and data evaluation, which we sketch in Fig. 1 and present in further detail as follows:

We now describe how we automatically collect the data from the literature, explaining the rationale behind our selection of venues and our automatic keyword-based filtering to identify the likely relevant papers regarding comment quality assessment. We justify the need for another step of data gathering based on the snowball approach in Section 2.2.5. Finally, we present our criteria for the careful evaluation of the relevant papers in Section 2.2.6.

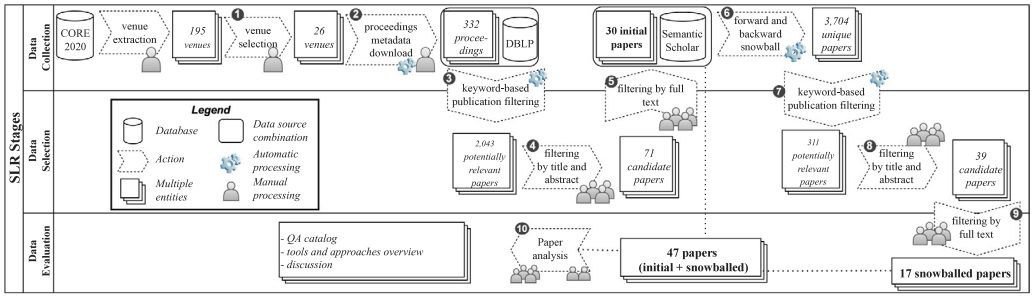


Fig. 1. SLR stages to collect relevant papers.

Venue Selection. Code comment analysis, generation, usage, and maintenance are of primary interest to the SE research community. Thus, in order to systematically review the literature on the comment quality assessment, we start by focusing on the SE venues. We use the latest 2020 updated version of the conference and journal database of the CORE ranking portal as a primary data source to identify all the potentially relevant SE venues.[2] The portal provides assessments of major conferences and journals in the computing disciplines, and it is a well-established and regularly-validated registry maintained by the academic community. We extract all ranked journals in SE (search code 803) from the CORE portal[3] and all top conferences and workshops in the SE field (search code 4612).[4] This process gives us an initial list of 85 journal and 110 conference venues. We select in step ❶ 26 software engineering (SE) conferences and journals from 195 candidate venues based on the likelihood of finding relevant papers in their proceedings.

We focus on A* and A conferences and journals, and add conferences of rank B or C if they are co-located with previously selected A* and A conferences to have venues, such as the *IEEE/ACM International Conference on Program Comprehension* (ICPC) or the *IEEE International Workshop on Source Code Analysis and Manipulation* (SCAM) that focus on source code comprehension and manipulation.

We prune venues that may not contain relevant contributions to source code comments. Specifically, we exclude a venue if its ten years of proceedings contain fewer than five occurrences of the words *documentation* or *comment*. This way, we exclude conferences, such as *IEEE International Conference on Engineering of Complex Computer Systems* (ICECCS), *Foundations of Software Science and Computational Structures* (FoSSaCS), and many others that primarily focus on other topics, such as verification or programming languages. Thus, we reduce our dataset to 20 conferences and six journals, as shown in Table 2.

In Table 2, the column *Type* specifies whether a venue is a conference (C) or a journal (J), and the column *Rank* denotes the corresponding CORE rank of the venue as of April 2021. The column *Selection* indicates the data collection phase in which the venue was first selected. The column *Papers per venue* indicates the total number of papers selected from this venue, both during the direct search and the snowball search.

We consider only full papers (published in a technical track and longer than five pages) since they are likely to be an extended or mature version of the papers published in other tracks, such as NIER, ERA, or Poster.

2.2.4. Data retrieval

We retrieve in step ❷ the proceedings from January 2011 to December 2020 of the selected venues from the DBLP digital library. From each paper, we collect its metadata using the GitHub repository,[5] such as the title, authors, conference track (if present), its page length, and its Digital Object Identifier (DOI), directly from DBLP for a total of 17554 publications. For each paper, the DOI is resolved and its abstract is collected from the publisher webpage.

Keyword-based filtering. We apply in step ❸ a keyword-based search (given in Section 2.2.1 using a prefix function) to the retrieved proceedings to select potentially relevant papers. We account for possible upper- and lowercase letters in the keywords, and sometimes use variations of keywords (*e.g.,* singular and plural forms).

Our filtering will get papers (whose title and abstract include keywords from K_1 and K_2 but not from K_3) that explicitly mention concepts we are interested in, *e.g., "A Human Study of Comprehension and Code Summarization"* from ICPC 2020 (Stapleton et al., 2020) is matched by keywords *summar* from K_1 in the title and *quality* from K_2 in the abstract, but will exclude papers not sufficiently close to our research subject, *e.g., "aComment: mining annotations from comments and code to detect interrupt related concurrency bugs"* from ICSE 2011 has two keywords *comment* and *annotation* from K_1 but none from the K_2.

The final set of keywords we use for filtering is the result of an iterative approach: we manually scan the full venue proceedings metadata to make sure the set of keywords did not prune relevant papers, and we refine the set of keywords during several iterative discussions. This iterative approach gives us confidence that our keyword-based filtering approach does not lead to false negatives for the selected venues. After applying the keyword-based filtering, we identify 2043 studies as potentially-relevant papers from a total of 17554, which we review manually.

2.2.5. Data selection

We analyze ❹ the 2043 selected papers following the protocol where four authors or evaluators manually evaluate the papers based on the inclusion and exclusion criterion to ensure that they indeed assess comment quality.

Inclusion criteria

I1 The topic of the paper is about code comment quality.
I2 The study presents a model/technique/approach to assess code comments or software documentation including code comments.

Exclusion criteria

E1 The paper is not in English.

[2] https://www.core.edu.au/conference-portal

[3] http://portal.core.edu.au/jnl-ranks/?search=803&by=for&source=CORE2020&sort=arank&page=1 accessed on 25 Mar, 2021

[4] http://portal.core.edu.au/conf-ranks/?search=4612&by=for&source=CORE2020&sort=arank&page=1 accessed on 25 Mar, 2021

[5] https://github.com/sbaltes/dblp-retriever

Table 2
Included Journals, Conferences, and Workshops.

Venue	Abbreviation	Rank	Type	Selection	Papers per venue
ACM Computing Surveys	CSUR	A*	J	Search	
ACM Transactions on Software Engineering and Methodology	TOSEM	A*	J	Search	
IEEE Transactions on Software Engineering	TSE	A*	J	Search	5
Empirical Software Engineering: an international journal	EMSE	A	J	Search	6
Journal of Systems and Software	JSS	A	J	Search	2
Information and Software Technology	IST	A	J	Search	1
ACM SIGSOFT Symposium on the Foundations of Software Engineering	ESEC/FSE	A*	C	Search	3
International Conference on Software Engineering	ICSE	A*	C	Search	6
Architectural Support for Programming Languages and Operating Systems	ASPLOS	A*	C	Search	
Computer Aided Verification	CAV	A*	C	Search	
International Conference on Functional Programming	ICFP	A*	C	Search	
ACM Conference on Object Oriented Programming Systems Languages and Applications	OOPSLA	A*	C	Search	1
ACM-SIGPLAN Conference on Programming Language Design and Implementation	PLDI	A*	C	Search	
ACM-SIGACT Symposium on Principles of Programming Languages	POPL	A*	C	Search	
Measurement and Modeling of Computer Systems	SIGMETRICS	A*	C	Search	
Automated Software Engineering Conference	ASE	A	C	Search	2
International Conference on Evaluation and Assessment in Software Engineering	EASE	A	C	Search	1
International Symposium on Empirical Software Engineering and Measurement	ESEM	A	C	Search	1
IEEE International Conference on Software Maintenance and Evolution	ICSME	A	C	Search	2
IEEE International Working Conference on Mining Software Repositories	MSR	A	C	Search	1
International Symposium on Software Reliability Engineering	ISSRE	A	C	Search	
IEEE International Working Conference on Software Visualisation	VISSOFT	B	C	Search	
IEEE International Conference on Global Software Engineering	ICGSE	C	C	Search	
IEEE International Conference on Program Comprehension	ICPC	C	C	Search	5
International Workshop on Modelling in Software Engineering	MISE	C	C	Search	
IEEE International Workshop on Source Code Analysis and Manipulation	SCAM	C	C	Search	1
International Conference on Web Information Systems and Applications	WISA	-	C	Snowball	1
Software Quality Journal	-	C	J	Snowball	1
Software: Practice and Experience	SPE	B	J	Snowball	1
ACM Symposium on Applied Computing	SAC	B	C	Snowball	1
IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation	MaLTeSQuE	-	C	Snowball	1
Journal of Software: Evolution and Process	JSEP	B	J	Snowball	1
Asia-Pacific Symposium on Internetware	Internetware	-	C	Snowball	1
IEEE International Joint Conference on Neural Networks	IJCNN	A	C	Snowball	1
International Computer Software and Applications Conference	COMPSAC	B	C	Snowball	1
Asia-Pacific Software Engineering Conference	APSEC	B	C	Snowball	1
International journal of software engineering and knowledge engineering	SEKE	-	J	Snowball	1

E2 It does not assess any form of quality aspects of comments *e.g.,* content, style, or language used.
E3 It is not published in a technical track.
E4 It is a survey paper.
E5 It is not a peer reviewed paper, or it is a pre-print.
E6 It covers other documentation artifacts, *i.e.,* not comments.
E7 It is shorter than 5 pages.

Manual analysis. The selected papers were equally divided among four evaluators (*i.e.,* two Ph.D. candidates and two faculty members) based on years of publications so that each evaluator gets papers from all venues, *e.g.,* the first author evaluate proceedings from 2011 to 2013. We make sure that evaluators do not take decisions on papers they co-authored to avoid conflicts of interest. Each evaluator has at least two years of experience in the domain of comment analysis. Each paper is reviewed by three evaluators. The evaluators follow a three-iteration-based process to evaluate the assigned papers. In the first iteration, the first evaluator independently assesses the relevance of a paper based on the criteria by inspecting each paper's title and abstract, to make an initial guess, then inspecting its conclusion to reach the final decision. In the next iteration, another evaluator reviews the paper and validates the previous decision by adding the label "agrees/disagrees with the first evaluator". With this process, every publication selected in the final set is reviewed by at least two researchers. In case they do not agree, the third evaluator reviews it (Kuhrmann et al., 2017), and the final decision is taken based on the majority voting mechanism.

We decide, for instance, to include the study by Hata et al. (2019), even though it only talks about links in comments. Though it does not explicitly describe any quality aspect of comments, it mentions the traceability of the links, which is a QA we consider in our study. All studies considered in our SLR together with their evaluation (the agreement and disagreement for each study) are available in our replication package.

Thus, we reduce 2043 papers to 71 candidate papers (*i.e.,*3%) with a fair agreement according to Cohen's Kappa (k=0.36). For all candidate papers, we read in step ❺ their introduction, conclusion, and the study design (if needed), and discuss them amongst ourselves to ensure their relevance. During this analysis process, some additional papers were found to be irrelevant. For example, the study by Aghajani et al. seems relevant based on the title and abstract, but does not really evaluate code comments, and we thus discarded it (Aghajani et al., 2020). With this process, 41 papers in total were discarded, reducing the relevant paper set to 30 papers.

Data gathering for snowballing. To include further relevant papers that we might have missed with the venue-based approach, we perform in step ❻ a forward and backward snowballing approach for the 30 papers and retrieve a total of 3704 unique papers.

Snowball papers	Total	Unique	*Selected*
from citations	2610	1624	741
from references	3369	2080	2021

The column *Total* reports the total number of references and citations collected. The *Unique* column reports a total number of unique items (*i.e.,* since relevant papers cover similar topics many references, and citations are shared across our set of studies). Finally, the column *Selected* reports the total number of unique references and citations whose publication year falls within our time frame range, *i.e.,* 2011–2020.

Table 3
Included studies.

Study ID	Title	Year	Reference
S01	How Good is Your Comment? A Study of Comments in Java Programs.	2011	Haouari et al. (2011)
S02	Quality Analysis of Source Code comments.	2013	Steidl et al. (2013)
S03	Evaluating Usage and Quality of Technical Software Documentation: An Empirical Study.	2013	Garousi et al. (2013)
S04	Inferring Method Specifications from Natural Language API Descriptions.	2012	Pandita et al. (2012)
S05	Using Traceability Links to Recommend Adaptive Changes for Documentation Evolution.	2014	Dagenais and Robillard (2014)
S06	On Using Machine Learning to Identify Knowledge in API Reference Documentation.	2019	Fucci et al. (2019)
S07	Detecting Fragile Comments.	2017	Ratol and Robillard (2017)
S08	Automatically Assessing Code Understandability: How Far are We?	2017	Scalabrino et al. (2017)
S09	Analyzing APIs Documentation and Code to Detect Directive Defects.	2017	Zhou et al. (2017)
S10	The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load.	2018	Fakhoury et al. (2018)
S11	A Large-Scale Empirical Study on Linguistic Antipatterns Affecting APIs.	2018	Aghajani et al. (2018)
S12	Improving API Caveats Accessibility by Mining API Caveats Knowledge Graph.	2018	Li et al. (2018)
S13	A Learning-Based Approach for Automatic Construction of Domain Glossary from Source Code and Documentation.	2019	Wang et al. (2019)
S14	A Framework for Writing Trigger-Action Todo Comments in Executable Format.	2019	Nie et al. (2019)
S15	A Large-Scale Empirical Study on Code-Comment Inconsistencies.	2019	Wen et al. (2019)
S16	Software Documentation Issues Unveiled.	2019	Aghajani et al. (2019)
S17	The Secret Life of Commented-Out Source Code.	2020	Pham and Yang (2020)
S18	Code Comment Quality Analysis and Improvement Recommendation: An Automated Approach	2016	Sun et al. (2016)
S19	A Human Study of Comprehension and Code Summarization.	2020	Stapleton et al. (2020)
S20	CPC: Automatically Classifying and Propagating Natural Language Comments via Program Analysis.	2020	Zhai et al. (2020)
S21	Recommending Insightful Comments for Source Code using Crowdsourced Knowledge.	2015	Rahman et al. (2015)
S22	Improving Code Readability Models with Textual Features.	2016	Scalabrino et al. (2016)
S23	Automatic Source Code Summarization of Context for Java Methods.	2016	McBurney and McMillan (2016a)
S24	Automatic Detection and Repair Recommendation of Directive Defects in Java API Documentation.	2020	Zhou et al. (2020)
S25	Measuring Program Comprehension: A Large-Scale Field Study with Professionals.	2018	Xia et al. (2018)
S26	Usage and Usefulness of Technical Software Documentation: An Industrial Case Study	2015	Garousi et al. (2015)
S27	What Should Developers be Aware of? An Empirical Study on the Directives of API Documentation.	2012	Monperrus et al. (2012)
S28	Analysis of License Inconsistency in Large Collections of Open Source Projects.	2017	Wu et al. (2017)
S29	Classifying Code Comments in Java Software Systems.	2019	Pascarella et al. (2019)
S30	Augmenting Java Method Comments Generation with Context Information based on Neural Networks.	2019	Zhou et al. (2019)
S31	Improving Source Code Lexicon via Traceability and Information Retrieval.	2011	Lucia et al. (2011)
S32	Detecting API Documentation Errors.	2013	Zhong and Su (2013)
S33	Analyzing Code Comments to Boost Program Comprehension	2018	Shinyama et al. (2018)
S34	Recommending Reference API Documentation.	2015	Robillard and Chhetri (2015)
S35	Some Structural Measures of API Usability	2015	Rama and Kak (2015)
S36	An Empirical Study of the Textual Similarity between Source Code and Source Code Summaries.	2016	McBurney and McMillan (2016)
S37	Linguistic Antipatterns: What They are and How Developers Perceive Them.	2016	Arnaoudova et al. (2016)
S38	Coherence of Comments and Method Implementations: A Dataset and An Empirical Investigation	2016	Corazza et al. (2018)
S39	A Comprehensive Model for Code Readability	2018	Scalabrino et al. (2018)
S40	Automatic Detection of Outdated Comments During Code Changes	2018	Liu et al. (2018)
S41	Classifying Python Code Comments Based on Supervised Learning	2018	Zhang et al. (2018)
S42	Investigating Type Declaration Mismatches in Python	2018	Pascarella et al. (2018)
S43	The Exception Handling Riddle: An Empirical Study on the Android API.	2018	Kechagia et al. (2018)
S45	Migrating Deprecated API to Documented Replacement: Patterns and Tool	2019	Xi et al. (2019)
S46	A Topic Modeling Approach To Evaluate The Comments Consistency To Source Code	2020	Iammarino et al. (2020)
S47	Comparing Identifiers and Comments in Engineered and Non-Engineered Code: A Large-Scale Empirical Study	2020	Lemos et al. (2020)

Data selection from snowballing. We repeat in step ❼ the same keyword-based filtering to these 3704 papers, as described in Section 2.2.3. As a result, 311 papers were added for manual analysis. We repeat in step ❽ the three-iteration based manual analysis process and find 39 additional candidate papers to analyze. After the second round of discussion ❾ we keep 17 additional relevant papers. We find a total of 47 papers shown in Table 3 published in the venues shown in Table 2. In Table 3, the column *Study ID* indicates the ID assigned to each paper, the column *Title* presents the title of the paper, and the column *Year* indicates the years in which the paper is published.

To further ensure the relevance of our search strategy, we search our keywords on popular publication databases, such as ACM, IEEE Xplore, Wiley *etc.* We search for our keywords in titles and abstracts.[6] We retrieve 13 144 results from IEEE Xplore, and 10 567 from ACM for the same timeline (2011-2020). We inspect first 200 results (sorted by relevance criterion on the publisher webpage) from each of these databases. We apply our inclusion and exclusion criterion to find the extent to which our venue selection criteria might have missed relevant papers. Our results from ACM show that 19% of the these papers are already covered by our search strategy but only 5% of them fulfilled our inclusion criterion. Nearly 81% of the papers are excluded due to their non-SE venue. Among these papers, 80% are unrelated to the code comment quality aspect while 1% of papers (two papers) that are related to code comments are missed due to two main reasons, (i) the venue not being indexed in CORE2020, and (ii) the paper being from a non-technical track.

Similarly, the results from IEEE show that 30% of the papers are already covered by our search strategy but only 5% of them fulfilled the inclusion criterion. Nearly 69% of the papers are excluded due to their non-SE venue and unrelated to code comment quality aspect. We also find 1% papers that are relevant to our topic of interest but excluded due to the length criteria, specifically one of the paper is a poster paper and another is a short paper.

2.2.6. Data evaluation

We work in step ❿ on the full versions of the 47 relevant papers to identify the QAs and the approaches to assess comments. In case we cannot retrieve the full PDF version of a paper, we use university resources to access it. This affects only one paper by Sun et al. which requires payment to access the full version (Sun et al., 2016). In case we cannot access a paper via any resource, we remove it from our list. We find no such inaccessible study.

We report all papers in an online shared spreadsheet on Google Drive to facilitate their analysis collaboratively. For each paper we extract common metadata, namely *Publication year*, *Venue*, *Title*, *Authors*, *Authors' country*, and *Authors' affiliation*. We then extract various dimensions (described in the following paragraphs) formulated to answer all research questions.

2.3. Data extraction for research questions

To answer *RQ1* (*What types of comments do researchers focus on when assessing comment quality?*), we record the *Comment scope* dimension. It lists the scope of comments under assessment such as class, API, method (function), package, license, or inline comments. In case the comment type is not mentioned, we classify it as "code comments". Additionally, we identify the programming languages whose comments are analyzed, and record this in the *Language analyzed* dimension.

To answer *RQ2* (*What QAs do researchers consider in assessing comment quality?*), we identify various QAs researchers mention to assess comment quality. This reflects the various quality aspects researchers perceive as important to have high-quality comments. Table 4 lists the QAs in the *Quality attribute (QA)* column and their brief summary in the *Description* column. Of these QAs, several are mentioned by Zhi et al. in their work (Zhi et al., 2015), and are highlighted by the bold text compared to QAs mentioned in other works. As Zhi et al. considered various types of documentation, such as requirement and architectural documents, not all attributes fit exactly into our study. For instance, the category "Format" includes the format of the documentation (*e.g.,* UML, flow chart) in addition to the other aspects such as writing style of the document, use of diagrams *etc.* Although the format of the documentation is not applicable in our case due to our comment-specific interest, we keep other applicable aspects (writing style, use of diagram) of this QA. In addition to their QAs, we include any additional attribute mentioned in our set of relevant papers. If a study uses different terminology but similar meaning to QAs in our list, we map such QAs to our list and update the list of possible synonyms as shown in the column *Synonyms* in Table 4. In case we cannot map a study to the existing QAs, we map it to the *Other* category.

For the cases where the studies do not mention any specific QA and mention comment quality analysis in general, we map the study to the list of existing QAs or classify it as *Other* based on their goal behind the quality analysis. For example, Pascarella et al. identify various information types in comments to support developers in easily finding relevant information for code comprehension tasks and to improve the comment quality assessment (Pascarella and Bacchelli, 2017). They do not mention any specific QA, but based on their study goal of finding relevant information easily, we map their study to the *content relevance* QA. Similarly, we map other comment classification studies such as Fucci et al. (2019), Pascarella et al. (2019), Shinyama et al. (2018), and Zhang et al. (2018) to the *content relevance* attribute. At the same time, the studies on linguistic anti-patterns (LAs) are mapped to the *consistency* attribute, given that LAs are practices that lead to lexical inconsistencies among code elements, or between code and associated comments (Arnaoudova et al., 2016; Fakhoury et al., 2018; Aghajani et al., 2018). Additionally, the studies that mention the negation of the QAs such as *inconsistency*, *incorrectness*, or *incompleteness* are mapped to their antonyms as *consistency*, *correctness*, or *completeness*, respectively to prevent duplication.

RQ3 (*Which tools and techniques do researchers use to assess comment QAs?*) concerns various methods researchers use or propose to assess comment QAs, for instance, whether they use machine-learning based methods to assess comment quality or not.

- *Technique type.* This identifies whether the technique used to assess a QA is based on natural language processing (NLP), heuristics, static analysis, metrics, machine-learning (ML), or deep neural network (DNN) approaches. The rationale is to identify which QAs are often assessed manually or using a specific automated approach. For instance, if the study uses specific heuristics related to the programming environment to assess a QA, it is classified as *heuristic-based* technique, if it uses abstract syntax tree (AST) based static analysis approaches, then it is assigned to *static analysis*, and if it uses machine-learning or deep-learning-based techniques (including any or both of the supervised or unsupervised learning algorithms), then it is classified as *ML-based*, or *DNN-based* respectively. A study can use mixed techniques to assess a specific QA and thus can be assigned to multiple techniques for the corresponding QA. We often find cases

[6] It is not possible to search the keywords in abstracts in Wiley.

Table 4
RQ2 QAs mentioned by Zhi et al. (highlighted in bold) and other works.

Quality Attribute (QA)	Synonyms	Description
QAs mentioned by Zhi et al.		
Accessibility	**Availability**, **information hiding**, **easiness to find**	Whether comment content can be accessed or retrieved by developers or not
Readability	**Clarity**	The extent to which comments can be easily read by other readers
Spelling and grammar	Natural language quality	Grammatical aspect of the comment content
Trustworthiness		The extent to which developers perceive the comment as trustworthy
Author-related		Identity of the author who wrote the comment
Correctness		Whether the information in the comment is correct or not
Completeness	Adequacy	How complete the comment content is to support development and maintenance tasks or whether there is missing information in comments or not
Similarity	**Uniqueness**, **duplication**	How similar the comment is to other code documents or code
Consistency	**Uniformity**, **integrity**	The extent to which the comment content is consistent with other documents or code
Traceability		The extent to which any modification in the comment can be traced, including who performed it
Up-to-datedness		How the comment is kept up-to-date with software evolution
Accuracy	**Preciseness**	Accuracy or preciseness of the comment content. If the documentation is too abstract or vague and does not present concrete examples, then it can seem imprecise.
Information organization		How the information inside a comment is organized in comments
Format	Including visual models, use of examples	Quality of documents in terms of writing style, description perspective, use of diagrams or examples, spatial arrangement, etc.
QAs mentioned by other works		
Coherence		How comment and code are related to each other, *e.g.,* method comment should be related to the method name(Steidl et al. (2013), Corazza et al. (2018))
Conciseness		The extent to which comments are not verbose and do not contain unnecessary information (McBurney and McMillan (2016a), Zhou et al. (2019), Lemos et al. (2020))
Content relevance		How relevant the comment or part of the comment content is to a particular purpose (documentation, communication) (Haouari et al. (2011), Garousi et al. (2013), Pascarella et al. (2019), Zhang et al. (2018), Lemos et al. (2020))
Maintainability		The extent to which comments are maintainable (Wen et al. (2019)–Pham and Yang (2020), Zhai et al. (2020)–Rahman et al. (2015))
Understandability		The extent to which comments contribute to understanding the system (Stapleton et al. (2020), McBurney and McMillan (2016a))
Usability	Usefulness	To which extent the comment can be used by readers to achieve their objectives (Steidl et al. (2013), Aghajani et al. (2019), Robillard and Chhetri (2015), Rama and Kak (2015))
Documentation technology		Whether the technology to write, generate, store documentation is current or not
Internationalization		The extent to which comments are correctly translated in other languages (Aghajani et al. (2019))
Other		The study does not mention any QA and cannot be mapped to any of the above attributes

where the studies do not use any automated technique to measure a QA and instead ask other developers to assess it manually, so we put such cases into the *manual assessment* category. In case the study mentions a different technique, we extend the dimension values.

- *Metrics or tools.* This further elaborates specific metrics, or tools the studies propose or use to assess a QA. A study can use an existing metric or can propose a new one. Similarly, one metric can be used to assess multiple QAs. We identify such metrics to highlight popular metrics amongst researchers.

RQ4 (*What kinds of contribution do studies often make?*) captures the nature of the study and the type of contribution researchers use or propose to assess comment quality. We first identify the nature of research of a study and then identify the type of contribution it provides. This can reflect the kind of research often conducted to assess comment quality and the kind of contribution they make to support developers in assessing comment quality, for instance, what kind of solutions the *Solution Proposal* research often propose, such as a method, metric, model, or tool.

To capture this information, we formulate the following dimensions:

- *Research type.* This identifies the nature of the research approach used in the studies, such as empirical, validation, evaluation, solution proposal, philosophical, opinion, or experience paper (Wieringa et al., 2006; Petersen et al., 2008). The dimension values are described in detail in Table 5.
- *Paper contribution.* This dimension describes the type of contribution the study provides in terms of a method/technique, tool, process, model, metric, survey, or empirical results (Petersen et al., 2008). The dimension values are described in

Table 5
Type of research approach studies use and type of contributions studies make.

Dimension	Category	Description
Research type	Empirical	This research task focuses on understanding and highlighting various problems by analyzing relevant projects, or surveying developers. These papers often provide empirical insights rather than a concrete technique.
	Validation	This research task focus on investigating the properties of a technique that is novel and is not yet implemented in practice, *e.g.,* techniques used for mathematical analysis or lab experimentation
	Evaluation	The paper investigates the techniques that are implemented in practice and their evaluation is conducted to show the results of the implementation in terms of its pros and cons and thus help researchers in improving the technique.
	Solution Proposal	The paper proposes a novel or a significant extension of an existing technique for a problem and describes its applicability, intended use, components, and how the components fit together using a small example or argumentation.
	Philosophical	These papers present a new view to look at the existing problems by proposing a taxonomy or a conceptual framework, *e.g.,* developing a new language or framework to describe the observations is a philosophical activity.
	Opinion	These papers describe the author's opinion in terms of how things should be done, or if a certain technique is good or bad. They do not rely on research methodologies and related work.
	Experience	These papers explain the personal experience of a practitioner in using a certain technique to show how something has been done in practice. They do not propose a new technique and are not scientific experiments.
Contribution type	Empirical	The paper provides empirical results based on analyzing relevant projects to understand and highlights the problems related to comment quality.
	Method/technique	The paper provides a novel or significant extension of an existing approach.
	Model	Provides a taxonomy to describe their observations or an automated model based on machine/deep learning.
	Metric	Provides a new metric to assess specific aspects of comments.
	Survey	Conducts survey to understand a specific problem and contribute insights from developers.
	Tool	Develops a tool to analyze comments.

detail in Table 5. If we cannot categorize it into any of these, we mark it "Other".

- *Tool availability.* This reflects whether the tool proposed in the study is accessible or not at the time of conducting our study. González et al. identified the reproducibility aspects characterizing empirical software engineering studies (González-Barahona and Robles, 2012) in which availability of the artifact (the tool proposed in the study, or the dataset used to conduct the study) is shown as an important aspect to facilitate the replication and extension of the study. Therefore, we record the availability of the proposed tool in this dimension and the availability of the dataset in the following dimension.
- *Dataset availability.* This reflects if the dataset used in the empirical study is accessible or not.

RQ5 (*How do researchers evaluate their comment quality assessment studies?*) concerns how various kinds of research (*Research type* dimension described in the previous RQ), and various kinds of contribution (*Paper contribution* dimension) are evaluated in the studies. For example, it helps us to observe that if a study proposes a new method/technique to assess comments, then the authors also conduct an experiment on open-source projects to validate the contribution, or they consult the project developers, or both. We capture the type of evaluation in the *Evaluation type* dimension, and its purpose in *Evaluation purpose*. The rationale behind capturing this information is to identify the shortcomings in their evaluations, *e.g.,* how often the studies proposing a tool are validated with practitioners.

- *Evaluation type.* It states the type of evaluation the studies conduct to validate their approaches, such as conducting an experiment on open-source projects (*Experiment*), or surveying students, practitioners, or both. For the automated approaches, we consider various performance metrics, also known as Information Retrieval (IR) metrics, that are used to assess the machine/deep learning-based models, such as Precision, Recall, F1 Measure, or Accuracy under the *performance metrics*. In case the approach is validated by the authors of the work, we identify the evaluation type as *Authors of the work*.
- *Evaluation purpose.* It states the motivation of evaluation by authors such as evaluate the functionality, efficiency, applicability, usability, accuracy, comment quality in general, or importance of attributes.

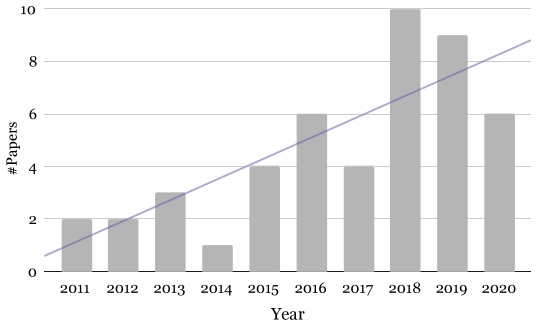
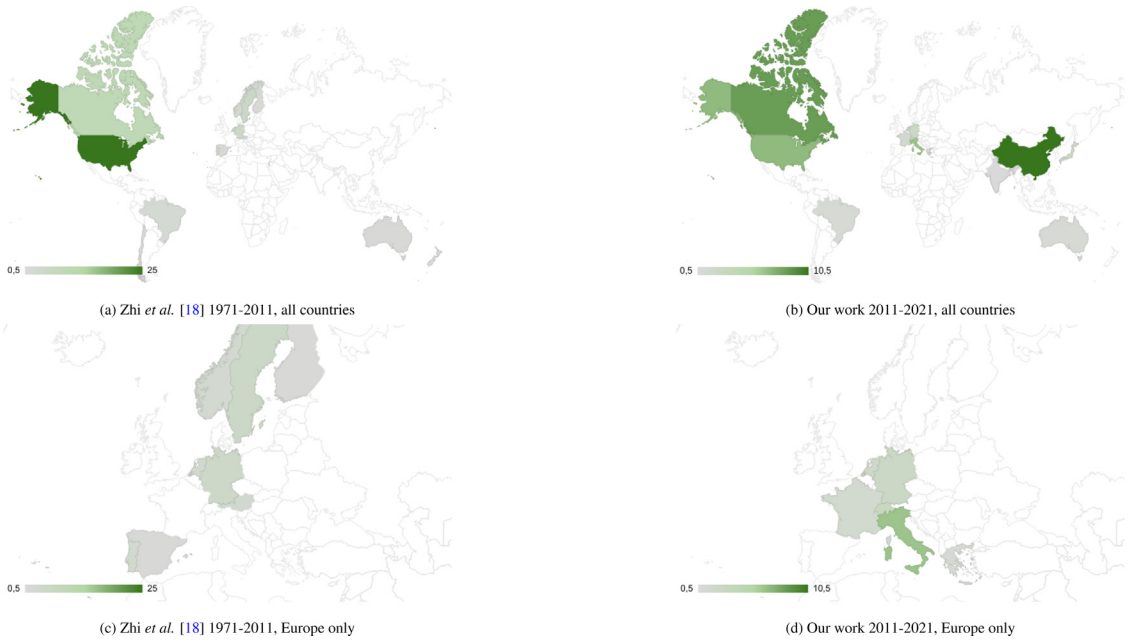


Fig. 2. Relevant papers by years.

3. Results

As mentioned in Section 2.2.5, we analyze 47 relevant papers in total. Before answering our four RQs, we present a brief overview of the metadata (publishing venues) of the papers.

Table 2 highlights the publication venues of these papers. Most studies were published in top-tier software engineering conferences (*e.g.,* ICSE) and journals, especially the ones with a focus on empirical studies (*e.g.,* EMSE). This means that the SE community agrees that assessing comment quality is an important topic deserving of research effort. Fig. 2 shows the paper distribution over the past decade, indicating a clear trend of increasing interest of the SE research community in comment quality assessment. Fig. 3 shows the author distribution of the selected papers by the institution. For the timeline 1971–2011,



(a) Zhi *et al.* [18] 1971-2011, all countries

(b) Our work 2011-2021, all countries

(c) Zhi *et al.* [18] 1971-2011, Europe only

(d) Our work 2011-2021, Europe only

Fig. 3. Relevant papers by countries.

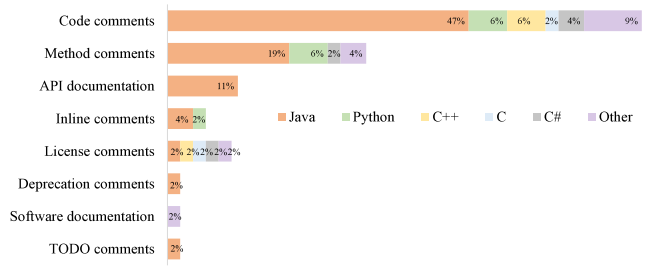


Fig. 4. Types of comments per programming language.

we rely on the geographical statistics data from the replication package of our reference study by Zhi et al. (2015), while for the period 2011–2021, and we collect these statistics as follows. For each paper, the primary affiliations of all authors are taken into account. If people from different countries co-authored a paper, we calculate the proportion of a country's contribution for each paper so that each paper gets a total score of one to avoid over-representing papers. For example, if five authors of a paper belong to Switzerland and one belongs to Spain, we assign 5/6 score for Switzerland and 1/6 for Spain for the paper. Comparison with the previous data allows us to see the evolution of the field, with more even distribution of researchers nowadays and (unsurprising) rise of contributions from southeast Asia, specifically from China.

Finding 1. The trend of analyzing comment quality has increased in the last decade (2011–2020), in part due to more researchers from southeast Asia working on the topic.

3.1. RQ_1: What types of comments do researchers focus on when assessing comment quality?

To describe the rationale behind code implementation, various programming languages use source code comments. Our results show that researchers focus more on some programming languages compared to others as shown in Fig. 4. This plot highlights the types of comments on the *y*-axis; each stack in the bar shows the ratio of the studies belonging to a particular language. For instance, the majority (87%) of the studies focus on code comments from Java, whereas only 15% of the studies focus on code comments from Python, and 10% of them focus on C# and C++. These results are in contrast to popular languages indicated by various developer boards, such as GitHub, Stack Overflow, or TIOBE. For instance, the TIOBE index show Python and C languages more popular than Java.[7] Similarly, the developer survey of 2019 and 2020 by Stack Overflow show that Java stands fifth after JavaScript, HTML/CSS, SQL, and Python among the most commonly used programming languages.[8] We find only one study (Hata et al. (2019)) that seems to address the comment quality aspect in JavaScript. Given the emerging trend of studies leveraging natural-language information in JavaScript code (Motwani and Brun, 2019; Malik et al., 2019), more research about comment quality may be needed in this environment. It indicates that researchers need to analyze comments of other languages to verify their proposed approaches and support developers of other languages.

Finding 2. 87% of the studies analyze comments from Java while other languages have not yet received enough attention from the research community.

As code comments play an important role in describing the rationale behind source code, various programming languages use different types of comments to describe code at various abstraction levels. For example, Java class comments should present high-level information about the class, while method comments should present implementation-level details (Nurvitadhi et al., 2003). We find that half of the studies (51% of the studies) focus on all types of comments whereas the other half focus on specific types of comments, such as inline, method, or TODO comments. However, we also see in Fig. 4 that studies frequently focus on method comments and API documentation. This proves

[7] https://www.tiobe.com/tiobe-index/ verified on Sep, 2021

[8] https://insights.stackoverflow.com/survey/2020

the effort the research community is putting into improving API quality. While some attention is given to often overlooked kinds of comments, such as license comments (Wu et al. (2017), Shinyama et al. (2018)), TODO comments (Nie et al. (2019)), inline comments (Pham and Yang (2020)), and deprecation comments (Xi et al. (2019)), no relevant paper seems to focus specifically on the quality of *class* or *package* comments. Recently Rani et al. studied the characteristics of class comments of Smalltalk in the Pharo environment[9] and highlighted the contexts they differ from Java and Python class comments, and why the existing approaches (based on Java, or Python) need heavy adaption for Smalltalk comments (Rani et al., 2021b,a). This may encourage more research in that direction, possibly for other programming languages.

Finding 3. Even though 50% of the studies analyze all types of code comments, the rest focus on studying a specific type of comments such as method comments, or API comments, indicating research interest in leveraging a particular type of comment for specific development tasks.

Previous work by Zhi et al. showed that a majority of studies analyze just one type of system (Zhi et al., 2015). In contrast, our findings suggest that the trend of analyzing comments of multiple languages and systems is increasing. For example, 80% of the studies analyzing comments from Python and all studies analyzing comments from C++ also analyze comments from Java. Only Pascarella et al. (Pascarella et al. (2018)) and Zhang et al. (Zhang et al. (2018)) focus solely on Python (Pascarella et al., 2018; Zhang et al., 2018). However, Zhang et al. (Zhang et al. (2018)) perform the comment analysis work in Python based on the Java study (Pascarella et al. (2019)) by Pascarella et al. (Zhang et al., 2018; Pascarella and Bacchelli, 2017). Such trends also reflect the increasing use of polyglot environments in software development (Tomassetti and Torchiano, 2014). The "Other" label in Fig. 4 comprises language-agnostic studies, *e.g.,* Aghajani et al. (2019) or the studies considering less popular languages, *e.g.,* Wu et al. (2017) focuses on COBOL. We find only one study (Hata et al. (2019)) that analyzes comments of six programming languages et al. (Hata et al., 2019).

Finding 4. The trend of analyzing multiple software systems of a programming language, or of several languages, shows the increasing use of polyglot environments in software projects.

3.2. RQ_2: Which QAs are used to assess code comments?

To characterize the attention that the relevant studies reserve to each QA over the past decade, Fig. 5 shows all the QAs on the *y*-axis and the corresponding years on the *x*-axis. Each bubble in the plot indicates both the number of papers by the size of the bubble and IDs of the studies. Comparing the *y*-axis with the QAs in Table 4 demonstrates that our analysis finds new QAs with respect to the previous work of Zhi et al. The 10 additional QAs are: *usefulness*, *use of examples*, *usability*, *references*, *preciseness*, *natural language quality*, *maintainability*, *visual models*, *internationalization*, *documentation technology*, *content relevance*, *conciseness*, *coherence*, and *availability*. However, not all QAs reported by Zhi et al. for software documentation quality (highlighted in bold in Table 4) are used in comment quality assessment. In particular, we find no mention of *trustworthiness*, and *similarity* QAs even though previous works have highlighted the importance of both QAs to have high-quality documentation (Visconti and Cook, 2004; Ambler, 2007; Dautovic et al., 2011). Also, Maalej et al. showed in their study that developers trust code comments more than other kinds of software documentation (Maalej et al., 2014), indicating the need to develop approaches to assess the trustworthiness of comments.

Finding 5. Compared to the previous work by Zhi et al. we find 10 additional QAs researchers use to assess code comment quality.

Although several QAs received attention in 2013, the detailed analysis shows that there were mainly two studies (Steidl et al. (2013), Garousi et al. (2013)) covering several QAs. There is only one study published in 2014 (Dagenais and Robillard (2014)), while 2015 sees the first studies focusing on assessing comment quality. One in particular, Garousi et al. (2015), attempts to cover multiple QAs. The plot also shows which QAs receive the most attention. A few QAs such as *completeness*, *accuracy*, *content relevance*, *readability* are often investigated. The QA *consistency* is by far the one that receives constant and consistent attention across the years, with several in 2017 (Ratol and Robillard (2017), Scalabrino et al. (2017), Zhou et al. (2017), Pascarella et al. (2019)) and 2018 (Fakhoury et al. (2018), Aghajani et al. (2018), Scalabrino et al. (2018), Pascarella et al. (2018), Kechagia et al. (2018)). Indeed, the problem of inconsistency has been studied from multiple points of view, such as inconsistency between code and comments that may emerge after code refactoring (Ratol and Robillard (2017)), or the inconsistencies revealed by so-called *linguistic antipatterns* (Aghajani et al. (2018), Arnaoudova et al. (2016)). Unsurprisingly, the plot shows that *up-to-dateness* increasingly has received attention in the last three years of the decade, given that comments that are not updated together with code are also a cause of inconsistency (Wen et al. (2019), Aghajani et al. (2019)).

A few attributes are rarely investigated, for instance the QAs investigated only by at most two studies over the past decade are *format*, *understandability*, *spelling & grammar*, *organization*, *internationalization*, *documentation technology*, *coherence*, *conciseness*, *author related* and *accessibility*. More research would be needed to assess whether such attributes are intrinsically less important than others for comments according to practitioners.

Finding 6. While QAs such as *consistency* and *completeness* are frequently used to assess comment quality, others are rarely investigated, such as *conciseness* and *coherence*.

Another aspect to analyze is whether researchers perceive the QAs as being the same or not. For example, do all studies mean the same by consistency, conciseness, accuracy of comments? We therefore collect the definition of each QA considered in the study. We find that for various QAs researchers refer to the same QA but using different terminology. We map such cases to the *Synonyms* column presented in Table 4. From this analysis we find that not all studies precisely define the QAs, or they refer to their existing definitions while evaluating comments using them. For instance, the studies (Haouari et al. (2011), Pandita et al. (2012), Wang et al. (2019), Pham and Yang (2020), Zhai et al. (2020), Pascarella et al. (2019), Zhang et al. (2018)) do not mention the specific QAs or their definition. We put such studies, classifying comment content with the aim to improve comment quality, under *content relevance*. On the other hand, in some studies researchers mention the QAs but not their definition. For instance, Garousi et al. (2015) refers to various existing studies for the QA definitions but which QA definition is extracted from which study is not very clear. Lack of precise definitions

[9] https://pharo.org/

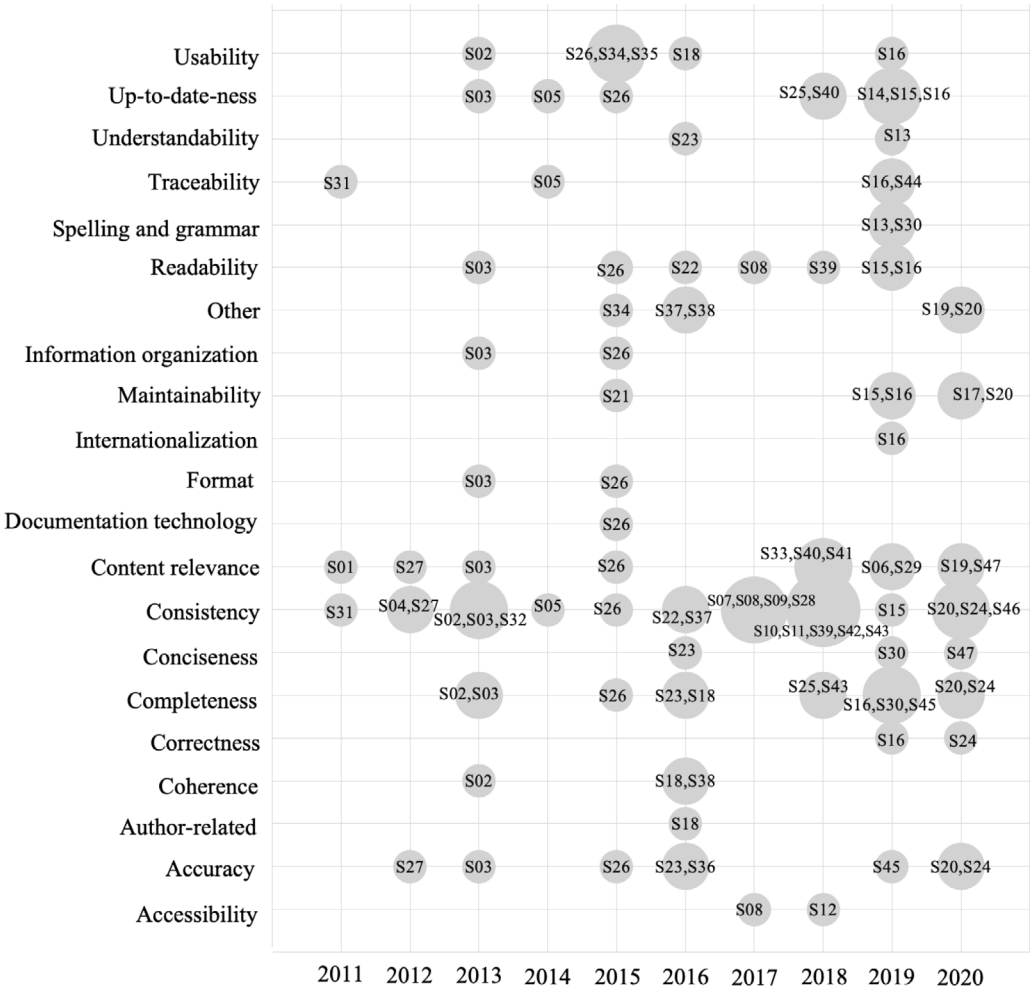


Fig. 5. Frequency of various comment quality QAs over year.

of QAs or having different definitions for the same QAs can create confusion among developers and researchers while assessing comment quality. Future work needs to pay attention to either refer to the existing standard definition of a QA or define it clearly in the study to ensure the consistency and awareness across developer and scientific communities. In this study, we focus on identifying the mention of QAs and their definition if given, and not on comparing and standardizing their definition. Such work would require not only the existing definitions available in the literature for QAs but also collecting how researchers use them in practice, and what developers perceive from each QA for source code comments, which is out of scope for this work. However, we provide the list of QAs researchers use for comment quality assessment to facilitate future work in mapping their definition and standardizing them for code comments.

Although each QA has its own importance and role in comment quality, they are not measured in a mutually exclusive way. We find cases where a specific QA is measured by measuring another QA. For example, *accuracy* is measured by measuring the *correctness* and *completeness* of comment, such as "*the documentation is incorrect or incomplete and therefore no longer accurate documentation of an API.*" (Zhou et al. (2020)) Similarly, *up-to-dateness* is measured through *consistency* of comments (Liu et al. (2018)) or *consistency* is evaluated and improved using *traceability* (Lucia et al. (2011)). This indicates the dependency of various QAs on each other, and improving one aspect of comments can automatically improve other related aspects. However, which techniques are used to measure which QAs is not yet known.

Finding 7. Many studies miss a clear definition of the QAs they use in their studies. This poses various challenges for developers and researchers, *e.g.*, understanding what a specific QA means, mapping a QA to other similar QAs, and adapting the approaches to assess the QA to a certain programming environment.

3.3. RQ_3: Which tools and techniques do researchers use to assess comment QAs?

With respect to each QA, we first identify which techniques have been used to measure them. We use the dimension *Technique type* to capture the type of techniques. Fig. 6 shows that the majority of the QAs are measured by asking developers to manually assess it (*manual assessment*). For instance, QAs such as *coherence*, *format*, *organization*, *understandability*, and *usability* are often assessed manually. This indicates the need and opportunities to automate the measurement of such QAs.

A significant number of studies experimented with various automated approaches based on machine or deep learning approaches, but they focus on specific QAs and miss other QAs such as *natural language quality*, *conciseness*, *correctness*, *traceability*, *coherence etc.* Similarly, another significant portion of studies uses heuristic-based approaches to measure various QAs. The limitation of such heuristic-based approaches is their applicability to other software systems and programming languages. More studies are required to verify the generalizability of such approaches.

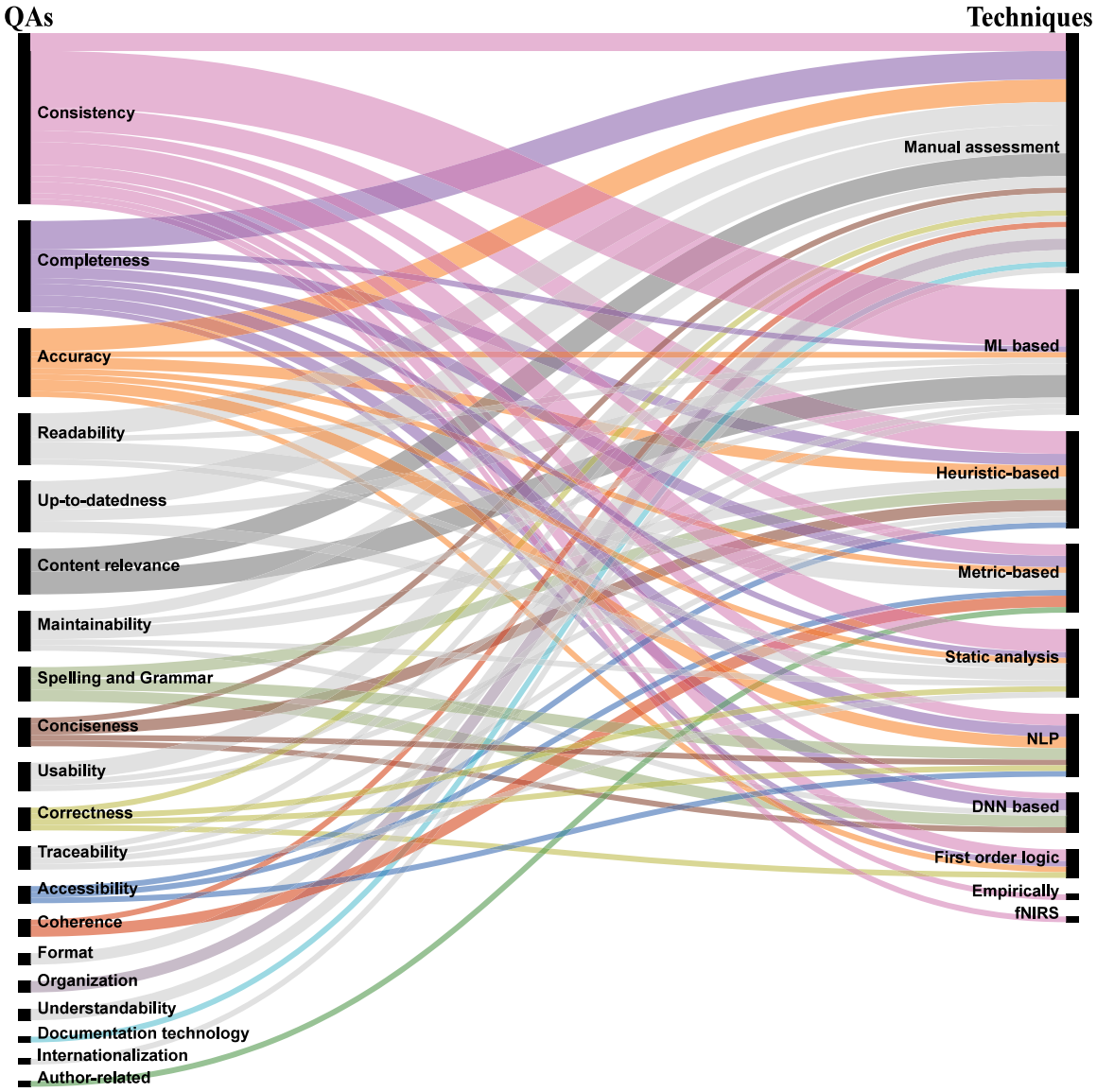


Fig. 6. Types of techniques used to analyze various QAs.

Finding 8. *Manual assessment* is still the most frequently-used technique to measure various QAs. Machine learning based techniques are the preferred automated approach to asses QAs, but the majority of them focus on specific QAs, such as *consistency*, *content relevance*, and *up-to-dateness*, while ignoring other QAs.

We find that the majority of the machine learning-based approaches are supervised ML approaches. These approaches require labeling the data and are therefore expensive in terms of time and effort. To avoid the longer training time and memory consumption of ML strategies, Kallis et al. used *fastText* to classify the issues reports on GitHub (Kallis et al., 2021). The *fastText* tool uses linear models and has achieved comparable results in classification to various deep-learning based approaches. A recent study by Minaee et al. shows that deep learning-based approaches surpassed common machine learning-based models in various text analysis areas, such as news categorization and sentiment analysis (Minaee et al., 2021). We also find some studies that use deep learning-based techniques partly (Fucci et al. (2019), Wang et al. (2019), Zhai et al. (2020)) along with machine learning techniques for a few QAs, such as assessing *conciseness*, *spelling and grammar*, and *completeness*. However, there are still many QAs that are assessed manually and require considerable effort to support developers in automatically assessing comment quality.

Finding 9. In the case of automated approaches to assess various QAs of comments, we observe that deep-learning based approaches are not yet explored even though various studies showed that they surpassed ML-based approaches in text analysis areas.

We see that machine learning-based approaches are used more often than deep-learning approaches, but whether it is due to their high accuracy, easy interpretation, or need for a small dataset is unclear and requires further investigation.

In addition to identifying general techniques, we collect which metrics and tools have been used to measure various QAs. Table 5 shows various QAs in the column *QAs*, and metrics and tools used for each QA in the column *Metrics*, and *Tools* respectively. The

Table 6
Metrics and tools used for various quality attributes.
Note: the description of each metric is given in Table 7.

QAs	Metrics	Tools
Accessibility	Scalabrino et al. (2017): Accessibility_1, Accessibility_2	Li et al. (2018): Text2KnowledgeGraph
Readability	Scalabrino et al. (2017): Readability_1 Scalabrino et al. (2016): Readability_1 Scalabrino et al. (2018): Readability_1	
Spelling and Grammar	Wang et al. (2019): SpellGrammar_1	
Correctness		Zhou et al. (2020): Drone
Completeness	Sun et al. (2016): Completeness_1, Author_1 Steidl et al. (2013): Completeness_2 Kechagia et al. (2018): Completeness_3	Zhou et al. (2020): Drone Xi et al. (2019): DAAMT
Consistency	Scalabrino et al. (2017): Consistency_1 Scalabrino et al. (2016): Consistency_1 Scalabrino et al. (2018): Consistency_1 Iammarino et al. (2020): Consistency_2 Wu et al. (2017): LicenseConsistency_1	Ratol and Robillard (2017): Fraco Fucci et al. (2019): RecoDoc, AdDoc Zhou et al. (2020): Drone Lucia et al. (2011): Coconut Zhou et al. (2017): Zhou et. al Arnaoudova et al. (2016): LAPD Pascarella et al. (2018): PyID
Traceability		Fucci et al. (2019): RecoDoc Lucia et al. (2011): Coconut
Up-to-datedness		Fucci et al. (2019): RecoDoc, AdDoc Nie et al. (2019): Trigit
Accuracy	McBurney and McMillan (2016): Accuracy_1	Zhou et al. (2020): Drone Xi et al. (2019): DAAMT
Coherence	Steidl et al. (2013). Coherence_1, Coherence_2 Sun et al. (2016): Coherence_3 Corazza et al. (2018): Coherence_4	
Maintainability	Rahman et al. (2015): Coherence_4	Pham and Yang (2020): Pham et. al.
Understandability	Wang et al. (2019): Understandability_1	
Usability	Rama and Kak (2015): Usability_1	Robillard and Chhetri (2015): Krec

description of the collected metrics is presented in Table 7. We can see that out of 21, only 10 QAs have metrics defined for them.

A software metric is a function that takes some software data as input and provides a numerical value as an output. The output provides the degree to which the software possesses a certain attribute affecting its quality (Committee et al., 1993). To limit the incorrect interpretation of the metric, threshold values are defined. However, the threshold value may change according to the type of comments analyzed, and the interpretation of the metric output may vary in turn. We report threshold values, if present, for the collected metrics.

For *readability* QA, researchers were often found to be using the same metric (Scalabrino et al. (2017), Scalabrino et al. (2016), Scalabrino et al. (2018)). As developers spend significant amount of time reading code, including comments, having readable comment can help them in understanding code easier. Yet readability remains a subjective concept. Several studies, such as Scalabrino et al. (2017), Scalabrino et al. (2016), Scalabrino et al. (2018) identified various syntactic and textual features for source code and comments. However, in context of code comments, they focus on the Flesch–Kincaid index method, which is typically used to assess readability of natural language text. Since comments often consist of a mix of source code and natural language text, such methods can have disadvantages. For example, developers can refer to the same code concept differently in comments, and they can structure their information differently. Thus, formulating metrics that consider the special context of code comments can improve the assessment of readability of comments.

Another popular metric is *Consistency_1* used for assessing consistency between comments and code (Scalabrino et al. (2017), Scalabrino et al. (2016), Scalabrino et al. (2018)). This metric measures the overlap between the terms of method comments and method body. These studies assume that the higher the overlap, better the readability of that code. Similarly, metrics (*coherence_1*, *coherence_3*, *coherence_4*) used for measuring the *coherence* QA suggest higher overlap between comments and code. However, having too many overlapping words can defy the purpose of comments and can lead to redundant comments. Using such metrics, a comment containing only rationale information about a method or class might be qualified as an incoherent or inconsistent comment whereas such comments can be very helpful in providing additional important information. Although metrics can help developers easily estimate the quality of comments, their sensitivity towards various QAs can degrade comment quality overall. More research is required to know the implication of given metrics on various QAs or combinations of QAs.

Finding 10. Nearly 25% of the studies use metric-based methods to measure comment quality. However, the metrics are defined or used for only 10 QAs out of 21QAs.

3.4. RQ4: What kinds of contribution do studies often make?

Research types. As a typical development cycle can contain various research tasks, such as investigation of a problem, or validation of a solution, we collect which types of research are performed for the comment quality assessment domain, and what kinds of solutions researchers often contribute. We categorize the papers according to the *research type* dimension and show its results in Fig. 7. The results show that the studies often conduct validation research (investigating the properties of a solution) followed by the solution proposal (offering a proof-of-concept method or technique). However, very few studies focus on evaluation research (investigating the problem or a technique implementation in practice). We find only one study performing a replication study (Stapleton et al. (2020)). Given the importance of research replicability in any field, future work needs to focus more on evaluating the proposed solution and testing their replicability in this domain.

Table 7
Description of each metric mentioned in Table 6.

Metrics	Description
Accessibility_1	MIDQ: (Documentable items of a method + readability of comments)/2. [Scalabrino et al. (2017)]
Accessibility_2	AEDQ: Identify all Stack overflow discussions that have "how to" words and the class name in the title. [Scalabrino et al. (2017)]
Accuracy_1	Short Text Semantic Similarity (STSS): the intersection of keywords between summaries and source code, STASIS (word semantic similarity, sentence semantic similarity, and word order similarity), LSS (Lightweight Semantic Similarity). [McBurney and McMillan (2016)]
Author_1	A class comment should contain authorship. Check the presence and absence of the @author tag with the following name. [Sun et al. (2016)]
Coherence_1	The similarity between words from method comments and method names where similarity is computed using Levenshtein distance. The value should be between 0 and 0.5 to have a coherent comment. [Steidl et al. (2013)]
Coherence_2	The length of comments should be between 2 words to 30 words. [Steidl et al. (2013)]
Coherence_3	Percentage of the number of class or method's words contained in the class or method comments divided by the total class or method's words. The value should be above or equal to 0.5. [Sun et al. (2016)]
Coherence_4	There is coherence between the comment and the implementation of a method when they have a high lexical similarity, where lexical similarity is computed using cosine similarity. [Corazza et al. (2018)]
Completeness_1	A class comment should contain a description and authorship. A method should contain comments if it is complex (more than three method invocation) and have 30 LOC. [Sun et al. (2016)]
Completeness_2	How many of the public classes, types, and methods have a comment preceding them. [Steidl et al. (2013)]
Completeness_3	Exceptions that are present in App Programs, Crashes, and API source code but not in API reference documentation. [Kechagia et al. (2018)]
Consistency_1	The overlap between the terms used in a method comment and the terms used in the method body. They correlate a higher value of CIC with a higher readability level of that code. [Scalabrino et al. (2017), Scalabrino et al. (2016), Scalabrino et al. (2018)]
Consistency_2	The Kullback–Leibler divergence is a measure that finds the difference between two probability distributions. [Iammarino et al. (2020)]
LicenseConsistency_1	Two similar source code files have different licenses. Find the number of files in a group, number of different licenses in the group, number of files with an unknown license in the group, number of files without any license in the group, and number of licenses in the GPL family. [Wu et al. (2017)]
Readability_1	Flesch reading-ease test. [Scalabrino et al. (2017), Scalabrino et al. (2016), Scalabrino et al. (2018)]
SpellGrammar_1	The sentence has no subject or predicate, or has incomplete punctuations (e.g., the right parenthesis is missing). [Wang et al. (2019)]
Understandability_1	Remove a sentence if it is incomplete, contains code elements, is a question, or it mentions the concept in its subordinate clauses. [Wang et al. (2019)]
Usability_1	ADI: number of words in the method comments. The threshold is decided based on the simple average of the ADI for all method declaration. [Rama and Kak (2015)]

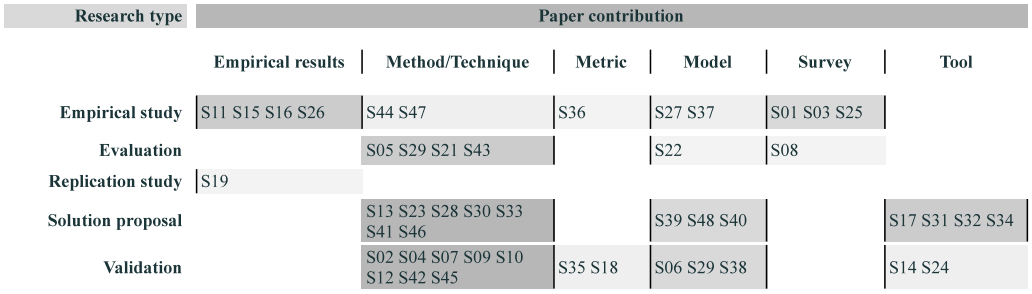


Fig. 7. Types of contribution for each research type.

Paper contribution types. By categorizing the papers according to the *paper contribution* definition, Figs. 7 and 8 show that over 44% of papers propose an approach (method/technique) to assess code comments. A large part (75%) of them are heuristics-based approaches, *e.g.,* Zhou et al. and Wang et al. present such NLP based heuristics (S9, Wang et al. (2019)). A few approaches rely on manual assessments. As an example, consider how taxonomies assessing comment quality have emerged (Wen et al., 2019; Aghajani et al., 2019). Models are the second contribution by frequency, which makes sense considering the increasing trend of leveraging machine learning during the considered decade: 60% of the relevant papers proposing models are based on such approaches. The label *Empirical results* comprises studies which mainly offer insights through authors' observations (*e.g.,* Aghajani et al. (2018), Wen et al. (2019), Aghajani et al. (2019), Stapleton et al. (2020), Garousi et al. (2015)). Finally, given the important role that metrics have in software engineering (Fenton and Bieman, 2014; Meneely et al., 2012), it is valuable to look into metrics that are proposed or used to assess code comment quality as well. For example, three studies (Sun et al. (2016), Rama and Kak (2015), and McBurney and McMillan (2016)) contribute metrics for *completeness*, *accuracy*, or *coherence* whereas other studies use existing established metrics, *e.g.,* Scalabrino et al. (2017), Scalabrino et al. (2016), or Scalabrino et al. (2018) compute the *readability* of comments using the metric named the Flesch–Kincaid index.

Paper contribution	Evaluation type					
	Authors of the work	Experiment	Performance metrics	Survey practitioners	Survey practitioners and students	Survey students
Empirical results	S15 S16	S11		S26	S19	
Method/Technique	S30 S42 S44 S47	S13 S28 S43 S45 S46	S02 S04 S05 S07 S09 S13 S20 S41 S42	S02 S05 S21	S20 S23 S43	S10 S12
Metric	S36	S35			S18	
Model	S27 S38	S06 S27 S37 S38 S39	S06 S22 S29 S33 S40		S37	S40
Survey				S01 S03 S25	S08	
Tool		S32 S34	S17 S24		S14 S34	S24 S31

Fig. 8. Types of evaluation for each paper contribution type.

Tool availability. Previous work indicates the developers' effort in seeking tools to assess documentation quality, and highlights the lack of such tools (Aghajani et al., 2019). In our study, we find that 32% of the studies propose tools to assess specific QAs, mainly for detecting inconsistencies between code and comments. Of these studies proposing tools, 60% provide a link to them. The lack of a direct link in the remaining 40% can hinder the reproducibility of such studies.

Dataset availability. In terms of dataset availability, 49% of the studies provide a link to a replication package. Of the remaining papers, some provide a link to the case studies they analyze (typically open-source projects) (Haouari et al., 2011), build on previously existing datasets (Scalabrino et al., 2018), or mention the reasons why they could not provide a dataset. For instance, Garousi et al. indicated the company policy as a reason to not to share the analyzed documentation in their case study (Garousi et al., 2015).

Finding 11. Nearly 50% of the studies still are lacking on the *replicability* dimension, with their respective dataset or tool often not publicly accessible.

3.5. RQ_5: How do researchers evaluate their comment quality assessment studies?

Fig. 8 shows how authors evaluate their contributions. We see that code comment assessment studies generally lack a systematic evaluation, surveying only students, or conducting case studies on specific projects only. Most of the time, an experiment is conducted without assessing the results through any kind of external expertise judgment. Hence, only 30% of the relevant studies survey practitioners to evaluate their approach. This tendency leads to several disadvantages. First, it is difficult to assess the extent to which a certain approach may overfit specific case studies while overlooking others. Second, approaches may be unaware of the real needs and interests of project developers. Finally, the approaches may tend to focus too little on real-world software projects (such as large software products evolving at a fast pace in industrial environments). Similarly, when a new *method* or *technique* or comment classification *model* is proposed, it is often assessed based on conventional performance metrics, such as Precision, Recall, or F1 (Steidl et al. (2013), Pandita et al. (2012), Ratol and Robillard (2017), Pascarella et al. (2019), Zhang et al. (2018) *etc.*.) and rarely are the results verified in an industry setting or with practitioners.

Finding 12. Many code comment assessment studies still lack systematic industrial evaluations for their proposed approaches, such as evaluating the *metric*, *model*, or *method/technique* with practitioners.

4. Discussion

Below we detail our observations about state of the art in comment quality analysis together with implications and suggestions for future research.

Comment types. The analysis of the comment quality assessment studies in the last decade shows that the trend of analyzing comments from multiple languages and systems is increasing compared to the previous decade where a majority of the studies focus on one system (Zhi et al., 2015). It reflects the increasing use of polyglot environments in software development (Tomassetti and Torchiano, 2014). Additionally, while in the past researchers focused on the quality of code comments in general terms, there is a new trend of studies that narrow their research investigation to particular comment types (methods, TODOs, deprecation, inline comments), indicating the increasing interest of researchers in supporting developers in providing a particular type of information for program comprehension and maintenance tasks.

Emerging QAs. Our analysis of the last decade of studies on code comment assessment shows that new QAs (*coherence*, *conciseness*, *maintainability*, *understandability etc.*.), which were not identified in previous work (Zhi et al., 2015), are now being investigated and explored by researchers. This change can be explained by the fact that while in the past researchers focused on the quality of code comments in general terms, in the last decade there has been a new trend of studies that narrow their research investigation to specific comment types (methods, TODOs, deprecation, inline comments) and related QAs.

Mapping QAs. As a consequence of this shift of focus towards specific comment types, the same QAs used in prior studies can assume different definition nuances, depending on the kind of comments considered. For instance, let us consider how the QA *up-to-dateness*, referred to in studies on code-comment *inconsistency*, assumes a different interpretation in the context of TODO comments. A TODO comment that *becomes outdated* describes a feature that is not being implemented, which means that such a comment should be addressed within some deadline, and then removed from the code base (Nie et al. (2019)) when either the respective code is written and potentially documented with a different comment, or the feature is abandoned altogether. At the same time, more research nowadays is conducted to understand the relations between different QAs.

Mapping taxonomies. In recent years, several taxonomies concerning code comments have been proposed, however, all of them are characterized by a rather different focus, such as the scope of the comments (Steidl et al. (2013)), the information embedded in the comment (Pascarella et al. (2019), Zhang et al. (2018)), the issues related to specific comment types (Fucci et al. (2019), Shinyama et al. (2018), Liu et al. (2018)), as well as the programming language they belong to. This suggests the need for a comprehensive code comment taxonomy or model that maps all these aspects and definitions in a more coherent manner to have a better overview of developer commenting practices across languages. Rani et al. adapted the code comment taxonomies of Java and Python (Pascarella et al. (2019), Zhang et al. (2018)) for class comments of Java and Python (Rani et al., 2021a). They mapped the taxonomies to Smalltalk class comments and found that developers write similar kinds of information in class comments across languages. Such a mapping can encourage building language-independent approaches for other aspects of comment quality evaluation.

5. Implication for future studies

Besides the aspects discussed above, future studies on code comment assessment should be devoted to filling the gaps of the last decade of research as well as coping with the needs of developers interested in leveraging comment assessment tools in different program languages.

Investigating specific comment types (RQ1). Several works showed the importance of different types of comments to achieve specific development tasks and understanding about code. Although, the trend of analyzing specific comment types has increased over the last decade, there are still comment types (*e.g.,* class and package comments) that need more attention.

Generalizing across languages (RQ1). Given the preponderance of studies focusing on the Java language, and considering that statistics from various developer boards (StackOverflow, GitHub) suggest that there are other popular languages as well (*e.g.,* Python and JavaScript), more studies on analyzing various types of comments in these languages are needed. Interesting questions in this direction could concern the comparison of practices (*e.g.,* given Python is often considered to be "self-explainable", do developers write fewer comments in Python?) and tools used to write code comments in different languages (*e.g.,* popularity of Javadoc v.s. Pydoc). Similarly, whether various programming language paradigms, such as functional versus object-oriented languages, or statically-typed versus dynamic-typed languages, play a role in the way developers embed information in comments, or the way they treat comments, needs further work in this direction.

Identifying QAs (RQ2). Our results show various QAs, *e.g., consistency*, *completeness*, and *accuracy* that are frequently considered in assessing comment quality. Additionally, various metrics, tools, and techniques that are proposed to assess them automatically. Indeed, some QAs are largely overlooked in the literature, *e.g.,* there is not enough research on approaches and automated tools that ensure that comments are *accessible*, *trustworthy*, and *understandable*, despite numerous studies suggesting that having good code comments brings several benefits.

Standardizing QAs (RQ2). We identify various QAs that researchers consider assessing comment quality. Not all of these QAs are unique *i.e.,* they have conceptual overlap (based on their definitions in Table 4 and measurement techniques in Table 6). For example, the definition of *up-to-datedness* and *consistency* mention of keeping comments updated. Similarly, the definition of *coherence* and *similarity* focus on the relatedness between code and comments. In this study, we mainly focus on identifying various QAs from the literature and on extracting metrics, tools, and techniques to measure them. Standardizing their definition can be an essential next step in the direction of comment quality assessment research. Since not every study provides the definition of mentioned QAs, such a work will require surveying the authors to understand how they perceive various QAs and where they refer to for QAs definitions.

Comment smells (RQ2). Although there is no standard definition of good or bad comments, many studies indicate bloated comments (or non-informative comments), redundant comments (contain same information as in the code), or inconsistent comments (*e.g.,* contain conflicting information compared to the code) as code or comment smells. Arnaoudva et al. identified various LAs that developers perceive as poor practices and should be avoided (Arnaoudova et al., 2016). Still, what information is vital in comments is a subjective concept and can sometimes be contradictory. For instance, Oracle's coding style guideline suggests including author information in class comments, whereas the Apache style guideline suggests removing it as it can be inferred from the version control system (Anon, 2020). We find that researchers use the *completeness* QA to identify informative comments. They define various metrics to assess the completeness of comments, as shown in Table 7. These metrics check the presence of specific information, such as summary, author, or exception information in class or method comments Future work can investigate the definition of good and bad comments by surveying various sources, such as documentation guidelines, researchers, and developers, and comparing the sources across to improve the understanding of high-quality comments. Such work can inspire the development of more metrics and tools to ensure the adherence of comments to the standards.

Automated tools and techniques (RQ3). Finally, concerning techniques to assess comment quality, we observed that those based on AI, such as NLP and ML, were increasingly used in the past decade. On the other hand, deep learning techniques do not yet seem to have gained a foothold within the community for assessing comment quality. Since code comment generation is becoming more and more popular also due to such advanced techniques emerging, we envision that future work may study techniques and metrics to assess the quality of automatically generated code comments.

Research evaluation (RQ4 and RQ5). Scientific methods play a crucial role in the growth of engineering knowledge (Vincenti et al., 1990). Several studies have indicated the weak validation in software engineering (Zelkowitz and Wallace, 1997). We also find that several studies propose solutions but do not evaluate their solution. Also, various approaches were validated only by the authors of the work or by surveying students. However, we need to do all steps as engineering science researchers do, empirically investigating the problems, proposing solutions, and validating those solutions.

In contrast to seven research types listed in Table 4, we observe only limited types of research studies. For example, we do not find any philosophical, opinion, or experience papers for the comment quality assessment domain even though this domain is more than a decade old now. Philosophical papers sketch a new perspective of looking at things, conceptual frameworks, metrics *etc.* Opinion papers present good or bad opinions of authors about something, such as different approaches to assess quality, using particular frameworks *etc.* Similarly, experience papers often present insights about lessons learned or anecdotes by authors in using tools or techniques in practice. Such papers help tool designers better shape their future tools.

6. Threats to validity

We now outline potential threats to the validity of our study. *Threats to construct validity* mainly concern the measurements used in the evaluation process. In this case, threats can be mainly due to (i) the imprecision in the automated selection of relevant papers (*i.e.,* the three-step search on the conference proceedings based on regular expressions), and to (ii) the subjectivity and error-proneness of the subsequent manual classification and categorization of relevant papers.

We mitigated the first threat by manually classifying a sample of relevant papers from a set of conference proceedings and compared this classification with the one recommended by the automated approach based on regular expressions. This allowed us to incrementally improve the initial set of regular expressions. To avoid any bias in the selection of the papers, we selected regular expression in a deterministic way (as detailed in Section 2): We first examined the definition of *documentation* and *comment* in *IEEE Standard Glossary of Software Engineering Terminology* (IEEE Standard 610.12–1990) and identified the first set of keywords *comment*, *documentation*, and *specification*; we further added comment-related keywords that are frequently mentioned in the context of code comments. Moreover, we formulated a set of keywords to discard irrelevant studies that presented similar keywords (e.g., code review comments). To verify the correctness of the final set of keywords, we manually scanned the full venue proceedings metadata to make sure the set of keywords did not prune relevant papers. This iterative approach allowed us to verify that our keyword-based filtering approach does not lead to false negatives for the selected venues.

We mitigated the second threat by applying multi-stage manual classification of conference proceedings, involving multiple evaluators and reviewers, as detailed in Section 2.

Threats to internal validity concern confounding factors that could influence our results and findings. A possible source of bias might be related to the way we selected and analyzed the conference proceedings. To deal with potential threats regarding the actual regular expressions considered for the selection of relevant studies, we created regular expressions that tend to be very inclusive, *i.e.,* that select papers that are marginally related to the topic of interest, and we take a final decision only after a manual assessment.

Threats to external validity concern the generalization and completeness of results and findings. Although the number of analyzed papers is large, since it involves studies spanning the last ten years of research, there is still the possibility that we missed some relevant studies. We mitigate this threat by applying various selection criteria to select relevant conference proceedings, considering the well-established venues and communities related to code comment-related studies, as detailed in Section 2. It is important to mention that this paper intentionally limits its scope in two ways, which threatens to the completeness of the study results and findings. First of all, we mainly focus on research work investigating code comment quality without integrating studies from industry tracks of conference venues (as was done in previous studies thematically close to ours (Ding et al., 2014; Zhi et al., 2015)). Second, we focus on those studies that involve manually written code comments in order to avoid auto-generated comments (already investigated in recent related work (Song et al., 2019; Nazar et al., 2016)). To further limit potential threats concerning the completeness of our study, we use the snowball approach to reach potentially relevant studies that we could have missed with our venue selection. However, we support the argument of Garousi et al. (2016) who report that a *multivocal* literature review, with further replications, is desirable to make the overall interpretation of code comment quality attributes more complete for future work.

7. Related work

This section discusses the literature concerning (i) studies motivating the importance of quality attributes for software documentation, (ii) comment quality aspects, and (iii) recent SLRs discussing topics closely related to our investigation.

Important quality attributes for software documentation. Various research works conducted surveys with developers to identify important quality attributes of good software documentation. Forward and Lethbridge surveyed 48 developers, and highlighted developer concerns about outdated documentation (Forward and Lethbridge, 2002). Chen and Huang surveyed 137 project managers and software engineers (Chen and Huang, 2009). Their study highlighted the typical quality problems developers face in maintaining software documentation: adequacy, complete, traceability, consistency, and trustworthiness. Robillard et al. conducted personal interviews with 80 practitioners and presented the important attributes for good documentation, such as including examples and usage information, complete, organized, and better design (Robillard, 2009). Similarly, Plosch et al. surveyed 88 practitioners and identified consistency, clarity, accuracy, readability, organization, and understandability as the most important attributes (Plösch et al., 2014). They also indicated that developers do not consider documentation standards important (*e.g.,* ISO 26514:2008, IEEE Std.1063:2001). Sohan et al. in their survey study highlighted the importance of examples in documentation (Sohan et al., 2017). The majority of the highlighted documentation quality attributes apply to code comments as well (as a type of software documentation). However, which specific quality attributes (*e.g.,* outdated, complete, consistent, traceable) researchers consider important to assess code comment quality and how these quality attributes are measured is yet to study.

Comment quality. Evaluating comment quality according to various aspects has gained a lot of attention from researchers, for instance, assessing their adequacy (Arthur and Stevens, 1989) and their content quality (Khamis et al., 2010; Steidl et al., 2013), analyzing co-evolution of comments and code (Fluri et al., 2009), or detecting inconsistent comments (Ratol and Robillard, 2017; Wen et al., 2019). Several works have proposed tools and techniques for the automatic assessment of comment quality (Khamis et al., 2010; Steidl et al., 2013; Yu et al., 2016). For instance, Khamis et al. assessed the quality of inline comments based on consistency and language quality using a heuristic-based approach (Khamis et al., 2010). Steidl et al. evaluated documentation comment quality based on four quality attributes, such as consistency, coherence, completeness, and usefulness of comments using a machine learning-based model (Steidl et al., 2013). Zhou et al. proposed a heuristic and natural language processing-based technique to detect incomplete and incorrect comments (Zhou et al., 2017). These works have proposed various new quality attributes to assess comment quality, such as completeness, coherence, and language quality, that are not included in previous quality models. However, a unifying overview of comment QAs and their assessment approaches is still missing. Our paper complements these previous works by investigating comment QAs discussed in the last decade of research.

Previous SLRs on code comments and software documentation. In recent years, SLRs have been conducted to investigate agile software development aspects in open-source projects (Silva et al., 2017), the usage of ontologies in software process assessment (Tarhan and Giray, 2017), and improvement aspects in DevOps process and practices (Badshah et al., 2020). Previous SLRs in the field investigated code comments and software documentation (Ding et al., 2014; Zhi et al., 2015), which are closely related to our work. Specifically, Ding et al. conducted an SLR to

explore the usage of knowledge-based approaches in software documentation (Ding et al., 2014). They identified twelve QAs. They also highlighted the need to improve QAs, especially conciseness, credibility, and unambiguity. Zhi et al. have explored various types of software documentation to see which QAs impact it (Zhi et al., 2015). Both of the studies considered the timeline until 2011. Additionally, they have not studied how the proposed comment quality assessment approaches are computed in practice for comments. Inspired by these related studies, we focused specifically on the code comment aspect. Song et al. conducted a literature review on code comment generation techniques, and indicated the need to design an objective comment quality assessment model (Song et al., 2019). Complementarily, Nazar et al. (2016) presented a literature review in the field of summarizing software artifacts, which included source code comment generation as well as bug reports, mailing lists, and developer discussion artifacts. Our work complements these previous studies since we mainly focus on manually written comments.

8. Conclusion

In this work, we present the results of a systematic literature review on source code comment quality evaluation practices in the decade 2011— 2020. We analyze 2353 publications and study 47 of them to understand of effort of Software Engineering researchers, in terms of what type of comments they focus their studies on, what QAs they consider relevant, what techniques they resort to in order to assess their QAs, and finally, how they evaluate their contributions. Our findings show that most studies consider only comments in Java source files, and thus may not generalize to comments of other languages, and they focus on only a few QAs, especially on consistency between code and comments. Some QAs, such as conciseness, coherence, organization, and usefulness, are rarely investigated. As coherent and concise comments play an important role in program understanding, establishing approaches to assess these attributes requires more attention from the community. We also observe that the majority of the approaches appear to be based on heuristics rather than machine learning or other techniques and, in general, need better evaluation. Such approaches require validation on other languages and projects to generalize them. Though the trend of analyzing comments appearing in multiple projects and languages is increasing compared to the previous decade, as reported by Zhi et al. the approaches still need more thorough validation (Zhi et al., 2015).

CRediT authorship contribution statement

Pooja Rani: Conceptualization, Data curation, Software, Methodology, Investigation, Validation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Arianna Blasi:** Conceptualization, Data curation, Software, Methodology, Investigation, Validation, Writing – review & editing, Visualization. **Nataliia Stulova:** Conceptualization, Data curation, Software, Methodology, Investigation, Validation, Writing – review & editing, Visualization, Project administration. **Sebastiano Panichella:** Conceptualization, Methodology, Investigation, Validation, Writing – review & editing. **Alessandra Gorla:** Conceptualization, Data curation, Software, Methodology, Writing – review & editing. **Oscar Nierstrasz:** Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I have shared a replication package

Acknowledgment

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project “Agile Software Assistance” (SNSF project No. 200020-181973, Feb 1, 2019–Apr 30, 2022) and the Spanish Government through the SCUM grant RTI2018-102043-B-I00, and the Madrid Regional through the project BLOQUES. We also acknowledge the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS.

References

Abidi, M., Khomh, F., 2020. Towards the definition of patterns and code smells for multi-language systems. In: EuroPLoP '20: European Conference on Pattern Languages of Programs 2020, Virtual Event, Germany, 1-4 July, 2020. ACM, pp. 37:1–37:13. http://dx.doi.org/10.1145/3424771.3424792.

Aghajani, E., Nagy, C., Bavota, G., Lanza, M., 2018. A large-scale empirical study on linguistic antipatterns affecting APIs. In: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018. IEEE Computer Society, pp. 25–35. http://dx.doi.org/10.1109/ICSME.2018.00012.

Aghajani, E., Nagy, C., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., Shepherd, D.C., 2020. Software documentation: the practitioners' perspective. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering. ICSE, IEEE, pp. 590–601.

Aghajani, E., Nagy, C., Vega-Márquez, O.L., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., 2019. Software documentation issues unveiled. In: Atlee, J.M., Bultan, T., Whittle, J. (Eds.), Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019. IEEE / ACM, pp. 1199–1210. http://dx.doi.org/10.1109/ICSE.2019.00122.

Allamanis, M., Barr, E.T., Bird, C., Sutton, C., 2014. Learning natural coding conventions. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE. ACM, New York, NY, USA, pp. 281–293. http://dx.doi.org/10.1145/2635868.2635883.

Ambler, S.W., 2007. Agile/lean documentation: strategies for agile software development. Retrieved June 20, 2007.

Anon, 0000. Oracle documentation guidelines, https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html, (verified on 10 2020).

Arnaoudova, V., Penta, M.D., Antoniol, G., 2016. Linguistic antipatterns: what they are and how developers perceive them. Empir. Softw. Eng. 21 (1), 104–158. http://dx.doi.org/10.1007/s10664-014-9350-8.

Arthur, J.D., Stevens, K.T., 1989. Assessing the adequacy of documentation through document quality indicators. In: Proceedings. Conference on Software Maintenance-1989. IEEE, pp. 40–49.

Auyang, S.Y., 2006. Engineering-an Endless Frontier. Harvard University Press.

Badshah, S., Khan, A.A., Khan, B., 2020. Towards process improvement in devops: A systematic literature review. In: Li, J., Jaccheri, L., Dingsoyr, T., Chitchyan, R. (Eds.), EASE '20: Evaluation and Assessment in Software Engineering, Trondheim, Norway, April 15-17, 2020. ACM, pp. 427–433. http://dx.doi.org/10.1145/3383219.3383280.

Chen, J.-C., Huang, S.-J., 2009. An empirical analysis of the impact of software development problem factors on software maintainability. J. Syst. Softw. 82 (6), 981–992.

Committee, S.E.S., et al., 1993. Ieee Standard for a Software Quality Metrics Methodology. IEEE Std 1061-1992, pp. 1–96. http://dx.doi.org/10.1109/IEEESTD.1993.115124.

Corazza, A., Maggio, V., Scanniello, G., 2018. Coherence of comments and method implementations: A dataset and an empirical investigation. Softw. Qual. J. 26 (2), 751–777.

Dagenais, B., Robillard, M.P., 2014. Using traceability links to recommend adaptive changes for documentation evolution. IEEE Trans. Softw. Eng. 40 (11), 1126–1146. http://dx.doi.org/10.1109/TSE.2014.2347969.

Dautovic, A., Plösch, R., Saft, M., 2011. Automated quality defect detection in software development documents. In: First International Workshop on Model-Driven Software Migration (MDSM 2011), p. 29.

de Souza, S.C.B., Anquetil, N., de Oliveira, K.M., 2005. A study of the documentation essential to software maintenance. In: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information, SIGDOC '05. ACM, New York, NY, USA, pp. 68–75. http://dx.doi.org/10.1145/1085313.1085331.

Dekel, U., Herbsleb, J.D., 2009. Reading the documentation of invoked API functions in program comprehension. In: 2009 IEEE 17th International Conference on Program Comprehension. IEEE, pp. 168–177.

Ding, W., Liang, P., Tang, A., Van Vliet, H., 2014. Knowledge-based approaches in software documentation: A systematic literature review. Inf. Softw. Technol. 56 (6), 545–567.

Fakhoury, S., Ma, Y., Arnaoudova, V., Adesope, O.O., 2018. The effect of poor source code lexicon and readability on developers' cognitive load. In: Khomh, F., Roy, C.K., Siegmund, J. (Eds.), Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018. ACM, pp. 286–296. http://dx.doi.org/10.1145/3196321.3196347.

Fenton, N., Bieman, J., 2014. Software Metrics: A Rigorous and Practical Approach. CRC Press.

Fluri, B., Würsch, M., Giger, E., Gall, H.C., 2009. Analyzing the co-evolution of comments and source code. Softw. Qual. J. 17 (4), 367–394.

Forward, A., Lethbridge, T.C., 2002. The relevance of software documentation, tools and technologies: A survey. In: Proceedings of the 2002 ACM Symposium on Document Engineering, DocEng '02. ACM, New York, NY, USA, pp. 26–33. http://dx.doi.org/10.1145/585058.585065.

Fucci, D., Mollaalizadehbahnemiri, A., Maalej, W., 2019. On using machine learning to identify knowledge in API reference documentation. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 109–119.

Garousi, V., Felderer, M., Mäntylä, M.V., 2016. The need for multivocal literature reviews in software engineering: complementing systematic literature reviews with grey literature. In: Beecham, S., Kitchenham, B.A., MacDonell, S.G. (Eds.), Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering, EASE 2016, Limerick, Ireland, June 01-03, 2016. ACM, pp. 26:1–26:6. http://dx.doi.org/10.1145/2915970.2916008.

Garousi, G., Garousi, V., Moussavi, M., Ruhe, G., Smith, B., 2013. Evaluating usage and quality of technical software documentation: An empirical study. In: da Silva, F.Q.B., Juzgado, N.J., Travassos, G.H. (Eds.), 17th International Conference on Evaluation and Assessment in Software Engineering, EASE '13, Porto de Galinhas, Brazil, April 14-16, 2013. ACM, pp. 24–35. http://dx.doi.org/10.1145/2460999.2461003.

Garousi, G., Garousi-Yusifoğlu, V., Ruhe, G., Zhi, J., Moussavi, M., Smith, B., 2015. Usage and usefulness of technical software documentation: An industrial case study. Inf. Softw. Technol. 57, 664–682.

González-Barahona, J.M., Robles, G., 2012. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empir. Softw. Eng. 17 (1), 75–89. http://dx.doi.org/10.1007/s10664-011-9181-9.

Haouari, D., Sahraoui, H.A., Langlais, P., 2011. [How good is your comment? a] study of comments in Java programs. In: Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada, September 22-23, 2011. IEEE Computer Society, pp. 137–146. http://dx.doi.org/10.1109/ESEM.2011.22.

Hata, H., Treude, C., Kula, R.G., Ishio, T., 2019. 9.6 Million links in source code comments: Purpose, evolution, and decay. In: Proceedings of the 41st International Conference on Software Engineering. IEEE Press, pp. 1211–1221.

Iammarino, M., Aversano, L., Bernardi, M.L., Cimitile, M., 2020. A topic modeling approach to evaluate the comments consistency to source code. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020. IEEE, pp. 1–8. http://dx.doi.org/10.1109/IJCNN48605.2020.9207651.

Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S., 2021. Predicting issue types on GitHub. Sci. Comput. Program. 205, 102598.

Kechagia, M., Fragkoulis, M., Louridas, P., Spinellis, D., 2018. The exception handling riddle: an empirical study on the android api. J. Syst. Softw. 142, 248–270. http://dx.doi.org/10.1016/j.jss.2018.04.034.

Keele, S., 2007. Guidelines for performing systematic literature reviews in software engineering. Tech. rep. Technical report, EBSE Technical Report EBSE-2007-01.

Kernighan, B.W., Pike, R., 1999. The Practice of Programming (Addison-Wesley Professional Computing Series), first ed. Addison-Wesley, URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/020161586X.

Khamis, N., Witte, R., Rilling, J., 2010. Automatic quality assessment of source code comments: the JavadocMiner. In: International Conference on Application of Natural Language to Information Systems. Springer, pp. 68–79.

Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering.

Kuhrmann, M., Fernández, D.M., Daneva, M., 2017. On the pragmatic design of literature studies in software engineering: An experience-based guideline. Empir. Softw. Eng. 22 (6), 2852–2891. http://dx.doi.org/10.1007/s10664-016-9492-y.

Lehman, M., Perry, D., Ramil, J., Turski, W., Wernick, P., 1997. Metrics and laws of software evolution–the nineties view. In: Proceedings IEEE International Software Metrics Symposium (METRICS'97). IEEE Computer Society Press, Los Alamitos CA, pp. 20–32. http://dx.doi.org/10.1109/METRIC.1997.637156.

Lemos, O.A.L., Suzuki, M., de Paula, A.C., Goes, C.L., 2020. Comparing identifiers and comments in engineered and non-engineered code: a large-scale empirical study. In: Hung, C., Cerný, T., Shin, D., Bechini, A. (Eds.), SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020. ACM, pp. 100–109. http://dx.doi.org/10.1145/3341105.3373972.

Li, H., Li, S., Sun, J., Xing, Z., Peng, X., Liu, M., Zhao, X., 2018. Improving [API] caveats accessibility by mining API caveats knowledge graph. In: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018. IEEE Computer Society, pp. 183–193. http://dx.doi.org/10.1109/ICSME.2018.00028.

Liu, Z., Chen, H., Chen, X., Luo, X., Zhou, F., 2018. Automatic detection of outdated comments during code changes. In: Reisman, S., Ahamed, S.I., Demartini, C., Conte, T.M., Liu, L., Claycomb, W.R., Nakamura, M., Tovar, E., Cimato, S., Lung, C., Takakura, H., Yang, J., Akiyama, T., Zhang, Z., Hasan, K. (Eds.), 2018 IEEE 42nd Annual Computer Software and Applications Conference, COMPSAC 2018, Tokyo, Japan, 23-27 2018, Volume 1. IEEE Computer Society, pp. 154–163. http://dx.doi.org/10.1109/COMPSAC.2018.00028.

Lucia, A.D., Penta, M.D., Oliveto, R., 2011. Improving source code lexicon via traceability and information retrieval. IEEE Trans. Softw. Eng. 37 (2), 205–227. http://dx.doi.org/10.1109/TSE.2010.89.

Maalej, W., Tiarks, R., Roehm, T., Koschke, R., 2014. On the comprehension of program comprehension. ACM TOSEM 23 (4), 31:1–31:37. http://dx.doi.org/10.1145/2622669, http://mobis.informatik.uni-hamburg.de/wp-content/uploads/2014/06/TOSEM-Maalej-Comprehension-PrePrint2.pdf.

Malik, R.S., Patra, J., Pradel, M., 2019. Nl2type: inferring javascript function types from natural language information. In: Atlee, J.M., Bultan, T., Whittle, J. (Eds.), Proceedings of the 41st International Conference on Software Engineering, ICSE 2019. IEEE / ACM, pp. 304–315. http://dx.doi.org/10.1109/ICSE.2019.00045.

McBurney, P.W., McMillan, C., 2016. An empirical study of the textual similarity between source code and source code summaries. Empir. Softw. Eng. 21 (1), 17–42. http://dx.doi.org/10.1007/s10664-014-9344-6.

McBurney, P.W., McMillan, C., 2016a. Automatic source code summarization of context for Java methods. IEEE Trans. Softw. Eng. 42 (2), 103–119. http://dx.doi.org/10.1109/TSE.2015.2465386.

McMillan, C., Poshyvanyk, D., Grechanik, M., 2010. Recommending source code examples via API call usages and documentation. In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, pp. 21–25.

Meneely, A., Smith, B.H., Williams, L.A., 2012. Validating software metrics: A spectrum of philosophies. ACM Trans. Softw. Eng. Methodol. 21 (4), 24:1–24:28. http://dx.doi.org/10.1145/2377656.2377661.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J., 2021. Deep learning–based text classification: A comprehensive review. ACM Comput. Surv. 54 (3), 1–40.

Monperrus, M., Eichberg, M., Tekes, E., Mezini, M., 2012. What should developers be aware of? An empirical study on the directives of API documentation. Empir. Softw. Eng. 17 (6), 703–737. http://dx.doi.org/10.1007/s10664-011-9186-4.

Motwani, M., Brun, Y., 2019. Automatically generating precise oracles from structured natural language specifications. In: Atlee, J.M., Bultan, T., Whittle, J. (Eds.), Proceedings of the 41st International Conference on Software Engineering, ICSE 2019. IEEE / ACM, pp. 188–199. http://dx.doi.org/10.1109/ICSE.2019.00035.

Nazar, N., Hu, Y., Jiang, H., 2016. Summarizing software artifacts: A literature review. J. Comput. Sci. Tech. 31 (5), 883–909.

Nie, P., Rai, R., Li, J.J., Khurshid, S., Mooney, R.J., Gligoric, M., 2019. A framework for writing trigger-action todo comments in executable format. In: Dumas, M., Pfahl, D., Apel, S., Russo, A. (Eds.), Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019. ACM, pp. 385–396. http://dx.doi.org/10.1145/3338906.3338965.

Nurvitadhi, E., Leung, W.W., Cook, C., 2003. Do class comments aid Java program understanding? In: 33rd Annual Frontiers in Education, 2003. FIE 2003. 1. IEEE, T3C–T3C.

Padioleau, Y., Tan, L., Zhou, Y., 2009. Listening to programmers — taxonomies and characteristics of comments in operating system code. In: Proceedings of the 31st International Conference on Software Engineering. IEEE Computer Society, pp. 331–341.

Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., Paradkar, A.M., 2012. Inferring method specifications from natural language API descriptions. In: Glinz, M., Murphy, G.C., Pezzè, M. (Eds.), 34th International Conference on Software Engineering, ICSE 2012, June (2012) 2-9. IEEE Computer Society, Zurich, Switzerland, pp. 815–825. http://dx.doi.org/10.1109/ICSE.2012.6227137.

Pascarella, L., Bacchelli, A., 2017. Classifying code comments in Java open-source software systems. In: Proceedings of the 14th International Conference on Mining Software Repositories, MSR '17. IEEE Press, pp. 227–237. http://dx.doi.org/10.1109/MSR.2017.63.

Pascarella, L., Bruntink, M., Bacchelli, A., 2019. Classifying code comments in Java software systems. Empir. Softw. Eng. 24 (3), 1499–1537. http://dx.doi.org/10.1007/s10664-019-09694-w.

Pascarella, L., Ram, A., Nadeem, A., Bisesser, D., Knyazev, N., Bacchelli, A., 2018. Investigating type declaration mismatches in Python. In: Fontana, F.A., Walter, B., Ampatzoglou, A., Palomba, F. (Eds.), 2018 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation, MaLTeSQuE, SANER 2018, Campobasso, Italy, March 20, 2018. IEEE Computer Society, pp. 43–48. http://dx.doi.org/10.1109/MALTESQUE.2018.8368458.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12, pp. 1–10.

Petticrew, M., Roberts, H., 2008. Systematic Reviews in the Social Sciences: A Practical Guide. John Wiley & Sons.

Pham, T.M.T., Yang, J., 2020. The secret life of commented-out source code. In: ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020. ACM, pp. 308–318. http://dx.doi.org/10.1145/3387904.3389259.

Plösch, R., Dautovic, A., Saft, M., 2014. The value of software documentation quality. In: 2014 14th International Conference on Quality Software, Allen, TX, USA, October 2-3, 2014. IEEE, pp. 333–342. http://dx.doi.org/10.1109/QSIC.2014.22.

Rahman, M.M., Roy, C.K., Keivanloo, I., 2015. Recommending insightful comments for source code using crowdsourced knowledge. In: Godfrey, M.W., Lo, D., Khomh, F. (Eds.), 15th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2015, Bremen, Germany, September 27-28, 2015. IEEE Computer Society, pp. 81–90. http://dx.doi.org/10.1109/SCAM.2015.7335404.

Rama, G.M., Kak, A.C., 2015. Some structural measures of API usability. Softw. Pract. Exp. 45 (1), 75–110. http://dx.doi.org/10.1002/spe.2215.

Rani, P., Panichella, S., Leuenberger, M., Di Sorbo, A., Nierstrasz, O., 2021a. How to identify class comment types? A multi-language approach for class comment classification. J. Syst. Softw. 181, 111047. http://dx.doi.org/10.1016/j.jss.2021.111047, arXiv:2107.04521, http://scg.unibe.ch/archive/papers/Rani21d.pdf.

Rani, P., Panichella, S., Leuenberger, M., Ghafari, M., Nierstrasz, O., 2021b. What do class comments tell us? An investigation of comment evolution and practices in Pharo Smalltalk. Empir. Softw. Eng. 26 (6), 1–49. http://dx.doi.org/10.1007/s10664-021-09981-5, arXiv:2005.11583, http://scg.unibe.ch/archive/papers/Rani21b.pdf.

Ratol, I.K., Robillard, M.P., 2017. Detecting fragile comments. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering. IEEE Press, pp. 112–122.

Robillard, M.P., 2009. [What makes APIs] hard to learn? answers from developers. IEEE Softw. 26 (6), 27–34. http://dx.doi.org/10.1109/MS.2009.193.

Robillard, M.P., Chhetri, Y.B., 2015. [Recommending reference API] documentation. Empir. Softw. Eng. 20 (6), 1558–1586. http://dx.doi.org/10.1007/s10664-014-9323-y.

Scalabrino, S., Bavota, G., Vendome, C., Vásquez, M.L., Poshyvanyk, D., Oliveto, R., 2017. Automatically assessing code understandability: how far are we?. In: Rosu, G., Penta, M.D., Nguyen, T.N. (Eds.), Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017. IEEE Computer Society, pp. 417–427. http://dx.doi.org/10.1109/ASE.2017.8115654.

Scalabrino, S., Linares-Vásquez, M., Oliveto, R., Poshyvanyk, D., 2018. A comprehensive model for code readability. J. Softw.: Evol. Process 30 (6), e1958.

Scalabrino, S., Linares-Vasquez, M., Poshyvanyk, D., Oliveto, R., 2016. Improving code readability models with textual features. In: 2016 IEEE 24th International Conference on Program Comprehension. ICPC, IEEE, pp. 1–10.

Shinyama, Y., Arahori, Y., Gondow, K., 2018. Analyzing code comments to boost program comprehension. In: 2018 25th Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 325–334.

Silva, A., Araújo, T., Nunes, J., Perkusich, M., Dilorenzo, E., de Almeida, H.O., Perkusich, A., 2017. A systematic review on the use of definition of done on agile software development projects. In: Mendes, E., Counsell, S., Petersen, K. (Eds.), Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE 2017, Karlskrona, Sweden, June 15-16, 2017. ACM, pp. 364–373. http://dx.doi.org/10.1145/3084226.3084262.

Sohan, S., Maurer, F., Anslow, C., Robillard, M.P., 2017. A study of the effectiveness of usage examples in REST API documentation. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE, pp. 53–61.

Song, X., Sun, H., Wang, X., Yan, J., 2019. A survey of automatic generation of source code comments: Algorithms and techniques. IEEE Access 7, 111411–111428.

Stapleton, S., Gambhir, Y., LeClair, A., Eberhart, Z., Weimer, W., Leach, K., Huang, Y., 2020. A human study of comprehension and code summarization. In: ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020. ACM, pp. 2–13. http://dx.doi.org/10.1145/3387904.3389258.

Steidl, D., Hummel, B., Juergens, E., 2013. Quality analysis of source code comments. In: Program Comprehension (ICPC), 2013 IEEE 21st International Conference on. IEEE, pp. 83–92.

Sun, X., Geng, Q., Lo, D., Duan, Y., Liu, X., Li, B., 2016. Code comment quality analysis and improvement recommendation: an automated approach. Int. J. Softw. Eng. Knowl. Eng. 26 (06), 981–1000.

Tan, L., Yuan, D., Krishna, G., Zhou, Y., 2007. /* iComment: Bugs or bad comments?*/. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, pp. 145–158.

Tarhan, A., Giray, G., 2017. On the use of ontologies in software process assessment: A systematic literature review. In: Mendes, E., Counsell, S., Petersen, K. (Eds.), Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE 2017, Karlskrona, Sweden, June 15-16, 2017. ACM, pp. 2–11. http://dx.doi.org/10.1145/3084226.3084261.

Tomassetti, F., Torchiano, M., 2014. An empirical assessment of polyglot-ism in GitHub. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–4.

Törngren, M., Sellgren, U., 2018. Complexity Challenges in Development of Cyber-Physical Systems. Springer International Publishing, Cham, pp. 478–503. http://dx.doi.org/10.1007/978-3-319-95246-8_27.

Vincenti, W.G., et al., 1990. What Engineers Know and how They Know It, 141. Johns Hopkins University Press, Baltimore.

Visconti, M., Cook, C.R., 2004. Assessing the state of software documentation practices. In: International Conference on Product Focused Software Process Improvement. Springer, pp. 485–496.

Wang, C., Peng, X., Liu, M., Xing, Z., Bai, X., Xie, B., Wang, T., 2019. A learning-based approach for automatic construction of domain glossary from source code and documentation. In: Dumas, M., Pfahl, D., Apel, S., Russo, A. (Eds.), Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019. ACM, pp. 97–108. http://dx.doi.org/10.1145/3338906.3338963.

Wen, F., Nagy, C., Bavota, G., Lanza, M., 2019. A large-scale empirical study on code-comment inconsistencies. In: Proceedings of the 27th International Conference on Program Comprehension. IEEE Press, pp. 53–64.

Wieringa, R., Maiden, N., Mead, N., Rolland, C., 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. 11 (1), 102–107.

Wu, Y., Manabe, Y., Kanda, T., Germán, D.M., Inoue, K., 2017. Analysis of license inconsistency in large collections of open source projects. Empir. Softw. Eng. 22 (3), 1194–1222. http://dx.doi.org/10.1007/s10664-016-9487-8.

Xi, Y., Shen, L., Gui, Y., Zhao, W., 2019. Migrating deprecated API to documented replacement: Patterns and tool. In: Proceedings of the 11th Asia-Pacific Symposium on Internetware, pp. 1–10.

Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A.E., Li, S., 2018. Measuring program comprehension: a large-scale field study with professionals. IEEE Trans. Softw. Eng. 44 (10), 951–976. http://dx.doi.org/10.1109/TSE.2017.2734091.

Yu, H., Li, B., Wang, P., Jia, D., Wang, Y., 2016. Source code comments quality assessment method based on aggregation of classification algorithms. J. Comput. Appl. 36 (12), 3448–3453.

Zelkowitz, M.V., Wallace, D., 1997. Experimental validation in software engineering. Inf. Softw. Technol. 39 (11), 735–743.

Zhai, J., Xu, X., Shi, Y., Tao, G., Pan, M., Ma, S., Xu, L., Zhang, W., Tan, L., Zhang, X., 2020. [CPC:] automatically classifying and propagating natural language comments via program analysis. In: Rothermel, G., Bae, D. (Eds.), ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020. ACM, pp. 1359–1371. http://dx.doi.org/10.1145/3377811.3380427.

Zhang, J., Xu, L., Li, Y., 2018. Classifying python code comments based on supervised learning. In: Meng, X., Li, R., Wang, K., Niu, B., Wang, X., Zhao, G. (Eds.), Web Information Systems and Applications - 15th International Conference, WISA 2018, Taiyuan, China, September 14-15, 2018, Proceedings. In: 11242 of Lecture Notes in Computer Science, Springer, pp. 39–47. http://dx.doi.org/10.1007/978-3-030-02934-0_4.

Zhi, J., Garousi-Yusifoğlu, V., Sun, B., Garousi, G., Shahnewaz, S., Ruhe, G., 2015. Cost, benefits and quality of software development documentation: A systematic mapping. J. Syst. Softw. 99, 175–198.

Zhong, H., Su, Z., 2013. [Detecting API] documentation errors. In: Hosking, A.L., Eugster, P.T., Lopes, C.V. (Eds.), Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, Part of SPLASH 2013, Indianapolis, in, USA, October 26-31, 2013. pp. 803–816. http://dx.doi.org/10.1145/2509136.2509523.

Zhou, Y., Gu, R., Chen, T., Huang, Z., Panichella, S., Gall, H., 2017. Analyzing APIs documentation and code to detect directive defects. In: Proceedings of the 39th International Conference on Software Engineering. IEEE Press, pp. 27–37.

Zhou, Y., Wang, C., Yan, X., Chen, T., Panichella, S., Gall, H.C., 2020. Automatic detection and repair recommendation of directive defects in Java API documentation. IEEE Trans. Softw. Eng. 46 (9), 1004–1023. http://dx.doi.org/10.1109/TSE.2018.2872971.

Zhou, Y., Yan, X., Yang, W., Chen, T., Huang, Z., 2019. [Augmenting Java] method comments generation with context information based on neural networks. J. Syst. Softw. 156, 328–340. http://dx.doi.org/10.1016/j.jss.2019.07.087.

Pooja Rani is a Postdoctoral researcher at the University of Bern (Switzerland). Her focus areas involve conducting empirical studies, developing methodology, and building tools to support developers in understanding code. Specifically, she studies code comments from various software systems and builds tools to improve the quality of comments. She finished her Ph.D. at the University of Bern in 2022 and masters at the Birla Institute of Technology and Science-Pilani (India) in 2017.

Arianna Blasi is a Ph.D. candidate at the Università della Svizzera italiana (Switzerland). Her research focuses on software testing. Her work is about automatically deriving test oracles from natural-language artifacts, such as code comments. She authored papers both about test oracles and code comments quality and served as a reviewer for journals and conferences in the SE community. Interested both in research and industry, she interned at Facebook to work on real-world software testing problems.

Nataliia Stulova is a Postdoctoral researcher at the University of Bern (Switzerland). Her research focuses on software documentation tools and techniques to keep software and its specifications aligned. Papers she (co-)authored have appeared in various international conferences and journals in the areas of software engineering, requirements engineering, and software verification. She has served as a reviewer for several international journals and conferences in the fields of software engineering and logic programming.

Sebastiano Panichella is a Computer Science Researcher at Zurich University of Applied Science (ZHAW). His main research goal is to conduct industrial research, involving both industrial and academic collaborations, to sustain the Internet of Things (IoT) vision, where future smart cities. Currently he is technical coordinator of H2020 and Innosuisse projects concerning DevOps for Complex Cyber-physical Systems. He authored (or co-authored) around seventhly papers appeared in International Conferences and Journals. He serves and has served as program committee member of various international conference and as reviewer for various international journals in the fields of software engineering. He was selected (results reported by the JSS journal) in 2019 as one of the top-20 (second in Switzerland) Most Active Early Stage Researchers Worldwide in SE, while in 2021 as one of the top-20 Most impactful SE researchers Worldwide.

Alessandra Gorla is an assistant professor at the IMDEA Software Institute, Spain. She completed her Ph.D. in informatics at the Università' della Svizzera Italiana in Lugano, Switzerland in 2011. In her Ph.D. thesis she defined and developed the notion of Automatic Workarounds, a self-healing technique to recover Web applications from field failures, a work for which she received the Fritz Kutter Award for the best industry related Ph.D. thesis in computer science in Switzerland. Before joining the IMDEA Software Institute, she was a postdoctoral researcher in the software engineering group at Saarland University in Germany. During her postdoc, she has also been a visiting researcher at Google in Mountain View. Alessandra is one of the recipients of the 2019 Facebook Testing and Verification research award. In 2020 she received the Emilio Aced award in privacy protection, given by the Agencia Española Protección de Datos for her research on Android applications, and she received the Ramon y Cajal fellowship in 2021, which is the top national grant for researchers in Spain.

Oscar Nierstrasz is Professor of Computer Science at the Institute of Computer Science (INF) in the Faculty of Science of the University of Bern, where he founded the Software Composition Group in 1994. He is co-author of over 300 publications and co-author of the open-source books Object-Oriented Reengineering Patterns and Pharo by Example.