UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Produce a Literature Review

Seminar in Green Software Engineering

Dr. Pooja Rani
rani@ifi.uzh.ch, pooja.rani@uni-mannheim.de
University of Zurich, Switzerland
University of Mannheim, Germany

Systematic literature review

Systematic literature review

- Individual studies contributing to a systematic review are called primary studies
- Literature review is a form of secondary study
- A systematic review of systematic reviews is called Tertiary study
- Aims at identifying, evaluating, and interpreting all available research about a particular research question, topic area, or phenomenon of interest

Common reasons

- Summarize the existing evidence
- Identify any gaps in current research to suggest areas for further investigation
- Provide a framework/background to appropriately position new research activities
- Support/contradict hypotheses
- Assist the generation of new hypotheses

- The well-defined methodology makes it less likely that results are biased
- Provides information about effects of some phenomenon across a wide range of empirical studies concerning the same research questions
- Requires considerably more effort than traditional reviews

Systematic literature reviews vs. literature surveys

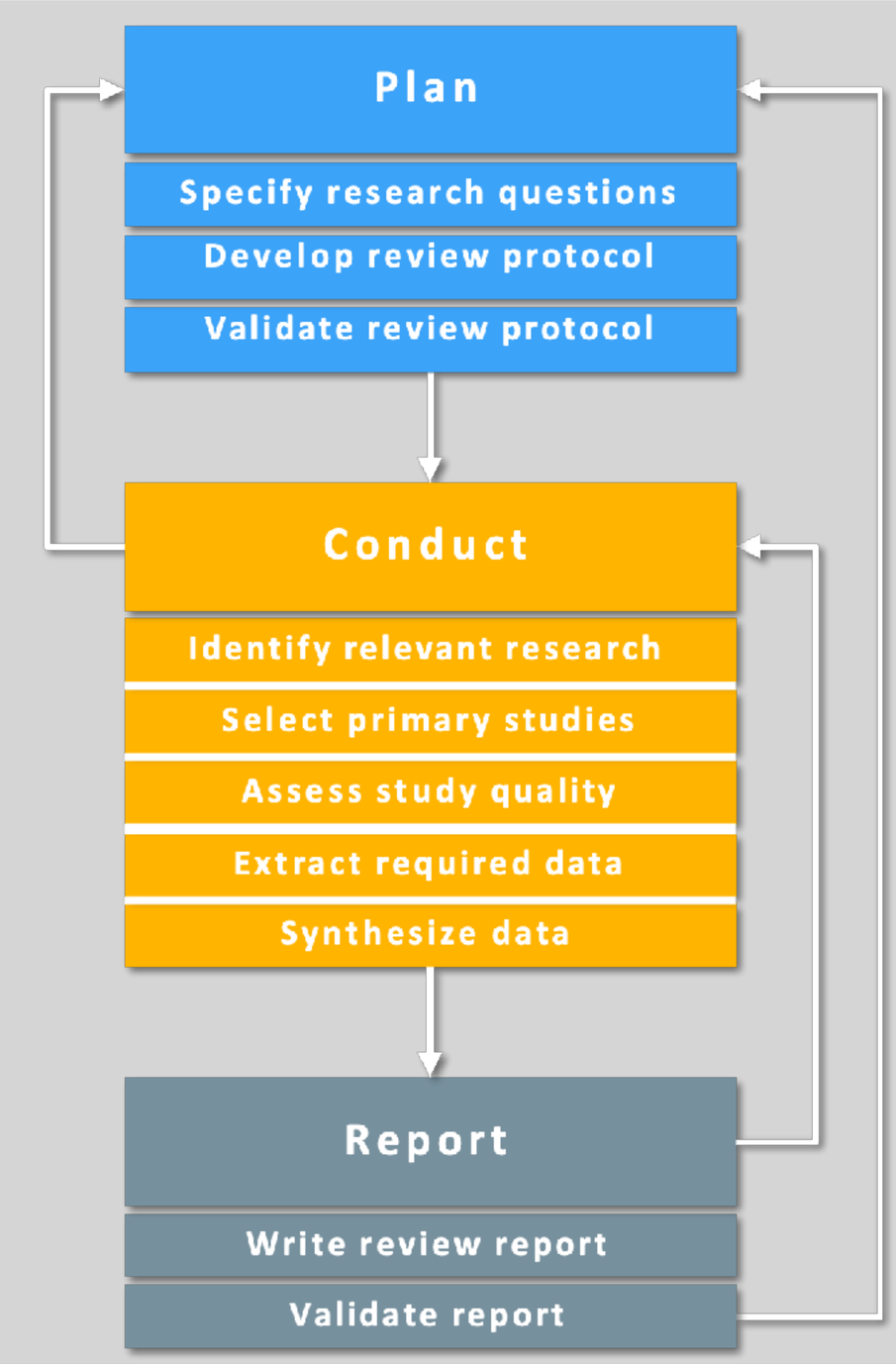
- With a survey or non-systematic literature review, we analyze, summarize, organize, and present novel conclusions from primary studies
- When systematic, the literature review technically and critically review primary studies to extract technical and scientific metadata from primary studies
- A systematic review requires a protocol, research questions, and methods
- Systematic reviews are replicable, comprehensive, objective

Systematic mapping study

- Systematic mapping studies are a wide overview of a research area
- The goal of systematic mapping is the identification of the quantity and type fo research
- A systematic review has the focus on establishing the state of empirical evidence
- The systematic reviews evaluate primary studies regarding their quality

Review process

- Plan: Define the research goals, questions, and a review protocol
- Conduct: Retrieve primary studies, select the studies, assess the quality, and extract meta-data
- Report: Create the final document



Reference example

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


Plan

Identify the goal

- We aim at identifying a main goal for the whole review

- It usually gives the title to the review

ELSEVIER

Information and Software Technology

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

Intelligent software engineering in the context of agile software development: A systematic literature review

Mirko Perkusich*, Lenardo Chaves e Silva, Alexandre Costa, Felipe Ramos, Renata Saraiva, Arthur Freire, Ednaldo Dilorenzo, Emanuel Dantas, Danilo Santos, Kyller Gorgônio, Hyggo Almeida, Angelo Perkusich

Federal University of Campina Grande, Campina Grande, PB, Brazil

ARTICLE INFO

Keywords:
Intelligent software engineering
Agile software development
Search-based software engineering
Machine learning
Bayesian networks
Artificial intelligence

ABSTRACT

CONTEXT: Intelligent Software Engineering (ISE) refers to the application of intelligent techniques to software engineering. We define an "intelligent technique" as a technique that explores data (from digital artifacts or domain experts) for knowledge discovery, reasoning, learning, planning, natural language processing, perception or supporting decision-making.

OBJECTIVE: The purpose of this study is to synthesize and analyze the state of the art of the field of applying intelligent techniques to Agile Software Development (ASD). Furthermore, we assess its maturity and identify adoption risks.

METHOD: Using a systematic literature review, we identified 104 primary studies, resulting in 93 unique studies.
RESULTS: We identified that there is a positive trend in the number of studies applying intelligent techniques to ASD. Also, we determined that reasoning under uncertainty (mainly, Bayesian network), search-based solutions, and machine learning are the most popular intelligent techniques in the context of ASD. In terms of purposes, the

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

Specify the research questions

- Once the goal has been defined, we have to think about the research questions
- The research questions will drive the whole review methodology
- The search process must identify primary studies that address the questions
- The data extraction process must extract data items needed to answer them
- The data analysis process must synthesize the data in such a way that the questions can be answered

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

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

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

Develop the review protocol

- We describe the plan and rules that will guide the process of reviewing

- It involves primary studies, data sources, and people

Search method

- A description of the methods involved for the search activity
- Database search
- Backward snowballing: starting from a primary study we retrieve related papers between reference
- Forward snowballing: we look at other studies that cite a target primary study (Google Scholar and Scopus)

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.

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.

Search terms

- The collection of keywords derived from the research questions

- The search string using boolean ANDs and ORs to query the digital sources

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

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.

Data sources

- The digital databases to retrieve primary studies
- Scientific literature, including journals and conference proceedings
- Grey literature, e.g., technical reports, work in progress, presentations
- The internet
- Other secondary studies

selected for the study (shown in Section 2.2.2) returned ten known relevant papers. The results are shown in Table 3. Only one paper was not returned, but we noticed that it was a limitation of the target databases (i.e., none of them indexed articles from the given journal). As shown in Section 3, the given paper was found after the first snowballing iteration. Finding the paper during the snowballing search is an evidence that it adds value to the research by minimizing the probability of missing relevant papers.

2.2.2. Data sources

After having a valid search string, we conducted the database search (**Step 3**) in the following digital libraries:

- ACM Digital Library
- Engineering Village
- ISI Web of Science
- Science Direct
- Scopus
- Springer

2.3. Selection criteria

To screen and clean data (**Step 4**), we defined a generic exclusion criteria:

1. A duplicate OR
2. Published in a non-peer reviewed channel (e.g., thesis) OR
3. Published in a book OR
4. Unavailable in English or Portuguese OR
5. Published before 2001.

Afterwards (**Step 5**), the remaining papers were evaluated through the following inclusion criteria:

- presents an intelligent technique applied to ASD *AND* is a primary

single researcher executed the screening and cleaning of data.

2.3.1. Selection method

To select the papers, we used a two-stage approach. **First**, we evaluated the papers by applying the adaptive reading approach [58], which included at least two reviewers. Later, the full-text of the papers were read by two reviewers (data extractor and data checker) [59,60]. They extracted the data from the papers and excluded them based on the same criteria used in the previous step and the quality criteria. We present details of the quality assessment and data extraction process in Sections 2.4 and 2.5, respectively.

The snowballing approach (**Step 6**) was performed with the papers identified during the database search as the seed set. For each paper in the seed set, we applied the backward and forward snowballing. Backward and forward snowballing refers to systematically analyzing the references and citations, respectively, of a given set of papers.

For the forward snowballing, we used Google Scholar and Scopus as the data sources. We applied the same approach as in **Steps 4** and **5** to select the papers and extract their data. For the backward snowballing, the papers were initially evaluated by only one reviewer. The reviewer was responsible for applying the generic exclusion criteria defined in **Step 4**, by evaluating the reference list for each paper, and, if necessary, the place of reference in the text. Later, for the included papers, we applied the same approach as in **Step 5** to select the papers and extract their data. After identifying that there were no more candidate papers, we analyzed the data regarding the Research Questions (RQs) (**Step 7**).

2.4. Quality assessment

To assess the empirical quality of the papers, we followed the criteria established by Dybå and Dingøyr [61], which are presented as follows. To evaluate each of the criteria listed above, we used a boolean scale, in which a score of “1” means “yes” and “0”, “no”.

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.

Selection criteria

- The queries can return many primary studies

- The selection criteria determine which studies are included in, or excluded from

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.

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.

Selection method

- How the selection criteria will be applied by the authors

- How the disagreements will be resolved, if need be

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.

Quality assessment

- Quality checklist to assess the individual studies

- For each of the papers, we check whether some characteristics are valid

Quality assessment

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.

Quality assessment

2.2.6. Data evaluation

We work in step 10 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.

Data extraction strategy

- How the information from each primary study will be obtained
- The description of the data collected
- Standard fields are always applied, such as type of article, name of the publication channel, year of publication
- It can also include some specific classification schema

Data extraction strategy

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 extraction strategy

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.

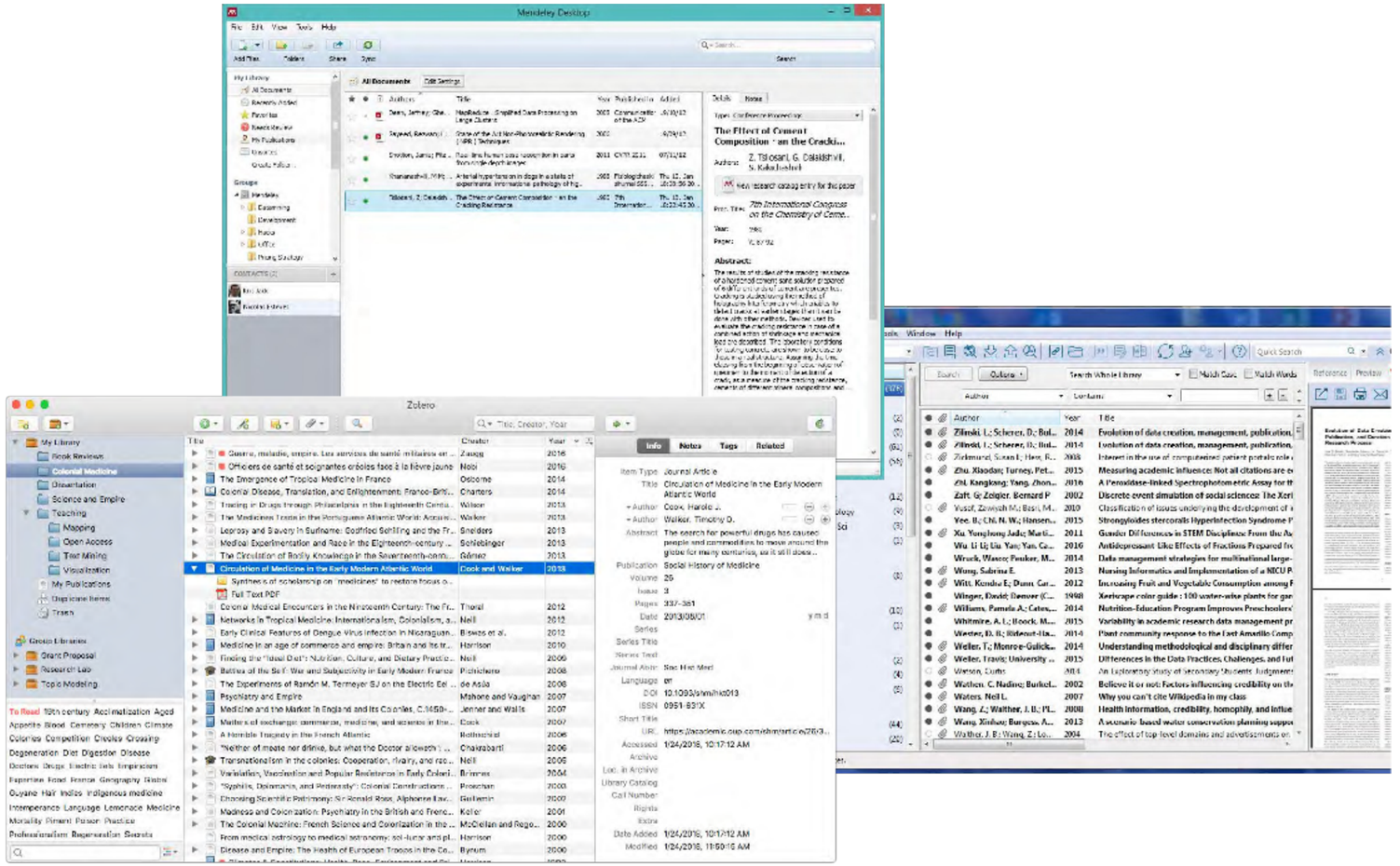

Conduct

Preliminary searches

- Help to identify existing literature reviews
- Asses the volume of potentially relevant studies
- Refine some of the steps in the review protocol

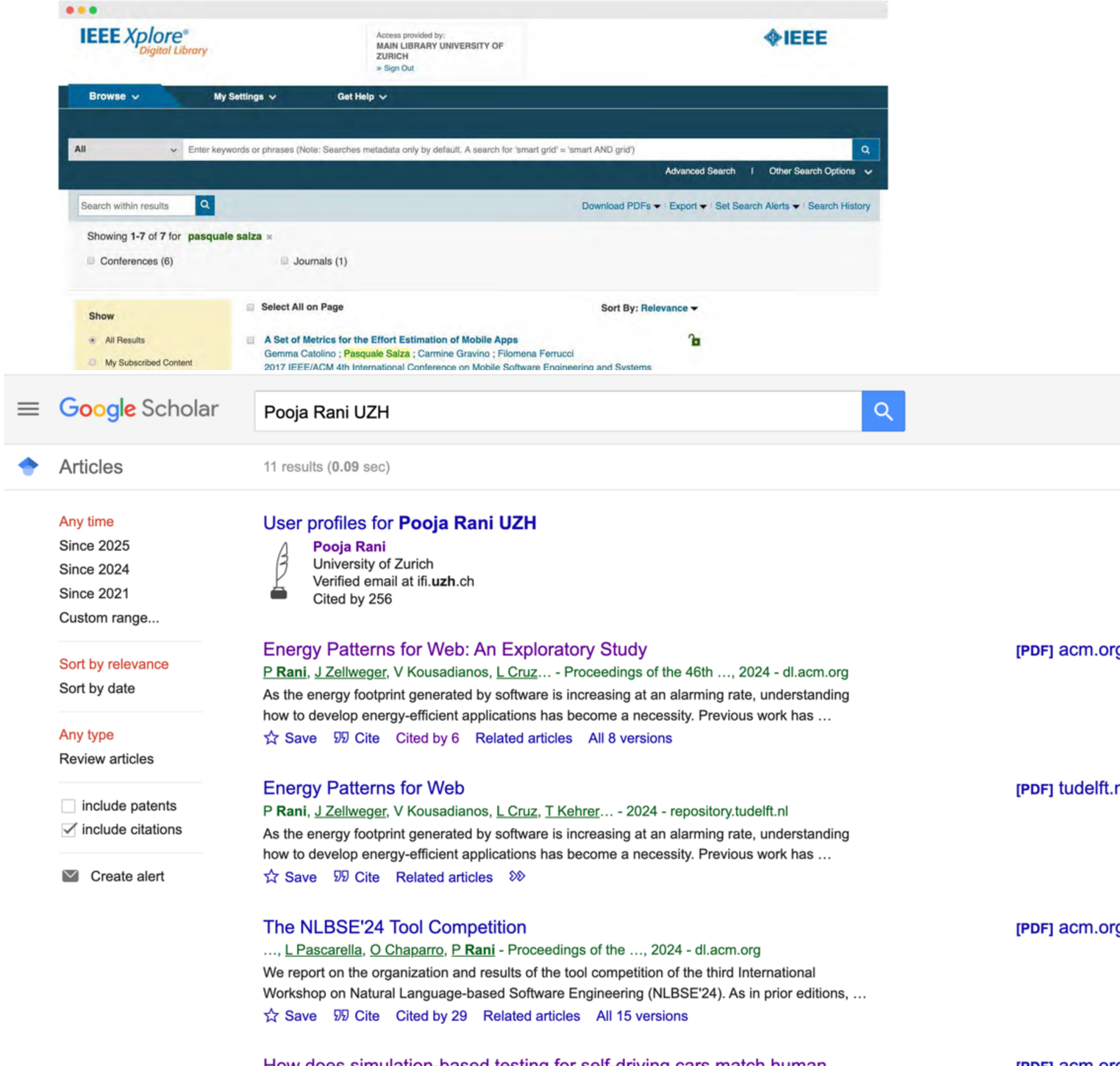
Reference managers

- Useful to store and put notes on retrieved papers
- Zotero
- Mendeley
- Endnote
- + JabRef



Digital Libraries

- Google Scholar
- DBLP
- ACM Digital Library
- IEEE Explore
- Citeseer
- Elsevier (Science Direct)



Study selection

- We first retrieve the full list of papers and their copies
- We should not include multiple publications of the same data (extensions) and get the most recent ones
- We filter part of them according to the titles
- We investigate about the pertinence using the abstracts
- Obtaining the final list might require to look at the contents

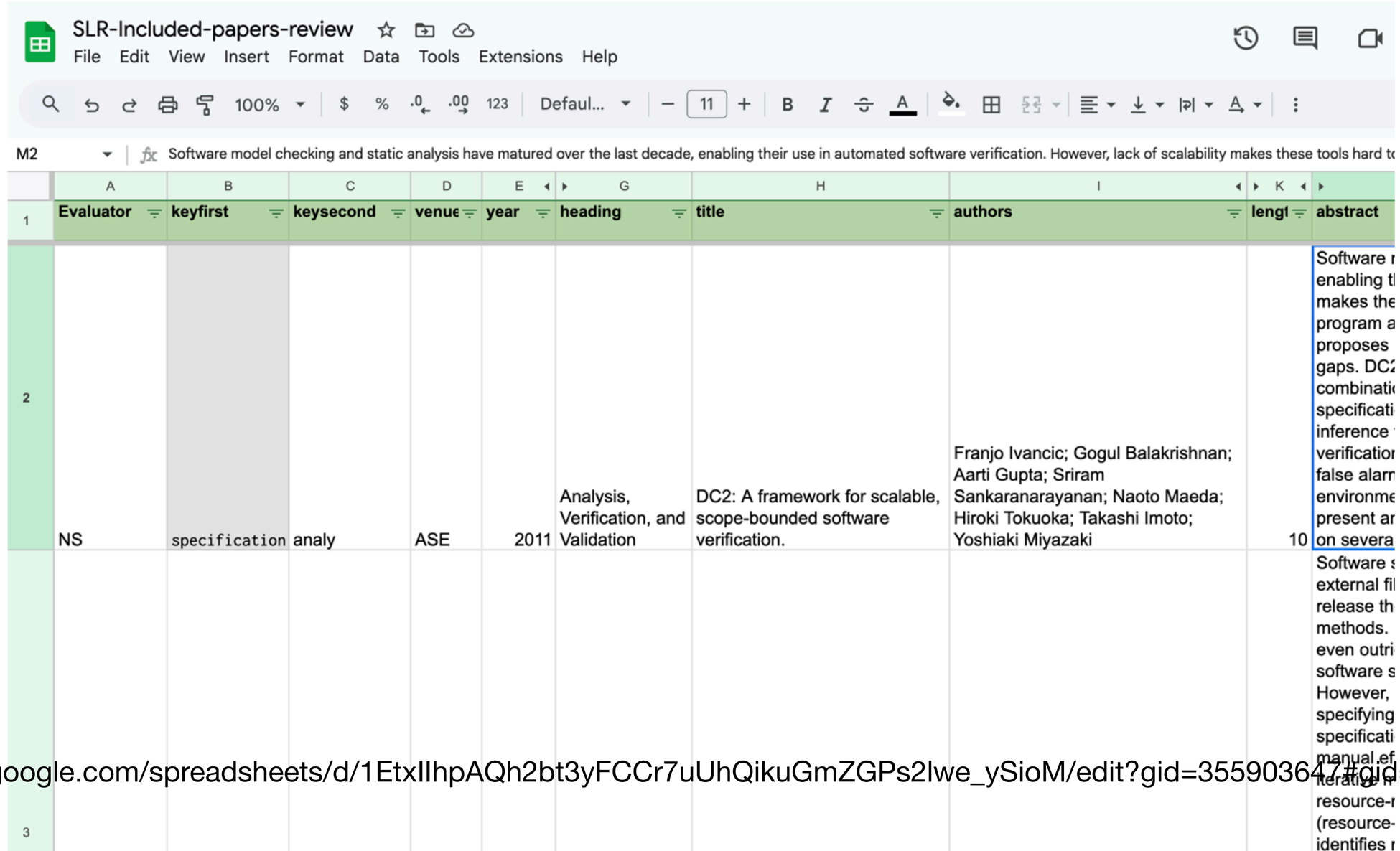
Documenting the search and selection

- The process has to be transparent and replicable (as much as possible)
- Every paper resulting from the search should be stored
- Also, the excluded studies should be saved and retained for possible reanalysis

Data extraction

- We extract relevant data from each of the selected primary studies
- A spreadsheet can be useful
- It should be done independently from two or more researchers
- Disagreements can be resolved either by consensus or arbitration by an additional independent researcher

Data extraction



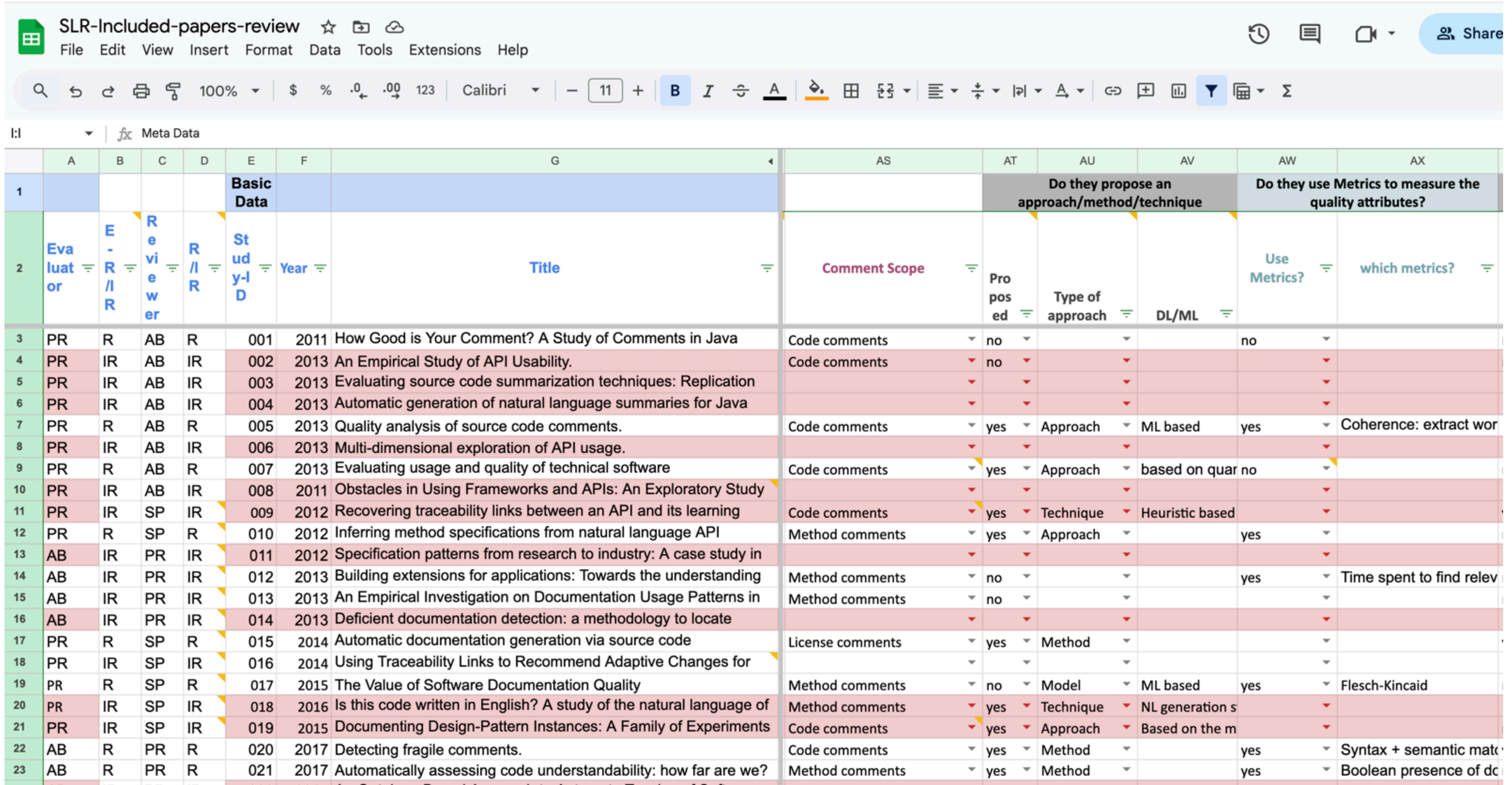
	A	B	C	D	E	G	H	I	K	
1	**Evaluator**	**keyfirst**	**keysecond**	**venue**	**year**	**heading**	**title**	**authors**	**lengt**	**abstract**
2	NS	specification	analy	ASE	2011	Analysis, Verification, and Validation	DC2: A framework for scalable, scope-bounded software verification.	Franjo Ivancic; Gogul Balakrishnan; Aarti Gupta; Sriram Sankaranarayanan; Naoto Maeda; Hiroki Tokuoka; Takashi Imoto; Yoshiaki Miyazaki	10	Software r enabling tl makes the program a proposes gaps. DC2 combinati specificati inference verificatio false alarr environme present ar on severa
3										Software s external fil release th methods. even outri software s However, specifying specificati manual ef iterative w resource-r (resource- identifies r

https://docs.google.com/spreadsheets/d/1EtxIIhpAQh2bt3yFCCr7uUhQikuGmZGPs2lwe_ySioM/edit?gid=355903647#gid=355903647

Data synthesis

- We collate and summarize the results of the included primary studies
- The synthesis can be descriptive (narrative) of the studies, useful for discussion
- It can be quantitative, based on the aggregation of quantitive data

Data synthesis



SLR-Included-papers-review

File Edit View Insert Format Data Tools Extensions Help

Share

100% $ % .0 .00 123 Calibri 11 B I

I:I fx Meta Data

	A	B	C	D	E	F	G	AS	AT	AU	AV	AW	AX
1					Basic Data				Do they propose an approach/method/technique			Do they use Metrics to measure the quality attributes?	
2	Evaluator	E-R/IR	Reviewer	R/IR	Study-ID	Year	Title	Comment Scope	Proposed	Type of approach	DL/ML	Use Metrics?	which metrics?
3	PR	R	AB	R	001	2011	How Good is Your Comment? A Study of Comments in Java	Code comments	no			no	
4	PR	IR	AB	IR	002	2013	An Empirical Study of API Usability.	Code comments	no				
5	PR	IR	AB	IR	003	2013	Evaluating source code summarization techniques: Replication						
6	PR	IR	AB	IR	004	2013	Automatic generation of natural language summaries for Java						
7	PR	R	AB	R	005	2013	Quality analysis of source code comments.	Code comments	yes	Approach	ML based	yes	Coherence: extract wor
8	PR	IR	AB	IR	006	2013	Multi-dimensional exploration of API usage.						
9	PR	R	AB	R	007	2013	Evaluating usage and quality of technical software	Code comments	yes	Approach	based on quar	no	
10	PR	IR	AB	IR	008	2011	Obstacles in Using Frameworks and APIs: An Exploratory Study						
11	PR	IR	SP	IR	009	2012	Recovering traceability links between an API and its learning	Code comments	yes	Technique	Heuristic based		
12	PR	R	SP	R	010	2012	Inferring method specifications from natural language API	Method comments	yes	Approach		yes	
13	AB	IR	PR	IR	011	2012	Specification patterns from research to industry: A case study in						
14	AB	IR	PR	IR	012	2013	Building extensions for applications: Towards the understanding	Method comments	no			yes	Time spent to find relev
15	AB	IR	PR	IR	013	2013	An Empirical Investigation on Documentation Usage Patterns in	Method comments	no				
16	AB	IR	PR	IR	014	2013	Deficient documentation detection: a methodology to locate						
17	PR	R	SP	R	015	2014	Automatic documentation generation via source code	License comments	yes	Method			
18	PR	IR	SP	IR	016	2014	Using Traceability Links to Recommend Adaptive Changes for						
19	PR	R	SP	R	017	2015	The Value of Software Documentation Quality	Method comments	no	Model	ML based	yes	Flesch-Kincaid
20	PR	IR	SP	IR	018	2016	Is this code written in English? A study of the natural language of	Method comments	yes	Technique	NL generation s		
21	PR	IR	SP	IR	019	2015	Documenting Design-Pattern Instances: A Family of Experiments	Code comments	yes	Approach	Based on the m		
22	AB	R	PR	R	020	2017	Detecting fragile comments.	Code comments	yes	Method		yes	Syntax + semantic mat
23	AB	R	PR	R	021	2017	Automatically assessing code understandability: how far are we?	Method comments	yes	Method		yes	Boolean presence of dc


Report

Write the review into a paper

- Once the review has been completed, it’s time to report everything into a paper
- The report should include everything needed to replicate the review
- Thus, it should include the results as well as the adopted process to retrieve them
- Papers have usually page limits

Title

- The title should be short but informative
- It needs to be based on the question being asked
- In the case of general purpose journals, it should indicate that the study is a systematic review

Abstract

- Some journals require systematic reviews to have a structured abstract
- Context: the importance of the research questions
- Objectives: the questions
- Methods: data sources, study selection, quality assessment, and data extraction
- Results: main findings
- Conclusions: implications for practice and future research

Alessandra Gorla[a], 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.

Introduction and background

- The introduction should set the context of the report
- The research questions have to be presented and justified
- It can include some background, useful to follow the rest of the paper

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).

Review questions

- Lists all the research questions with separate identifiers, e.g., RQ1
- When possible, we define subquestions to simplify the understandability of the paper
- Every research questions should be then motivated

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

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

Review methods

- Presents the review protocol
- It is structured into subsections according to the components of the review protocol
- It should also report any change to the original protocol

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.

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.

Included and excluded studies

- List all the final set of included primary studies

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)

Included and excluded studies

- List all the final set of included primary studies
- If the paper space allows, it should also includes the list of the excluded papers
- A diagram can show the process of selection

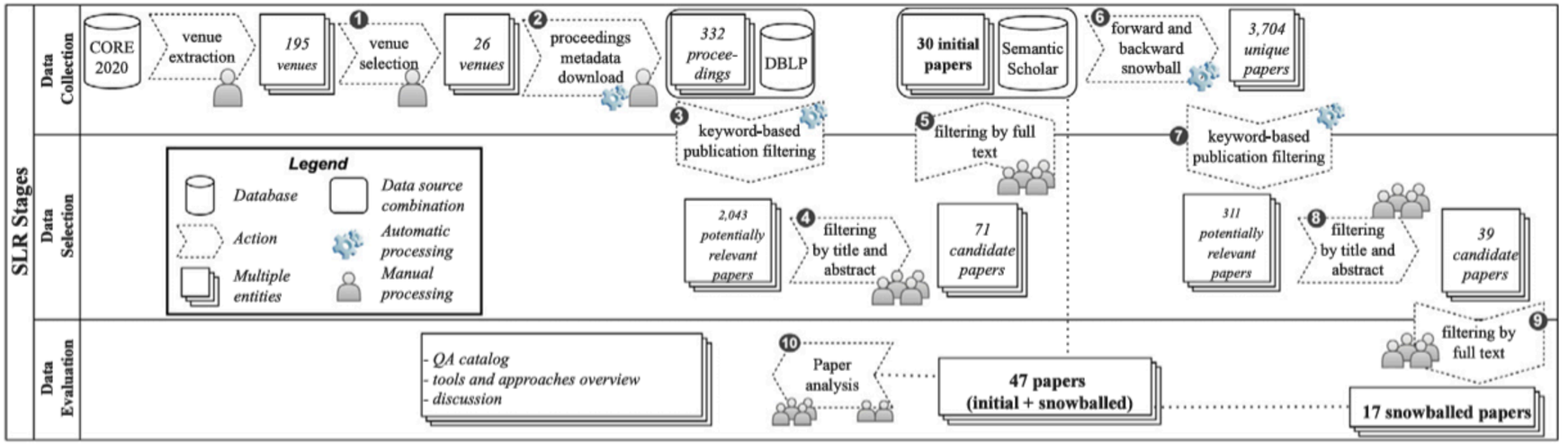


Fig. 1. SLR stages to collect relevant papers.

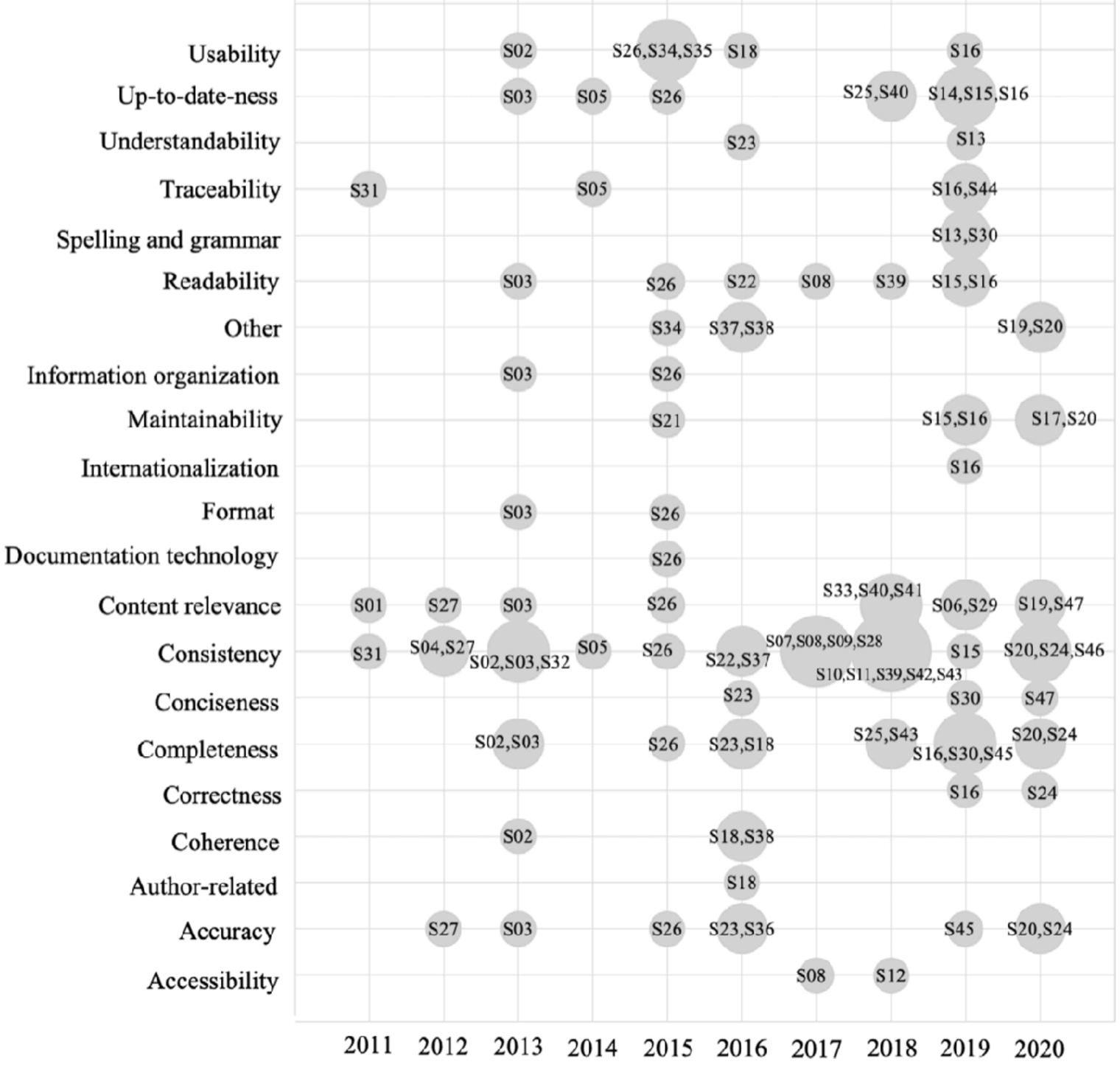
Results

- Non-quantitative studies should be provided to summarize each of the studies and presented in textual form

- Quantitative summary results should be presented in tables and graphs

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.


Usability
Up-to-date-ness
Understandability
Traceability
Spelling and grammar
Readability
Other
Information organization
Maintainability
Internationalization
Format
Documentation technology
Content relevance
Consistency
Conciseness
Completeness
Correctness
Coherence
Author-related
Accuracy
Accessibility
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
S02
S26,S34,S35
S18
S16
S03
S05
S26
S25,S40
S14,S15,S16
S23
S13
S31
S05
S16,S44
S13,S30
S03
S26
S22
S08
S39
S15,S16
S34
S37,S38
S19,S20
S03
S26
S21
S15,S16
S17,S20
S16
S03
S26
S26
S33,S40,S41
S01
S27
S03
S26
S06,S29
S19,S47
S31
S04,S27
S02,S03,S32
S05
S26
S22,S37
S07,S08,S09,S28
S10,S11,S39,S42,S43
S15
S20,S24,S46
S23
S30
S47
S02,S03
S26
S23,S18
S25,S43
S16,S30,S45
S20,S24
S16
S24
S02
S18,S38
S18
S27
S03
S26
S23,S36
S45
S20,S24
S08
S12

Discussion

- This section can be used to discuss the findings of the review
- Highlights strengths and weaknesses of the evidence included in the review
- Discusses the applicability of the findings

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

Related 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

Conclusions

- Shows the practical implications of the review for the research community
- Highlights some unanswered questions and implications for future work

8. Conclusion

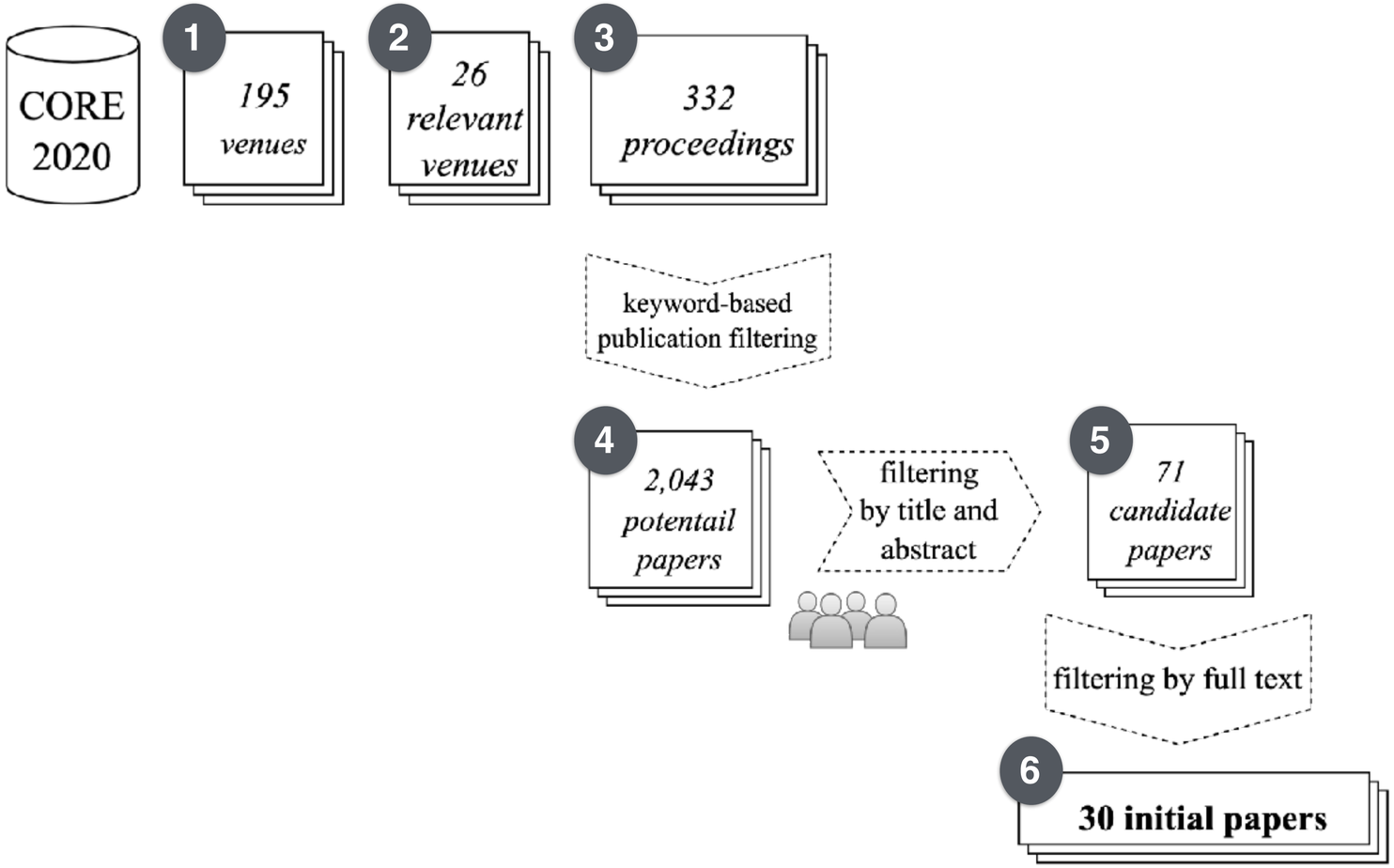
In this work, we present the results of a systematic literature review on source code comment quality evaluation practices in the decade 2011– 2020. We analyze 2353 publications and study 47 of them to understand of effort of Software Engineering researchers, in terms of what type of comments they focus 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

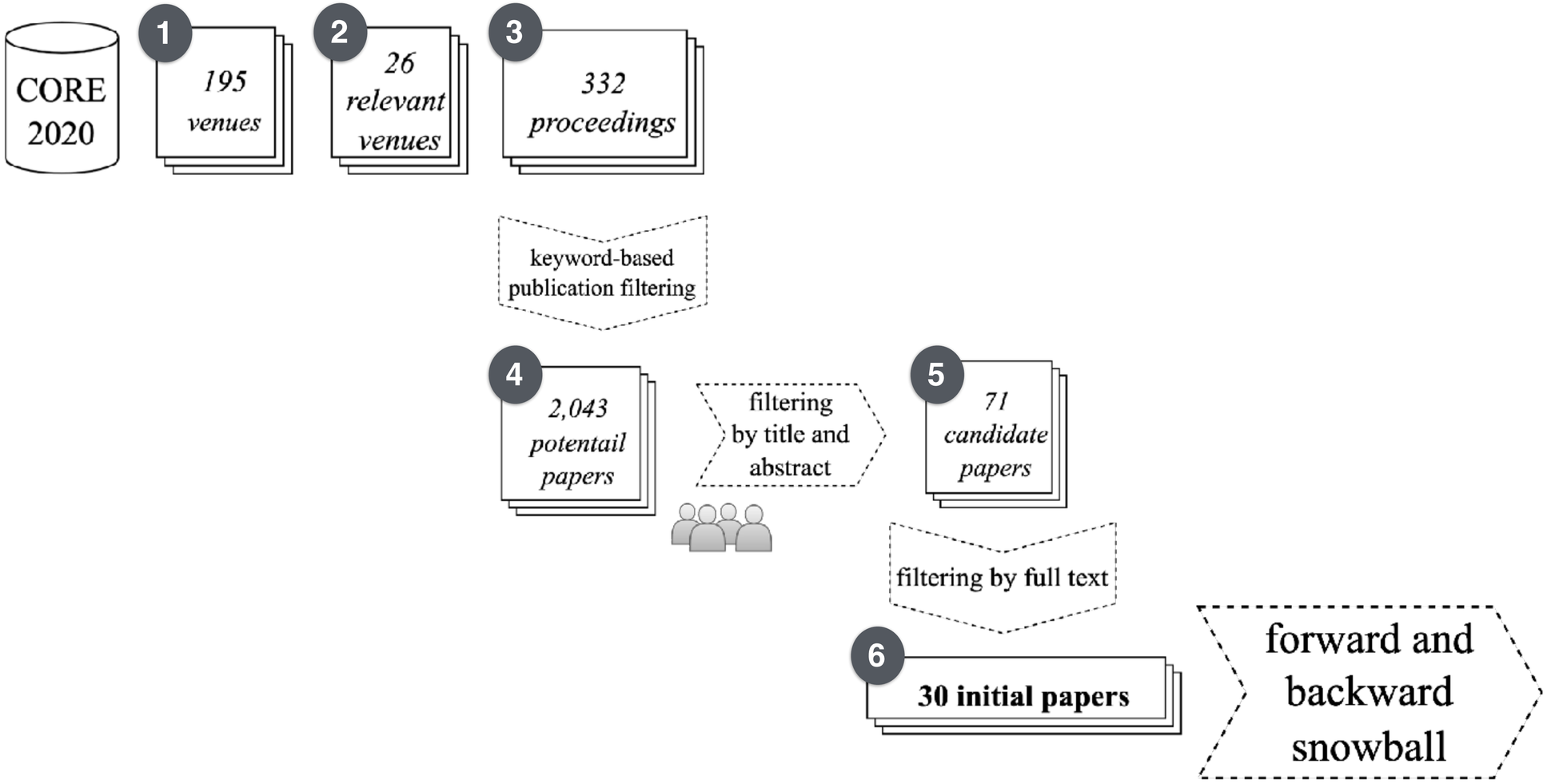
Threats to Validity

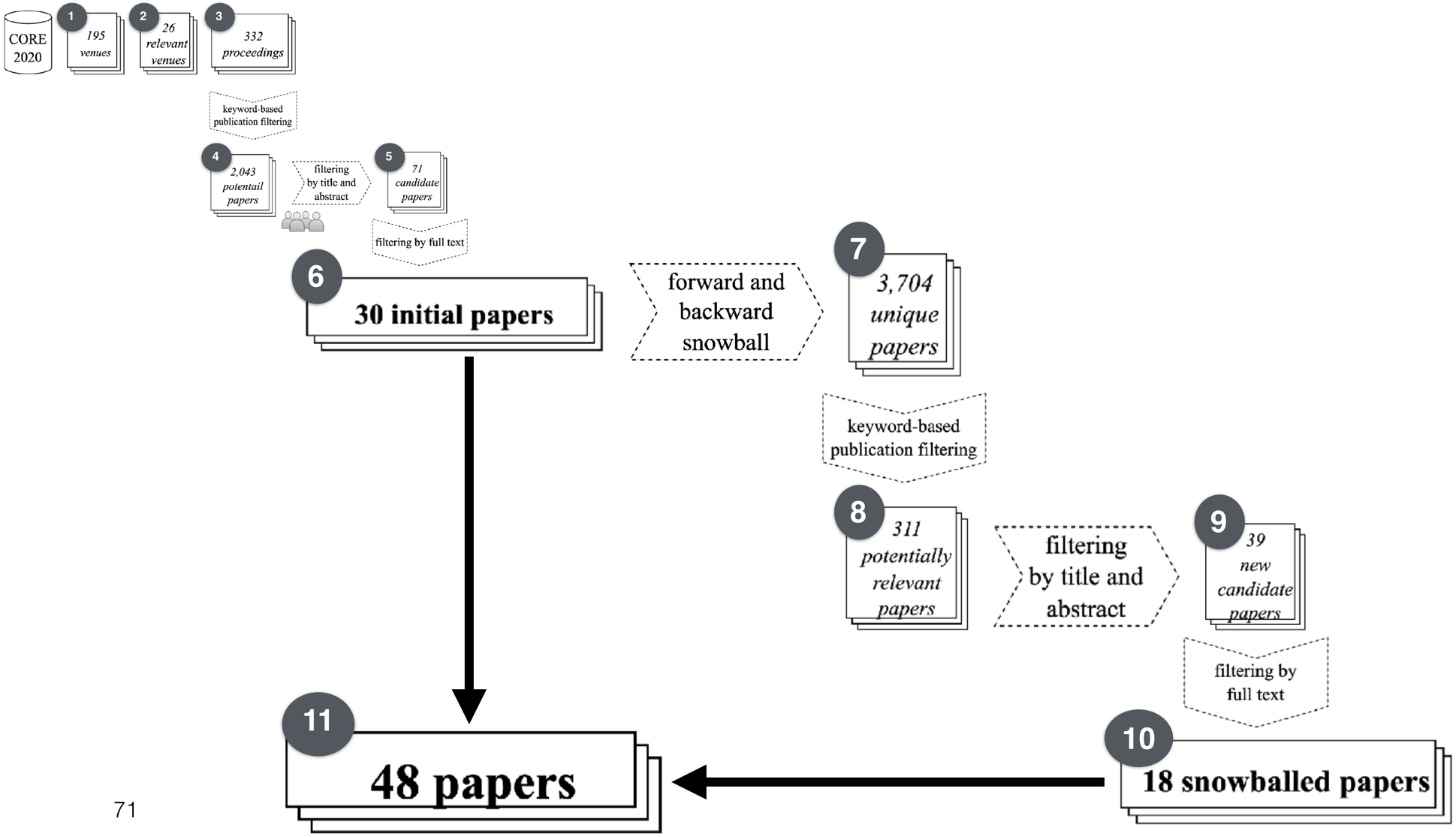
6. Threats to validity

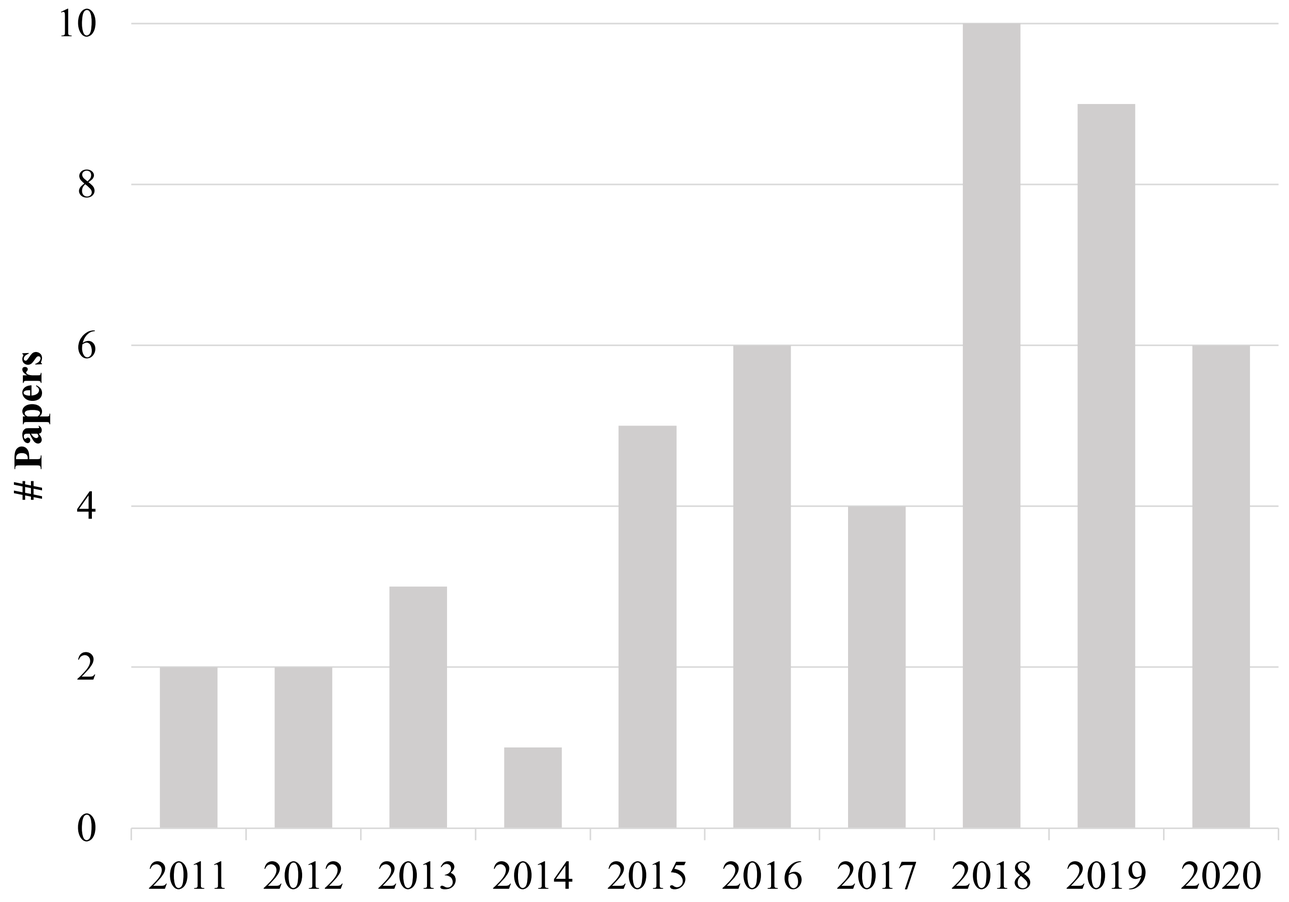
We now outline potential threats to the validity of our study. *Threats to construct validity* mainly concern the measurements used in the evaluation process. In this case, threats can be mainly due to (i) the imprecision in the automated selection of relevant papers (*i.e.*, the three-step search on 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


CORE 2020
1
195 venues
2
26 relevant venues
3
332 proceedings
keyword-based publication filtering
4
2,043 potentail papers
filtering by title and abstract
5
71 candidate papers
filtering by full text
6
30 initial papers


CORE 2020
1
195 venues
2
26 relevant venues
3
332 proceedings
keyword-based publication filtering
4
2,043 potentail papers
filtering by title and abstract
5
71 candidate papers
filtering by full text
6
30 initial papers
forward and backward snowball


CORE 2020
1
195 venues
2
26 relevant venues
3
332 proceedings
keyword-based publication filtering
4
2,043 potentail papers
filtering by title and abstract
5
71 candidate papers
filtering by full text
6
30 initial papers
forward and backward snowball
7
3,704 unique papers
keyword-based publication filtering
8
311 potentially relevant papers
filtering by title and abstract
9
39 new candidate papers
filtering by full text
10
18 snowballed papers
11
48 papers



48 papers over years

Analyzed dimensions

Comment types

method comments, class comments

Analyzed dimensions

Comment types

method comments, class comments

Quality attributes

consistency, completeness

Analyzed dimensions

Comment types

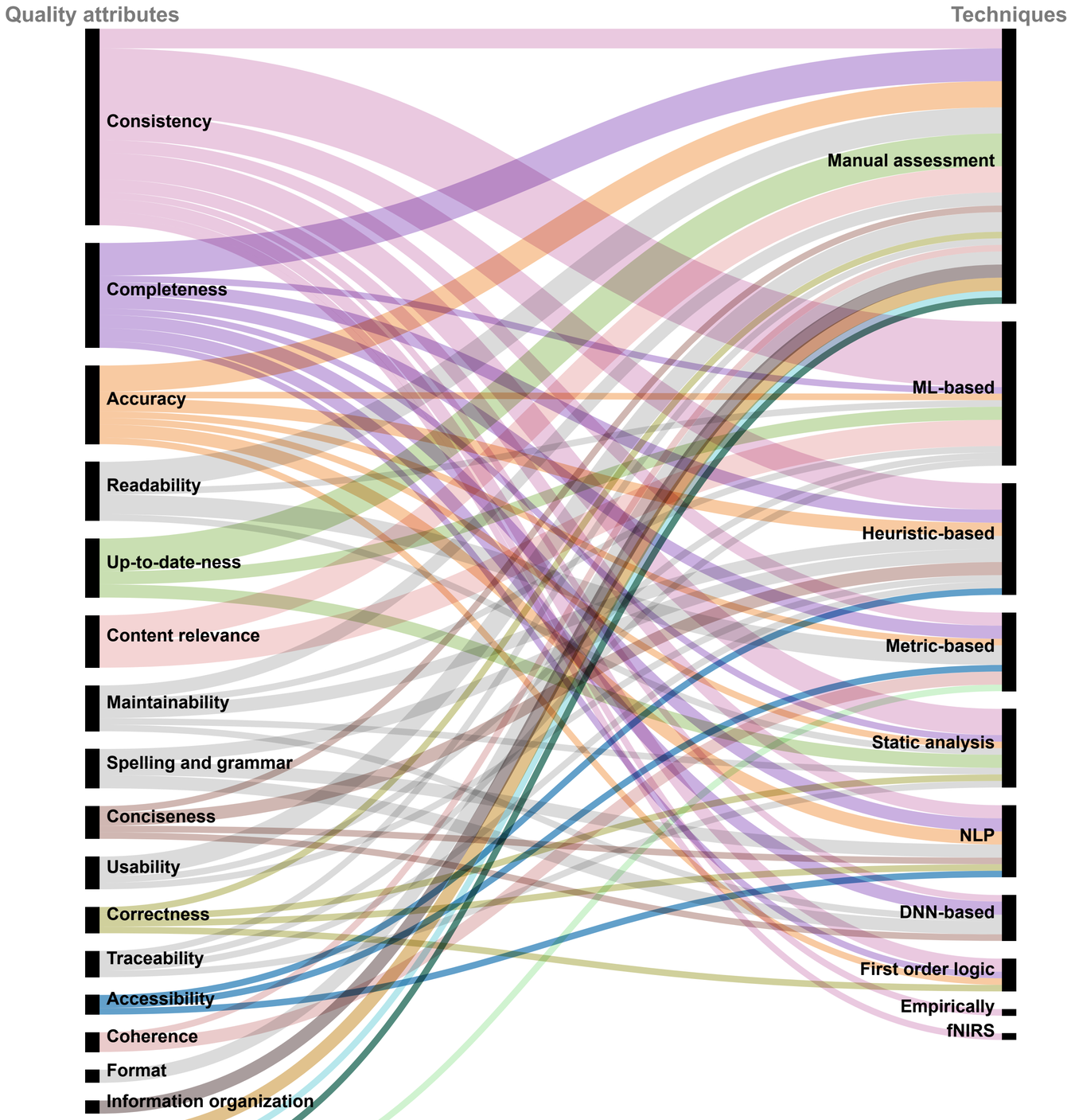
method comments, class comments

Quality attributes

consistency, completeness

Techniques

heuristic-based, machine learning-based


Quality attributes
Techniques
Consistency
Completeness
Accuracy
Readability
Up-to-date-ness
Content relevance
Maintainability
Spelling and grammar
Conciseness
Usability
Correctness
Traceability
Accessibility
Coherence
Format
Information organization
Manual assessment
ML-based
Heuristic-based
Metric-based
Static analysis
NLP
DNN-based
First order logic
Empirically
fNIRS

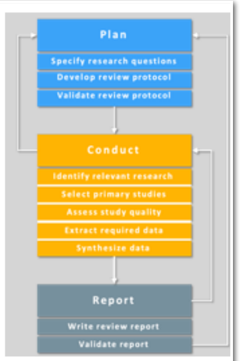
Submit your research questions.

Systematic literature review

- Individual studies contributing to a systematic review are called primary studies
- Literature review is a form of secondary study
- A systematic review of systematic reviews is called Tertiary study
- Aims at identifying, evaluating, and interpreting all available research about a particular research question, topic area, or phenomenon of interest

Review process

- Plan: Define the research goals, questions, and a review protocol
- Conduct: Retrieve primary studies, select the studies, assess the quality, and extract meta-data
- Report: Create the final document



Search method

- A description of the methods involved for the search activity
- Database search
- Backward snowballing: starting from a primary study we retrieve related papers between reference
- Forward snowballing: we look at other studies that cite a target primary study (Google Scholar and Scopus)

Data synthesis

- We collect and summarize the results of the included primary studies
- The synthesis can be descriptive (narrative) of the studies, useful for discussion
- It can be quantitative, based on the aggregation of quantitive data

Write the review into a paper

- Once the review has been completed, it's time to report everything into a paper
- The report should include everything needed to replicate the review
- Thus, it should include the results as well as the adopted process to retrieve them
- Papers have usually page limits

Dr. Pooja Rani
rani@ifi.uzh.ch, pooja.rani@uni-mannheim.de
University of Zurich, Switzerland
University of Mannheim, Germany