IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 14, 2021, accepted February 15, 2022, date of publication February 21, 2022, date of current version March 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3152397

Technologies for GQM-Based Metrics Recommender Systems: A Systematic Literature Review

MIRKO FARINA[1,2], ANNA GORB[2], ARTEM KRUGLOV[2], AND GIANCARLO SUCCI[2], (Member, IEEE)
[1]Human Machine Interaction Laboratory, Faculty of Humanities and Social Sciences, Innopolis University, 420500 Innopolis, Russia
[2]Faculty of Computer Science and Engineering, Innopolis University, 420500 Innopolis, Russia

Corresponding author: Giancarlo Succi (g.succi@innopolis.ru)

This work was supported by the Russian Science Foundation under Grant 19-19-00623.

ABSTRACT **Purpose:** With this Systematic Literature Review (SLR), we aim to discover technologies to construct a Goal-Question-Metrics (GQM) based metrics recommender for software developers. Since such a system has not yet been described in the literature, we decided to analyse the technologies used in three main components of recommender systems - data sets, algorithms, and recommendations - independently. **Methods:** To achieve our goal we performed - following the best norms in our discipline - a systematic literature review (SLR). We first identified, through searches aptly performed, 422 potentially relevant papers, from which we selected - after applying inclusion and exclusion criteria - 30 papers, which we eventually included in our final log. **Results:** Systems with textual data set preprocess information in nearly the same way and the majority use similarity scores to create recommendations. Systems with GQM-based algorithms consist of questionnaires and require users to explicitly answer questions to produce suggestions. With respect to the recommendations of reviewed systems, they range from application programming interfaces (APIs) to requirements, but no system presently recommends metrics. **Conclusion:** In our SLR we: (a) identified a sequence of the most popular steps for preprocessing in recommender systems; (b) proposed an optimisation strategy for such steps; (c) found out that the most promising approach includes both ranking and classification; and (d) established that there are no recommendation systems developed to date to process metrics.

INDEX TERMS Goal question metrics model, recommender systems, text processing.

I. INTRODUCTION

Given the software's immaterial nature, researchers attempted to study its characteristics and influencing factors using software metrics, that is, numeric values attached to particular attributes of artifacts related to the software development process [1], [2]. The selection of software metrics, though, is not a simple task [3]. Properly selected metrics can indicate deficiencies in the process in the early stages of development [4], and vice versa, inappropriate metrics may interfere with the achievement of the project's goals [5] or cause developers to waste precious time [6, p. 630]. Consequently, a number of structured approaches to derive metrics have been proposed [7]–[10]. Among them one of the most relevant is the Goal-Question-Metrics (GQM), developed by Basili *et al.* in the '80s [11], [12]. In recent decades this approach has gained substantial popularity [6, p. 45]; however, the proper formulation of GQM is not a trivial task and requires substantial expertise. The purpose of this SLR is to investigate and find out which technologies can be used to develop a recommendation algorithm for software engineers, so as to automatically suggest appropriate metrics for GQM models.

Such an algorithm can be integrated into "metrics collectors" - applications that collect products and process metrics. Usually such systems have many metrics [13] from which users can select useful ones. Manual selection is costly and time consuming; moreover, users often cannot determine by themselves, which metrics they need. To solve these problems dashboards with predefined set of metrics have been proposed [14]. However, such dashboards cannot cope with the specifics of each case [15], and this is where recommender systems can be used [16].

The associate editor coordinating the review of this manuscript and approving it for publication was Wuliang Yin.

Recommender systems became popular in the 90s [17]. Since then, different types of recommender systems have emerged, which can be classified as follows:

1) **Collaborative filtering** [18] is the most widely used type of recommender systems. It is based on similarities that can be calculated in different ways:
   - user-based [19]: construct recommendations based on ratings of similar users;
   - item-based [20]: calculate similarity levels between already rated items and the targeted one.

   The advantage of this type of filtering is its suitability for cases where one cannot create a detailed description for an object (e.g. recommending music or movies). However, this type of filtering also has some weaknesses, namely: cold-start, data-sparsity, scalability, and synonymy [21].
2) **Content-based filtering** [22], unlike collaborative filtering, generates suggestions based on profiles and descriptions. The main advantage of this approach is that it overcomes the cold-start problem. In other words, even if the user did not interact with the system before, the system will still be able to generate accurate suggestions. However, this type of filtering has a potential disadvantage: the system requires very detailed information about each item in question and the user profiles in order to make recommendations [23].
3) **Hybrid filtering** [24], the combination of different recommender techniques to overcome the limitations of the two filtering methods described above.

There are many types of recommender systems with hybrid filtering used for various purposes [25], but in conducting this SLR we found that they have not yet been used to automatically generate metrics based on the GQM model. This suggests that our review is the first in this specific area. Though, we should also note that a consistent part of our work is devoted to surveying recommender systems and that several SLRs covered this issue before us [26], [27]. The first SLR [26] we have encountered focused on issues related to I/O in recommender systems in software engineering, while the second one [27] focused on the distribution of recommender systems over the development phases, which is beyond the scope of our current research.

In section II, we introduce the review protocol in this SLR, specifying our methodology. In section III, we coherently and systematically present the results of our work, while in section IV we discuss and analyse them critically. Section V points out a series of limitations and threats to validity potentially affecting our research, while Section VI, describes the significance of our findings for the field. Finally, Section VII summarises what we have achieved in our work and discusses potential future developments and new research directions.

II. REVIEW PROTOCOL

In this section, we describe the "Review Protocol" adopted for this SLR. We describe it upfront here to show that the work we conducted is unbiased, rigorous, systematic, and reproducible. To plan this SLR we used as a guideline the work done by Kitchenham and Charters [28], thus identifying three basic steps for our SLR (see Table 1).

TABLE 1. Stages of the SLR.

Planning the review	Specification of research questions Identification of resources and search strategy Inclusion and exclusion criteria Quality assessment
Conducting the review	Selection of studies Data extraction
Reporting the review	Results, analysis, and shortcomings

A. PRISMA 2020 CHECKLIST

Having specified the steps required for our SLR, we then followed the standard protocol for performing SLRs adopted by researchers internationally (the PRISMA Checklist 2020 [29]). Our checklist is available for review in the appendix section (see Appendix A).

B. RESEARCH QUESTIONS

To set our research towards a specific direction we first figured out the goal of our SLR based on the following concepts and points:

Purpose: generate useful metrics
Viewpoint: software developers
Issue: using GQM model
Object: recommendation algorithm

This translates to the following research question, which are the bases of our SLR.

RQ$_1$: Are there recommender systems for software developers whose data set includes textual information?
RQ$_2$: Which algorithms underlie recommender systems for software developers?
RQ$_3$: Are there recommender systems capable of suggesting metrics for software developers?

There are three main components in any recommender system: data set, algorithm, and recommendations [30]. GQM-based metrics recommenders should be based on a GQM approach, which means that goals and questions will constitute a data set and metrics will represent the recommendations. Since goals and questions are textual information, with RQ$_1$ we will identify recommender systems that also receive textual information as input and will understand how they preprocess it. Then, with RQ$_2$ we will find which algorithms are used by recommender systems to create suggestions. Answering this question will allow us to determine how to construct the central part of our system - the recommendation algorithm. Finally, with RQ$_3$ we will find out which types of suggestions recommender systems for software developers make and whether metrics are present among them.

It is of paramount importance to answer this set of questions because with the information obtained, it will be possible to understand the advantages and disadvantages related to each approach. Furthermore, this information will allow us to select the most appropriate technologies to create GQM-based metrics recommenders for software developers. Since we have not found any systematic works that cover these important topic, we believe there is strong merit in this research.

C. SEARCH STRATEGY

A crucial step in the development of any SLR is the formulation of a search strategy. Given the amount of papers potentially available on the topic, we decided to use automatic database search (filtering) in this work. Firstly, we selected a list of databases for our searches. The databases we used include: *Scopus, IEEE Xplore,* and *ACM Digital Library*. We did not use other renowned databases for our searches, for instance, *Science Direct*, because it has a restriction on the number of Boolean operators that one can list.

We subsequently formulated a number of generally relevant keywords that could be representatives of our research questions. These are as follows: *recommender systems* and *software developers*. The keywords selected were sufficiently general to ensure that we would not miss any important paper. However, we realized they might have been too general to guarantee the precision needed for our work, so we created synonyms for each of them (see Table 2).

TABLE 2. Synonyms for keywords.

Keyword	Synonyms
software developers	engineering, programming, computer science
recommender algorithms	system, platform, engine, suggestion

By using this improved list of keywords, we subsequently generated a series of Search Queries. We used Boolean operators to do so, in accordance with the best practices of our discipline. After that we searched the databases originally selected with these Search Queries. Below we report our preliminary results :

- **IEEE Xplore** - (39 papers initially)
  (“Document Title”:software OR “Document Title” :programm* OR “Document Title”:“computer scien*”) AND ((“Document Title”:recommend* OR “Document Title”:suggest*) NEAR/1 (“Document Title”: system OR “Document Title”:platform OR “Document Title”:engine OR “Document Title”:algorithm))
- **Scopus** - (120 papers initially)
  TITLE (software OR programm* OR “computer scien*” AND (recommend* OR suggest* PRE/0 system OR platform OR engine OR algorithm))
- **ACM Digital Library** - (328 papers initially)
  Title: (software OR programm* OR ”computer scien*”) AND (recommend* OR suggest*) AND (system OR platform OR engine OR algorithm)

We gathered a total of 487 potentially relevant papers for our SLR. All these papers were added to our reading log for further processing. We then excluded 65 duplicates, reducing the number of papers potentially relevant for inclusion in the study to 422.

D. INCLUSION AND EXCLUSION CRITERIA

We formulated Exclusion Criteria (EC) and Inclusion Criteria (IC). EC and IC help deciding, in a systematic and coherent way, which - among all the papers selected as potentially relevant (in our case 422 papers) - should be included into the final reading log [31]. To be included, a paper needed to satisfy all the following Inclusion Criteria.

IC_1: the work is written in English AND
IC_2: the work helps solving a software development problem AND
IC_3: the work contains an algorithm for the construction of a recommender system AND
IC_4: the work describes recommender system with textual data set, or suggests software metrics

However, it was decided that even if a work met all the ICs above, we would exclude it, if it satisfied one of the following ECs.

EC_1: the work did not satisfy one or more of the inclusion criteria stated above OR
EC_2: the work is similar to others produced later by the same authors OR
EC_3: the work has no evaluation phase OR
EC_4: the work is a conference proceeding OR
EC_5: the work can be classified as grey literature (e.g., technical report or dissertation)

Having specified our IC and EC, we then consistently applied them to the set of 422 papers previously selected as potentially relevant for our SLR, which have then been filtered as outlined in Fig. 1.

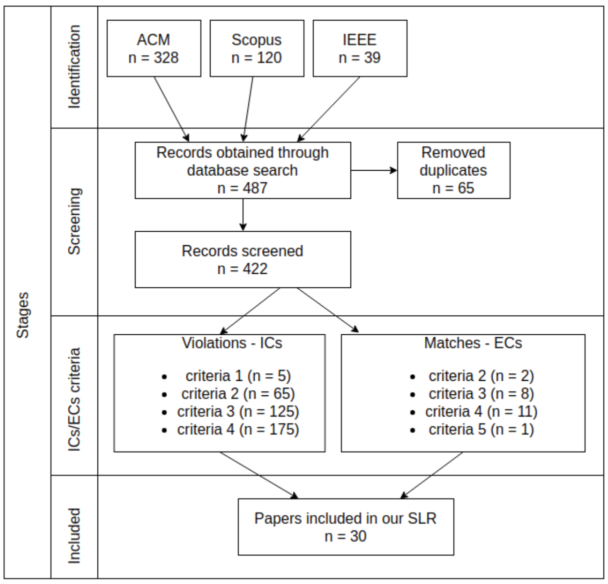


FIGURE 1. Prisma Flow Chart Diagram.

E. QUALITY ASSESSMENT

Establishing an object metric to determine the quality of the papers selected for inclusion is an important step in the development of a systematic literature review, as it ensures that the findings obtained through it are reliable and academically sound. We prepared a list of questions with relative scores (0, 0,5,1), which could offer a reliable indication about the quality of the reviewed paper.

The list of questions we used for our quality assessment may be found in the appendix section together with the detailed results of the quality assessment, which was conducted by two people independently to maximise objectivity and minimize bias (see Appendix).

There were indeed many high quality papers among those we selected for inclusion in our final log. This is demonstrated by the scores reported in Table 3 below. The average quality score was 6.38 out of 8, which indeed confirms the reliability of the findings on which we based our SLR.

TABLE 3. Papers distribution by quality scores.

Quality score	4.5	5	5.5	6	6.5	7	7.5	8
Quantity	3	2	5	7	1	2	4	6

III. RESULTS

In this section we coherently and systematically present the results of our investigation. We first provide a critical assessment of the databases we used for this study. We then carefully review all the papers included in our final log and classify them in meaningful ways that help us extracting important information or relevant data for our SLR.

A. PRELIMINARY CLUSTERING

Table 4 displays the main advantages and disadvantages of the databases chosen to carry out our search. From the information provided in this table, the reader can conclude that all the databases we used are trustworthy and academically sound.

TABLE 4. Databases - advantages and disadvantages.

Database	Advantages	Disadvantages
IEEExplore Digital Library	peer-reviewed papers	require an individual or institutional subscription
Scopus	paper verified by experts	require an individual or institutional subscription
ACM Digital Library	peer-reviewed papers	access to some materials only through a individual or institutional subscription

Figure 2 shows the distribution of papers by selected databases, and Figure 3 illustrates their distribution by publishers. These figures demonstrate that the papers we included in our final log are either from well-known computer science conferences or have been published in top-ranked journals.

Figure 4 illustrates the distribution of papers by year of publication. For convenience, we also offer to our reader a visual representation of the distribution of articles by period of 5 years (see Table 5).

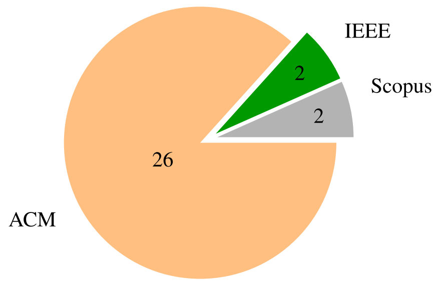


FIGURE 2. Papers distribution by database.

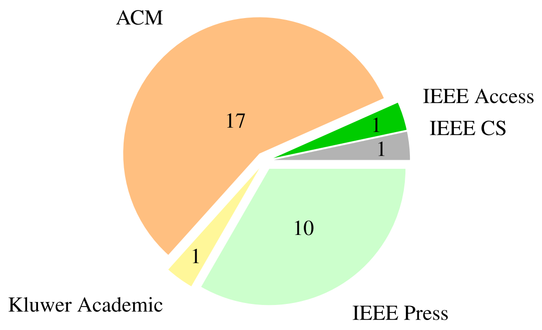


FIGURE 3. Papers distribution by publishers.

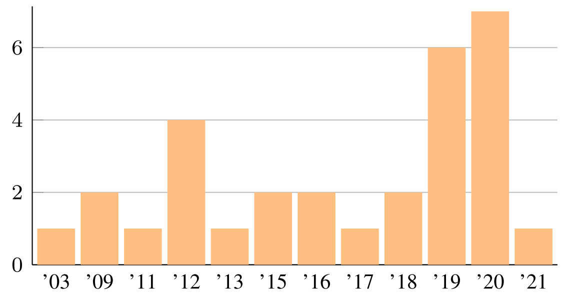


FIGURE 4. Papers distribution by year of publication.

TABLE 5. Papers distribution over 5 year periods.

Year	2002-2006	2007-2011	2012-2016	2017-2021
Quantity	1	3	9	17

We noticed that most of the papers included in our log were written in the last five years, 9 of them were written between 2012 and 2016, and only 4 of them were written earlier (between 2002 and 2011). This can probably be justified with the observation that the field is relatively new and researchers have only recently started investigating it.

B. SUMMARY

The recommender system developed by Azeem *et al.* [32] classified pull requests in three major categories: rejection, response, and acceptance. Its data set included such features as title, body and review comments. The preprocessing phase for this textual information encompassed stop words deletion and stemming and the resulted text was processed by a Skip-gram model. Together these steps prepared the foundations for the XGboost classification algorithm, which eventually classified pull requests.

Xie *et al.* [33] constructed the data set for their algorithm from APIs functionality descriptions. The authors concluded that such descriptions can be assigned to one of 87 functionality categories and 523 phrase patterns they individuated. In particular, they applied Part-of-Speech tagging and lemmatization with SpaCy and Bidirectional Encoder Representations from Transformers to determine under which category a description would fall. After that the query was compared with all the sentences from the identified category by functionality category similarity, semantic role similarity, and Word2Vec based text similarity.

Castro-Herrera *et al.* [34] proposed a recommender system to suggest forums of interest to stakeholders in large-scale projects. The idea was to collect stakeholders' needs in text, preprocess them using stemming, delete stop words, apply Term Frequency - Inverse Term Frequency (TF-IDF) method, and then cluster them in an optimal number of clusters, using adopted consensus clustering algorithms [35]. After those preliminary operations, collaborative filtering with a k-Nearest Neighbors (kNN) algorithm was used to suggest new forums for the stakeholder.

McMillan *et al.* [36] created a data set from the source code with textual features descriptions to facilitate the reuse of source code packages in software prototypes. In this research, the features descriptions were clustered with an incremental diffusive clustering algorithm and feature naming approach [37]. Then they were mapped to related source code modules and processed by kNN algorithm with cosine similarity. Finally, three indicators were optimized: features coverage, number of projects, and external coupling.

Zhang *et al.* [38] used tuples of questions and tags (like in Stack Overflow) as a data set to automatically recommend appropriate tags for new questions. These tuples were passed through several stages: word embedding, multi-tasking like CNN, and gradient descent algorithm. The data obtained was used to recommend top-k tags for the new question.

The recommender system developed by Hamza and Walker [39] applied several heuristics, such as actor-action-object tree construction, to suggest features for Software Product Lines (SPLs) based on the manually extracted functional requirements. Its preprocessing stage included: PoS tagging to these sections, stop words removal, sorting, and deletion of duplicates.

Gomez *et al.* [40] retrieved reviews of mobile apps from Google Play Store, filter stop words, and then applied a Latent Dirichlet Allocation algorithm to cluster all reviews into 100 topics and find out error-suspicious apps. Using J48 decision tree algorithm and correlation between error-sensitive permissions and error-related reviews, their recommender system predicted if there were any bugs in the app.

The artifacts recommender system developed by Cubranić and Murphy [41] indexed textual query and turned them into a document vector. To define similarity between artifacts the system used vector-space cosine similarity.

Ashok *et al.* [42], proposed a recommender system that assisted the process of bug fixing. Initially, the system calculated custom similarity score and then converted different information sources, including those containing text, into typed documents. Then it transformed the documents obtained into a factor graph, while creating a ranked list of relevant bug descriptions. This list could be used to retrieve recommendations about workers, source files, and functions that could be used to fix a bug.

The recommender system developed by Gao *et al.* [43] received as input a technical question without answer. Using a three-step algorithm, the system then generated possible answers for this question.

In developing their recommender system, Jiang *et al.* [44] encountered the problem of automatic features detection from apps descriptions. To partition the text into sentences they firstly used LingPipe and removed all irrelevant symbols. Then, the authors used the Stanford PoS tagger to remove the sentences that did not meet specific criteria. Finally, in using the Naïve Bayes classifier application, which helps to determine whether the sentence describes a given feature, they extracted names and formed a feature vector. Subsequently, the authors applied Latent Dirichlet Allocation to find most similar apps and suggest features to be implemented.

To solve the problem of insufficient information while fixing bugs, the following algorithm was proposed by Rejaul [45]: title/summary of the report is tokenized, stop words are removed, everything is converted to initial forms and then TF-IDF is applied. Then cosine similarity is used to find out the most similar bug reports to the new one, from which the key features is then extracted.

To recommend appropriate APIs according to the natural language query, the system proposed by Cat *et al.* [46] extracted from Stack Overflow the top-50 questions similar to the given one. The recommender then extracted the content of certain Stack Overflow posts, tokenized sentences with NLTK, reduced words to the base forms, transformed them into word embedding with Gensim and then applied the inverse document frequency. Word embedding in combination with IDF values helped to define similarity. Subsequently, APIs entities were extracted from the resulted text and the query was computed.

Palomba *et al.* [47] classified user reviews for mobile apps in 4 predetermined groups using ARdoc [48]. The preprocessing of the input included misspellings checks, contractions expansion, PoS tagging, removing all but verbs and nouns, and tokenization. The second step involved: singularization, stop words removal, stemming, repetitions and short tokens (less then 3 elements) removal. The third step required clusterization with Hierarchical Dirichlet Process. This was done to group together similar requests. Finally, using An asymmetric dice similarity coefficient, they are linked with the code that need to be changed.

The recommender system developed by Kamel *et al.* [49] suggested tasks for crowdsourcing developers. For this purpose, the recommender system extracted features from tasks descriptions and developers profiles. Keywords were then allocated from the input information and DBpedia ontology assigned classes to them. Then features were transformed into bag-of-words and classified using an SVM algorithm.

Di Sipio *et al.* [50] introduced a recommender system that used the READMEs and the source code from GitHub repositories to classify them by topics. Each README went through the stop words removal, stemming, lemmatization and TF-IDF vectorisation. Then they were all classified with a Multinomial Naive Bayesian Network.

The system developed by Wang *et al.* [51] used test reports and requirements to suggest the appropriate set of crowd workers for the given task. Input parameters went through word segmentation, stop words removal, synonym replacement, and vectorization. Then the "descriptive terms list" was constructed and for each document a "task terms vector" was compiled, based on the presence -in the document- of the term from the "descriptive terms list". From this and some other data 26 features were extracted by the learning-based ranking algorithm - LambdaMART.

Cerezo *et al.* [52] proposed a chatbot that could find the mentor for any developer. To categorize user messages into several predefined categories, all sentences were weighted using raw term frequency and subsequently classified, according to the highest weight. To identify the key-concept in the next step IDF was then used and then the algorithm recommended relevant experts.

In the approach suggested by Almhana *et al.* [53] to recommend relevant classes for bug report, its description was passed through tokenization, stop words deletion,and stemming. Next, to compare it with the source code and API descriptions, cosine similarity with weights from TF-IDF was applied.

Another one recommender system by Thung *et al.* [54] suggested methods from an API library, which could be used to accomplish feature requests. For that goal, textual description of a feature request went through tokenization and stemming. Then TF-IDF was applied to generate weights for the next step - cosine similarity with various API methods.

Lin *et al.* [55] proposed a recommender system to suggest architectural refactorings. It compared the target and source models using lexical similarity and design conformance. The description of the model and program elements was transformed using tokenization, stop words removing, stemming, encoding into TF-IDF vector, and cosine similarity. Using the resulted lexical similarity and design conformance, authors subsequently applied a genetic algorithm to determine the optimal solution.

To improve Stack Overflow recommendations Zagalsky *et al.* [56] proposed a system that retrieved the questions from Stack Overflow and their accepted answers by means of code snippets. Then, it applied the keyword search based on the Apache Lucene library to recommend code.

The recommender system developed by Abhinav *et al.* [57] suggested an appropriate task to the crowd worker. To rich this goal it needed only to compare the worker profile with the task description. For that it used tokenization and Doc2Vec for the preprocessingnstage and the cosine similarity to define the set of tasks that suits better the worker interests and skills.

In the work by Xu *et al.* [58] the system - in order to make proper API recommendations - compared feature requests text with the API descriptions and the text in the source code using preprocessing, vectorization, and cosine similarity.

One of the most important stages for developers recommendations in collaborative bug fixing, suggested by Xie *et al.* [59] is the topic determination for the bug report. For this purpose, the bug reports were tokenized, stop words were eliminated, stemming was applied and then Latent Dirichlet Allocation was used to define the topic.

Anvik and Murphy [60] proposed a recommender system that suggested report writers. For preprocessing it removed stop words and non alphabetic characters. It then calculated and normalised terms frequency. The algorithm subsequently generated labels (developers names), using project-specific heuristics. Finally, it created recommendations using SVM.

Rodas-Silva *et al.* [61] developed system which recommends implementation components based on SPLs features. For this purpose, features of the component went through the TF-IDF and cosine similarity.

Patterns are useful for programmers because - in using them - they can save precious time. However, it is quite challenging to find the right pattern in terms of both code and structure. To facilitate this process Palma *et al.* [62] used a GQM based system to propose the design of some patterns that can make the design of the system reusable at a later stage. For each pattern developed, the authors identified a set of circumstances (or layers) in which such patterns should be used. In particular, the first layer represented the primary conditions, whereas the second layers described the subconditions. For each layer users were then asked to give answers ('yes' 'no', or 'do not know'). These answers were weighted and ranged from 0 to 9. After all answers were gathered and the total combined weight was calculated, the pattern with the highest weight was suggested to the user by the system itself.

A comparable approach was proposed by Clara *et al.* [63], the authors attempted to develop an appropriate design-pattern category. In particular, the researchers developed a GQM-based tree model that consisted of chains of questions with associated positive or negative answers, that when combined together, led the user to the right design pattern. However, unlike in the first recommender system, the questions in this one were aimed at identifying the category of the design patterns. In particular, in this paper the authors presented a table with all questions and the weights they used. Depending on the user's answer these weights were added to a certain category score (such as behavioral, structural, or creational). At the end of the process the users received the pattern with the highest score as a suggestion.

Finally, the recommender system proposed by Wang *et al.* [64] suggested tags for software information sites. The object was passed through the chain of transformations including tokenization, Camel Case splitter, special sign splitter, numbers and stop words removal, and stemming. After that, with the combination of classification layers and probability estimators, the system was able to generate appropriate tags.

C. FURTHER CLUSTERING

While performing preliminary clustering on the papers selected for inclusion in our log, we noticed that they could be further divided into different classes based on their areas of application.

Figure 5 below shows the areas of software development for which proposed recommender systems were developed. We noticed that the absolute leaders are bugs followed by APIs and crowdsourcing.

IV. DISCUSSION

Having systematically presented our results, we next critically discuss them in context.

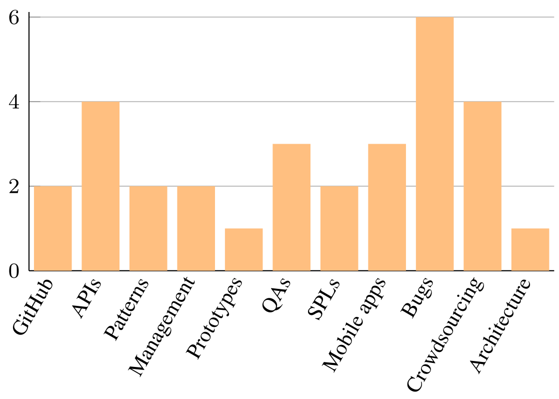


FIGURE 5. Papers distribution by areas of application.

A. RQ1: ARE THERE RECOMMENDER SYSTEMS FOR SOFTWARE DEVELOPERS WHOSE DATA SET INCLUDES TEXTUAL INFORMATION?

All the papers we reviewed except [62] and [63] described recommender systems that receive textual information as input. However, to work with text, any system needs to preprocess it. This means that texts must be standardized. However, preprocessing is not the only step that needs to be performed by the system. Any system also needs to vectorize the text; that is, convert letters into numbers. These two processes are beneficial because they limit the number of words, reduce scattering of data, and optimize the possibility of processing such data by computers.

Figure 6 shows the technologies used for preprocessing in the selected papers. The height of the columns denotes the number of occurrences this type of technology has been encountered.

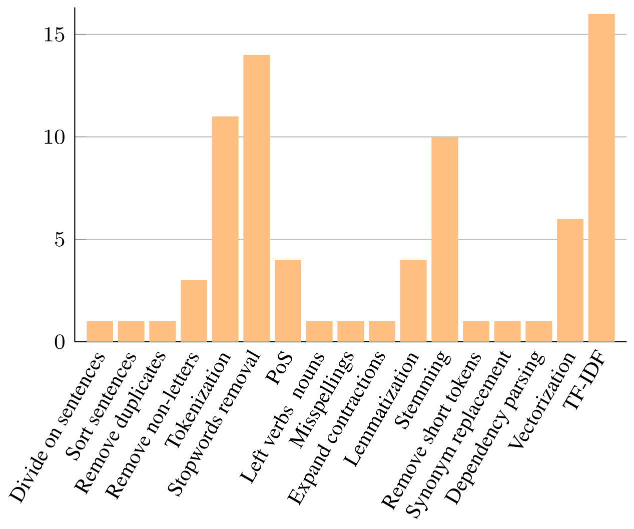


FIGURE 6. Preprocessing steps and usage.

Some of these terms are quite general concepts and are therefore discussed in different ways in the papers we analyzed. For example:

- **Vectorization** is mentioned as Word2Vec (2), word embeddings (2), Doc2Vec (1), and vector space model (1)
- **TF-IDF** is mentioned as TF (1), IDF (3), TF-IDF (11), customized version (1)

Based on Figure 6, we can rank preprocessing technologies by popularity, as follows: (1) TF-IDF, (2) Stop words removal, (3) Tokenization, (4) Stemming, (5) Vectorization, (6) PoS and Lemmatization, (7) Non-letter symbols removal.

Now we are in position to form from the most popular methods a sequence of logical actions that researchers may use in case of preprocessing for any GQM-based metrics recommender system. To do so, we make a few preliminary considerations.

- **lemmatization** and **stemming** do the same things; that is, they reduce the words to their normal forms - however, they do that in different ways. In particular, lemmatization lowers the sparsity of the data, because it does not just remove the endings, but uses morphological analysis - PoS (so these two will always be in sequence).
- **vectorization** and **TF-IDF** also do the same things; namely, they transform text into numbers. However, TF-IDF was used more frequently than vectorization, so we decided to choose TF-IDF.
- **TF-IDF** will always be the last step because other steps produce texts as output while TF-IDF receives text and produces numbers.

- **tokenization** should go before **stop words removing** and **PoS with lemmatization**, because these two steps works with words, not with letters.

With these important preliminary considerations, the question remains about the optimal sequence for **stop words removal**, **tokenization**, **PoS and lemmatization**, and **non-letters symbols removal**. To answer this question we carried out the small experiment. We took the train dataset of 500 sentences from Kagle: https://www.kaggle.com/theoviel/improve-your-score-with-some-text-preprocessing/notebook and run it 1000 times for each arrangement of preprocessing steps. The machine used in the experiment had the following characteristics: Intel Core i5-8250U, 4GHz, 7862MiB RAM. The resulted mean runtime and standard deviation are shown in Table 6.

TABLE 6. Execution time for each combination.

Group	Steps combination	Average time, ms	Std.dev., ms
A	Tokenization, Non-letters, Stop words, PoS and lemmatization	43.436	4.083
B	Non-letters, Tokenization, Stop words, PoS and lemmatization	41.003	1.068
C	Tokenization, PoS and lemmatization, Non-letters, Stop words	60.459	2.200
D	Tokenization, Non-letters, PoS and lemmatization, Stop words	59.080	2.667
E	Non-letters, Tokenization, PoS and lemmatization, Stop words	58.092	1.722
F	Tokenization, Stop words, Non-letters, PoS and lemmatization	45.810	0.757
G	Tokenization, Stop words, PoS and lemmatization, Non-letters	43.394	1.257
H	Tokenization, PoS and lemmatization, Stop words, Non-letters	57.518	0.848

Then we performed multiple pairwise comparison test using Tukey method with familywise error rate 0.05 [65]. The results of the test (see Table 7) show that in our configuration the most efficient sequence of remaining preprocessing steps appears to be B: (1) Non-letter symbols removal, (2) Tokenization, (3) Stop words removal, (4) PoS definition and Lemmatization.

Therefore, based on the above considerations and from the result of our test we assume the following order of preprocessing steps as optimal for metrics recommenders: (1) Non-letter symbols removal, (2) Tokenization, (3) Stop words removal, (4) PoS definition, (5) Lemmatization, (6) TF-IDF.

One can notice at least a couple of issues in the analysis we conducted so far. Firstly, we cannot say that the most common approaches are the most effective ones. This is because preprocessing is often context-sensitive. Secondly, potential issue is that the authors' choices of preprocessing techniques in the papers we selected are generally subjective and lack an objective evaluation. This can make it difficult, if not impossible, to know whether a particular choice was done on purpose.

Moreover, the results of the experiment are only preliminary, because test was run on one machine and on one dataset.

TABLE 7. Pairwise group comparisons by Tukey test, FWER = 0.05. ***Lower and Upper are the boundaries of the 95% confidence intervals. * All differences are significant apart from the one between A and G.***

Group1	Group2	Meandiff	p-adj	Lower	Upper
A	B	-2.433	0.001	-2.720	-2.147
A	C	17.024	0.001	16.738	17.310
A	D	15.644	0.001	15.358	15.930
A	E	14.657	0.001	14.371	14.943
A	F	2.374	0.001	2.088	2.660
A	G	-0.042	0.9*	-0.328	0.244
A	H	14.082	0.001	13.796	14.368
B	C	19.457	0.001	19.171	19.743
B	D	18.078	0.001	17.792	18.364
B	E	17.09	0.001	16.804	17.376
B	F	4.808	0.001	4.522	5.094
B	G	2.392	0.001	2.106	2.678
B	H	16.516	0.001	16.230	16.802
C	D	-1.379	0.001	-1.665	-1.093
C	E	-2.367	0.001	-2.653	-2.081
C	F	-14.650	0.001	-14.936	-14.364
C	G	-17.065	0.001	-17.352	-16.779
C	H	-2.942	0.001	-3.228	-2.656
D	E	-0.988	0.001	-1.274	-0.702
D	F	-13.270	0.001	-13.556	-12.984
D	G	-15.686	0.001	-15.972	15.400
D	H	-1.562	0.001	-1.848	-1.276
E	F	-12.283	0.001	-12.569	-11.996
E	G	-14.698	0.001	-14.984	-14.412
E	H	-0.574	0.001	-0.861	-0.288
F	G	-2.416	0.001	-2.702	-2.130
F	H	11.708	0.001	11.422	11.994
G	H	14.124	0.001	13.838	14.41

Nevertheless, this can work as a point of reference for further investigations.

B. RQ2: WHICH ALGORITHMS UNDERLINE RECOMMENDER SYSTEMS FOR SOFTWARE DEVELOPERS?

Among all the algorithms used by the authors of the papers we reviewed in our SLR we can identify 5 major groups: (a) classification, (b) clustering, (c) ranking, (d) heuristics based approaches, and (e) questionnaires. Figure 7 shows the distribution of the algorithms used by recommender systems.

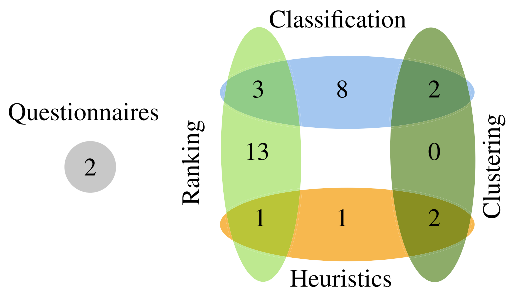


FIGURE 7. Algorithms classification.

- **Classification:** XGBoost [32], Tree based classification [62], [63], SVM [49], [60], Multinomial Naive Bayes [50], TF-similarity [52], Convolutional Neural Networks [64].

- **Ranking:** Cosine similarity [41], [45], [53]–[55], [57], [58], [61], Markow chain [42], Custom similarity measure [46], LambdaMART [51], TF-IDF similarity [56], LDA [59].
- **Heuristics:** Heuristics [39].
- **Classification and ranking:** BERT and matching score [33], Convolution Neural Networks and Gradient Descent [38], Naive Bayes and cosine similarity [44].
- **Classification and clustering:** LDA and decision tree [40], ARdoc and clusterization hierarchical Dirichlet process [47].
- **Heuristics and ranking:** Heuristics and Convolution Neural Networks [43].
- **Clustering and ranking:** Incremental diffusive clustering algorithm and kNN [36], Average link hierarchical agglomerative clustering algorithm and kNN [34].
- **Questionnaires:** [62], [63].

In GQM-based metrics recommender systems we can not classify or cluster textual data reliably [32], [44], [49], [50], [52], [60], as a virtually infinite number of metrics can be conceived. It can also be difficult to construct some heuristics [39]), or a huge number of pre-arranged questions and goals [62], [63]. The authors of the last papers succeeded in their goals because the number of existing patterns was limited (under 20). In our case, the unlimited number of metrics potentially conceivable represents a virtually insurmountable obstacle. Even if we could take the most basic ones as a point of reference, their number would still be quite large. Moreover, answering such questionnaires in software development can take a lot of time, which can be better spent on the development process instead.

Ranking algorithms are also quite problematic. In any GQM-based metrics recommender system there will be a huge number of goals and questions from all users, and comparing all this information with cosine or any other similarity can take a long time.

The idea of deleting everything besides verbs and nouns from the sentences [47], or using functionality categories and phrase patterns (citeAPI) can help to reduce the fragmentation of information and prepare the data for similarity metrics; however, this approach needs more scientific research and a lot of data to produce a meaningful analysis.

Another way could be to use the combination of any classification and ranking algorithm. To reduce the amount of information for processing, goals and questions could be classified [34], [36] into some groups from which then the most similar to the input items could be extracted. It is worth noting though, that because of the strong context dependency of such algorithms, it would be almost impossible to choose the standard one; yet, the best or most optimal in a particular combination can be established through a series of experiments on real data.

To summarize, we realized that pure classification, clustering, heuristics, and ranking are not suitable for solving our problem because of the large number of goals, questions and metrics. However, we noticed that ranking in combination with classification could contribute to solving our problem.

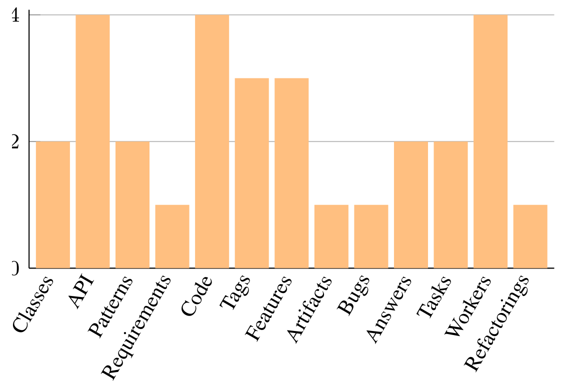


FIGURE 8. **Papers distribution over output types.**

C. RQ3: ARE THERE RECOMMENDER SYSTEMS CAPABLE OF SUGGESTING METRICS FOR SOFTWARE DEVELOPERS?

To answer this question we carefully checked the outputs used or proposed in the papers we selected for inclusion in our log (see Figure 8). Recommending the appropriate **APIs** was the goal of 4 systems in 4 papers [33], [46], [54], [58]. The same number of recommender systems proposed as outputs **code parts**, such as packages [36], parts of code to change [47], software product line implementation components [61], classes where can be a bug [53]. In 4 other papers the algorithms proposed as outputs **workers** - experts/specialists who can accomplish the task [51], [52], [59], [60]. In addition, 2 systems suggested as outputs the **classes** of the input data; that is, acceptance, response or rejection of the pull request [32] and whether the app contained a bug [40]. 2 other papers proposed algorithms for **design patterns** suggestion [62], [63]. Only for 1 system the result turned out to be **requirements** [34] and for 1 more - **artifacts** [41]. Furthermore, 3 papers suggested as outputs **tags** [38], [64], while 2 papers recommended **answers** for the questions asked on the same sites [43], [56]. 3 other systems recommended **features**: for software product lines [39], for mobile app descriptions [44] and for bug reports improvement [45]. 1 additional paper described a recommender system focused on **bugs** proposal. 2 systems suggested as outputs **tasks** for the developers [49], [57], while 1 more paper suggested **refactorings** generation [55].

Among the papers we screened there were several that fed metrics into the proposed recommendation system as inputs. However, as we said before, we did not find systems that could generate metrics as outputs. This fact prompts us to think that we were the first to discover the need of such an algorithm in the area of software development.

We can assume, that metrics are used as input data for algorithms, because the values obtained by measuring a process, product, or resources usually need to be analyzed and used to generate solutions. Because of the nature of such metrics, it is not surprising that there are recommender systems that base their proposals on the analysis of a variety of metrics and on their combinations.

However, the process of metrics determination is also complex and should be automated.

V. LIMITATIONS, THREATS TO VALIDITY, AND REVIEW ASSESSMENT

A. LIMITATIONS

We begin this section by examining the obstacles that may have prevented an objective and unbiased review.

1) The search was conducted with a limited number of databases; namely IEEExplore, Scopus, ACM Digital Library. A sceptical reader may object that we should have taken into considerations more databases for our searches (such as Google Scholar, Science Direct). We acknowledge this as a good objection; however, we provided good reasons for not searching Science Direct in section 2.3 above. As for Google Scholar, we notice that this database tends to simply aggregate papers from other databases, often with no filter with respect to academic quality (low quality conference proceedings as well as dubious piece of grey literature are often found through Google Scholar). So, although our database selection may have been perfected, we firmly believe that it was broad enough to ensure the academic soundness of our findings.
2) Another potential shortcoming affecting this work might be that we did not take into account grey literature. It is customary for SLR to concentrate on primary studies, still, we notice that including grey literature in a SLR is becoming an increasingly acceptable practice in software engineering, leading to a multivocal approach [66]–[68]. However, we realized that such a practice is often implemented when there is a scarcity of secondary sources, which was not our case.
3) Finally, one could argue that one of the inclusion criteria adopted in this work (selecting only papers written in English) is severely constraining the kind of research we were able to get. Issues in cross-cultural research are rightly emerging as vitally important in science [69], [70]. We are fully aware of such issues and acknowledge this as a very serious point of contention. However, most of the literature in the field is in English, therefore the requirement we adopted in this SLR is neither unusual nor uncommon for our field.

B. THREATS TO VALIDITY

In this subsection we discuss potential biases that may have affected our research and the production of our findings. To do this, we review the actions that were taken to avoid the emergence of such biases [71].

1) **Retrieval bias** - to avoid this bias we used the widest possible search queries. This means that we screened all papers related to recommender systems and software development, and used a wide set of databases for our searches.
2) **Publication bias** - we relied on peer reviewed, quality papers published in either good academic journals or in prestigious/reputable conferences. This means that we covered a variety of levels of analysis and of experimental protocols. We believe that this ensured that we did not report only statistically significant results
3) **Inclusion criteria bias** - to avoid this bias, we attempted to formulate the most general and appropriate criteria for our topic, following the best norms of our discipline.
4) **Selector bias** - we acknowledge that this bias may be present in our work, because we did not adopt a blind review process; however, every author involved in the research cross-checked the methodology we adopted as well as the results obtained for objectivity and consistency. As we follow the best norms of our discipline [72], we believe we did not fall for this bias.

C. REVIEW ASSESSMENT

The last step in the critical assessment of our findings involves a general reflection on the overall quality of the work we presented. Following [73], we formulated a set of questions, which can be used as a point of reference to assess the overall quality of a SLR. The questions follow below:

1) **Are the inclusion/exclusion criteria objective and reasonable?** All criteria were mentioned upfront in the methodology (see Section II), before we started the search process. The criteria we selected are congruent with those generally used in the field and fit the topic of our study perfectly. Thus, we believe that the inclusion/exclusion criteria we used are reasonable and objective.
2) **Does the search process cover all possible relevant papers?** To perform our searches, we used a set of reliable bibliographic databases and gathered papers from both prestigious academic journals and reputable conferences proceedings. On these grounds, we can say that the papers selected for inclusion in our final log are representative for the field.
3) **Has there been a quality review?** We developed a metric to assess the papers' quality (see Section 2.6 above). Indeed we proved that the quality of the papers we included in the log was high. Therefore, we can also say that the findings on which we developed our SLR were reliable and accurate.

VI. SYNOPTIC SUMMARY AND SIGNIFICANCE OF RESEARCH

In this section we highlight for the reader the most important results we achieved in this SLR and then briefly reflect on their significance for the field.

Important Results:

1) We have observed that the sequence of the most frequently used preprocessing steps in recommender

systems for software developers is the following: (1) TF-IDF, (2) Stop words removal, (3) Tokenization, (4) Stemming, (5) Vectorization, (6) PoS and Lemmatization, (7) Non-letter symbols removal.
2) We have found that preprocessing steps combined in the following order constitute the most promising (optimal) sequence for GQM-based metrics recommender for software developers: (1) Non-letter symbols removal, (2) Tokenization, (3) Stopwords removal, (4) PoS definition, (5) Lemmatization, (6) TF-IDF.
3) We have noticed that recommender systems for software developers can be based on one of the following categories of algorithms: classification, ranking, heuristics, classification and ranking, classification and clustering, heuristics and ranking, clustering and ranking, questionnaires.
4) We have determined that a combination of classification and ranking suits the GQM-based metrics recommender construction more than others categories of algorithms.
5) We have observed that there are no recommender systems developed to date to generate metrics for software developers.

The combination of the results obtained in this SLR can help in the construction of future ’metrics collectors’. For the development of such systems, it is crucially important to generate and present metrics in the most user-friendly way, which can be achieved by the use of recommender systems. However, such systems need to work quickly and accurately, for a better user experience. In this context, the results listed above are beneficial because they can help us increasing the accuracy, while lowering the sparcity of the data. They also allow us to maximize the algorithms’ execution time.

VII. CONCLUSION

The goal of this research was to determine the technologies needed to build a GQM-based metrics recommender system for software developers. The aim of this system would be to automatically generate metrics based on users’ goals and questions. Since there are no mentions of such systems in the literature, to achieve our goal, our research has focused on the three basic components characterising any recommender systems, independently: data sets, algorithms, and recommendations.

To deal with the problem of assigning metrics for user’s goals and questions, we showed that a combination of ranking and classification technologies can be relevant for the construction of an algorithm capable of dealing with this issue. However, we did not find papers covering the issue of metrics generation. That may point out a research gap, which we partially contributed to fill.

We hope that our SLR will help drawing researchers’ attention to the problem of metric generation and hence, provide a basis for other investigations in this fascinating yet unexplored area of research.

APPENDIX A

Template taken and revised from: http://prisma-statement.org/documents/PRISMA_2020_checklist.pdf

TABLE 8. PRISMA 2020 Checklist.

Num	Location
Title	
1	Page 1: Title
Abstract	
2	Page 1: Abstract
Introduction	
3	Page 1: Section I
4	Page 1: Section I
Methods	
5	Page 3: Section II-D
6	Page 3: Section II-C
7	Page 3: Section II-C
8-15	not applicable
16a 16b	Page 3: Section II-D not applicable
Results	
17	Page 5: Section III
18-22	not applicable
Discussion	
23a 23b	Page 7: Section IV-A, Page 8: Section IV-B, Page 9: Section IV-C Page 8: Section IV-A
Limitations	
23c	Page 10: Section V
Conclusion	
23d	Page 11: Section VI
Other information	
24	not applicable
25	Page 1

APPENDIX B

Questions for Quality Assessment:

1) Was the motivation for the development of the recommender system clearly specified?
   - 1 point if the motivation for the development of the recommender system was clearly stated;
   - 0.5 points if the motivation was provided, but could be further elaborated;
   - 0 points if the motivation was hard to identify or if it was not mentioned;
2) Does the proposed recommender system solve the problem for which it was developed?
   - 1 point if the system solves the problem;
   - 0.5 points if the system partially solves the problem;
   - 0 points if the system does not solve the problem;
3) Is the recommender system’s construction algorithm clear and reproducible?
   - 1 point if it is possible to replicate the algorithm described in the article;
   - 0.5 points if in general the algorithm is clear, but it is difficult to replicate it because of lack of crucial details;
   - 0 points if If the algorithm has been described only in general terms and it is not possible to replicate it;

4) Was the proposed recommender system objectively evaluated?
   - 1 point if the authors conducted a fair and unbiased review of their algorithm or if they performed a critical analysis of its results;
   - 0.5 points if the authors performed an analysis of their algorithm but such an analysis was partially biased or is not clear or critical enough;
   - 0 points if the authors did not conduct a fair and unbiased analysis or if the results were not critically analysed;

Sum of scores, gathered during the answering on these questions above, are presented in Table 9.

TABLE 9. Quality scores assigned to the papers selected for inclusion.

Paper reference	Motivation	Solution	Clear Algorithm	Evaluation
[32]	2	2	1	2
[33]	2	2	1.5	2
[62]	2	2	2	0.5
[34]	2	1.5	1	1.5
[36]	2	1.5	1.5	1
[38]	1.5	2	0.5	1
[39]	1.5	2	0.5	1.5
[40]	1.5	1.5	1	2
[41]	2	1.5	0.5	1.5
[42]	2	2	1.5	2
[43]	2	2	2	2
[44]	2	2	2	2
[45]	2	1.5	0.5	0.5
[46]	2	1	1.5	1
[63]	2	1.5	2	1.5
[47]	2	2	1.5	2
[49]	2	2	0	1
[50]	2	1	0.5	2
[51]	2	1.5	1	1.5
[52]	2	1	1	0.5
[53]	2	2	1.5	2
[54]	2	2	2	2
[55]	2	1	1.5	1.5
[56]	1.5	1.5	0.5	1
[60]	2	1.5	1	1.5
[57]	2	2	2	2
[58]	2	2	0	1.5
[59]	2	1	1	2
[61]	2	2	2	2
[64]	2	2	2	2

REFERENCES

[1] E. B. Belachew, "Analysis of software quality using software metrics," *Int. J. Comput. Sci. Appl.*, vol. 8, no. 4/5, pp. 11–20, Oct. 2018.

[2] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2014.

[3] E. Bouwers, J. Visser, and A. Van Deursen, "Getting what you measure: Four common pitfalls in using software metrics for project management," *Queue*, vol. 10, no. 5, pp. 50–56, May 2012.

[4] G. Concas, M. Marchesi, G. Destefanis, and R. Tonelli, "An empirical study of software metrics for assessing the phases of an agile project," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 22, no. 4, pp. 525–548, Jun. 2012.

[5] M. Singh, A. Mittal, and S. Kumar, "Survey on impact of software metrics on software quality," *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 1, pp. 137–141, 2012.

[6] M. Lines and S. Ambler, *Choose Your WoW: A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working (WoW)* (Choose Your WoW Series). Queens, NY, USA: The Science and Information Organization, 2019.

[7] T. Tahir, G. Rasool, and C. Gencel, "A systematic literature review on software measurement programs," *Inf. Softw. Technol.*, vol. 73, pp. 101–121, May 2016.

[8] F. Castro. *THE Beginner's Guide to OKR*. Accessed: Feb. 23, 2022. [Online]. Available: https://felipecastro.com/resource/The-Beginners-Guide-to-OKR.pdf

[9] K. Hoffmann-Burdzińska and O. Flak, "Management by objectives as a method of measuring teams' effectiveness," *J. Positive Manage.*, vol. 6, p. 67, Apr. 2016.

[10] M. Ishaq Bhatti, H. M. Awan, and Z. Razaq, "The key performance indicators (KPIs) and their impact on overall organizational performance," *Quality Quantity*, vol. 48, no. 6, pp. 3127–3143, Nov. 2014.

[11] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 6, pp. 728–738, Nov. 1984.

[12] V. Basili, R. Solingen, G. Caldiera, and D. Rombach, *The Goal Question Metric Approach*, vol. 1. Hoboken, NJ, USA: Wiley, 1994, pp. 528–532.

[13] M. Alhamadi, "Challenges, strategies and adaptations on interactive dashboards," in *Proc. 28th ACM Conf. User Modeling, Adaptation Personalization*, New York, NY, USA, 2020, pp. 368–371.

[14] V. Ivanov, A. Rogers, G. Succi, J. Yi, and V. Zorin, "Precooked developer dashboards: What to show and how to use," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, New York, NY, USA, May 2018, pp. 402–403.

[15] V. Ivanov, V. Pischulin, A. Rogers, G. Succi, J. Yi, and V. Zorin, "Design and validation of precooked developer dashboards," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, New York, NY, USA, Oct. 2018, pp. 821–826.

[16] P. Singh, P. D. Pramanik, A. Dey, and P. Choudhury, "Recommender systems: An overview, research trends, and future directions," *Int. J. Bus. Syst. Res.*, vol. 15, pp. 14–52, Jan. 2021.

[17] R. Burke, A. Felfernig, and M. H. Göker, "Recommender systems: An overview," *AI Mag.*, vol. 32, no. 3, pp. 13–18, Jun. 2011.

[18] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, *Collaborative Filtering Recommender Systems*. Berlin, Germany: Springer, 2007, pp. 291–324.

[19] H. Wang, Z. Shen, S. Jiang, G. Sun, and R.-J. Zhang, "User-based collaborative filtering algorithm design and implementation," *J. Phys., Conf. Ser.*, vol. 1757, no. 1, Jan. 2021, Art. no. 012168.

[20] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, "Item-based collaborative filtering recommendation algorithms," in *Proc. 10th Int. Conf. World Wide Web*, 2001, pp. 285–295.

[21] F. O. Isinkaye, Y. O. Folajimi, and B. A. Ojokoh, "Recommendation systems: Principles, methods and evaluation," *Egyptian Inform. J.*, vol. 16, no. 3, pp. 261–273, 2015.

[22] M. J. Pazzani and D. Billsus, *Content-Based Recommendation Systems*. Berlin, Germany: Springer, 2007, pp. 325–341.

[23] P. Lops, M. D. Gemmis, and G. Semeraro, *Content-based Recommender Systems: State of the Art and Trends*. Boston, MA, USA: Springer, Jan. 2011, pp. 73–105.

[24] E. Çano and M. Morisio, "Hybrid recommender systems: A systematic literature review," *Intell. Data Anal.*, vol. 21, no. 6, pp. 1487–1524, Nov. 2017.

[25] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, *Recommendation Systems in Software Engineering*. New York, NY, USA: Springer, 2014.

[26] M. Gasparic and A. Janes, "What recommendation systems for software engineering recommend: A systematic literature review," *J. Syst. Softw.*, vol. 113, pp. 101–113, Mar. 2016.

[27] W. Siricharoen, U. Pakdeetrakulwong, and P. Wongthongtham, "Recommendation systems for software engineering: A survey from software development life cycle phase perspective," in *Proc. 9th Int. Conf. Internet Technol. Secured Trans.*, Dec. 2014, pp. 137–142.

[28] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ., Keele, U.K., Durham Univ. Joint Rep., Durham, U.K., Tech. Rep. EBSE 2007-001, 2007.

[29] M. J. Page, J. E. McKenzie, P. M. Bossuyt, I. Boutron, T. C. Hoffmann, C. D. Mulrow, and L. Shamseer, "The prisma 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, vol. 372, Apr. 2021, Art. no. 105906.

[30] F. Rehman, O. Khalid, and S. A. Madani, "A comparative study of location-based recommendation systems," *Knowl. Eng. Rev.*, vol. 32, no. e7, pp. 1–30, Jan. 2017.

[31] J. Harris, C. Quatman, M. Manring, R. Siston, and D. Flanigan, "How to write a systematic review," *Amer. J. Sports Med.*, vol. 42, no. 11, pp. 2761–2768, Aug. 2013.

[32] M. I. Azeem, S. Panichella, A. Di Sorbo, A. Serebrenik, and Q. Wang, "Action-based recommendation in pull-request development," in *Proc. Int. Conf. Softw. Syst. Processes*, New York, NY, USA, Jun. 2020, pp. 115–124.

[33] W. Xie, X. Peng, M. Liu, C. Treude, Z. Xing, X. Zhang, and W. Zhao, "API method recommendation via explicit matching of functionality verb phrases," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, New York, NY, USA, Nov. 2020, pp. 1015–1026.

[34] C. Castro-Herrera, C. Duan, J. Cleland-Huang, and B. Mobasher, "A recommender system for requirements elicitation in large-scale software projects," in *Proc. ACM Symp. Appl. Comput.*, New York, NY, USA, 2009, pp. 1419–1426.

[35] S. Zhong and J. Ghosh, "A unified framework for model-based clustering," *J. Mach. Learn. Res.*, vol. 4, pp. 1001–1037, Dec. 2003.

[36] C. Mcmillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher, "Recommending source code for use in rapid software prototypes," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 848–858.

[37] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli, "On-demand feature recommendations derived from mining public product descriptions," in *Proc. 33rd Int. Conf. Softw. Eng.*, New York, NY, USA, May 2011, pp. 181–190.

[38] J. Zhang, H. Sun, Y. Tian, and X. Liu, "Semantically enhanced tag recommendation for software CQAs via deep learning," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, New York, NY, USA, May 2018, pp. 294–295.

[39] M. Hamza and R. J. Walker, "Recommending features and feature relationships from requirements documents for software product lines," in *Proc. IEEE/ACM 4th Int. Workshop Realizing Artif. Intell. Synergies Softw. Eng.*, May 2015, pp. 25–31.

[40] M. Gomez, R. Rouvoy, M. Monperrus, and L. Seinturier, "A recommender system of buggy app checkers for app store moderators," in *Proc. 2nd ACM Int. Conf. Mobile Softw. Eng. Syst.*, May 2015, pp. 1–11.

[41] D. Cubranic and G. C. Murphy, "Hipikat: Recommending pertinent software development artifacts," in *Proc. 25th Int. Conf. Softw. Eng.*, Portland, OR, USA, 2003, pp. 408–418.

[42] B. Ashok, J. Joy, H. Liang, S. K. Rajamani, G. Srinivasa, and V. Vangala, "DebugAdvisor: A recommender system for debugging," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. Eur. Softw. Eng. Conf. Found. Softw. Eng. Symp.*, New York, NY, USA, 2009, pp. 373–382.

[43] Z. Gao, X. Xia, D. Lo, and J. Grundy, "Technical Q8A site answer recommendation via question boosting," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 1, pp. 1–34, Jan. 2021.

[44] H. Jiang, J. Zhang, X. Li, Z. Ren, D. Lo, X. Wu, and Z. Luo, "Recommending new features from mobile app descriptions," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 4, pp. 1–29, Oct. 2019.

[45] M. R. Karim, "Key features recommendation to improve bug reporting," in *Proc. IEEE/ACM Int. Conf. Softw. Syst. Processes (ICSSP)*, May 2019, pp. 1–4.

[46] L. Cai, H. Wang, Q. Huang, X. Xia, Z. Xing, and D. Lo, "BIKER: A tool for bi-information source based API method recommendation," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, New York, NY, USA, Aug. 2019, pp. 1075–1079.

[47] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 106–117.

[48] S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. Gall, "How can I improve my app? Classifying user reviews for software maintenance and evolution," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2015, pp. 281–290.

[49] M. M. Kamel, A. Gil-Solla, and M. Ramos-Carber, "Tasks recommendation in crowdsourcing based on workers' implicit profiles and performance history," in *Proc. 9th Int. Conf. Softw. Inf. Eng. (ICSIE)*, New York, NY, USA, Nov. 2020, pp. 51–55.

[50] C. D. Sipio, R. Rubei, D. D. Ruscio, and T. P. Nguyen, "A multinomial naïve Bayesian (MNB) network to automatically recommend topics for GitHub repositories," in *Proc. Eval. Assessment Softw. Eng.*, New York, NY, USA, 2020, pp. 71–80.

[51] J. Wang, Y. Yang, S. Wang, Y. Hu, D. Wang, and Q. Wang, "Context-aware in-process crowdworker recommendation," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, New York, NY, USA, 2020, pp. 1535–1546.

[52] J. Cerezo, J. Kubelka, R. Robbes, and A. Bergel, "Building an expert recommender chatbot," in *Proc. IEEE/ACM 1st Int. Workshop Bots Softw. Eng. (BotSE)*, May 2019, pp. 59–63.

[53] R. Almhana, W. Mkaouer, M. Kessentini, and A. Ouni, "Recommending relevant classes for bug reports using multi-objective search," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, New York, NY, USA, Aug. 2016, pp. 286–295.

[54] F. Thung, S. Wang, D. Lo, and J. Lawall, "Automatic recommendation of API methods from feature requests," in *Proc. 28th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2013, pp. 290–300.

[55] Y. Lin, X. Peng, Y. Cai, D. Dig, D. Zheng, and W. Zhao, "Interactive and guided architectural refactoring with search-based recommendation," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, New York, NY, USA, Nov. 2016, pp. 535–546.

[56] A. Zagalsky, O. Barzilay, and A. Yehudai, "Example overflow: Using social media for code recommendation," in *Proc. 3rd Int. Workshop Recommendation Syst. Softw. Eng. (RSSE)*, Jun. 2012, pp. 38–42.

[57] K. Abhinav, G. K. Bhatia, A. Dubey, S. Jain, and N. Bhardwaj, "TasRec: A framework for task recommendation in crowdsourcing," in *Proc. 15th Int. Conf. Global Softw. Eng.*, New York, NY, USA, Jun. 2020, pp. 86–95.

[58] C. Xu, B. Min, X. Sun, J. Hu, B. Li, and Y. Duan, "MULAPI: A tool for API method and usage location recommendation," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion*, May 2019, pp. 119–122.

[59] X. Xie, W. Zhang, Y. Yang, and Q. Wang, "DRETOM: Developer recommendation based on topic models for bug resolution," in *Proc. 8th Int. Conf. Predictive Models Softw. Eng.*, New York, NY, USA, 2012, pp. 19–28.

[60] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.

[61] J. Rodas-Silva, J. A. Galindo, J. García-Gutiérrez, and D. Benavides, "Selection of software product line implementation components using recommender systems: An application to wordpress," *IEEE Access*, vol. 7, pp. 69226–69245, 2019.

[62] F. Palma, H. Farzin, Y.-G. Gueheneuc, and N. Moha, "Recommendation system for design patterns in software development: An DPR overview," in *Proc. 3rd Int. Workshop Recommendation Syst. Softw. Eng. (RSSE)*, Jun. 2012, pp. 1–5.

[63] C. K. Youssef, F. M. Ahmed, H. M. Hashem, V. E. Talaat, N. Shorim, and T. Ghanim, "GQM-based tree model for automatic recommendation of design pattern category," in *Proc. 9th Int. Conf. Softw. Inf. Eng. (ICSIE)*, New York, NY, USA, Nov. 2020, pp. 126–130.

[64] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "EnTagRec++: An enhanced tag recommendation system for software information sites," *Empirical Softw. Eng.*, vol. 23, no. 2, pp. 800–832, Apr. 2018.

[65] S. Lee and D. K. Lee, "What is the proper way to apply the multiple comparison test?" *Korean J. Anesthesiol.*, vol. 71, no. 5, pp. 353–360, Oct. 2018.

[66] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Inf. Softw. Technol.*, vol. 106, pp. 101–121, Feb. 2019.

[67] V. Garousi, M. Felderer, M. V. Mäntylä, and A. Rainer, "Benefitting from the grey literature in software engineering research," in *Contemporary Empirical Methods in Software Engineering*. Cham, Switzerland: Springer, 2020, pp. 385–413.

[68] Q. Mahood, D. Van Eerd, and E. Irvin, "Searching for grey literature for systematic reviews: Challenges and benefits," *Res. Synth. Methods*, vol. 5, no. 3, pp. 221–234, Sep. 2014.

[69] J. Henrich, S. J. Heine, and A. Norenzayan, "The weirdest people in the world?" *Behav. Brain Sci.*, vol. 33, nos. 2–3, pp. 61–83, Jun. 2010.

[70] J. Henrich, *The WEIRDest People in the World: How the West Became Psychologically Peculiar and Particularly Prosperous*. London, U.K.: Penguin, 2020.

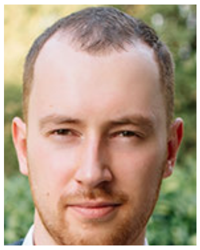
[71] C. F. Durach, J. Kembro, and A. Wieland, "A new paradigm for systematic literature reviews in supply chain management," *J. Supply Chain Manage.*, vol. 53, no. 4, pp. 67–85, Oct. 2017.

[72] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Eng.*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[73] B. Kitchenham, "Procedures for performing systematic reviews," Dept. Comput. Sci., Keele Univ., Keele, U.K., Tech. Rep. tr/se-0401, 2004.

MIRKO FARINA received the B.A. and M.Sc. degrees in Milan, the M.Phil. degree in Edinburgh, and the Ph.D. degree in Sydney. He is currently an Assistant Professor of philosophy and the Head of the Human Machine Interaction Laboratory, Innopolis University. He has been a Honorary Member of the Laboratory for Industrializing Software Production (LIPS), Faculty of Computer Science and Engineering, Innopolis University, since 2021; an Expert Member of the UNESCO Inclusive Policy Laboratory, since 2020; and a Contributing Member of the Astana Club, since 2021.

ANNA GORB is currently pursuing the degree with the Computer Science Program, Innopolis University, Russia. She also works at the Laboratory of Industrial Software Production, Innopolis University. Her research interests include recommender systems, agile methodologies, and software metrics.



ARTEM KRUGLOV graduated from Ural Federal University, in 2013, and received the Ph.D. degree, in 2017. He is currently an Assistant Lecturer with the Faculty of Computer Science and Software Engineering, Innopolis University. His research interests include the aspects of software development processes, agile methodologies, product and project management, and empirical methods.

GIANCARLO SUCCI (Member, IEEE) is currently a Full Professor at Innopolis University, Russia, where he directs the Laboratory of Industrial Software Production. Before joining the Innopolis University, he was a Professor with tenure at the Free University of Bozen-Bolzano, Italy; a Professor with tenure at the University of Alberta, Edmonton, AB, Canada; an Associate Professor at the University of Calgary, AB, Canada; and an Assistant Professor at the University of Trento, Italy. His research interests include multiple areas of software engineering, including open source development, agile methodologies, experimental software engineering, software engineering over the internet, and software product lines and software reuse.