The Journal of Systems & Software 168 (2020) 110669

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

# Similarity-based analyses on software applications: A systematic literature review

Maximilian Auch [a,*], Manuel Weber [a], Peter Mandl [a], Christian Wolff [b]

[a] Munich University of Applied Science, Lothstrasse 64, 80335 Munich, Germany
[b] University of Regensburg, Germany

ARTICLE INFO

*Article history:*
Received 17 September 2019
Received in revised form 15 May 2020
Accepted 25 May 2020
Available online 28 May 2020

*Keywords:*
Software similarity
Secondary study
Machine learning

ABSTRACT

In empirical studies on processes, practices, and techniques of software engineering, automation and machine learning are gaining popularity. In order to extract knowledge from existing software projects, a sort of similarity analysis is often performed using different methodologies, data and metadata. This systematic literature review focuses therefore on existing approaches of similarity-, categorization- and relevance-based analysis on software applications. In total, 136 relevant publications and patents were identified between 2002 and 2019 according to the established inclusion and exclusion criteria, which perform a calculation of software similarity in general or to support certain software engineering phases.

## 1. Introduction

Empirical studies have been carried out in the field of software engineering for several decades to support and improve specific processes, practices, and techniques. For this purpose, empirical studies have recently focused more and more on automation and machine learning to derive knowledge from the amount of data. This data, consisting of completed and already running software projects, is today available in large numbers in open source repositories and includes information such as the success or applicability of new programming languages, paradigms and techniques, new tool support for implementation, debugging and testing, as well as new visualizations of concepts. More and more published works are based on large amounts of these software projects in order to gain knowledge from the abundance of data. These findings, which can be gained from past projects, can help to achieve their goals. To learn from the right projects, often a similarity analysis needs to be performed to retrieve repositories, which are relevant for the targeted purpose. An increasing number of available projects and project-related information in collaborative repositories, including open source content and in application stores can also cause the problem of searchability and recommendability of software. To meet this use case, the software similarity is also needed for automated categorization or software tagging, which is why these targets are also relevant for research within this field. Therefore the software similarity represents a supporting function in the area of software engineering. In various project tasks of software engineering the retrieval of similar software projects has already been investigated for software engineering support. These tasks include project management, requirements engineering, software design, software development and testing. One aspect of this work is the automatic retrieval of similar software projects. Finding similar projects usually helps to derive decisions, identify action steps, support tasks, support documentation or to control the project. Therefore, it can be considered in the context of software engineering as a kind of supporting cross-sectional task. In addition, there are further possible use cases such as finding suitable developers or managing large quantities of applications.

To cover most application areas in the field of software engineering and to give a comprehensive overview of the research area, a systematic literature review (SLR) was carried out. The focus of the review was set on detecting similar or related applications, classifying software into groups or tagging software by their common characteristics. Therefore, a research question is phrased as a first step of the literature search according to Neely et al. (2010). This step serves as an orientation aid and offer a well-defined structure for the literature review process (Aveyard, 2014). The following research question with the corresponding sub-questions were examined by this SLR:

> Can similar software for a given software be determined automatically?
>
> – Which machine learning algorithms can be applied in the context of automated detection of similar software?
> – On which elements of a software can similar software be detected, including source code, textual descriptions, software libraries and meta data?

[*] Corresponding author.
*E-mail address:* maximilian.auch@hm.edu (M. Auch).

https://doi.org/10.1016/j.jss.2020.110669
0164-1212/© 2020 Elsevier Inc. All rights reserved.

**Table 1**
All electronic databases used as literature sources for this SLR.

| Database | Link |
|---|---|
| ACM Digital Library | https://dl.acm.org |
| IEEExplore | https://ieeexplore.ieee.org |
| Elsevier Scopus | https://www.scopus.com |
| Science Direct | https://www.sciencedirect.com |
| Springer Link | https://link.springer.com |
| Web of Science | apps.webofknowledge.com.emedien.ub.uni-muenchen.de |
| ArXiv | https://arxiv.org |
| Google Scholar | https://scholar.google.de |

The overall SLR was designed according to Kitchenham et al. (2004), by following the defined process steps (1) planning, (2) conducting and (3) reporting on the review. To give some further details about the methodology, the following section provides details about the applied process for reproducibility before describing the review results.

## 2. Research methodology

By examining the published articles, this study aims to find and understand the common methods and possible applications for automatic detection of software. For a general overview, this section summarizes the developed review protocol, which was applied in this study.

### 2.1. Data sources and search strategy

Basically, a computer-based search strategy was applied, which included an automatic search in electronic databases. Since not all primary studies can be found within a single literature source (Brereton et al., 2007), a first literature search was undertaken to reveal all most relevant literature sources for the SLR.[1] These electronic databases are summarized in Table 1.

### 2.2. Identification of keywords and search terms

While all relevant keywords were extracted from the research questions, all available keywords of the related work found in each publication of the first literature search were utilized as well. These resulting keyword lists were then consolidated and summarized with a corresponding prioritization according to their frequency. Similar keywords and synonyms have been merged. The resulting list of searchable keywords is presented by a word cloud in Fig. 1.

The search terms described below link the respective keywords using the Boolean operators AND and OR, according to the recommendation of Brereton et al. (2007). Due to the high number of possible combinations of the keywords, these were analysed in advance, promising combinations were formed which lead back to the research questions and a first sample search was carried out. The following combinations yielded promising results:

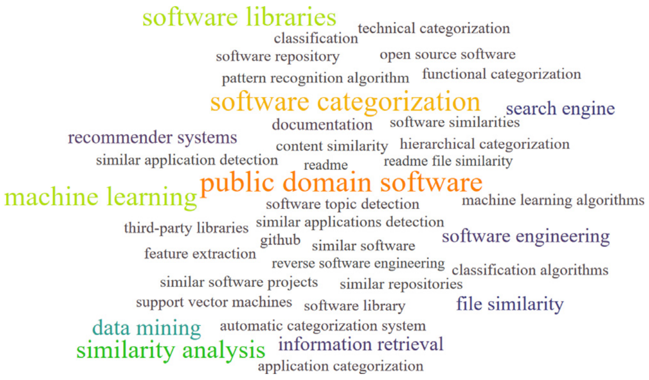

**Fig. 1.** Word cloud of the most relevant identified search terms for this review.

**ST1** ((“software libraries” OR “software library” OR “java libraries” OR “third-party libraries” OR “external libraries”) AND (“information retrieval” OR “IR” OR “document retrieval”) AND (“recommender systems” OR “project recommendation” OR “recommendation system” OR “software recommendation system” OR “recommendation algorithms”))

**ST2** ((“software libraries” OR “software library” OR “java libraries” OR “third-party libraries” OR “external libraries”) AND (“documentation” OR “system documentation”) AND (“software categorization” OR “functional categorization” OR “technical categorization” OR “application categorization” OR “automatic categorization system” OR “hierarchical categorization”))

**ST3** ((“public domain software” OR “open source software” OR “open-source repository” OR “software repository” OR “github” OR “gitlab” OR “repository”) AND (“file similarity” OR “readme file similarity” OR “readme”) AND (“software categorization” OR “functional categorization” OR “technical categorization” OR “application categorization” OR “automatic categorization system” OR “hierarchical categorization”))

**ST4** ((“software libraries” OR “software library” OR “java libraries” OR “third-party libraries” OR “external libraries”) AND (“similarity analysis” OR “similar software projects” OR “similar projects” OR “similar software” OR “software similarities” OR “similar application detection” OR “content similarity” OR “similar github repositories” OR “similar repositories” OR “similar applications detection” OR “similar apps”))

**ST5** ((“public domain software” OR “open source software” OR “open-source repository” OR “software repository” OR “github” OR “gitlab” OR “repository”) AND (“data mining” OR “pattern recognition algorithm”) AND (“similarity analysis” OR “similar software projects” OR “similar projects” OR “similar software” OR “software similarities” OR “similar application detection” OR “content similarity” OR “similar github repositories” OR “similar repositories” OR “similar applications detection” OR “similar apps”) AND (“software engineering” OR “reverse software engineering” OR “computer aided software engineering”))

**ST6** ((“software libraries” OR “software library” OR “java libraries” OR “third-party libraries” OR “external libraries”) AND (“machine learning” OR “classification” OR “classification algorithms” OR “support vector machine” OR “support vector machines” OR “svm” OR “feature extraction” OR “software topic detection”) AND (“recommender systems” OR “project recommendation” OR “recommendation system” OR “software recommendation system” OR “recommendation algorithms”))

[1] Other databases, such as Wiley Online Library and Taylor and Francis, were sampled but remained without result and are therefore not included in this SLR.

**ST7** ((“public domain software” OR “open source software” OR “open-source repository” OR “software repository” OR “github” OR “gitlab” OR “repository”) AND (“software categorization” OR “functional categorization” OR “technical categorization” OR “application categorization” OR “automatic categorization system” OR “hierarchical categorization”) AND (“search engine”))

**ST8** ((“software libraries” OR “software library” OR “java libraries” OR “third-party libraries” OR “external libraries”) AND (“software categorization” OR “functional categorization” OR “technical categorization” OR “application categorization” OR “automatic categorization system” OR “hierarchical categorization”) AND (“software engineering” OR “reverse software engineering” OR “computer aided software engineering”))

**ST9** ((“software engineering” OR “reverse software engineering” OR “computer aided software engineering”) AND (“similarity analysis” OR “similar software projects” OR “similar projects” OR “similar software” OR “software similarities” OR “similar application detection” OR “content similarity” OR “similar github repositories” OR “similar repositories” OR “similar applications detection” OR “similar apps”) AND (“recommender systems” OR “project recommendation” OR “recommendation system” OR “software recommendation system” OR “recommendation algorithms”))

### *2.3. Inclusion and exclusion criteria*

The publications, which were determined by applying the described search terms to the selected literature sources, were filtered according to certain criteria. These criteria were established according to Budgen and Brereton (2006) in inclusion and exclusion criteria. While the inclusion criteria include work based on the research question and required characteristics of the publication, several exclusion criteria were determined to filter according to specific topics and characteristics of the work. In the SLR process described later, each process step applied the inclusion criteria first, followed by the exclusion criteria in a subsequent step.

In order to filter the approaches to the recognition of similar software projects, the criteria were chosen accordingly. Related research areas, such as the detection of similar or relevant code locations, algorithms, (API-)functions and their calls, as well as code components for reuse, malware classification, software theft detection or plagiarism detection should be excluded from the targeted results scope. Even if these works partly perform basic preliminary research for the classification of entire software projects or use a similar methodology, they are excluded from the focus of this literature review in order to provide a better overview in the large field of research. The excluded research areas may be analysed by other SLRs, such as the one of Nazir et al. (2019) for the analysis of theft detection and software piracy approaches. A large field of research on effort estimation has also been identified, which includes many publications. The approaches usually have less focus on the software, but concentrate more on the project with a project management view. Therefore effort estimation was excluded. For completeness, other literature reviews such as those of Wen et al. (2012), Idri et al. (2015) and Sharma and Singh (2017) can be consulted for this field of research as well. The same applies to work that aims to support documentation. For instance, this includes model-driven engineering, focusing for example on finding similarities in UML diagrams to give support in modelling tools. Finally, user-focused analyses for the identification of similar software usage concepts were found. Since these also only focused on a partial aspect of the software to be recognized, they were also excluded from this SLR.

All inclusion and exclusion criteria are listed below for an overview and traceability.

**Inclusion criteria:**

- Application Similarity Detection: The work classifies services or recommends similar software projects.
- The most detailed or if similar, the most recently found work must be stated if several papers report on the same study.
- The paper is written in english.
- It is a primary study.
- According to the taxonomy of Cooper (1988), the material includes a scientific research result, a developed theory, or a documented and evaluated application.

**Exclusion criteria:**

- The detection of similar software or categorization is not automated.
- Partial application similarity detection: The resulting similarity is only applied on aspects of the software, like the similarity of source code, API-functions, issues, contributors or reported bugs. The approach does not aim to categorize the whole application.[2]
- The software similarity refers to find identical, e.g. duplicated, instead of similar software. Using methods such as birthmarks or fingerprints, goals such as malware classification, software theft detection, plagiarism detection and code clone detection are achieved. This can be for example used to find repackaged Android applications, as proposed by Cesare and Xiang (2012).[3]
- The work focuses on component reuse, which is limited to the categorization of source code or modular parts of the software.
- A categorization is exclusively based on meta-models by having a focus on “model-driven engineering”.
- The work focuses on a categorization or similarity-calculation of web-services.
- Effort estimation is the main target of the research, even if a similarity analysis is based on project characteristics such as project size, time required, number of team members.
- The focus of the study lies mainly on a usage-centric evaluation of application users and user search queries.
- Publications which do not make a new contribution to gaining knowledge about the research questions posed.

### *2.4. Snowballing*

As an additional step, the SLR process was extended by the backward snowballing and forward snowballing. Snowballing refers to the use of the reference list or quotations for the respective work to identify additional relevant work. While the analysis of the references or quotations in the respective work is called backward snowballing, forward snowballing refers to the analysis of the list of works referencing the respective work. Our approach followed the proposal of Wohlin (2014). According to Randolph (2009), the number of snowballing-levels is not linked to a number of works found. Instead, it is recommended

[2] Work that only uses certain aspects to categorize the entire application is not excluded.

[3] Since the application of a birthmark could be used as a method to detect similar software, it is mentioned for completeness. For an overview, the SLR on Software Birthmark published in 2019 by Nazir et al. (2019) can be considered.

to stop the process if no new results are found after a search operation, such as the snowball search.

In advance, it was determined that the listed works for the forward snowballing on the platforms vary in part. The different literature sources provided different citations and references. For example Google Scholar, which was proposed by Wohlin (2016) in case of snowballing-related search, provides in many cases more results for each paper and in some cases also less than the actually identified literature source of the paper. Therefore, the procedure was chosen to use both the origin page of the retrieved paper, if it offers the function of searching for citing works, and Google Scholar as a supplementary source.

## 3. Results

The SLR was executed according to the procedure described in the previous section. The results are shown in the following sections. This includes an overview of the process steps and the examined publications as well as the synthesized data.

The review itself was carried out over a period of approximately 4 months. The initial search of the described search terms in all described databases was done between 2019-03-28 and 2019-03-30. The following level 1 snowballing was performed until 2019-06-16. The subsequent level 2 snowballing was completed in 2019-07-21, while the remaining level 3 to level 5 snowballing work was done until 2019-08-07. The first search using the defined search terms resulted in 3387 items, which were filtered in several steps to 58 relevant papers. For filtering purposes, duplicates were first automatically removed on the basis of the available DOI numbers. This was followed by a manual review on the content of each title. For papers that could not be excluded unambiguously, a further check was carried out by using the abstracts. Afterwards, all remaining work was completely inspected. The main reason for this was the often ambiguous abstracts, which generally exhibit variable quality in the area of software engineering according to Budgen and Brereton (2006). On the final result of 58 papers the planned snowballing was carried out. On the first level a total of 3066 referenced and referencing papers were checked. The completed second level of the snowballing included another 2169 papers for inclusion and exclusion, while in the following levels 1146 and 486 papers were inspected. The final result of the searches and filters was 136 publications.

Fig. 2 summarizes this process steps and describes the number of works examined as well as the resulting numbers by filtering and snowballing.

Both the search process, including snowballing, and the resulting 136 publications were inclusive of studies that probably did not undergo a peer-review process. These papers are master theses, student projects, patents and other papers that were not published by any apparent publisher. However, possible papers that were written in addition to the patent were treated as independent papers in this literature search. We hereby follow the recommendation of Kitchenham and Charters (2007) to search other sources of evidence as well. This procedure is intended to provide a more complete picture of the current field of research. In order to inform the reader of this fact, all mentions of these studies will be followed by a corresponding note.

### 3.1. Overview

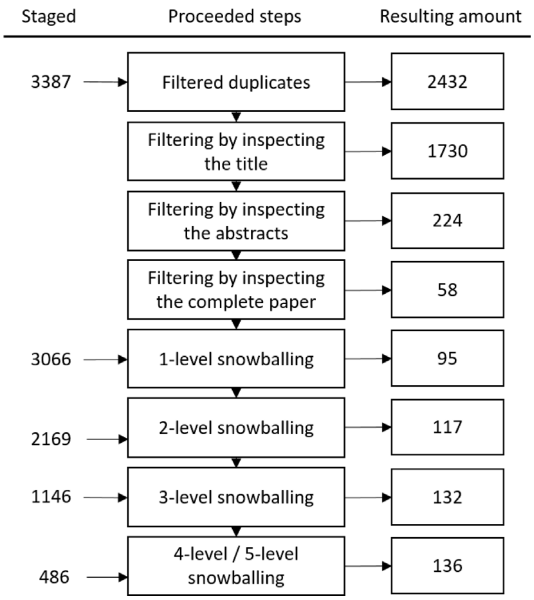

**Fig. 2.** Proceeding of the steps in the SLR process.

**Table 2**
Targeted domains and frequency of occurrences of found publications.

| Primary objectives of findings | Occurrence |
|---|---|
| Software categorization | 66 |
| Project management action recommendation/ Project monitoring/Success determination | 3 |
| Similar application detection | 20 |
| Software library categorization/recommendation | 4 |
| Tag-/Topic-/Software domain recommendation | 9 |
| Requirements-/Software architecture elicitation | 4 |
| Maintenance facilitation/defect prediction/ usability evaluation/quality assurance | 4 |
| Mobile app recommendation | 13 |
| Relevant software/Feature recommendation | 7 |
| Development improvement/onboarding/education | 6 |

The introduced, broad range of possible use cases for software similarity on the field of software engineering becomes visible by the rough categorization of the goals of all found publications from this SLR. The categories describing the main domain of the work are listed with the assigned frequency of occurring results in Table 2. It became clear that not all publications exclusively focus on the detection of similar software projects or some form of categorization, such as supervised classification or unsupervised clustering. Some calculate the similarity between software as an intermediate step to, for example, feed a recommendation system with appropriate data. Among other things, some of them tried to support certain process steps in the field of software engineering. Some of the work tried to support certain process steps in the area of software engineering. Even the automated recommendation of tags, which on some platforms among other things lead to the improvement of search results, is the goal of some discovered publications. GitHub has already tried an approach described in 2017 by Ganesan (2017), in which they launched *Topics* as a new feature and for tagging the repositories they recommend these tags to it's users. In this respect of tag-recommendation, some rely on the recommendation of tags that are used in similar software projects. Therefore, this part of the research area is also relevant in this literature review.

It is noticeable that the majority of the work concentrates mainly on a form of categorization of software. This categorization is achieved either by an automated form of supervised classification or unsupervised clustering. Which of the publications pursues which primary objectives is shown partly below in Section 3.2 within the detailed description.

For the detection of similar software, different methods for information extraction and categorization were used. Some more

**Table 3**
Extracted amount of explicitly mentioned categorization or similarity detection methods used by the approaches.

| Described methods for categorization | Occurrence |
|---|---|
| K-nearest neighbour (kNN), IBK, Weighted nearest neighbour | 20 |
| Support Vector Machine (SVM), TreeRankSVM | 19 |
| Naïve bayes, Multinomial naïve bayes, Tree augmented naïve (TAN) | 18 |
| Decision Tree, Boosted decision tree | 16 |
| Artificial neural network (ANN), DNN, CNN, Fast RCNN, C-LSTM, Multilayer perceptron, Kohonen clustering networks | 13 |
| K-means clustering, Spherical k-means, K-means++, K-medoids | 12 |
| Hierarchical clustering | 9 |
| Bayesian networks, Boosted bayesian networks, K2 | 8 |
| Graph-based/Ontology-based classification | 8 |
| Random forest (RF) | 8 |
| Collaborative filtering (CF) | 5 |
| Dirichlet process clustering (DPC) | 3 |
| Expectation maximization (EM) clustering | 1 |
| RIPPER | 1 |
| Voting feature intervals (VFI) | 1 |

**Table 4**
Amount of extracted information retrieval methods used by the approaches.

| Described methods | Occurrence |
|---|---|
| Term frequency–inverse document frequency (TF–IDF) | 33 |
| Latent Dirichlet allocation (LDA),<br>LDA-TR, LDA-GA, L-LDA, CombineLDA | 32 |
| LSI/LSA, PLSA | 15 |
| BM25 weighting function | 1 |
| Collaborative topic modelling | 1 |
| Relative category frequency (RCF) | 1 |

**Table 5**
Number of software-related data grouped into categories used by the relevant publications.

| Information categories | Occurrence |
|---|---|
| Online software profiles | 73 |
| Software code related | 61 |
| Additional information/semantic categories | 29 |
| Calculated metrics | 15 |
| Repository activities | 13 |
| Mobile-specific data | 8 |
| Issue-related sources | 4 |

frequently than others. For a better overview, these methods were extracted and an aggregated overview is provided in Tables 3 and 4. While Table 3 describes the usage of typical categorization and filtering techniques, Table 4 summarizes information retrieval (IR) and ranking functions. These IR techniques are sometimes only used in the context of pre-processing, for example for data extraction. In some approaches, however, these techniques are also used as a mechanism for finding similar software. For example Zhang et al. (2014) use the *Term Frequency–Inverse Document Frequency* (TF–IDF) score as a "relevance scoring" without an additional classification or clustering step. In addition, it was not possible to identify the exact method from every paper, since some papers do not give details, but only describe an unspecific categorization, such as "Clustering" Venkataramani et al. (2013).

As a further evaluation on the research field for software similarity, the publication years of the identified research papers were analysed. An increasing interest in the field can be observed by looking at the diagram of Fig. 3. While before 2010 no more than two publications per year were found to be relevant for this review, the trend continued to increase in the following years and reached a temporary peak in 2018. Since the original literature search for this work was carried out at the end of the first quarter of 2019 and snowballing in the second and parts of the third quarter, only six publications were found for 2019.

In order to get an overview of the applied approaches for similarity detection, the identified 136 publications are shortly described and afterwards summarized in the following.

### 3.2. Similarity detection approaches

For the automated calculation of software similarities, many different approaches have already been applied, which differ according to the targeted domains from Table 2, the used data and the applied methodology. Almost 50 different aspects of software-related data for similarity analysis were identified in the SLR-results. All of the data was taken from the software code itself, calculated metrics, online software profiles, issue-related sources, activities on repository-platforms, mobile-specific data (e.g. sensor-usages) and further additional information as well as semantic categories. In Table 5 the total number of number of utilized data grouped into categories is summarized.

In order to provide an overview of all the work identified in the field of software engineering in advance, a taxonomy was developed into which each work can be classified. The categories *Sources, Retrieved Input, Categorization method* and *Research target* were thereby selected for a comprehensive classification. While *Sources* and *Retrieved Input* describe the data source and the information used in the study, *Categorization method* shows the methods used to process the data in order to identify similar software. All relevant methods were listed, which we were able to identify and which could be classified into the defined taxonomy. However, some approaches do not use typical machine learning methods for categorization, but limit the work to IR-specific methods. These methods include topic detection (e.g. LSI or LDA), term frequencies like TF–IDF, relative category frequency, semantic filtering, weighting and ranking functions like BM25. In the case that approaches rely solely on categorization methods, they are grouped under the corresponding position in the taxonomy. With this taxonomy it is possible to classify the different approaches and get an overview. In order to achieve a better understanding of each work, the taxonomy also contains the *Research target*. This is to show what the overall goal of the applied approach in the study was. The Tables 6, 7, and 8 present the articles and a classification of their approaches within the dimensions of our taxonomy. In this regard, the publications are sorted by publication date in order to identify possible trends in the procedures. Within the publication years, the studies were again sorted by research target and sources.

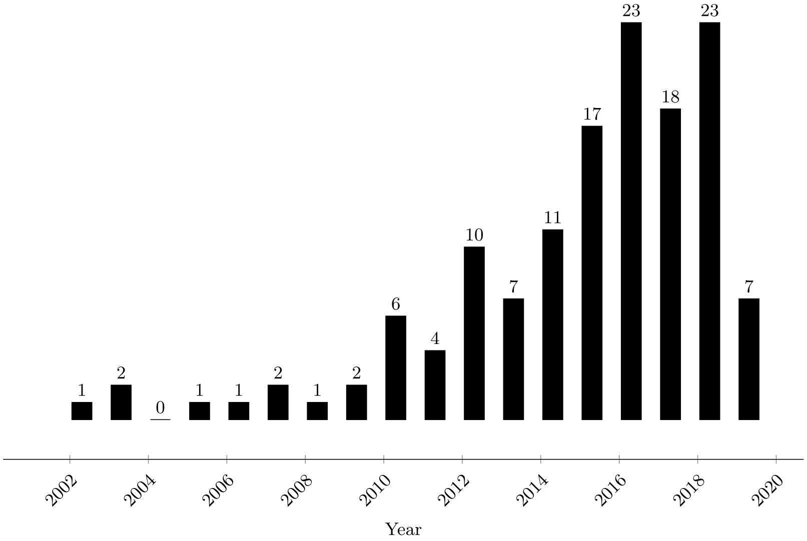

**Fig. 3.** Distribution of relevant publications by year.

Below is a brief description of each relevant publication. Although the studies often rely on multiple outlined categories of Table 2, they are described according to their focus on used data as far as possible. Complementary the most relevant results of the studies will be discussed in the subsequent section.

**Based on software engineering processes:** Some of the publications concentrate on metrics from different areas of software engineering in order to establish a measure of similarity between software projects and thus classify software or support a process area of software engineering.[4] We can differentiate between metrics related to project management, development-related metrics and platform-related metrics, which can be collected from the software project. From the SLR results, two publications rely on a combination of various project management metrics, development-related metrics and platform-related metrics. While Haitao et al. (2012) performed a similarity calculation using euclidean distance similarity and weighted euclidean distance similarity on these metrics, Liu et al. (2018) described another approach based on a neural network. The exclusive use of metrics related to project management is proposed in a patent by Creel and Chappel (2008)[5] as they calculated the vector distance on potential success indicators and then performed a clustering using Kohonen computation. Srinivas and Rao (2015) also applied clustering on this type of metrics, but using k-means. While Barreto and Rocha (2010), on the other hand, described a classification approach on certain project management-related metrics based on weighted nearest neighbour. The approaches of Paithankar and Ingle (2009), Allaho (2014) and Prieto (2014),[6] rely on a combination of project management-related metrics and calculated metrics from source code, such as lines of code (LoC), byte-size, McCabe's cyclomatic complexity, PRG-file-number, reports-number, interface-number and usability attributes. Different from the others, the procedure of Prieto is described in a patent. Other approaches that also use source code metrics, but do not use other metrics from the software engineering process, are described in the following section.

**Based on source code:** A larger part of the SLR-relevant publications rely on the source code itself for a similarity calculation. This similarity, which relies on code-related characteristics of the software, can provide different results than e.g. the similarity measure based on project management metrics described before. In comparison to other sources of information, the data extracted from source code is based on the realization-level of the applications. Therefore, the similarity analyses based on source code are applied on a lower abstraction level compared to meta-information or architectural documentations of the software. Similar to the approaches of the previous chapter, some studies also use metrics, but directly from the source code. Code-related metrics are thereby calculated as key figures directly from the implemented code. Since 2010, several approaches have been published that establish a similarity of software by source code metrics alone. This has been presented using classifiers such as multilayer perceptron, bayesian network, *k-nearest neighbours* (kNN) and J48 by Yusof and Ramadan (2010) and discriminant analysis, C4.5 decision tree and kNN by Yusof and Rana (2010). Hierarchical clusterings and k-means clustering was performed on such metrics by Santos et al. (2014) and Jureczko and Madeyski (2010). Another approach was taken by Guendouz et al. (2015), who used a collaborative filtering technique. In contrast to the previous approaches, Yaremchuck et al. (2018) described in their publication a combination of source code metrics and information from the source code itself. The metrics refer to reliability indicators. In contrast to these approaches, other studies were identified using general terms, keywords, comments, function names, constant names, variable names, function arguments, etc. from source code. Based on this data, two publications of Kawaguchi et al. (2003), Kawaguchi et al. (2006) describe a focus on source code with the aim of categorizing software. In these frequently referenced papers, they describe the usage of *latent semantic analysis* (LSA) on code-specific keywords or "identifiers" to classify software. In a similar way, Sandhu et al. (2007a) used LSA as well to identify keywords in source code and applied a classifier like naïve bayes and decision trees with a cosine similarity for calculation. Further

[4] The research field of effort estimation for software projects also frequently applies project management metrics to support project planning. Since these publications which are focused on effort estimation were excluded by the SLR criteria, only a fraction of the work based on project management is listed here.

[5] The approach was published as a patent and therefore appears to lack a peer-review.

[6] The approach was published as a patent and therefore appears to lack a peer-review.

**Table 6**
Classification of all found approaches into our taxonomy sorted by year.

| Approaches | Research target | | | | | | | | | | Sources | | | | | Retrieved Input | | | | | | | Categorization method | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software categorization | Project management/monitoring | Similar application detection | Software library categorization | Tag/Domain recommendation | Requirement/Architecture elicitation | Maintenance facilitation/... | Mobile app recommendation | Relevant software/Feature recom. | Development improvement/support | Software projects/images | Mobile applications | Project metadata/documentation | Domain knowledge/context logs | Developer/user-generated information | Source-/Bytecode[a] | Project-metrics | Documentation | Software profile[b] | Open issues/Bugs | Repository activities | Runtime data[c] | NB/Bayesian network/VFI classifier | K-nearest neighbour classifier | Support vector classifier | Decision tree classifier | (Deep) Neural network classifier | Graph-based classification | Random forest classifier | Rule based classifier | K-means variants/DPC/EM clustering | Hierarchical clustering | Collaborative filtering | Information retrieval methods[d] |
| 2002 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ugurel et al. (2002) | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | ● | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · |
| 2003 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Kawaguchi et al. (2003) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Krovetz et al. (2003) | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | ● | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · |
| 2005 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Inoue et al. (2005) | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · |
| 2006 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Kawaguchi et al. (2006) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| 2007 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Sandhu et al. (2007a) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · |
| Sandhu et al. (2007b) | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · |
| 2008 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Creel and Chappel (2008)[e] | · | ● | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · |
| 2009 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Tian et al. (2009) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Paithankar and Ingle (2009) | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | ● | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| 2010 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Yusof and Ramadan (2010) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | ● | · | ● | ● | · | · | · | · | · | · | · |
| Yusof and Rana (2010) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | ● | · | ● | · | · | · | · | · | · | · | · |
| Shabtai et al. (2010) | ● | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | ● | · | · | ● | · | · | ● | · | · | · | · | · |
| Barreto and Rocha (2010) | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |
| Jureczko and Madeyski (2010) | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · |
| Grechanik et al. (2010) | · | · | · | · | · | · | · | · | ● | · | ● | · | ● | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| 2011 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| McMillan et al. (2011) | ● | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | ● | · | · | · | · | · | · | ● | · | ● | ● | · | · | · | · | · | · | · | · |
| Kanda et al. (2011) | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Kelly et al. (2011) | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Dumitru et al. (2011) | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |

(continued on next page)

**Table 6** (*continued*).

| Approaches | Research target | | | | | | | | | | Sources | | | | | Retrieved Input | | | | | | | Categorization method | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software categorization | Project management/monitoring | Similar application detection | Software library categorization | Tag/Domain recommendation | Requirement/Architecture elicitation | Maintenance facilitation/... | Mobile app recommendation | Relevant software/Feature recom. | Development improvement/support | Software projects/images | Mobile applications | Project metadata/documentation | Domain knowledge/context logs | Developer/user-generated information | Source-/Bytecode[a] | Project-metrics | Documentation | Software profile[b] | Open issues/Bugs | Repository activities | Runtime data[c] | NB/Bayesian network/VFI classifier | K-nearest neighbour classifier | Support vector classifier | Decision tree classifier | (Deep) Neural network classifier | Graph-based classification | Random forest classifier | Rule based classifier | K-means variants/DPC/EM clustering | Hierarchical clustering | Collaborative filtering | Information retrieval methods[d] |
| 2012 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Haitao et al. (2012) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| McMillan (2012a) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| McMillan (2012b) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Yusof et al. (2012) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · |
| Yang and Tu (2012) | ● | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Sanz et al. (2012) | ● | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | ● | · | · | · | ● | ● | ● | ● | · | · | ● | · | · | · | · | · |
| Zhu et al. (2012) | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · |
| Thung et al. (2012) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| McMillan et al. (2012) | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Wang et al. (2012) | · | · | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| 2013 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Venkataramani et al. (2013) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Wang et al. (2013) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · |
| Qiu et al. (2013) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Thung et al. (2013) | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | ● | · |
| Bhandari et al. (2013) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Yin et al. (2013) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Wang et al. (2013) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| 2014 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Grechanik (2014)[e] | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Wang et al. (2014b) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | ● | · | · | · | ● | ● | ● | · | · | · | · | · | · | ● | · | · |

[a] Beside source and binary code, also configuration files, file-/folder-structure, api-usage/references, code- and platform-related metrics as well as mobile app configuration are included.

[b] Includes tags, reviews, wikis, search engine results, descriptions, licensing, offered in-app purchases.

[c] Includes usage-/context-logs, recorded time/usage duration and UI-images.

[d] Includes topic detection, term frequencies, relative category frequency, semantic filtering, ranking functions.

[e] The work appears to lack a peer-review.

**Table 7**
Continuing table on classification of all found approaches into our taxonomy sorted by year.

| Approaches | Research target | | | | | | | | | | Sources | | | | | Retrieved Input | | | | | | | Categorization method | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software categorization | Project management/monitoring | Similar application detection | Software library categorization | Tag/Domain recommendation | Requirement/Architecture elicitation | Maintenance facilitation/... | Mobile app recommendation | Relevant software/Feature recom. | Development improvement/support | Software projects/images | Mobile applications | Project metadata/documentation | Domain knowledge/context logs | Developer/user-generated information | Source-/Bytecode[a] | Project-metrics | Documentation | Software profile[b] | Open issues/Bugs | Repository activities | Runtime data[c] | NB/Bayesian network/VFI classifier | K-nearest neighbour classifier | Support vector classifier | Decision tree classifier | (Deep) Neural network classifier | Graph-based classification | Random forest classifier | Rule based classifier | K-means variants/DPC/EM clustering | Hierarchical clustering | Collaborative filtering | Information retrieval methods[d] |
| Linares-Vásquez et al. (2014) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | ● | ● | ● | · | · | · | · | · | · | · | · |
| Vakulenko et al. (2014) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Zhu et al. (2014) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · |
| Zhang et al. (2014) | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | ● | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Prieto (2014)[e] | · | ● | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Allaho (2014) | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | ● | ● | · | · | ● | ● | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Mens et al. (2014) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Wang et al. (2014a) | · | · | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · |
| Santos et al. (2014) | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · |
| 2015 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Baldrich (2015)[e] | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Kim et al. (2015) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | · | · |
| Avila (2015)[e] | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Escobar-Avila et al. (2015) | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Berardi et al. (2015) | ● | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | ● | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · |
| Chen et al. (2015) | ● | · | · | · | · | · | · | · | · | · | · | ● | ● | · | ● | ● | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Shewale et al. (2015) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | ● | ● | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · |
| Srinivas and Rao (2015) | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Hernández and Costa (2015) | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Bu et al. (2015) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Escobar-Avila (2015) | · | · | · | ● | · | · | · | · | · | · | ● | · | ● | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Vargas-Baldrich et al. (2015) | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Portugal et al. (2015) | · | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Sun et al. (2015) | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Su et al. (2015) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Yang et al. (2015) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |
| Guendouz et al. (2015) | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · |
| 2016 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Liao et al. (2016) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · |

(*continued on next page*)

**Table 7** (*continued*).

| Approaches | Research target | | | | | | | | | | Sources | | | | | Retrieved Input | | | | | | | Categorization method | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software categorization | Project management/monitoring | Similar application detection | Software library categorization | Tag/Domain recommendation | Requirement/Architecture elicitation | Maintenance facilitation/... | Mobile app recommendation | Relevant software/Feature recom. | Development improvement/support | Software projects/images | Mobile applications | Project metadata/documentation | Domain knowledge/context logs | Developer/user-generated information | Source-/Bytecode[a] | Project-metrics | Documentation | Software profile[b] | Open issues/Bugs | Repository activities | Runtime data[c] | NB/Bayesian network/VFI classifier | K-nearest neighbour classifier | Support vector classifier | Decision tree classifier | (Deep) Neural network classifier | Graph-based classification | Random forest classifier | Rule based classifier | K-means variants/DPC/EM clustering | Hierarchical clustering | Collaborative filtering | Information retrieval methods[d] |
| Dong et al. (2016) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · |
| Zhou (2016)[e] | ● | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Yuan et al. (2016) | ● | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | ● | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · |
| Hao et al. (2016) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | ● | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · |
| Radosavljevic et al. (2016) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Al-Subaihin et al. (2016) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · |
| Xin Li et al. (2016) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · |
| Liu et al. (2016) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Babatunde (2016)[e] | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · |
| Lavid Ben Lulu and Kuflik (2016) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Ye et al. (2016) | · | · | ● | · | · | · | · | · | · | · | ● | · | ● | · | · | ● | · | ● | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · |
| Linares-Vásquez et al. (2016) | · | · | ● | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Bu et al. (2016) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Ma et al. (2016) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Petrovic et al. (2016) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · |
| Chen et al. (2016) | · | · | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Cai et al. (2016) | · | · | · | · | ● | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |
| Yu et al. (2016) | · | · | · | · | · | ● | · | · | · | · | · | ● | ● | · | · | ● | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |
| Pan et al. (2016) | · | · | · | · | · | · | · | ● | · | · | · | ● | ● | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Sanap and Vaidya (2016) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Li et al. (2016) | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Yang et al. (2016) | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · |

[a] Beside source and binary code, also configuration files, file-/folder-structure, api-usage/references, code- and platform-related metrics as well as mobile app configuration are included.

[b] Includes tags, reviews, wikis, search engine results, descriptions, licensing, offered in-app purchases.

[c] Includes usage-/context-logs, recorded time/usage duration and UI-images.

[d] Includes topic detection, term frequencies, relative category frequency, semantic filtering, ranking functions.

[e] The work appears to lack a peer-review.

**Table 8**
Continuing table on classification of all found approaches into our taxonomy sorted by year.

| Approaches | Research target | | | | | | | | | | Sources | | | | | Retrieved Input | | | | | | | Categorization method | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software categorization | Project management/monitoring | Similar application detection | Software library categorization | Tag/Domain recommendation | Requirement/Architecture elicitation | Maintenance facilitation/... | Mobile app recommendation | Relevant software/Feature recom. | Development improvement/support | Software projects/images | Mobile applications | Project metadata/documentation | Domain knowledge/context logs | Developer/user-generated information | Source-/Bytecode[a] | Project-metrics | Documentation | Software profile[b] | Open issues/Bugs | Repository activities | Runtime data[c] | NB/Bayesian network/VFI classifier | K-nearest neighbour classifier | Support vector classifier | Decision tree classifier | (Deep) Neural network classifier | Graph-based classification | Random forest classifier | Rule based classifier | K-means variants/DPC/EM clustering | Hierarchical clustering | Collaborative filtering | Information retrieval methods[d] |
| 2017 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bodó and Indurkhya (2017) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | ● | ● | ● | · | · | ● | · | · | · | · | · |
| Catal et al. (2017) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | ● | ● | ● | · | · | ● | · | · | · | · | · |
| Nguyen and Nguyen (2017) | ● | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · |
| Surian et al. (2017) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · |
| Yoon et al. (2017) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Chen et al. (2017b) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | ● | ● | ● | · | · | · | · | · | · | ● | · | · |
| Chen et al. (2017a) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | ● | ● | ● | · | · | · | · | · | · | ● | · | · |
| Sharma et al. (2017) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Soll and Vosgerau (2017) | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | ● | ● | ● | · | · | ● | ● | ● | · | · | ● | · | ● | · | · | · | · | · | · | · | · |
| Xu et al. (2017a) | · | · | ● | · | · | · | · | · | · | · | ● | · | ● | · | ● | ● | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Xu et al. (2017b) | · | · | ● | · | · | · | · | · | · | · | ● | · | ● | · | ● | ● | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Zhang et al. (2017) | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Yu et al. (2017) | · | · | · | ● | · | · | · | · | · | · | · | ● | ● | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · |
| Portugal et al. (2017) | · | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Cao et al. (2017) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |
| Rustgi et al. (2017) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Yao et al. (2017) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Thung et al. (2017) | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| 2018 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Altarawy et al. (2018) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · |
| Ma et al. (2018) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | · | ● | ● | · | · | ● | · | · | · | · | · |
| Kim et al. (2018) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · |
| Leclair et al. (2018) | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | ● | · | ● | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · |
| Singla et al. (2018a) | ● | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | · | · | ● | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · |
| Reyhani Hamedani et al. (2018) | ● | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | ● | · | ● | · | · | · | ● | ● | ● | · | ● | · | ● | · | · | · | · | · |
| Singla et al. (2018b) | ● | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · |
| Mingshan Jr (2018)[e] | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | ● | ● | · | · | ● | · | · | · | · |
| ZhangChao and WanLili (2018) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Chao and Lili (2018) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |

(continued on next page)

**Table 8** (*continued*).

| Approaches | Research target | | | | | | | | | | Sources | | | | | Retrieved Input | | | | | | | Categorization method | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Software categorization | Project management/monitoring | Similar application detection | Software library categorization | Tag/Domain recommendation | Requirement/Architecture elicitation | Maintenance facilitation/... | Mobile app recommendation | Relevant software/Feature recom. | Development improvement/support | Software projects/images | Mobile applications | Project metadata/documentation | Domain knowledge/context logs | Developer/user-generated information | Source-/Bytecode[a] | Project-metrics | Documentation | Software profile[b] | Open issues/Bugs | Repository activities | Runtime data[c] | NB/Bayesian network/VFI classifier | K-nearest neighbour classifier | Support vector classifier | Decision tree classifier | (Deep) Neural network classifier | Graph-based classification | Random forest classifier | Rule based classifier | K-means variants/DPC/EM clustering | Hierarchical clustering | Collaborative filtering | Information retrieval methods[d] |
| 2017 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Bodó and Indurkhya (2017) | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | ● | ● | ● | · | · | ● | · | · | · | · | · |
| Krishna and Babu (2018) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · |
| Sun et al. (2018) | · | · | ● | · | · | · | · | · | · | · | ● | · | ● | · | ● | ● | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Nguyen et al. (2018a) | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Su (2018) | · | · | ● | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · |
| Nguyen et al. (2018b) | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | ● | ● | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Yin et al. (2018) | · | · | · | · | ● | · | · | · | · | · | ● | · | ● | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Yin et al. (2018) | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Chen et al. (2018) | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● |
| Humm and Ossanloo (2018) | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | · | · | · | · | ● | · | · | · | ● | · | · | ● | ● | · | · | · | · | · | · | · |
| Zheng et al. (2018) | · | · | · | · | · | · | · | · | ● | · | · | · | ● | · | ● | · | · | · | ● | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Nafi et al. (2018) | · | · | · | · | · | · | · | · | ● | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| Liu et al. (2018) | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | ● | ● | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · | · |
| Yaremchuck et al. (2018) | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · | · |
| 2019 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Al-Subaihin et al. (2019) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Ochiai et al. (2019) | ● | · | · | · | · | · | · | · | · | · | · | · | ● | ● | · | · | · | · | ● | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · |
| Hamednai et al. (2019) | · | · | ● | · | · | · | · | · | · | · | · | ● | ● | · | · | ● | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · |
| Zhou et al. (2019) | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | · | · | · | ● |
| Nadezhda et al. (2019) | · | · | · | · | · | ● | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | ● | · | · | · | · | · | · |
| Raja and Pushpa (2019) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | · | · | · | · | ● | · | · | · | · | · | · | · | · | · | · | · | ● | · | · | · |
| Liu et al. (2019) | · | · | · | · | · | · | · | ● | · | · | · | · | ● | · | ● | · | · | · | ● | · | · | ● | · | · | · | · | · | · | · | · | · | · | ● | · |

[a] Beside source and binary code, also configuration files, file-/folder-structure, api-usage/references, code- and platform-related metrics as well as mobile app configuration are included.

[b] Includes tags, reviews, wikis, search engine results, descriptions, licensing, offered in-app purchases.

[c] Includes usage-/context-logs, recorded time/usage duration and UI-images.

[d] Includes topic detection, term frequencies, relative category frequency, semantic filtering, ranking functions.

[e] The work appears to lack a peer-review.

LSA-based approaches for the recognition of relevant terms in source code were presented by Inoue et al. (2005) and Yusof et al. (2012) as well as Sandhu et al. (2007b). The latter additionally used *probabilistic latent semantic analysis* (PLSA) and described the usage of naïve bayes classifier. Instead of LSA, the newer *latent Dirichlet allocation* (LDA) and a further similarity analysis was presented by Altarawy et al. (2018). They applied hierarchical clustering with cosine similarity and j Jensen–Shannon divergence on the extracted terms. Earlier approaches proposed the application of similar methods as well. Yang and Tu (2012) as well as Sun et al. (2015) applied LDA and subsequently cosine similarity. While Yang and Tu wanted to support software maintenance tasks with their approach, Sun et al. aimed to categorize the software. An automatic categorization was also targeted by Tian et al. (2009) as well as Kelly et al. (2011). To achieve this goal, Tian et al. extracted domain vocabulary from source code by utilizing LDA. A recommendation of domain-related topics of software projects was aimed by Kelly et al. who used a k-means clustering on the basis of the determined topics to compare current domain concepts. They were also using LDA for the recognition of these topics. As another, primary goal Li et al. (2016) aimed for an improvement of the education of developers, developer-onboarding and the actual development work. For this purpose they analysed function arguments in source code to determine behavioural similarity. Therefore, they used different metrics, including single-programme symbolic execution (SSE), paired-programme symbolic execution (PSE) and random sampling (RS). The execution of behavioural patterns was also targeted by Liao et al. (2016). To categorize software, they applied an unsupervised hierarchical clustering combined with a Levenshtein distance on the extracted pattern-usage. In contrast, supervised approaches were discussed by Bodó and Indurkhya (2017) as well as Catal et al. (2017) to categorize the software. Bodó and Indurkhya also used low level source code features like names of variables, methods, classes, packages and modules to evaluate different classifiers like SVM, random forest, decision tree and kNN. Catal et al. used similar data and evaluated the classifiers SVM, naïve bayes, J48, kNN and random forest. In addition to this work, the other published code-based approaches use a combination of source code and other data to calculate similarity between software.

Attempts based on source code and software descriptions were increasingly carried out. Ugurel et al. (2002) published results on similarity calculations between software based on certain code aspects including code comments and readme-files. The same authors published another paper (Krovetz et al., 2003) in the following year in which they also included programme variables in the analysis. More unspecifically described, Leclair et al. (2018) recently also used source code in combination with software descriptions using a neural text classification, which is according to the author similar to *CLSTM*.[7] To support requirements elicitation, Yu et al. (2016) applied a kNN and a self-adaptive similarity using data based on functional features in source code, software description and API calls in addition. Similar data is also used by Ye et al. (2016), where they explicitly analyse developer guides as software descriptions. They extract relevant terms using TF–IDF and calculate a similarity measure using *Support Vector Machine* (SVM) and cosine similarity. In another approach, Sun et al. (2018) and Xu et al. (2017a,b) did analyse the behaviour of developers alongside features extracted from the source code and description documents. They analysed repository activities which include aspects like stars, forks, watches, subscriptions, ownerships and the temporal behaviour when creating new repositories. The three approaches explicitly describe the use of stars, forks and repository creation using cosine similarity. As in the previous paper, they applied TF–IDF to the text corpus and combine calculated similarities, as in the case of Sun et al. in a project similarity matrix.

Other works that almost exclusively use repository activities and code-related data for the calculation of software similarity are described below. Similar to the previous data, Yuan et al. (2016) and Hamednai et al. (2019) use the source code in combination with an application description. However, their focus lies on mobile applications from an Android platform and therefore take additional information from the manifest.xml, which is a configuration file attached to the android applications. While Yuan el al. concentrate only on the app permissions defined in the manifest.xml via naïve bayes classifier, Hamednai, Kim and Cho use the whole manifest file and additionally the software name, software tags and assigned app store categories. For a similarity analysis TreeRankSVM[8] in combination with TF–IDF was applied. Sanz et al. (2012) made us of strings contained in an application's source code, as well as information about app permissions and services or rather features from the attached manifest file. As a further addition they also included user comments as part of the available data from an android app-market itself. For the categorization of the data they also compare different classifiers like random forest, weka's C4.5, kNN, bayesian network, tree augmented naïve (TAN), naïve bayes and SVM. Shabtai et al. (2010) were also investing in using data from the xml-based manifest file. They additionally use dex files from source code, general resource files available in apps and images as features, which are available in apks. McMillan (2012a) published a conference paper as well as a dissertational thesis (McMillan, 2012b) in which an approach based on LSA for the analysis of source code in combination with API calls was published. Unlike all previous authors, Nguyen et al. (2018a) did use graphs for their approach to calculate a graph-based similarity. This graph represents specific relations occurring between the source code files and connections between two developers. Hereby the assumption is made that same developers in software projects contribute to a certain level of similarity. In a second publication, Nguyen et al. (2018b) additionally included third-party libraries in the graph for the aimed categorization of software projects. Lately, this categorization is also achieved by Nguyen and Nguyen (2017) by using code tokens as features from source code and methods as well as classes from library-APIs. For this automatic categorization they applied a *deep neural network* (DNN). The last work identified that refers to the whole source code is Shewale et al. (2015), who extracts contextual information from mobile application code. In addition, the software name, usage records of the app users and search snippets are used. These snippets were extracted from a search for the app in a web search engine and should serve as an enrichment of the applied corpus. No direct analysis of the source code, but only file names and file extensions was done by Ma et al. (2018). They compared seven different machine learning classifiers, like decision tree, SVM and bayesian network to classify open source applications.

**Based on binary files:** Unlike the previous section, the following publications rely on binary files instead of source code. With such an approach, it is possible that information which would be useful for a similarity analysis is lost during the compilation process. On the other hand, for example one of the arguments of Avila (2015)[9]

[7] CLSTM or Contextual LSTM is an extension of the recurrent neural network LSTM model. According to Ghosh et al. (2016) LSTM stands for Long-Short Term Memory and incorporates contextual features (e.g., topics) into the model.

[8] TreeRankSVM is a linear ranking support vector machine, described by Lee and Lin (2014).

[9] The approach was published as a master thesis and therefore appears to lack a peer-review.

is that the source code is not always available. The following described literature therefore present approaches for the application on closed source software. In 2015 some approaches applied TF–IDF in combination with other methods to achieve software categorization. Avila (2015) did analyses on vocabulary in the bytecode of libraries by applying TF–IDF, Dirichlet process clustering (DPC)[10] and cosine similarity. In contrast, Vargas-Baldrich et al. (2015) calculated relevance scoring using TF–IDF on the basis of class names, fields, method names and method arguments. At the same time another publication of Baldrich (2015)[11] was found within the SLR, in which he presented a graph-based approach for similarity calculation. For the software categorization he used TF–IDF and weight matching based on similar identifiers from project's bytecode as he applied in the previously described work. The published work of Su (2018) uses similarity functionality, execution and topic distributions in programme binaries for a software similarity. Methodically, Su applied kNN for a classification of the software.

Additional publications, which describe the use of binary files, are Berardi et al. (2015) and Pan et al. (2016). However, unlike the previous ones, they do not only use binaries, but also certain aspects of online software profiles. The focus of both works lies on mobile applications. Berardi et al. used the file-size from the binaries in combination with information from app descriptions, app names, app store categories, average user ratings and the amount of user ratings. For a desired software categorization they used SVM and a *best match 25* (BM25) weighting function. Pan et al. on the other hand, applied forms of LDA and a Jensen–Shannon distance metric to mobile app binaries and users' app descriptions. Escobar-Avila (2015) relied on a similar combination of information. In this second publication he used the clustering algorithm DPC and cosine similarity to calculate the similarity in a vector space model (VSM), as in the master thesis of Avila (2015) described above. Unlike before, he also used the software profile, which can contain arguments like app name and description. Similar to this, Escobar-Avila et al. (2015) published an approach using cosine similarity, TF–IDF, vectors built in a VSM, and a final clustering by DPC.

Finally the conference papers of Zhou et al. (2019) and Yin et al. (2018), Yin et al. (2018) also need to be listed in this section. These paper all use information from docker repositories or simply docker software to introduce a recommendation function of suitable tags. Also they all used *labelled LDA* (L-LDA) for topic modelling to pass on suitable tag sets. In a latter literature of Yin et al. (2018) the usage of app descriptions and the calculation of a similarity based rankings between two repositories was additionally discussed. For the calculation a jaccard similarity, described by Niwattanakul et al. (2013), was applied to generate parameter vectors.

**Based on online software profiles:** Some work relies mainly on online software profiles, which can include software descriptions and documentation of developers or users, the name or title of the software, collections of user-/developer-defined software tags or even automatically generated tags, as well as categories of marketplaces and repository-platforms. Additional information was used by analysing version-descriptions or update information such as publicly available change-logs or commit-messages, application-reviews, user ratings, any kind of comments provided by the source-platform and even web-links pointing to the software. The data available here have a more abstract meta-level focus than source code or code-based metrics. Information that might not be available through the actual implementation can be used. However, the same applies in the other direction: Conclusions about the actual implementation of the software may only be possible to a limited extent. The following 15 found studies, all published between 2011 and 2018, exclusively use textual software descriptions. However, many of them applied different methodologies. Dumitru et al. (2011) can be listed first, as their goal was the recommendation of features identified in similar products. For this they used kNN in connection with a cosine similarity. With the aim of a mobile app recommendation, Yin et al. (2013) described an approach with function overlap and also a calculated cosine similarity. Vakulenko et al. (2014) also implemented a mobile app recommendation using an overlap coefficient and a topic modelling based on LDA. Whereas Su et al. (2015) applied LDA and subsequently a cosine similarity for a software categorization.

TF–IDF was also increasingly used to identify relevant terms in software descriptions. While Wang et al. (2013) also used intent relation between mobile apps and a jaccard similarity coefficient, Yang et al. (2015) and Zhou (2016),[12] Radosavljevic et al. (2016) and Al-Subaihin et al. (2016) used exclusively the software descriptions and applied TF–IDF on them. To implement a mobile app recommendation, Wang et al. used a jaccard similarity coefficient. For a software categorization Yang et al. presented an approach that includes cosine similarity, *kNN classification based on stratified sampling* (SS-kNN) and coverage-weighted similarity besides TF–IDF. Clustering using k-means was used by Zhou, while Al-Subaihin et al. described several clusterings on software, such as hierarchical clustering, spherical k-means (skmeans) and agglomerative hierarchical clustering. Subsequently, Hao et al. (2016) used a graph on the software descriptions and user's behaviour log to calculate similarity and categorization of software. Surian et al. (2017), on the other hand, followed an approach for identifying false categorizations in app markets by applying LDA to the texts and performing their own categorization using SVM and k-means clustering algorithms. More focus on mobile app recommendation based on app description was placed by Rustgi et al. (2017) and Yao et al. (2017). For this purpose they described the usage of an overlap measure or a cosine similarity. In 2018 a software categorization based on the software descriptions with different approaches was pursued. While Mingshan Jr (2018)[13] used decision trees (DT), artificial neural network (ANN), rule-based models and LSA to achieve the objectives, Al-Subaihin et al. (2019) compared several approaches with LDA, VSM, VSM with LSA, Finite State Machine (FSM) with collocations and FSM with Dependencies. Also LDA, but additionally using SVM and k-means clustering, was described by Krishna and Babu (2018).

In addition to the app description, Yu et al. (2017) also used third-party libraries of the mobile apps in 2017 to provide a software library categorization and recommendation for Android mobile app libraries. For this approach they used a collaborative filtering approach as well as LDA and calculated the cosine similarity on generated vectors.

Like the previously mentioned work of Berardi et al. (2015), the following literature also relies on user reviews, user ratings or comments from app-users in combination with other software profile information. The other profile information includes a software description, the name of the software and categories of the apps in app-markets. The publication of Chen et al. (2015),

[10] DPC or Dirichlet process clustering is referred to as a non-parametric clustering algorithm because it does not require the number of clusters as parameters, according to Avila (2015).

[11] The approach was published as a master thesis and therefore appears to lack a peer-review.

[12] The approach was published as a student project and therefore appears to lack a peer-review.

[13] The approach was published as a master thesis and therefore appears to lack a peer-review.

Bu et al. (2015) and Lavid Ben Lulu and Kuflik (2016) have the characteristic that they focus on mobile applications and besides the described attributes also include app images like ui-screenshots. Chen et al. use *scale invariant feature transform* (SIFT) descriptors for each screenshot image and additionally the list of developers involved and a list of required permissions of the app. Methodically TF–IDF and LDA with an euclidean distance were used to categorize the mobile applications. The only difference between Bu et al. is that they do not include the permissions of the app and only describe the usage of a cosine similarity. Lavid Ben Lulu and Kuflik also used several other arguments for a software categorization by applying a similarity calculation to a bag of words. The bag of words was additionally filled with information from descriptions of similar apps, number of downloads or rather pull requests, rating data, growth curve as a function of time, web-links referring to the application and "Bing search API"-descriptions. As images, all available icons of an app were taken. Ma et al. (2016) used a bag-of-words with data from software descriptions, app names, app market categories, user reviews and enriched information from search engine results. On this set of data, they applied a calculation of distance similarity. Most recently, Raja and Pushpa (2019) also described an approach that utilizes the application description page for each app that includes name, genre, average rating score and description exclusively. Similar to the previously described literature, Babatunde (2016)[14] focused on the categorization of mobile applications using the online profile data app name, description and content rating. Also similar is the methodical approach using TF–IDF to extract relevant keywords and then applying a classifier. In contrast to the others, they used naïve bayes and instead of software categories they used the licensing of mobile apps as well as whether they offer in-app purchases as categorical features. Bhandari et al. (2013) and Sanap and Vaidya (2016) limited themselves to app names, reviews which were handled as app summary and user comments compared to the previous work. Sanap and Vaidya additionally described only superficially the usage of additional information by extracted web based features in a first phase and contextual features in a second phase. However, it was not possible to identify a methodological approach that could be classified in this study. Bhandari et al. on the other hand, describe the application of TF–IDF with a subsequent calculation of similarity based on graphs. The last literature to be listed here which relies on application reviews comes from Cao et al. (2017). In addition to the reviews, they used app's version descriptions for a mobile app recommendation. For this recommendation they compared several approaches, which among others rely on a nearest neighbour method combined with LDA, *matrix factorization* (MF) and a combination of MF for latent factors representation and *evolution progress modelling (EPM)* for version series association.

A focus on words in app titles and words in app descriptions was pursued by Bu et al. (2016) and Yoon et al. (2017). Same with additional consideration of app usage logs described by Ochiai et al. (2019). Bu et al. used LSA, TF–IDF and a similarity calculation based on cosine similarity. Yoon et al. instead compared some clustering and classification methods. The paper of Ochiai et al. published in 2019, on the other hand, describes the application of a DNN for software categorization. Based on application descriptions and their tags, as well as semantic categories as facets, Humm and Ossanloo (2018) published in a book comparing 10 machine learning techniques. These techniques including DNN, bayesian classifiers and decision trees. As semantic categories, they used diverse software-related data like programming language and licensing.

A whole series of publications between 2012 and 2014 by Wang et al. (2012), Wang et al. (2013), Wang et al. (2014a) were identified as relevant in this study. In all three publications, they used textual project profiles, including software names, descriptions and labels or tags with different methods. While in the workshop paper of Wang et al. (2012) and the journal article of Wang et al. (2014a) they pursued a recommendation of tags comparing a new approach against SVM and *trace ratio criterion LDA* (TR-LDA), in the conference paper of Wang et al. (2013) they described a software categorization using TF–IDF and hierarchical categorization. Also a hierarchical categorization on software names, descriptions and tags was proposed by Chen et al. (2017b,a). In addition, they applied TF–IDF, SVM, kNN and naïve bayes to implement software categorization. They used the same data and the same methods to analyse system operation services and configuration management modules from repositories of Puppet, Ansible and Chef. Also a categorization on app information like title, description and market category was published by Liu et al. (2016) based on TF–IDF and cosine similarity.

As previously identified by Cao et al. (2017), three further papers were identified in this literature review which include change-logs, version-descriptions or update information in the similarity analysis. Chen et al. (2016) followed a tagging of mobile applications by analysing combined software descriptions and update-texts using TF–IDF. ZhangChao and WanLili (2018) and Chao and Lili (2018), on the other hand, had software categorization as their goal in two publications. While in the first journal article of ZhangChao and WanLili (2018) TF–IDF to app name, subtitle, description and update information was applied, in the second article of Chao and Lili (2018) a VSM and LSA was applied to the same information, enriched by, among others, the responsible software developers.

While the work discussed above used any kind of software description and documentation in general, some of the included literature described a concrete usage of readme files from repositories. These text files that typically introduce and explains a project, containing information that is required to understand what the project is about, how it is used and in general characteristics about the project. Portugal et al. (2015) analysed these readme files applying semantic filtering to support requirements elicitation, followed by a publication of Portugal et al. (2017), aiming a requirements elicitation by analysing readme files as well. Therefore, they applied TF–IDF combined with clustering techniques such as k-medoids and k-means as well as included additionally a GitHub relevance via the GitHub API to order the software projects. Meanwhile, Sharma et al. (2017) applied LDA-GA, VSM, TF–IDF and cosine similarity on readme files to categorize the software.

Another option identified was the use of readme-files as well as metadata related to activities around the repository. This information is typically provided via platforms like GitHub using their APIs and includes stars, forks, watches, creation time of repository, wiki, contributors and subscriptions. This starts with Zhang et al. (2017), who analysed readme files and GitHub stars using TF–IDF, VSM and cosine similarity. Petrovic et al. (2016) used repository forks, stars and user subscriptions, which they generally interpreted as user relationships. Descriptions such as readme files were also used. For this procedure they applied an ANN and calculated a similarity measure using euclidean distance. An even larger part of metadata was analysed by Soll and Vosgerau (2017), who used GitHub's API to include forks, stars, watchers and subscribers in a similarity calculation. In addition, they used information from the API, like the size of repository, if it has a website, available wikis and bugs. Furthermore, readme-files, extensions of available files and folder-names as repository-structure, as well as programming language, repository-names

[14] This work appears to lack a peer-review.

and commit-messages were used. This way they analysed, similar to some approaches based on source code or binaries, both low-level and high-level features for a similarity determination of software. For the analysis Soll and Vosgerau made use of decision trees and a kNN classifier in which they calculated the jaccard distance. For the prediction of relevance ratings between different software projects, Zheng et al. (2018) utilized LDA on readme-files, GitHub's topic-tags, repo-names as well as the owner of repositories. Further work that relies on repository activities will be discussed separately in one of the next sections.

Finally, five other papers are presented, which also used certain aspects from online software profiles and partly combined them with other related information than described earlier in this section. Cai et al. (2016) utilized domain knowledge extracted from repositories, software tags of other software and community questions to recommend suitable tags. These software-related questions were determined from platforms, like GitHub and StackOverflow. For extraction and further processing they used TF–IDF, LDA, kNN and L-LDA. Similarly, Thung et al. (2012) used tags of applications to characterize each of them by their set of tags. To finally determine similar applications, they applied a cosine similarity. The same way Mens et al. (2014) used cosine similarity on tags from SourceForge to determine similar applications. For this purpose TF–IDF was applied additionally. Liu et al. (2019) also used cosine similarity, but in combination with collaborative filtering, to generate recommendations for mobile applications. For this purpose, a record was created for each application, consisting of user id, app name, record time, and usage duration. Singla et al. (2018a) on the other hand used a fast region-based convolutional neural network (Fast R-CNN) to categorize apps by analysing images of these apps and their textual metadata.

A combination of online software profiles and the API references described below were used by Wang et al. (2014b) and Thung et al. (2017). Wang et al. used profile attributes like the software description, collaborative tags and software categories from "high-level summaries" in combination with the evaluation of API-calls. This evaluation was done using a hierarchical categorization that included naïve bayes, kNN, SVM and LDA. Thung et al. used VSM and a cosine similarity to analyse project features like "textual description and keywords" as well as "API features".

**Based on API references:** The usage of information extracted from API calls, their terms, classes, packages and call-frequency was observed in various studies. Compared to source code analysis, a targeted analysis of API usage could focus more on the integration of the software rather than on the software itself. The analysis of similar interactions of software systems is therefore another approach that has been pursued by some publications. Like the previously described work of Wang et al. (2014b) and Thung et al. (2017), which also used online software profiles, 11 other papers were identified as relevant that made mainly use of API-references.

At first there is a publication of Grechanik et al. (2010) based on TF–IDF for the extraction of keywords of API calls and API documentations. To find highly relevant applications, a term-document-matrix score computation was executed. Grechanik (2014)[15] also got a patent which describes the usage of LSA, VSM and cosine similarity on packages and classes of the called APIs for categorization of software. McMillan et al. (2011) has published a conference paper with Linares-Vásquez, Poshyvanyk and Grechanik, who have also worked on other previously mentioned publications. In this conference paper, they described a software categorization based on an evaluation of used terms, classes and packages of called APIs by using decision trees, naïve bayes and SVM. A year later McMillan et al. (2012) used LSA on shared semantic anchors, like API calls, to determine a similarity of software. The number of called APIs and the implementation of requirements by using API calls were also analysed. Similar to a previously described journal article of McMillan, API calls were analysed by Linares-Vásquez et al. (2014). The goal was an API-based categorization and single words-based categorization using methods like SVM. Reyhani Hamedani et al. (2018) focused on mobile applications for the Android platform and described in their journal article a classification based on random forest, kNN, naïve bayes, SVM and a DNN based on TensorFlow. In their process, they rely on the analysis of packages, classes, methods and the full-method signatures of the APIs used. Kanda et al. (2011) and Dong et al. (2016) have also published an approach with a focus on android applications. While Kanda et al. analysed method calls of API classes by sequence comparison for a software similarity, Dong et al. applied a multinomial naïve bayes (MNB) on "characteristic APIs" to categorize the applications. These characteristic APIs are intended to enable the detection of category information. Nafi et al. (2018), on the other hand, used the documentation of the APIs called by the respective software to determine software relevant for users by means of extracted paragraphs in vectors and an applied cosine similarity.

Finally two publications, Kim et al. (2015, 2018), have to be described in this section, which both describe a software categorization as a goal. The first publication in 2015 discusses the application of a random forest and an ANN on API call frequencies by the respective software. In 2018 only a random forest was used, but besides the API call frequencies also string information extracted from *.rsrc* files was used. This can be considered as an additional analysis of resource files.

**Based on repository activities:** The usage of repository activities includes platform-dependent features provided by APIs and GitHub. These features mainly include social aspects such as the stars assigned by users of the platform, the number of repository watches, subscriptions, members and forks as well as the creation time, available wikis, pull requests and download-numbers of repositories. Furthermore, repo-owner, user-relations and platform-specific scoring, such as a GitHub relevance score, were included. These different aspects of features describe characteristics of the repository itself, the dependency relations among repositories and the social activities among developers. When using repository activities, the first advantage to be mentioned is that links on social media repository platforms or data operated by the community can already indicate a connection or commonality between software projects. In addition, it may be helpful in cases where projects do not provide much useable data. However, it should be noted that data does not come directly from the software project, and indicators such as user behaviour may change over time or vary between different platforms and societies. In addition, a sufficiently large community is required to use the activities as an information value.

Some publications have already been described in the previous sections that considered repository activities in combination with other software-related data for a similarity analysis (Allaho, 2014; Lavid Ben Lulu and Kuflik, 2016; Petrovic et al., 2016; Portugal et al., 2017; Soll and Vosgerau, 2017; Sun et al., 2018; Xu et al., 2017a,b; Zhang et al., 2017; Zheng et al., 2018).

Yang et al. (2016) applied an approach for development improvement, as well as onboarding and education of developers. For this purpose they used a custom equation distance on the social activities among developers, which includes repository-forks, total reference times from repositories that the user joined and social relation among users. In addition, they nonspecifically

[15] The approach was published as a patent and therefore appears to lack a peer-review.

describe an additional use of characteristics of the repository and dependency relations. Zhang et al. (2014) used five features from the repository activities, consisting of forks, watches, comments on issues, pull-requests and members. In combination with a relevance scoring provided by GitHub on all repositories, a software categorization was performed by calculating a cosine similarity.

For a subsequent filtering of categorized software Venkataramani et al. (2013) also used information like last updated project, number of forks and number of watchers. In addition, a best string matching was set as parameter. Methodically, TF–IDF and a subsequent clustering were used. First they calculated a graph-based similarity using jaccard and overlap similarity. For the graph the developers between the projects were modelled and finally the number of common developers was determined.

**Further approaches:** In this last section, all studies that could not be conclusively assigned to one of the other categories were collected. The remaining 10 publications mostly use high level information or data from third-party services. A utilization of search engines was performed by Xin Li et al. (2016) as well as Zhu et al. (2014). Xin Li et al. described the usage of VSM, naïve bayes and LDA on the application name in combination with enriched information from search engine results. Zhu et al. also used vectors in a VSM. However, they calculated similarities using cosine distance similarity and made use of collected data from context logs taken from mobile users, from which they extracted emerging real-world contextual features for the apps and additionally collected relevant web knowledge. Already two years earlier, Zhu et al. (2012) published an approach that relies only on the used context logs from different mobile users and 8.8 million app usage records of 680 unique apps. This rich contextual information from usage records was transferred as vectors to a VSM and a cosine distance was calculated for categorization of applications.

An ontology-based similarity for a software architecture analysis was proposed by Nadezhda et al. (2019), utilizing implemented design pattern of each software. A similarity was calculated on the basis of elements, like delegators, adapters, builders, abstracts/super-classes or interfaces, modelled in an UML language. Qiu et al. (2013) also published a graph-based approach to calculate software similarity based on documented diagrams. They present the concrete analysis of class diagrams by applying a weight matching in the created graph.

Another approach had the focus on third-party libraries used for the respective software. For this purpose, Thung et al. (2013) calculated in 2013 the similarity of two projects based on the set of libraries that are used in common between two projects. They used a nearest-neighbour-based collaborative filtering technique and calculated the cosine similarity between these projects. Hernández and Costa (2015) also published an approach in 2015 in which they extracted software libraries from all apache maven “Project Object Model” (POM) files specified in each project and calculated a jaccard similarity as well as a cosine similarity. Chen et al. (2018) introduced an approach that applies LDA to recorded UI information from mobile app recommendation applications. Singla et al. (2018b) used an approach based on app images, analysed in a *convolutional neural network* (CNN) for categorization of software. Finally, there is a publication of Linares-Vásquez et al. (2016) which analyses similar android applications with manifest-specific data like intents, permissions needed and sensors used through LSI and cosine similarity.

Another aspect is the retrieval of similar software that can be used for an effort prediction. This topic is part of the defined exclusion criteria and will not be examined further in this SLR. For completeness further literature reviews around topic effort estimation models like Wen et al. (2012), Idri et al. (2015) and Sharma and Singh (2017) can be consulted.

### 3.3. Results of studies and knowledge gain

An overview of the results obtained and the knowledge gained is summarized below. However, many of the papers show no evaluation details or comparable results. Therefore, only the relevant output was extracted.

**Baseline evaluation:** First of all, work that has been evaluated against another relevant approach from this SLR as a baseline should be considered. In order to better capture these successive evaluations, a graphical overview is provided in Fig. 4. Other publications are not included in this illustration, since these used different approaches on different software data as well as methods, with different evaluation methodologies and metrics. A direct comparison can therefore not be made in most cases.

One of the original studies is the approach MUDABlue of Kawaguchi et al. (2006), against which many of the relevant publications evaluated their work directly or indirectly. MUDABlue itself was again evaluated against the earlier study of Ugurel et al. (2002). As a result, MUDABlue categorized software with higher precision and recall. Based on MUDABlue, other approaches like Sally (Vargas-Baldrich et al., 2015; Baldrich, 2015),[16] CLAN of McMillan et al. (2012) and LACT of Tian et al. (2009) were evaluated. Results revealed, that Maven repositories were better categorized by Sally than using MUDABlue. However, MUDABlue was not conceived as a tool for categorizing Maven repositories. The approach of CLAN reached a higher recall and precision on finding similar applications than MUDABlue. LACT as well was compared against the existing approach of MUDABlue und provided an improvement over this state-of-the-art approach by generating more readable names of categories.

CLAN again was used as a baseline by an approach of Zhang et al. (2017) called RepoPal as well as the other approaches of Mens et al. (2014) and Thung et al. (2012). The latter approach achieved better results than CLAN, but both approaches have relatively low confidences and precisions. The approach of Mens et al. outperformed CLAN as well, by increasing the success rate by 23.08%. RepoPal outperformed CLAN in a experiment in terms of success-rate, confidence, and precision. There are additionally even two further approaches of Nguyen et al. (2018a) called CROSSSIM and of Nafi et al. (2018) called CroLSim, which used RepoPal as a state-of-the-art approach for evaluation. Both approaches outperformed RepoPal which outperformed CLAN.

The approach LACT which was evaluated against MUDABlue as well provided an improvement over MUDABlue by generating more readable names of categories. Since its publication, LACT has also been used as baseline by other studies. First of all, the approach LACTA of Yang and Tu (2012) is noteworthy. LACTA outperformed LACT in specific cases applied to android applications. Furthermore, the approaches LASCAD of Altarawy et al. (2018) and LSTD (Wang et al., 2012; Wang et al., 2013) were evaluated against LACT. LASCAD categorized software stably better than the prior approaches MUDABlue and LACT on different data sets. Compared to the previous work like MUDABlue and LACT, Wang et al. (2013) presents competitive results with finer-grained and multi-layered categories. While both previous approaches presented an improvement against their baselines, LSTD reached an average precision of 0.75 and 0.68 respectively, which was nearly the same as the result of LACT.

Unlike the previous studies, Thung et al. (2013) validated their LibRec approach separately. As a result they achieved a promising result with a 0.894 recall rate. The published approaches AppLibRec of Yu et al. (2017) and CrossRec of Nguyen

[16] The approach of Baldrich (2015) was published as a master thesis and therefore appears to lack a peer-review.

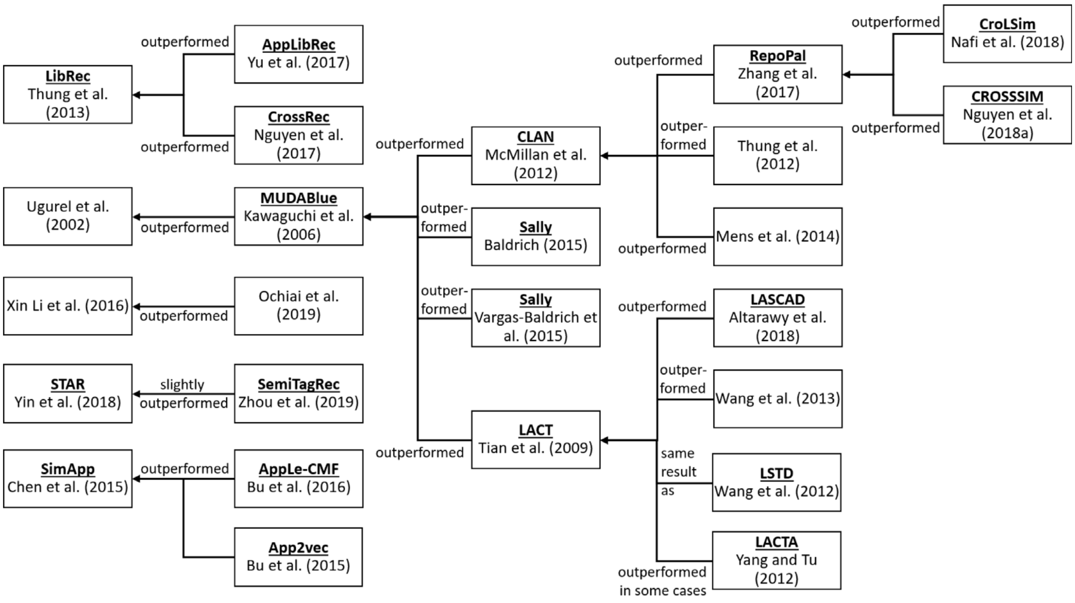

**Fig. 4.** Baseline evaluation between SLR-studies.

et al. (2018b) were evaluated against LibRec and outperformed the older approach on recommending third-party libraries. The maximum improvement of AppLibRec was described by +40.91% on precision and 35.34% on recall. Hereby, a maximum precision of 0.569 and a maximum recall of 0.508 was achieved.

Similarly, on the field of tag recommendation the earlier approach of Yin et al. (2018) called STAR was slightly outperformed by the 2019 approach SemiTagRec of Zhou et al. (2019).

The similarity calculation on mobile applications was achieved by SimApp, proposed by Chen et al. (2015), gaining a precision@1 and precision@5 of 0.875 and 0.819, respectively. The App2vec method presented by Bu et al. (2015) outperformed the baseline method SimApp significantly because of the additionally used similarity relationship from source app stores. The proposed method AppLe-CMF also used SimApp among others as baseline. Thereby, it outperformed all state-of-the-art baselines, like Doc2vec, TF–IDF, LSI, LDA and SimApp.

Experimental results of Xin Li et al. (2016) showed that LDA outperforms other classification methods, like VSM and naïve bayes. This demonstrated to them that semantic analysis is worthwhile. Additionally, they discovered that adding search snippets led to improvements. The proposed approach of Ochiai et al. (2019) used a CNN and outperformed other baseline approaches like the one of Xin Li et al. (2016) The experiment brought an improvement of the f1-measure by 5.5% and greater.

**Evaluation on software categorization:** Approaches, which were not evaluated against other SLRs, are mentioned here. Although most results cannot be directly compared, the obtained values could give an indication about the effectiveness when applied. All available results of experiments were extracted and the most as well as less successful approaches are outlined below.

For software categorization, mostly the results are presented by a calculated accuracy. A maximum accuracy above 0.8 was achieved by experiments of Shabtai et al. (2010), Liu et al. (2016), Yuan et al. (2016), Singla et al. (2018a), Escobar-Avila et al. (2015), Reyhani Hamedani et al. (2018) and Escobar-Avila (2015) (Xu et al., 2017a,b) and (Avila, 2015; Escobar-Avila, 2015).[17] The experiments of Krovetz et al. (2003) also reached a classification accuracy of over 0.90, but showed differences depending on the categories. While the high result of over 0.90 was obtained for distinct categories, about 0.72 accuracy was achieved for related categories. An accuracy for software categorization between 0.7 and 0.8 was reached by Yusof and Ramadan (2010) and Sandhu et al. (2007b,a). A comparison between functional and object-oriented software code was made by Sandhu et al. (2007a), who discovered that function oriented software was compared best by multinomial naïve bayes. In comparison the best result on object oriented software was reached by complement naïve bayes with 0.552 accuracy. While Singla et al. (2018a) showed a 0.89 accuracy, their classifier, combining image and text based approach, also reached a f1-measure slightly above 0.8. Other studies on software categorization using f1-measure was Chen et al. (2017a) with 0.88, Sharma et al. (2017) with 0.7 and Lavid Ben Lulu and Kuflik (2016) with 0.665. The classification approach of Ma et al. (2018) combines different ML algorithms through ensemble learning and reached an average precision of 0.85 and recall of 0.82. Su (2018) even achieved with over 0.9 precision on programme classification. Thung et al. (2013) evaluated their approach by taking the recall rate, which was 0.894.

While the previous described results can be considered as better results, Nguyen and Nguyen (2017) and Berardi et al. (2015) were described as less than optimal for the accuracy. Nguyen and Nguyen (2017) reached by the use of J.48 and naïve bayes an accuracy of 0.223 and 0.307. Their best result was 0.587 reached by their implemented DNN. From this result they considered the DNN as a promising technique in automated software categorization. Berardi et al. (2015) concluded, that there was a large margin for improvement in further studies.

**Evaluation on other targets:** Beside software categorization, also the studies on other targets achieved reasonable results. On the field of software tagging, the results of Cai et al. (2016) show that the approach GRETA achieves up to 0.35 of f1-measure, outperforming four baseline methods when tagging GitHub repositories. The proposed clustering approach AAOKL of Chen et al. (2016) reached an average precision of 0.266 in maximum and a highest hit rate of 0.628. Better numbers of measures can be found in other approaches beside software tagging. On studies with focus on mobile app recommendation the proposed method of Chen et al. (2018) brought a result in experiments of 0.457 average

[17] The approach of Avila (2015) was published as a master thesis and therefore appears to lack a peer-review.

precision, 0.418 MAP and 0.36 MRR. The new topic model called combineLDA to calculate the similarity based on extracted topics outperformed LDA in the study of Pan et al. (2016). While LDA reached a f1-measure of 0.4, the new approach got 0.56. For the recommendation of software Thung et al. (2017) proposed WebAPIRec and reached for Hit@N, MAP and MRR in the peak 0.880, 0.697 and 0.750 respectively. When performing relevance ratings on software, the approach Zheng et al. (2018) reached a high hit rate of 0.983 as well, but the average accuracy was at 0.298.

**Additional extracted knowledge from results:** In addition to the many possible data that can be used to calculate the software similarity, some insights into its effectiveness were also gained. Singla et al. (2018b) discovered, that the classification accuracy can be improved for some classes when app images are considered. However, Zhang et al. (2014) found forks, watches, pull-requests and members as helpful for identifying relevant projects, while issue comments was not found as suitable. McMillan et al. (2011) and Zhou (2016) described separately, that package-information from APIs and from source code are more effective attributes than API and code classes. Also API packages were found to be a good alternative to terms, which includes identifiers and comments extracted from source code. In contrast, Wang et al. (2014b) suggested based on their results that software profiles are a more effective alternative to software APIs for categorization.

Concerning the methodology, it was found by Zhou (2016) that clustering software text descriptions with the help of topic modelling seems to outperform TF–IDF vectorizer slightly. Beside this, the authors also pointed out that SVM was the best-performing algorithm compared to decision tree and naïve bayes. In case of naïve bayes, Babatunde (2016)[18] observed that a multinomial naïve bayes performed better than a bernoulli naïve bayes classifier on text classification where the number of occurrence of a word is important. Another study, which used TF–IDF, was performed by ZhangChao and WanLili (2018). They discovered that they could achieve a more effective approach using methods such as LSA or LDA to extract relevant terms. Furthermore, Yoon et al. (2017) described that the state-of-the-art method LDA performed better than k-means for their proposed clustering. Also their clustering method "rdoc2vec" showed the highest performance on average, while it performed regardless of the training corpus size. In the study of Wang et al. (2014a) it was observed that TRG and SVM achieved a higher precision than LDA-TR and Yusof and Rana (2010) came to the conclusion, that a better classification can be performed using kNN compared to C4.5 and discriminant analysis.

Finally, neural networks were evaluated against some state-of-the-art approaches. According to Mingshan Jr (2018),[19] neural networks provide a slightly better performance than decision trees in their experiment. In the earlier stated study of Reyhani Hamedani et al. (2018), the authors concluded that a DNN can improve the results in general.

## 4. Summarization of results

In the previous section, the detailed description shows similarities and differences between the different approaches on the data and methodology used. Also an overview of the performance of particular studies and general findings was given. In the following, the findings of the publications will be summarized and a better overview will be provided. In addition, this section explains some details about validation and restrictions.

### 4.1. Comprehensive overview

By the above description it might be difficult to capture the field of research initially. Many of the published approaches that are relevant to this literature review not only use a particular type of data, but also describe the analysis of several different sources. Thus, the software is analysed according to different aspects, such as functional and technical properties. At the same time, these approaches are based on different levels of abstraction. While about 45 of the presented papers rely on source code or binaries, about 25 of the articles performed an analysis on software design descriptive diagrams, used libraries as technical basis or various project metrics. These metrics from the project environment or the software itself were identified 13 times in the result set of this work. About 72, almost half, of the examined research papers focus on software profiles, which describe, evaluate and categorize the software. This category also includes change log and commit messages, which are describing versions of the software. Four of the publications also used, in combination with other sources, textual data from bug reports, open issues and issue comments. The rest of the work used repository-based metadata from the respective platforms. In order to further enrich the data obtained, almost 10 approaches also used additional sources of information such as software logs, information from Q/A portals such as StackOverflow or search engine query results.

Over the past few years, numerous goals, data combinations and methods have led to different sectors of research in the field of software similarity, which has made providing an overview a difficult task. Therefore, an attempt was made to extract the central work from the result set of the reviews. In order to extract the central publications from the 136 results, all citations between this literature were examined. The papers cited most frequently by the other SLR-relevant publications can be regarded as central elements on the examined field of research. The 15 most cited and therefore roughly 10% most fundamental publications are listed in Table 9.

All the 136 identified publications are available in different types and levels of detail. While most articles come from journals and conferences, there are also patents, workshop paper, technical reports, books, master theses and student projects included in this review. For an overview of all results, the relevant literature is listed in Fig. 5 according to their publication type and the year of publication.

### 4.2. Trends and opportunities

The field of similarity-based analyses on software applications has experienced a steadily growing trend on the timeline in recent years. This is already presented and described in Fig. 3. Besides the increasing trend of publications, other trends and resulting opportunities can be identified. If we consider the data sources and extracted information used in the approaches in chronological order, as it is presented in Tables 6–8, we identified some trends. When new technologies were established, they were also considered in the research field of software similarity analyses. One example is the inclusion of mobile applications, which have been used since 2010. Given the growing size of these app stores, a classification, clustering or tagging on the mobile apps makes sense to provide better searchability and recommendations in mobile app stores. Something similar can be seen in the increasing use of container technologies. As solutions such as Docker and Kubernetes are increasingly used and online repositories such as Docker Hub[20] are growing, studies have

[18] This work appears to lack a peer-review.

[19] The approach was published as a master thesis and therefore appears to lack a peer-review.

[20] The repository Docker Hub is a service hosted by Docker Inc. and can be accessed on https://hub.docker.com/ (last accessed: 2020-01-28).

**Table 9**
The 15 most fundamental publications on the examined research field of software similarity by the amount of citations between all SLR-relevant papers.

| Authors | Title | Cites |
|---|---|---|
| Kawaguchi et al. (2006) | MUDABlue: An automatic categorization system for Open Source repositories | 41 |
| McMillan et al. (2011) | Categorizing software applications for maintenance | 29 |
| Tian et al. (2009) | Using Latent Dirichlet Allocation for automatic categorization of software | 26 |
| Ugurel et al. (2002) | What is the code?: automatic classification of source code archives | 24 |
| McMillan et al. (2012) | Detecting similar software applications | 23 |
| Linares-Vásquez et al. (2014) | On using machine learning to automatically classify software applications into domain categories | 20 |
| Wang et al. (2013) | Mining Software Profile across Multiple Repositories for Hierarchical Categorization | 18 |
| Thung et al. (2012) | Detecting similar applications with collaborative tagging | 16 |
| Zhu et al. (2014) | Mobile App Classification with Enriched Contextual Information | 16 |
| Yin et al. (2013) | App recommendation: a contest between satisfaction and temptation | 16 |
| Kawaguchi et al. (2003) | Automatic categorization algorithm for evolvable software archive | 14 |
| Grechanik et al. (2010) | A search engine for finding highly relevant applications | 13 |
| Chen et al. (2015) | SimApp: A Framework for Detecting Similar Mobile Applications by Online Kernel Learning | 13 |
| Dumitru et al. (2011) | On-demand Feature Recommendations Derived from Mining Public Product Descriptions | 12 |
| Zhu et al. (2012) | Exploiting Enriched Contextual Information for Mobile App Classification | 11 |

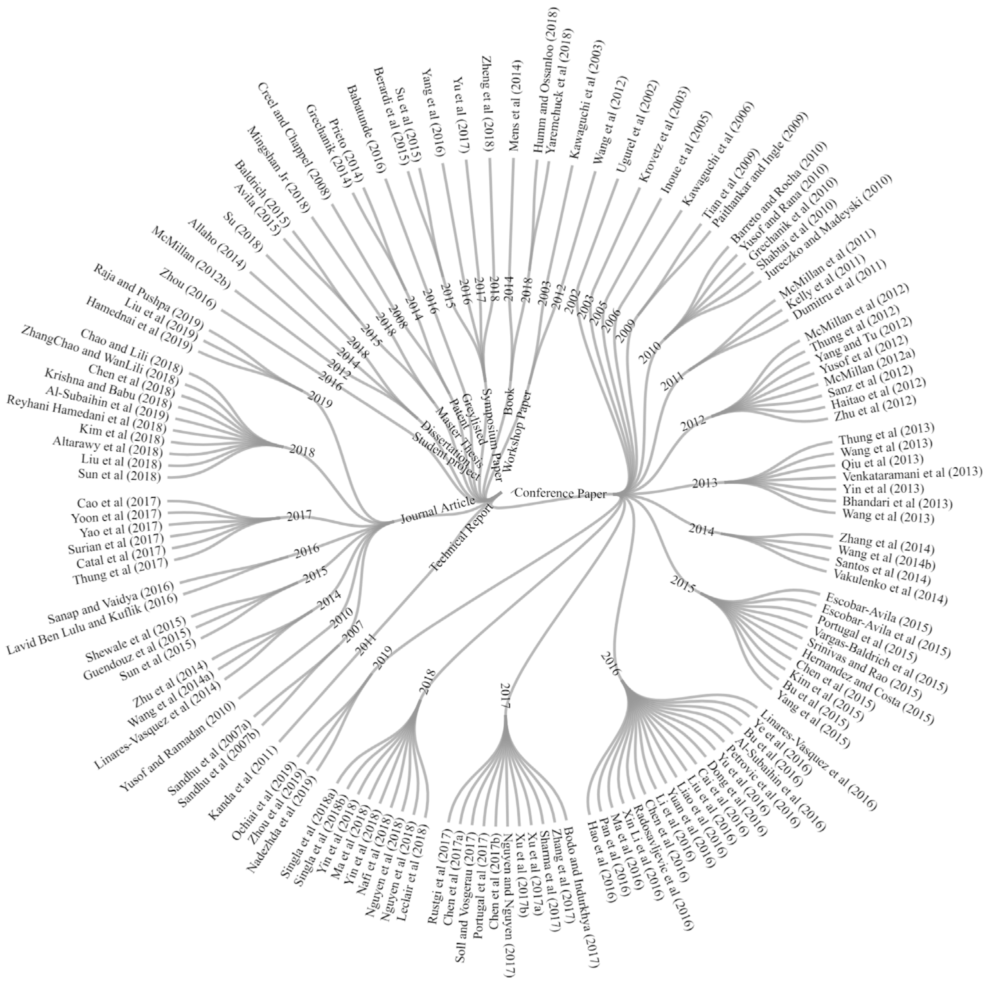

**Fig. 5.** Overview of identified types and publishing dates on SLR-results.

been applied on these openly available containers for similarity analyses since 2018.

In recent years, the focus of the found studies on using source platforms such as Sourceforge, Ohloh, and Freecode has also shifted towards other platforms such as Github as the version control tool Git has become more popular. Thereby most of the work relies on publicly available open source software and information from publicly available APIs. Especially in this aspect we see the opportunity to evaluate the approaches on enterprise repositories in order to enable a better generalization of the findings. The further development of the software platforms also brought new features such as reviews, ratings and comments as well as stargazing, forking, watches and subscriptions. In addition, there is a trend towards the formation of online communities since 2013, which has brought new possibilities for matching similar software. In addition, the number of approaches relying on repository activities has been increasing only since 2017, whereas in previous years few publications appeared. We therefore see a potential demand for further case studies in applying software similarity approaches on social relationships, such as the networking of developers in communities, as well as the analysis of specific usage behaviour on company-internal platforms. This may produce results that differ from those approaches performed on open source software.

Further trends, in particular with regard to shifts in the use of certain information, could not be clearly identified. Information from source and byte code as well as software documentation is constantly used in studies nowadays as it was used previously. Among other things, this could be due to the circumstance that information sources are mostly not interchangeable. Rather, information sources contain different aspects on different levels of abstraction for similarity analysis. Accordingly, only approaches that have the same objectives should be comparable. However, we also see opportunities for further studies to draw on more sources of information and to check the relevance of the individual information for the similarity calculation. For example, we are not aware of any work that combines all the collected repository activities in a similarity calculation for an evaluation and determines their significance. On the source code level, on the other hand, we see the chance to apply more programming languages, after most studies use collections of Java programmes as data sets. For example, software sources based on Javascript and Typescript were hardly considered in the publications identified. In addition, there may be an opportunity to draw on additional, language-specific aspects from the software for a similarity analysis and to use further repositories, such as the javascript-oriented NPM,[21] with possibly different information offerings and community structures.

Looking at the trend development of the software engineering goals of all approaches, no clear trend shifts can be identified in this research area as well. However, we can determine that the recommendation of tags, topics and software domains has only been a goal of the publications since 2011. The mobile app recommendation described above has been described as a software engineering goal since 2013. Similarly, the focus of some publications on software library categorization and recommendation emerged starting in 2013. This could also be related to the fact that the range of third-party libraries and their use has grown strongly over the years. According to Thung et al. (2013), many third-party libraries were already publicly available in 2013. Of the software projects analysed by them in 2013, already 93.3% used an average of 28 third-party libraries in their described data set.

Finally, from the perspective of the application of methods, the recently increased use of neural networks can be highlighted. While before 2018 neural networks were only used sporadically by the reviewed approaches, in 2018 and 2019 there were already 8 approaches with often promising results. Therefore, we also see an opportunity to improve the results in the detection of similar software projects by using different kinds of neural networks.

### 4.3. Result validation and limitations

The SLR was performed as part of a dissertation process and therefore includes limitations established by the underlying research questions, exclusion and inclusion criteria. Within this framework, a review process was carried out to ensure the quality and to validate the results considered as relevant for the SLR. A sample of 22% of the finally included publications was spot-checked with regard to the exclusion and inclusion criteria. For this, the literature found by one author was reviewed by another author. This validation step was carried out in order to prevent personal bias in the filtering process as well as unclear wording in the stated criteria. The sample was obtained by dividing the publications into those published before 2010 and those published later and after randomly selecting 15 publications from each of the two groups.

Approaches that were not published in literature, but are described, for example, in the form of blog posts, were not considered in this literature review. Since there may also be publicly accessible blog posts or general websites describing attempts to determine software similarity, this should be noted as a limitation of this study.

## 5. Conclusion

The literature review shows that software similarity is a broad field of research with different approaches, methods and goals. Furthermore, there is a growing interest in the field of machine learning in software development and software engineering in general. The research shows a trend in the number of publications and the utilization of a wide variety of data in the last 20 years. We expect that the trend towards open source currently continues and online repositories such as GitHub are becoming increasingly popular. This is also confirmed by data from the online hoster Github (2018) in an annual report, which describes that the platform hosted around 96 million repositories until then. This was an increase of 40% over the previous year.

GitHub and similar repositories as well as app stores often provide additional information, which is referred to as online profiles in this work. In the context of this SLR it was determined that already most of the relevant publications rely completely or partly on available online profiles of the software. Approaches like RepoPal or CroLSim demonstrated that a classification improvement through the usage of these online profiles can be achieved compared to approaches like the frequently referenced MUDABlue and CLAN, which exclusively rely on source code. In addition to the online profiles, many studies have also used certain forms of usage behaviour on platforms such as Github, information from bug trackers and, in general, data from third-party sources. Also on the basis of this information, promising results were achieved, which were described in this work. The conclusion can therefore be drawn that for current computations of the software similarity, additional data from third-party systems are at least relevant and can even help to improve calculations based on the software code exclusively.

Nevertheless, it should be noted that because of the large number of approaches, a context-independent recommendation cannot be given. The data used for a similarity analysis as well as the methodologies applied are diverse and have been combined in many ways by the analysed publications. The amount of the resulting options complicates the comparability of all approaches and makes a derivation for new, planned similarity analyses case-dependent. In order to derive a procedure, the target must be considered and the appropriate data and methodology should be selected. The described result of this SLR, can help with such a procedure selection and point out possible gaps for further research by providing a structured overview.

## CRediT authorship contribution statement

**Maximilian Auch:** Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Writing - original draft. **Manuel Weber:** Validation, Writing - review & editing. **Peter Mandl:** Supervision, Validation, Writing - review & editing, Project administration, Resources, Funding acquisition. **Christian Wolff:** Supervision, Project administration.

[21] The repository NPM is a service hosted by npm Inc. and can be accessed on https://www.npmjs.com/ (last accessed: 2020-02-06).

## Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.jss.2020.110669.

During the research work Maximilian Auch was employed part-time by Pentasys AG and Professor Dr. Peter Mandl was a partner of iSYS Software GmbH.

## References

Al-Subaihin, A., Sarro, F., Black, S., Capra, L., 2019. Empirical comparison of text-based mobile apps similarity measurement techniques. Empir. Softw. Eng. 1–26.

Al-Subaihin, A., Sarro, F., Black, S., Capra, L., Harman, M., Jia, Y., Zhang, Y., 2016. Clustering mobile apps based on mined textual features. In: International Symposium on Empirical Software Engineering and Measurement, Vol. 08-09-September-2016.

Allaho, M.Y., 2014. Recommendation services for open source software community.

Altarawy, D., Shahin, H., Mohammed, A., Meng, N., 2018. Lascad: Language-agnostic software categorization and similar application detection. J. Syst. Softw. 142, 21–34.

Aveyard, H., 2014. Doing a Literature Review in Health and Social Care: A Practical Guide. Open University Press.

Avila, J.R.E., 2015. A Model for Automatic Categorization of Software Applications Using Non-Parametric Clustering and Bytecode Analysis. Universidad Nacional de Colombia, Magister en Ingeniería - Sistemas y Computación.

Babatunde, O., 2016. Applying naive bayes classification to google play apps categorization. CoRR, abs/1608.08574.

Baldrich, S.V., 2015. Automatic Multi-Label Categorization of Java Applications Using Dependency Graphs. Universidad Nacional de Colombia, Maestría en Ingeniería - Ingeniería de Sistemas y Computación. Research Line: Software Engineering.

Barreto, A.O.S., Rocha, A.R., 2010. Analyzing the similarity among software projects to improve software project monitoring processes. In: 2010 Seventh International Conference on the Quality of Information and Communications Technology. pp. 441–446.

Berardi, G., Esuli, A., Fagni, T., Sebastiani, F., 2015. Multi-store metadata-based supervised mobile app classification. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. In: SAC '15, ACM, New York, NY, USA, pp. 585–588.

Bhandari, U., Sugiyama, K., Datta, A., Jindal, R., 2013. Serendipitous recommendation for mobile apps using item-item similarity graph. In: Banchs, R.E., Silvestri, F., Liu, T.-Y., Zhang, M., Gao, S., Lang, J. (Eds.), Information Retrieval Technology. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 440–451.

Bodó, Z., Indurkhya, B., 2017. Software categorization using low-level distributional features. Front. Artif. Intell. Appl. 297, 88–98.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80 (4), 571–583, Software Performance.

Bu, N., Niu, S., Yu, L., Ma, W., Long, G., 2016. Bridging semantic gap between app names: Collective matrix factorization for similar mobile app recommendation. In: Cellary, W., Mokbel, M.F., Wang, J., Wang, H., Zhou, R., Zhang, Y. (Eds.), Web Information Systems Engineering – WISE 2016. Springer International Publishing, Cham, pp. 324–339.

Bu, N., Yu, L., Ma, W., Du, C., Niu, S., Long, G., 2015. Detect similar mobile applications with transfer learning. In: 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity). pp. 856–859.

Budgen, D., Brereton, P., 2006. Performing systematic literature reviews in software engineering. In: Proceedings of the 28th International Conference on Software Engineering. In: ICSE '06, ACM, New York, NY, USA, pp. 1051–1052.

Cai, X., Zhu, J., Shen, B., Chen, Y., 2016. Greta: Graph-based tag assignment for github repositories. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), Vol. 1. pp. 63–72.

Cao, D., Nie, L., He, X., Wei, X., Shen, J., Wu, S., Chua, T.-S., 2017. Version-sensitive mobile app recommendation. Inform. Sci. 381, 161–175.

Catal, C., Tugul, S., Akpinar, B., 2017. Automatic software categorization using ensemble methods and bytecode analysis. Int. J. Softw. Eng. Knowl. Eng. 27 (07), 1129–1144.

Cesare, S., Xiang, Y., 2012. Software Similarity and Classification. Springer London, London, pp. 77–86.

Chao, Z., Lili, W., 2018. A feasibility study on adopting individual information cognitive processing as criteria of categorization on apple itunes store. 27 (2), 1–28.

Chen, N., Hoi, S.C., Li, S., Xiao, X., 2015. Simapp: A framework for detecting similar mobile applications by online kernel learning. In: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining. In: WSDM '15, ACM, New York, NY, USA, pp. 305–314.

Chen, N., Hoi, S.C., Li, S., Xiao, X., 2016. Mobile app tagging. In: Proceedings of the Ninth ACM International Conference on Web Search and Data Mining. In: WSDM '16, ACM, New York, NY, USA, pp. 63–72.

Chen, W., Xu, P., Dou, W., Wu, G., Gao, C., Wei, J., 2017a. A hierarchical categorization approach for configuration management modules. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Vol. 1. pp. 160–169.

Chen, W., Xu, P., Wu, G., Dou, W., Gao, C., Wei, J., 2017b. A hierarchical categorization approach for system operation services. In: 2017 IEEE International Conference on Web Services (ICWS). pp. 700–707.

Chen, X., Zou, Q., Fan, B., Zheng, Z., Luo, X., 2018. Recommending software features for mobile applications based on user interface comparison. Requir. Eng..

Cooper, H.M., 1988. Organizing knowledge syntheses: A taxonomy of literature reviews. Knowl. Soc. 1 (1), 104.

Creel, C.T., Chappel, O.A., 2008. Project Management System and Method for Assessing Relationships Between Current and Historical Projects. Google Patents, US Patent 7, 366, 680.

Dong, F., Guo, Y., Li, C., Xu, G., Wei, F., 2016. Classifydroid: Large scale android applications classification using semi-supervised multinomial naive bayes. In: 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS). pp. 77–81.

Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M., 2011. On-demand feature recommendations derived from mining public product descriptions. In: Proceedings of the 33rd International Conference on Software Engineering. In: ICSE '11, ACM, New York, NY, USA, pp. 181–190.

Escobar-Avila, J., 2015. Automatic categorization of software libraries using bytecode. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 2. pp. 784–786.

Escobar-Avila, J., Linares-Vásquez, M., Haiduc, S., 2015. Unsupervised software Categorization using bytecode. In: 2015 IEEE 23rd International Conference on Program Comprehension. pp. 229–239.

Ganesan, K., 2017. Topic suggestions for millions of repositories.

Ghosh, S., Vinyals, O., Strope, B., Roy, S., Dean, T., Heck, L., 2016. Contextual lstm (clstm) models for large scale nlp tasks.

Github, 2018. State of the octoverse.

Grechanik, M., 2014. Systems and Methods for Finding Project-Related Information by Clustering Applications into Related Concept Categories. Google Patents, US Patent 8, 832, 655.

Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., Cumby, C., 2010. A search engine for finding highly relevant applications. In: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1. In: ICSE '10, ACM, New York, NY, USA, pp. 475–484.

Guendouz, M., Amine, A., Hamou, R.M., 2015. Recommending relevant open source projects on github using a collaborative-filtering technique. Int. J. Open Source Softw. Process. (IJOSSP) 6 (1), 1–16.

Haitao, L., Ru-xiang, W., Guo-ping, J., 2012. Similarity measurement for data with high-dimensional and mixed feature values through fuzzy clustering. In: 2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE), Vol. 3. pp. 617–621.

Hamednai, M.R., Kim, G., Cho, S.-j., 2019. Simandro: an effective method to compute similarity of android applications. Soft Comput. 23 (17), 7569–7590.

Hao, Y., Wang, Z., Xu, X., 2016. Global and personal app networks: Characterizing social relations among mobile apps. In: 2016 IEEE International Conference on Services Computing (SCC). pp. 227–234.

Hernández, L., Costa, H., 2015. Identifying similarity of software in apache ecosystem – an exploratory study. In: 2015 12th International Conference on Information Technology - New Generations. pp. 397–402.

Humm, B.G., Ossanloo, H., 2018. Domain-specific semantic search applications: Example softwarefinder. In: Semantic Applications: Methodology, Technology, Corporate Use. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 243–258.

Idri, A., azzahra Amazal, F., Abran, A., 2015. Analogy-based software development effort estimation: A systematic mapping and review. Inf. Softw. Technol. 58, 206–230.

Inoue, K., Garg, P.K., Iida, H., Matsumoto, K., Torii, K., 2005. Mega software engineering. In: Bomarius, F., Komi-Sirvi¨o, S. (Eds.), Product Focused Software Process Improvement. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 399–413.

Jureczko, M., Madeyski, L., 2010. Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the 6th International Conference on Predictive Models in Software Engineering. In: PROMISE '10, ACM, New York, NY, USA, pp. 9:1–9:10.

Kanda, T., Manabe, Y., Ishio, T., Matsushita, M., Inoue, K., 2011. A prototype of comparison tool for Android applications based on difference of API calling sequences, 111 (107), 35–40.

Kawaguchi, S., Garg, P.K., Matsushita, M., Inoue, K., 2003. Automatic categorization algorithm for evolvable software archive. In: Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings. pp. 195–200.

Kawaguchi, S., Garg, P.K., Matsushita, M., Inoue, K., 2006. Mudablue: An automatic categorization system for open source repositories. J. Syst. Softw. 79 (7), 939–953, Selected papers from the 11th Asia Pacific Software Engineering Conference (APSEC2004).

Kelly, M.B., Alexander, J.S., Adams, B., Hassan, A.E., 2011. Recovering a balanced overview of topics in a software domain. In: 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation. pp. 135–144.

Kim, Y., Cho, S.-j., Han, S., You, I., 2018. A software classification scheme using binary-level characteristics for efficient software filtering. Soft Comput. 22 (2), 595–606.

Kim, Y., Park, J., Cho, S.-j., Nah, Y., Han, S., Park, M., 2015. Machine learning-based software classification scheme for efficient program similarity analysis. In: Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems. In: RACS, ACM, New York, NY, USA, pp. 114–118.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Citeseer.

Kitchenham, B.A., Dyba, T., Jorgensen, M., 2004. Evidence-based software engineering. In: Proceedings of the 26th International Conference on Software Engineering. In: ICSE '04, IEEE Computer Society, Washington, DC, USA, pp. 273–281.

Krishna, P., Babu, P., 2018. Detection of the app in google play by categorization.

Krovetz, R., Ugurel, S., Giles, C.L., 2003. Classification of source code archives. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval. In: SIGIR '03, ACM, New York, NY, USA, pp. 425–426.

Lavid Ben Lulu, D., Kuflik, T., 2016. Wise mobile icons organization: Apps taxonomy classification using functionality mining to ease apps finding. Mob. Inf. Syst. 2016.

Leclair, A., Eberhart, Z., McMillan, C., 2018. Adapting neural text classification for improved software categorization. pp. 461–472.

Lee, C.-P., Lin, C.-J., 2014. Large-scale linear ranksvm. Neural Comput. 26 (4), 781–817.

Li, S., Xiao, X., Bassett, B., Xie, T., Tillmann, N., 2016. Measuring code behavioral similarity for programming and software engineering education. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C). pp. 501–510.

Liao, Y., Li, J., Li, B., Zhu, G., Yin, Y., Cai, R., 2016. Automated detection and classification for packed android applications. In: 2016 IEEE International Conference on Mobile Services (MS). pp. 200–203.

Linares-Vásquez, M., Holtzhauer, A., Poshyvanyk, D., 2016. On automatically detecting similar android apps. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC). pp. 1–10.

Linares-Vásquez, M., McMillan, C., Poshyvanyk, D., Grechanik, M., 2014. On using machine learning to automatically classify software applications into domain categories. Empir. Softw. Eng. 19 (3), 582–618.

Liu, C., Cao, J., Feng, S., 2019. Leveraging kernel-incorporated matrix factorization for app recommendation. ACM Trans. Knowl. Discov. Data 13 (3), 31:1–31:27.

Liu, X., Song, H.H., Baldi, M., Tan, P., 2016. Macro-scale mobile app market analysis using customized hierarchical categorization. In: IEEE INFOCOM 2016 - the 35th Annual IEEE International Conference on Computer Communications. pp. 1–9.

Liu, C., Yang, D., Zhang, X., Ray, B., Rahman, M.M., 2018. Recommending github projects for developer onboarding. IEEE Access 6, 52082–52094.

Ma, Y., Fakhoury, S., Christensen, M., Arnaoudova, V., Zogaan, W., Mirakhorli, M., 2018. Automatic classification of software artifacts in open-source applications. In: 2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR). pp. 414–425.

Ma, Q., Muthukrishnan, S., Simpson, W., 2016. App2vec: Vector modeling of mobile apps and applications. In: Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. In: ASONAM '16, IEEE Press, Piscataway, NJ, USA, pp. 599–606.

McMillan, C., 2012a. CLAN: Closely Related Applications. Citeseer.

McMillan, C., 2012b. Searching, Selecting, and Synthesizing Source Code Components (PhD thesis). College of William & Mary, Williamsburg, VA, USA, AAI3533674.

McMillan, C., Grechanik, M., Poshyvanyk, D., 2012. Detecting similar software applications. In: 2012 34th International Conference on Software Engineering (ICSE). pp. 364–374.

McMillan, C., Linares-Vásquez, M., Poshyvanyk, D., Grechanik, M., 2011. Categorizing software applications for maintenance. In: 2011 27th IEEE International Conference on Software Maintenance (ICSM). pp. 343–352.

Mens, T., Serebrenik, A., Cleve, A., 2014. Evolving software systems. pp. 1–404,

Mingshan Jr, H., 2018. Leveraging Mobile App Classification and User Context Information for Improving Recommendation Systems. Brock University.

Nadezhda, Y., Gleb, G., Pavel, D., Vladimir, S., 2019. An approach to similar software projects searching and architecture analysis based on artificial intelligence methods. In: Abraham, A., Kovalev, S., Tarassov, V., Snasel, V., Sukhanov, A. (Eds.), Proceedings of the Third International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'18). Springer International Publishing, Cham, pp. 341–352.

Nafi, K.W., Roy, B., Roy, C.K., Schneider, K.A., 2018. [Research paper] crolsim: Cross language software similarity detector using api documentation. In: 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM). pp. 139–148.

Nazir, S., Shahzad, S., Mukhtar, N., 2019. Software birthmark design and estimation: A systematic literature review. Arab. J. Sci. Eng. 44 (4), 3905–3927.

Neely, J.G., Magit, A.E., Rich, J.T., Voelker, C.C., Wang, E.W., Paniello, R.C., Nussenbaum, B., Bradley, J.P., 2010. A practical guide to understanding systematic reviews and meta-analyses. Otolaryngol. - Head Neck Surg. 142 (1), 6–14.

Nguyen, P., Di Rocco, J., Di Ruscio, D., 2018a. Crosssim: Exploiting mutual relationships to detect similar oss projects. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 388–395.

Nguyen, P., Di Rocco, J., Di Ruscio, D., 2018b. Mining software repositories to support oss developers: A recommender systems approach. In: CEUR Workshop Proceedings, Vol. 2140.

Nguyen, A.T., Nguyen, T.N., 2017. Automatic categorization with deep neural network for open-source java projects. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). pp. 164–166.

Niwattanakul, S., Singthongchai, J., Naenudorn, E., Wanapu, S., 2013. Using of Jaccard coefficient for keywords similarity. In: Proceedings of the International Multiconference of Engineers and Computer Scientists, Vol. 1. pp. 380–384.

Ochiai, K., Putri, F., Fukazawa, Y., 2019. Local app classification using deep neural network based on mobile app market data.

Paithankar, K., Ingle, M., 2009. Characterization of software projects by restructuring parameters for usability evaluation. In: 2009 Second International Conference on Computer and Electrical Engineering, Vol. 2. pp. 436–441.

Pan, T., Zhang, W., Wang, Z., Xu, L., 2016. Recommendations based on lda topic model in android applications. In: 2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). pp. 151–158.

Petrovic, G., Dimitrieski, V., Fujita, H., 2016. A deep learning approach for searching cloud-hosted software projects.. In: SoMeT. pp. 358–368.

Portugal, R., Casanova, M., Li, T., Do Prado Leite, J., 2017. Gh4re: Repository recommendation on github for requirements elicitation reuse. In: CEUR Workshop Proceedings, Vol. 1848. pp. 113–120.

Portugal, R., Do Prado Leite, J., Almentero, E., 2015. Time-constrained requirements elicitation: Reusing github content. In: 1st International Workshop on Just-in-Time Requirements Engineering, JIT RE 2015 - Proceedings. pp. 5–8.

Prieto, R., 2014. Project Categorization and Assessment Through Multivariate Analysis. Google Patents, US Patent App. 14/053, 890.

Qiu, D., Li, H., Sun, J., 2013. Measuring software similarity based on structure and property of class diagram. In: 2013 6th International Conference on Advanced Computational Intelligence, ICACI 2013 - Proceedings. pp. 75–80.

Radosavljevic, V., Grbovic, M., Djuric, N., Bhamidipati, N., Zhang, D., Wang, J., Dang, J., Huang, H., Nagarajan, A., Chen, P., 2016. Smartphone app Categorization for interest targeting in advertising marketplace. In: Proceedings of the 25th International Conference Companion on World Wide Web. In: WWW '16 Companion, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, pp. 93–94.

Raja, D.R.K., Pushpa, S., 2019. Diversifying personalized mobile multimedia application recommendations through the latent dirichlet allocation and clustering optimization. Multimedia Tools Appl..

Randolph, J.J., 2009. A guide to writing the dissertation literature review. Pract. Assess. Res. Eval. 14 (13), 1–13.

Reyhani Hamedani, M., Shin, D., Lee, M., Cho, S.-J., Hwang, C., 2018. Androclass: An effective method to classify android applications by applying deep neural networks to comprehensive features. Wirel. Commun. Mob. Comput. 2018.

Rustgi, P., Fung, C., Rashidi, B., McInnes, B., 2017. Droidvisor: An android secure application recommendation system. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). pp. 1071–1076.

Sanap, A., Vaidya, M., 2016. By analyzing the app reviews profile based mobile app recommendation using contextual information of app.

Sandhu, P.S., Bala, M., Singh, H., 2007a. Automatic categorization of software modules. IJCSNS 7 (8), 114.

Sandhu, P.S., Singh, J., Singh, H., 2007b. Approaches for Categorization of reusable software components.

Santos, M., Amador, R., de Souza Bermejo, P.H., Costa, H., 2014. Similar characteristics of internal software quality attributes for object-oriented open-source software projects.

Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., 2012. On the automatic categorisation of android applications. In: 2012 IEEE Consumer Communications and Networking Conference (CCNC). pp. 149–153.

Shabtai, A., Fledel, Y., Elovici, Y., 2010. Automated static code analysis for classifying android applications using machine learning. In: 2010 International Conference on Computational Intelligence and Security. pp. 329–333.

Sharma, P., Singh, J., 2017. Systematic literature review on software effort estimation using machine learning approaches. In: 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS). pp. 43–47.

Sharma, A., Thung, F., Kochhar, P., Sulistya, A., Lo, D., 2017. Cataloging github repositories. In: ACM International Conference Proceeding Series, vol. Part F128635, pp. 314–319.

Shewale, S.K., Gayakee, V.V., Ugale, P.D., Sonawane, H.D., 2015. Personalized app service system algorithm for effective classification of mobile applications. Int. J. Eng. Technol. Res.(IJETR) 3 (1).

Singla, K., Mukherjee, N., Bose, J., 2018a. Multimodal language independent app classification using images and text. In: Silberztein, M., Atigui, F., Kornyshova, E., Métais, E., Meziane, F. (Eds.), Natural Language Processing and Information Systems. Springer International Publishing, Cham, pp. 135–142.

Singla, K., Mukherjee, N., Koduvely, H.M., Bose, J., 2018b. Evaluating usage of images for app classification.

Soll, M., Vosgerau, M., 2017. Classifyhub: An algorithm to classify github repositories. In: Kern-Isberner, G., Fürnkranz, J., Thimm, M. (Eds.), KI 2017: Advances in Artificial Intelligence. Springer International Publishing, Cham, pp. 373–379.

Srinivas, C., Rao, C.V.G., 2015. A feature vector based approach for software component clustering and reuse using k-means. In: Proceedings of the the International Conference on Engineering & MIS 2015. In: ICEMIS '15, ACM, New York, NY, USA, pp. 67:1–67:5.

Su, F.-H., 2018. Uncovering Features in Behaviorally Similar Programs (PhD thesis). Columbia University.

Su, X., Zhang, D., Li, W., Li, W., 2015. Android app recommendation approach based on network traffic measurement and analysis. In: 2015 IEEE Symposium on Computers and Communication (ISCC). pp. 988–994.

Sun, X., Li, B., Leung, H., Li, B., Li, Y., 2015. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks. Inf. Softw. Technol. 66, 1–12.

Sun, X., Xu, W., Xia, X., Chen, X., Li, B., 2018. Personalized project recommendation on github. Sci. China Inf. Sci. 61 (5).

Surian, D., Seneviratne, S., Seneviratne, A., Chawla, S., 2017. App miscategorization detection: A case study on google play. IEEE Trans. Knowl. Data Eng. 29 (8), 1591–1604.

Thung, F., Lo, D., Jiang, L., 2012. Detecting similar applications with collaborative tagging. In: IEEE International Conference on Software Maintenance, ICSM. pp. 600–603.

Thung, F., Lo, D., Lawall, J., 2013. Automated library recommendation. In: 2013 20th Working Conference on Reverse Engineering (WCRE). pp. 182–191.

Thung, F., Oentaryo, R.J., Lo, D., Tian, Y., 2017. Webapirec: Recommending web APIs to software projects via personalized ranking. IEEE Trans. Emerg. Top. Comput. Intell. 1 (3), 145–156.

Tian, K., Revelle, M., Poshyvanyk, D., 2009. Using latent dirichlet allocation for automatic categorization of software. In: 2009 6th IEEE International Working Conference on Mining Software Repositories. pp. 163–166.

Ugurel, S., Krovetz, R., Giles, C.L., 2002. What's the code?: Automatic classification of source code archives. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. In: KDD '02, ACM, New York, NY, USA, pp. 632–638.

Vakulenko, S., Müller, O., Brocke, J.v., 2014. Enriching itunes app store categories via topic modeling.

Vargas-Baldrich, S., Linares-Vásquez, M., Poshyvanyk, D., 2015. Automated tagging of software projects using bytecode and dependencies (n). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). pp. 289–294.

Venkataramani, R., Asadullah, A., Bhat, V., Muddu, B., 2013. Latent co-development analysis based semantic search for large code repositories. In: 2013 IEEE International Conference on Software Maintenance. pp. 372–375.

Wang, T., Wang, H., Yin, G., Ling, C.X., Li, X., Zou, P., 2013. Mining software profile across multiple repositories for hierarchical categorization. In: 2013 IEEE International Conference on Software Maintenance. pp. 240–249.

Wang, T., Wang, H., Yin, G., Ling, C., Li, X., Zou, P., 2014a. Tag recommendation for open source software. Front. Comput. Sci. 8 (1), 69–82.

Wang, T., Wang, H., Yin, G., Yang, C., Li, X., Zou, P., 2014b. Hierarchical categorization of open source software by online profiles. IEICE Trans. Inf. Syst. E97-D (9), 2386–2397.

Wang, T., Yin, G., Li, X., Wang, H., 2012. Labeled topic detection of open source software from mining mass textual project profiles. In: Proceedings of the First International Workshop on Software Mining. In: SoftwareMining '12, ACM, New York, NY, USA, pp. 17–24.

Wang, F., Zhang, Z., Sun, H., Zhang, R., Liu, X., 2013. A cooperation based metric for mobile applications recommendation. In: Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 03. In: WI-IAT '13, IEEE Computer Society, Washington, DC, USA, pp. 13–16.

Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C., 2012. Systematic literature review of machine learning based software development effort estimation models. Inf. Softw. Technol. 54 (1), 41–59.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. In: EASE '14, ACM, New York, NY, USA, pp. 38:1–38:10.

Wohlin, C., 2016. Second-generation systematic literature studies using snowballing. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. In: EASE '16, ACM, New York, NY, USA, pp. 15:1–15:6.

Xin Li, Ya-hong Lian, Hong Yu, 2016. Classification of mobile apps with combined information. In: 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA). pp. 193–198.

Xu, W., Sun, X., Hu, J., Li, B., 2017a. Repersp: recommending personalized software projects on github. pp. 648–652.

Xu, W., Sun, X., Xia, X., Chen, X., 2017b. Scalable relevant project recommendation on github. In: ACM International Conference Proceeding Series, vol. Part F130951.

Yang, C., Fan, Q., Wang, T., Yin, G., Wang, H., 2016. Repolike: Personal repositories recommendation in social coding communities. In: Proceedings of the 8th Asia-Pacific Symposium on Internetware. In: Internetware '16, ACM, New York, NY, USA, pp. 54–62.

Yang, C.-Z., Tu, M.-H., 2012. LACTA: An enhanced automatic software categorization on the native code of android applications. Lect. Notes Eng. Comput. Sci. 2195, 769–773.

Yang, S., Yu, H., Deng, W., Lai, X., 2015. Mobile application recommendations based on complex information. In: Ali, M., Kwon, Y.S., Lee, C.-H., Kim, J., Kim, Y. (Eds.), Current Approaches in Applied Artificial Intelligence. Springer International Publishing, Cham, pp. 415–424.

Yao, Y., Zhao, W.X., Wang, Y., Tong, H., Xu, F., Lu, J., 2017. Version-aware rating prediction for mobile app recommendation. ACM Trans. Inf. Syst. 35 (4), 38:1–38:33.

Yaremchuck, S., Kharchenko, V., Gorbenko, A., 2018. Search of similar programs using code metrics and big data-based assessment of software reliability. In: Alani, M.M., Tawfik, H., Saeed, M., Anya, O. (Eds.), Applications of Big Data Analytics: Trends, Issues, and Challenges. Springer International Publishing, Cham, pp. 185–211.

Ye, X., Shen, H., Ma, X., Bunescu, R., Liu, C., 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings - International Conference on Software Engineering, Vol. 14-22-May-2016. pp. 404–415.

Yin, K., Chen, W., Zhou, J., Wu, G., Wei, J., 2018. Star: A specialized tagging approach for docker repositories. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC). pp. 426–435.

Yin, P., Luo, P., Lee, W.-C., Wang, M., 2013. App recommendation: A contest between satisfaction and temptation. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. In: WSDM '13, ACM, New York, NY, USA, pp. 395–404.

Yin, K., Zhou, J., Chen, W., Wu, G., Zhu, J., Wei, J., 2018. D-tagger: A tag recommendation approach for docker repositories. In: Proceedings of the Tenth Asia-Pacific Symposium on Internetware. In: Internetware '18, ACM, New York, NY, USA, pp. 3:1–3:10.

Yoon, Y.-C., Lee, J., Park, S.-Y., Lee, C., 2017. Fine-grained mobile application clustering model usingretrofitted document embedding. ETRI J. 39 (4), 443–454.

Yu, H., Lian, Y., Yang, S., Tian, L., Zhao, X., 2016. Recommending features of mobile applications for developer. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10086 LNAI, pp. 361–373.

Yu, H., Xia, X., Zhao, X., Qiu, W., 2017. Combining collaborative filtering and topic modeling for more accurate android mobile app library recommendation. In: Proceedings of the 9th Asia-Pacific Symposium on Internetware. In: Internetware'17, ACM, New York, NY, USA, pp. 17:1–17:6.

Yuan, C., Wei, S., Wang, Y., You, Y., ZiLiang, S., 2016. Android applications categorization using bayesian classification. In: 2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). pp. 173–176.

Yusof, Y., Alhersh, T., Mahmuddin, M., Mohamed Din, A., 2012. Classification of machine learning engines using latent semantic indexing.

Yusof, Y., Ramadan, Q.H., 2010. Automation of software artifacts classification. Int. J. Soft Comput. 5 (3), 109–115.

Yusof, Y., Rana, O.F., 2010. Classification of software artifacts based on structural information. In: Setchi, R., Jordanov, I., Howlett, R.J., Jain, L.C. (Eds.), Knowledge-Based and Intelligent Information and Engineering Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 546–555.

Zhang, Y., Lo, D., Kochhar, P.S., Xia, X., Li, Q., Sun, J., 2017. Detecting similar repositories on github. In: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 13–23.

Zhang, L., Zou, Y., Xie, B., Zhu, Z., 2014. Recommending relevant projects via user behaviour: An exploratory study on github. In: 1st International Workshop on Crowd-Based Software Development Methods and Technologies, CrowdSoft 2014 - Proceedings. pp. 25–30.

ZhangChao, WanLili, 2018. Evaluation and functionality stems extraction for app categorization on apple itunes store by using mixed methods: Data mining for categorization improvement. 17 (2), 111–128.

Zheng, Z., Wang, L., Xu, J., Wu, T., Wu, S., Tao, X., 2018. Measuring and predicting the relevance ratings between FLOSS projects using topic features. In: Proceedings of the Tenth Asia-Pacific Symposium on Internetware. In: Internetware '18, ACM, New York, NY, USA, pp. 12:1–12:10.

Zhou, J., 2016. Automated app Categorization using API analysis.

Zhou, J., Chen, W., Wu, G., Wei, J., 2019. Semitagrec: A semi-supervised learning based tag recommendation approach for docker repositories. In: Peng, X., Ampatzoglou, A., Bhowmik, T. (Eds.), Reuse in the Big Data Era. Springer International Publishing, Cham, pp. 132–148.

Zhu, H., Cao, H., Chen, E., Xiong, H., Tian, J., 2012. Exploiting enriched contextual information for mobile app classification. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management. In: CIKM '12, ACM, New York, NY, USA, pp. 1617–1621.

Zhu, H., Chen, E., Xiong, H., Cao, H., Tian, J., 2014. Mobile app classification with enriched contextual information. IEEE Trans. Mob. Comput. 13 (7), 1550–1563.

**Maximilian Auch** has been a research assistant at the University of Applied Sciences Munich since 2017 (at the Department of Computer Science and Mathematics) and is simultaneously employed as a software developer at Pentasys AG since 2017. Prior to this, he completed a Master of Science in Business Information Systems at the University of Applied Sciences in Munich. His research interests include software engineering, recommendation systems and knowledge management.

**Manuel Weber** studied information systems at the University of Stuttgart and the University of Hohenheim. Since March 2019, he works as a research assistant at the University of Applied Sciences Munich (Department of Computer Science and Mathematics) within the research group for distributed cognitive computing. His research interests are in machine learning, transfer learning and building occupancy estimation.

**Professor Dr. Peter Mandl** has been a professor with a focus on "Distributed Systems" at the Faculty of Computer Science and Mathematics at the University of Applied Sciences Munich since 2002, spokesman of the Competence Center for Information Systems (CCWI) at the University of Applied Sciences in Munich since 2005, and an executive board member of iSYS Software GmbH since 1996.

**Professor Dr. phil. habil. Christian Wolff** has been a professor in the Department of Media Informatics at the Institute for Information and Media, Language and Culture at the University of Regensburg since 2003. He is also a member of numerous national and international professional societies (including ACM and IEEE Computer Society). Since April 2009 he has been chairman of the University Association of Information Science. Since September 2016 he has again been a member of the leadership of the specialist group Media Informatics in the "Gesellschaft für Informatik" (GI) and since October 2016 he has been a member of the leadership of the European Chapter of the Association for Information Science and Technology (ASIST). In addition, he is a (provisional) spokesman of the regional group East Bavaria of the "Gesellschaft für Informatik".