2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR) | 979-8-3503-1184-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/MSR59073.2023.00016

Dealing with Popularity Bias in Recommender Systems for Third-party Libraries: How far Are We?

Phuong T. Nguyen
Università degli studi dell'Aquila, Italy
phuong.nguyen@univaq.it

Riccardo Rubei
Università degli studi dell'Aquila, Italy
riccardo.rubei@graduate.univaq.it

Juri Di Rocco
Università degli studi dell'Aquila, Italy
juri.dirocco@univaq.it

Claudio Di Sipio
Università degli studi dell'Aquila, Italy
claudio.disipio@graduate.univaq.it

Davide Di Ruscio
Università degli studi dell'Aquila, Italy
davide.diruscio@univaq.it

Massimiliano Di Penta
Università degli studi del Sannio, Italy
dipenta@unisannio.it

***Abstract*—Recommender systems for software engineering (RSSEs) assist software engineers in dealing with a growing information overload when discerning alternative development solutions. While RSSEs are becoming more and more effective in suggesting handy recommendations, they tend to suffer from popularity bias, i.e., favoring items that are relevant mainly because several developers are using them. While this rewards artifacts that are likely more reliable and well-documented, it would also mean that missing artifacts are rarely used because they are very specific or more recent. This paper studies popularity bias in Third-Party Library (TPL) RSSEs. First, we investigate whether state-of-the-art research in RSSEs has already tackled the issue of popularity bias. Then, we quantitatively assess four existing TPL RSSEs, exploring their capability to deal with the recommendation of popular items. Finally, we propose a mechanism to defuse popularity bias in the recommendation list. The empirical study reveals that the issue of dealing with popularity in TPL RSSEs has not received adequate attention from the software engineering community. Among the surveyed work, only one starts investigating the issue, albeit getting a low prediction performance.**

***Index Terms*—Recommender systems; Third-party libraries; Popularity bias; Fairness**

I. Introduction

In online shopping market, recommender systems have been conceived to enable customers to approach goods that suite their needs [8], [41]. Nonetheless, while being able to provide accurate results, these systems tend to present frequently seen items [6], [7], [16], i.e., *popularity bias* is a common phenomenon of various recommender systems. In fact, the ability to exhibit rare but useful items is considered as an added value [40]. The *long tail effect* indicates that a handful of items are extremely popular, whilst most of the remaining ones, so-called the long tail, are not known by customers [10], [55]. Essentially, products belonging to the long tail are considered a social good [7] and recommending them benefits both customers and shop owners [51].

Recommender systems for software engineering (RSSEs) effectively help developers cope with the proliferation of various data sources, and avoid information overload [21], [42]. RSSEs could suggest, for example, relevant developers' discussions, experts to handle issues or to fix bugs, code completions, or bug/vulnerability fixing. Among different types of RSSEs, in this paper, we focus on third-party library (TPL) RSSEs, that provide libraries relevant to the project under development [17], [25], [32], [36], [44], [49]. Previous work has shown such systems to be highly accurate and valuable for developers. Furthermore, TPL RSSEs enable developers to use existing infrastructures to build software without reinventing the wheel by offering tailored, ready-to-be-used pieces of functionality.

In TPL recommendation, the *Novelty* metric assesses if a system can retrieve libraries in the long tail and expose them to projects [36]. This increases the possibility of coming across *serendipitous* libraries [23], e.g., those that are seen by chance but turn out to be useful for the project under development [21]. For example, there could be a recent library, yet to be widely used, that can better interface with new hardware or achieve faster performance than popular ones. In summary, recommending only popular TPLs would harm the novelty of the results. We conjecture that the existing research in RSSEs, while achieving enhancement in accuracy, neglects the issue of dealing with popularity bias in RSSEs.

In this paper, we perform a threefold investigation on the issue of popularity bias in TPL RSSEs, contributing to advance the knowledge in the domain as follows:

– By means of a literature review on premier Software Engineering venues, we show that state-of-the-art research overlooks the issue of popularity bias in TPL RSSEs.
– We perform a qualitative analysis on recently developed RSSEs, studying their ability to deal with popularity bias.
– Through a quantitative study on four TPL RSSEs, we assess their sensitivity to popularity bias.
– We propose a practical solution to increase fairness in the recommendation lists.

2574-3864/23/$31.00 ©2023 IEEE
DOI 10.1109/MSR59073.2023.00016

The scripts and data produced in our study are available to foster future research [37].

Structure. Section II presents a motivating example and background related to popularity bias. Section III elaborates on the study materials and methods. Afterward, the study outcomes are reported and analyzed in Section IV. In Section V, we present empirical evidence on a possible counteraction. The related work is reviewed in Section VI. We sketch future work and conclude our paper in Section VII.

II. Motivating Analysis

We motivate our work by analyzing the nature of the recommendations provided by a state-of-the-art TPL RSSE, i.e., CrossRec [36][1] The system has been chosen for this motivating study because of: *(i)* original implementation availability and full documentation [36], allowing us to monitor the internal process; *(ii)* being less computationally-expensive than other tools such as LibSeek [25] and GRec [28].

We are interested in understanding how CrossRec and similar RSSEs treat libraries having different frequencies of occurrences. To this end, we leveraged a dataset of generic Java applications with 5,200 projects using 31,817 libraries. To assess how different TPLs are recommended, we employed a typical recommendation scenario for each project: half of its used libraries are used as query, and the other half are removed to be the ground-truth data. Given the TPLs as query, CrossRec is expected to return the ones in the ground truth.

The ability to provide matching libraries is deemed to be among the most important quality traits of RSSEs [42], including TPL RSSEs. Even though in practice a recommended library that does not belong to the ground-truth data might still be suitable for the project under consideration [23], [36]. However, the usefulness, as well as the relevance of such a library needs to be carefully evaluated with a proper user study [21], and this is out of scope for this work.

For each library, we consider the following parameters: $\mathbb{F}$ is the frequency of occurrences of a library in the training data; $\mathbb{G}$ is the number of projects containing the library in the ground-truth data; and $\mathbb{R}$ is the number of projects getting the library as a recommendation by means of CrossRec.

A. Popular and rare libraries

TABLE I: Libraries and CrossRec recommendations.

	Library	$\mathbb{F}$	$\mathbb{G}$	$\mathbb{R}$
Popular	junit:junit	3,926	3,468	4,615
	org.slf4j:slf4j-api	1,565	341	2,328
	log4j:log4j	1,106	890	2,592
	commons-io:commons-io	1,000	209	2,035
	com.google.guava:guava	956	414	2,105
	com.google.code.gson:gson	820	487	2,007
	org.mockito:mockito-core	689	548	1,658
	mysql:mysql-connector-java	627	97	1,065
	joda-time:joda-time	513	314	1,054
	com.h2database:h2	410	325	588
Rare	bsf:bsf	12	9	0
	net.oauth.core:oauth	9	5	1
	simple-jndi:simple-jndi	8	7	0
	javax.jdo:jdo-api	7	4	0
	javax.el:javax.el	6	4	0
	org.joda:joda-money	6	4	0
	taglibs:request	5	1	0
	javatar:javatar	4	1	0
	batik:batik-codec	3	1	0
	org.neo4j:parent	2	1	0

Table I reports how recommendation results vary for the 10 most popular (i.e., large $\mathbb{F}$) and 10 least popular (i.e., low $\mathbb{F}$) libraries in the dataset. Table I shows that the most popular libraries are also recommended very often ($\mathbb{R}$ is large). Unsurprisingly, junit:junit is the most popular library in the corpus as it appears $\mathbb{F}$=3,926 times in the training data. Looking into the ground-truth data, we see that the library belongs to $\mathbb{G}$=3,468 projects. However, it is recommended to $\mathbb{R}$=4,615 projects, meaning that its popularity may generate several false positives. This is not particularly surprising, considering that one could think about filtering out from recommendations libraries not used in production code, as is the case of junit:junit. However, results are also true for other popular libraries, i.e., they are frequently recommended by the RSSE, although they are not necessarily useful for several projects. Among others, mysql:mysql-connector-java belongs to the ground-truth data of $\mathbb{G}$=97 projects, however it is recommended $\mathbb{R}$=1,065 times. This means that, most of the times, the recommendation turns out to be a false positive. We computed the Spearman's correlation coefficient and obtained $\rho(\mathbb{F}, \mathbb{R})$=0.976, i.e., there is a strong correlation between the frequency of a library usage and the number of times it gets recommended. Interestingly, $\rho(\mathbb{G}, \mathbb{R})$=0.29, and this corresponds to a weak, and even no correlation between the number of times that a library appears in the ground-truth data and its frequency of recommendations. In other words, *a popular library is recommended just because it is popular, not because it is needed by projects*.

The bottom of Table I highlights a striking outcome: almost all the rare TPLs are never discovered by CrossRec, i.e., $\mathbb{R}$=0. For example, simple-jndi:simple-jndi belongs to eight and seven projects in the training and ground-truth data, respectively, however, it is never recommended. This means that projects such as apache/openjpa that contain simple-jndi:simple-jndi in their ground-truth data will never get such a library recommended. In summary, the rare TPLs in Table I get almost no chance to be recommended.

⋆ **Remark 1.** Popular libraries are excessively recommended to projects, leading to false positives. Given that TPL RSSEs present only popular libraries, the rare ones will never be recommended.

B. Solution-specific libraries

As previously mentioned, RSSEs suggest TPLs by relying on some definitions of project similarity. Unfortunately, this might not be sufficient to recommend TPLs specific to peculiar goals or used technology of a project at hand. Table II shows examples of true positive and false negative recommendations produced by CrossRec for some explanatory projects.

Eclipse Kapua [4] is an integration platform for IoT devices and smart sensors. Thus, it manages the connectivity of

[1] The original implementation of CrossRec has been used.

TABLE II: Explanatory CrossRec True Positive and False Negative recommendations.

Project	**True Positive**	**False Negative**
org.eclipse.kapua	commons-fileupload:commons-fileupload, com.h2database:h2, org.elasticsearch.client:transport, commons-io:commons-io, ...	org.apache.tomcat:tomcat-servlet-api, org.reflections:reflections, org.apache.activemq:artemis-amqp-protocol ...
org.eclipse.mylyn	junit:junit com.google.code.gson:gson, ...	org.eclipse.emf.ecore.xmi, org.eclipse.emf.common, org.eclipse.emf.ecore, org.eclipse.core.runtime, ...
hadoop.apache.org	org.hsqldb:hsqldb, org.mockito:mockito-all, commons-daemon:commons-daemon, commons-net:commons-net, ...	woodstox-core, kerb-simplekdc, ...

different devices in heterogeneous environments. By asking CrossRec to recommend libraries for org.eclipse.kapua by intentionally removing some actual dependencies, we only get generic libraries, like commons-io and spring-security-core. Unfortunately, libraries specific to the project's goal, i.e., managing connectivity and service development, such as tomcat-servlet-api and artemis-amqp-protocol, are not recommended.

Eclipse Mylyn [5] is the task and application lifecycle management (ALM) framework for Eclipse. CrossRec can recommend for such a project generic libraries like junit:junit, and com.google.code.gson:gson, whereas it fails to recommend libraries that are related to the Eclipse architecture such as OSGi [3] and EMF [2] specific libraries.

Apache Hadoop [1] is a framework for managing the distributed processing of large data sets across clusters of computers. Also in this case, CrossRec can correctly recommend libraries such as commons-daemon, common-net. In contrast, it is not able to provide useful libraries related to the specific goal of the project, like fast processing of XML documents (woodstox-core) and distributed authentication (kerb-simplekdc).

> ⋆ **Remark 2.** As found for CrossRec, state-of-the-art TPL RSSEs may not properly leverage peculiar aspects of a project, e.g., related to implementation solutions. Hence, they fail to recommend some relevant goals, architecture- or solution-specific libraries.

Altogether, we conclude that *popularity bias does exist in TPL RSSEs*. Such an issue needs to be tackled in order to decrease the number of frequent (while not necessarily useful) libraries, and to increase the number of the rare (but useful) ones in the recommendations.

III. Investigation materials and methods

The *goal* of this study is to investigate the extent to which TPL RSSEs have addressed popularity bias. The *quality focus* concerns the accuracy of the provided recommendations, which can be affected by the presence of popular artifacts in the training data. Finally, the *context* consists of *(i)* TPL RSSEs published in the recent software engineering literature; and *(ii)* three datasets from Java and Android open-source projects used to assess the recommenders.

This study addresses the following research questions:

- **RQ**$_1$: *Has the issue of dealing with popularity bias in TPL recommender systems been studied by the existing literature?* Through a literature analysis, we investigate to what extent the popularity bias in TPL RSSEs has ever been studied by state-of-the-art research. Though we do not aim for a complete, detailed systematic literature review, we adhere to existing guidelines for such type of study in Software Engineering research [26], [33], [54].
- **RQ**$_2$: *How well do state-of-the-art TPL recommender systems tackle popularity bias?* We analyze the most representative TPL RSSEs to investigate the extent to which they are susceptible to popularity bias. Afterward, we perform an empirical evaluation on four representative systems, i.e., LibRec [49], CrossRec [36], LibSeek [25], and GRec [28] further validating our hypothesis. Their source implementation is available, allowing us to experiment on real-world datasets according to our needs.
- **RQ**$_3$: *Can the re-ranking mechanism counteract popularity bias in TPL recommender systems?* Based on an existing Web search diversification technique [46], we derive a re-ranking strategy to defuse bias in the recommendation results provided by two TPL RSSEs.

A. RQ$_1$ Methodology: Literature analysis

Listing 1: Excerpt of the Scopus query string.

```
TITLE-ABS((‘‘recommendation’’ OR ‘‘recommender’’ OR ‘‘
        recommendation systems’’ OR ‘‘recommender systems’’)
AND (‘‘library’’ OR ‘‘libraries ’’ OR ‘‘TPL’’ OR ‘‘TPLs’’ OR ‘‘
        third-party libraries’’ OR ‘‘third-party ’’)  [...]
AND (LIMIT-TO (PUBYEAR, 2022) OR [...] OR LIMIT-TO (
        PUBYEAR, 2017))
```

To address RQ$_1$, we perform a literature analysis to gain a first understanding, based on the description available in scientific papers, of the extent to which the bias problem has been addressed for RSSEs. Aiming at a reasonable trade-off between efficiency and the coverage of state-of-the-art studies on popularity bias in TPL RSSEs, we employed a search strategy guided by four W-questions [57], i.e., "*Which?*" "*Where?*" "*What?*" "*When?*" explained as follows.

- *Which?* We conducted a combined search with automatic and manual activities to collect papers from various venues comprising conferences and journals.
- *Where?* The literature analysis was performed on premier software engineering venues including *(i)* nine conferences: ASE, ESEC/FSE, ESEM, ICSE, ICSME, ICST, ISSTA, MSR, and SANER; and *(ii)* five journals: EMSE, IST, JSS, TOSEM, and TSE.[2] The *Scopus* database[3] was chosen for the automatic search, and all the papers published by a given year of a given venue were retrieved through the advanced Scopus search and export features. An excerpt of the queries is shown in Listing 1.

[2] Venues listed in alphabetical order, for full names, interested readers can refer to the online appendix [37].

[3] https://scopus.com

- *What?* Title and abstract of each article were fetched following a set of predefined keywords.
- *When?* Since bias and fairness is a recent research theme, the search was confined to the most five recent years, i.e., from 2017 to 2022.

The search resulted in a corpus of 9,435 articles. Then, various filtering steps were conducted to scale down the search and retrieve only those that satisfy our predefined requirements. In particular, we are interested in studies dealing with RSSEs together with the relevant topics, i.e., TPL recommendation, bias, and fairness. The metadata of the collected studies is available in an online appendix [37].

B. RQ_2 Methodology: Quantitative analysis of popularity bias in four TPL RSSEs

To address RQ_2, we select four TPL RSSEs, i.e., LibRec [49], CrossRec [36], LibSeek [25], and GRec [28]. The rationale behind this selection is as follows. LibRec and CrossRec are notable systems for recommending TPL libraries for generic applications, while LibSeek and GRec are devised to work with Android apps. The four systems are also representative in terms of the working mechanisms. In particular, LibRec[4] and CrossRec[5] are based on collaborative filtering techniques and associate rule mining; whilst LibSeek[6] employs matrix factorization, and GRec[7] works on top of graph neural networks. We conjecture that, pending some possible external validity threats, an analysis of these systems could be generalizable to other TPL RSSEs.

1) Datasets and configurations: We use datasets curated by existing studies to evaluate the four TPL RSSEs. The datasets are summarized in Table III. Overall, we consider three datasets from Java open source software, two (DS_1 and DS_2) related to generic software systems, and one (DS_3) specific to Android apps.

TABLE III: Datasets and configurations.

	Dataset		
	DS$_1$ [36]	**DS**$_2$	**DS**$_3$ [25], [28]
# projects	1,200	5,200	56,091
# libraries	13,497	31,817	762
Type	Generic software	Generic software	Android apps
System	**Experimental configurations**		
LibRec	✔	✔	✖
CrossRec	✔	✔	✖
LibSeek	✖	✔	✔
GRec	✖	✔	✔

The bottom of Table III enumerates the considered experimental settings, where the columns represent datasets, and the rows correspond to systems. A cell is filled with a check mark (✔) if the tool in the corresponding row is executed on the dataset in the column, otherwise, it is filled with an uncheck mark (✖). In the ideal case, we should experiment with every possible pair of systems and datasets. This would lead to 12 configurations in total. Indeed, this is computationally expensive, considering the fact that we may spend several hours to execute one configuration. Thus we have to find the most representative configurations by reasoning as follows. DS_1 was used to evaluate and compare LibRec and CrossRec in the original work [36]. Instead, DS_3–originally named MALib[8]–was used in the original LibSeek [25] and GRec [28] papers. Accordingly, in this paper, DS_1 is used for the evaluation of LibRec and CrossRec, and DS_3 is for the evaluation of LibSeek and GRec. Moreover, we curated DS_2, an extended version of DS_1 with 4,000 additional projects. DS_2 is the only dataset applied in the evaluation of LibRec, CrossRec, LibSeek, and GRec. In this respect, DS_2 is the common ground to compare the performance of the four systems.

To give a first idea of how the dataset could cause popularity bias, we counted the number of occurrences for each library in a dataset, and sorted all the libraries in descending order of this number. The frequency for the libraries of DS_1, DS_2, and DS_3 is shown in Fig. 1. As it can be seen, there is always a long tail in all three datasets. For instance, in DS_1 the most popular library is `junit:junit`, being included in 969 over $1,200$ projects. Meanwhile, the majority of the TPLs are infrequent i.e., we count $10,212$ among $13,497$ libraries appearing only once in the whole dataset. The datasets reflect how real developers choose TPLs for their systems, either based on their own knowledge/searching on the Web, or possibly, leveraging some recommenders, yet we have no trace of that.

To aim for reliability in the evaluation, we performed the training and testing using different rounds of validation. In particular, we used the ten-fold cross-validation methodology to train and test the systems on the mentioned datasets.

2) Evaluation metrics: Given a set of projects P with a set of libraries L, we evaluate if the ranked top-N items returned by the systems contain long tail items, as well as if they match with the ground-truth data using *Expected popularity complement (EPC)*, *Catalog coverage*, *Precision* and *Recall*.

▷ **Expected popularity complement.** The metric measures *novelty*, i.e., the ability to provide libraries that are rarely seen, but match with the ground-truth data [36], [38], [50]:

$$EPC@N = \frac{\sum_{p\in P}\sum_{r=1}^{N}\frac{rel(p,r)*[1-pop(REC_r(p))]}{log_2(r+1)}}{\sum_{p\in P}\sum_{r=1}^{N}\frac{rel(p,r)}{log_2(r+1)}} \tag{1}$$

where $rel(p,r)$ is 1 if the library at the r position of the *top-N* list to project p belongs to the ground-truth data, 0 otherwise; $pop(REC_r(p))$ quantifies the popularity of the library at the position r of the recommended list. It is the ratio between the number of projects getting $REC_r(p)$ as a recommendation, and the number of projects getting the most frequently suggested library as a recommendation, i.e., $pop(REC_r(p)) = num(REC_r(p))/max_{l\in L}(num(l))$.

▷ **Coverage.** This metric gauges *sales diversity*, i.e., the ability of a system to suggest a wide range of libraries [36], as well as to spread the concentration among all items,

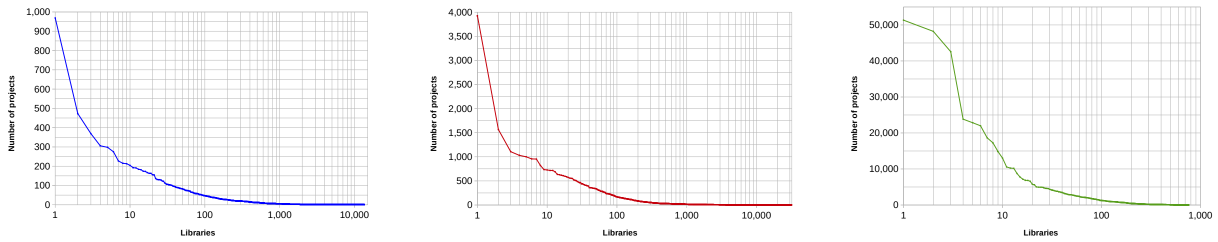
[4] The original implementation of LibRec was kindly provided by its authors.

[5] https://github.com/crossminer/CrossRec

[6] https://github.com/libseek/LibSeek

[7] https://github.com/fio1982/GRec

[8] https://github.com/malibdata/MALib-Dataset



(a) **Dataset DS**$_1$ (b) **Dataset DS**$_2$ (c) **Dataset DS**$_3$

Fig. 1: Frequency of occurrence for the libraries in the three considered datasets.

rather than focusing only on a limited set of libraries [42]: $Coverage@N = |\cup_{p \in P} REC_N(p)| \, / \, |L|$.

▷ **Precision and Recall.** *Precision* $P@N$ is the ratio of items matching with the ground-truth data to the total number of provided items; *Recall* $R@N$ is the ratio of matched items to the total number of ground-truth items.

C. RQ$_3$ *Methodology: Re-ranking the recommendations*

Based on xQuAD [46]–a framework for diversifying Web search results–we derive a re-ranking technique to rearrange the recommendation lists, aiming to reduce the number of popular (yet not necessarily useful) libraries, as well as to increase the number of unpopular (but useful) ones in the results. Starting from an initial ranked list R, we look for the long-tail items in the list, and promote them to upper positions to yield a new list S. S is initially set to $\varnothing$, and iteratively populated by enrolling new items using the following formula.

$$(1-\gamma)\mathcal{P}(l \mid p) + \gamma\mathcal{P}(l, \overline{S} \mid p) \tag{2}$$

where γ is the trade-off factor to harmonize accuracy and diversity; $\mathcal{P}(l \mid p)$ is the probability that project p invokes library l; $\mathcal{P}(l, \overline{S} \mid p)$ is the probability that p invokes l that does not belong to S, computed as follows [6]:

$$\mathcal{P}(l, \overline{S} \mid p) = \sum_{c \in H,T} \mathcal{P}(l, \overline{S} \mid c)\mathcal{P}(c \mid p) \tag{3}$$

where H and T are the set of head (popular) and tail (long tail) items, respectively. Under the assumption that the libraries are independent of each other with respect to the head and tail categories, $\mathcal{P}(l, \overline{S} \mid c)$ is identified below [6]:

$$\mathcal{P}(l, \overline{S} \mid c) = \mathcal{P}(c \mid p)\mathcal{P}(\overline{S} \mid c) = \mathcal{P}(c \mid p)\prod_{i \in S}(1-\mathcal{P}(i \mid c, S)) \tag{4}$$

where $\prod_{i \in S}(1-\mathcal{P}(i \mid c, S))$ is set to 1 if item i belongs to both S and category c, and 0 otherwise. From Equations 2 and 4, the final ranking score for a library l is computed as:

$$s_l = (1-\gamma)\mathcal{P}(l \mid p) + \gamma \sum_{c \in H,T} \mathcal{P}(c \mid p)\mathcal{P}(l \mid c)\prod_{i \in S}(1-\mathcal{P}(i \mid c, S)) \tag{5}$$

The γ parameter is used to adjust the trade off between accuracy and fairness [7]. By increasing γ, we give more weight to the second part of Equation 2, i.e., improving fairness, but less weight to the first part, i.e., decreasing accuracy. In fact, γ can be tuned to favor fairness or accuracy. In our experiments, we set γ to 0.2 so as to increase fairness, but avoid excessively interfering the accuracy at the same time.

IV. Results

This section reports the study results, addressing the research questions formulated in Section III.

A. ***RQ***$_1$*:* Has the issue of dealing with popularity bias in TPL recommender systems been studied by the existing literature?

The process in Section III-A resulted in a corpus with 9,435 articles from the considered venues in the five most recent years, i.e., from 2017 to 2022. We focused on research contributions related to TPL RSSEs and the associated theme, including bias and fairness. From the collected corpus we narrowed down the scope by using three sets of keywords as follows: *(i)* **REC**: "*recommendation*," "*recommender*," or "*recommendation systems*"; *(ii)* **LIB**: "*library*," "*libraries*," "*TPL*," "*TPLs*," "*third-party libraries*," and "*third-party*"; *(iii)* **BIF**: "*bias*," and "*fairness*." In Table IV we report the number of papers that contain both keywords in the corresponding column and row.[9] Our search targets papers containing one of the following three combinations: either *(i)* REC and LIB; or *(ii)* REC and BIF; or *(iii)* BIF and LIB. We mark the associated cells using the green color, and carefully examine these papers.

TABLE IV: Number of papers for the related topics.

	REC	LIB	BIF
REC	908		
LIB	17	104	
BIF	25	2	200

Following Table IV, we notice that the combinations of keywords expressed by REC, BIF, and LIB terms occur in 44 papers. In particular, for the combinations REC and BIF, REC and LIB, and BIF and LIB, we found 25, 17, and 2 articles, respectively. Starting from those, we read the title and abstract to elicit approaches and methodologies, checking whether the papers explicitly deal with popularity bias. The main findings of the study are discussed in the following.

[9] As the matrix is symmetric, we only list the numbers on the lower left part, and leave the upper right part blank, for the sake of clarity.

TABLE V: State-of-the-art recommender systems for mining TPLs (Listed in chronological order).

System	Venue	Year	Data source	Working mechanism	Prone to popularity bias?	Avail.
LibRec [49]	WCRE	2013	GitHub	LibRec is built on top of a light collaborative-filtering technique and association mining, looking for libraries that are used by popular projects	The system is exposed to popularity bias by its nature, retrieving only popular libraries thanks to association mining [9]	✔
LibCUP [44]	JSS	2017	GitHub	Usage patterns are discovered by means of DB-SCAN [22] – a hierarchical clustering algorithm	DBSCAN groups libraries that are most frequently co-used by projects. Therefore, popular libraries tend to get recommended more often	✖
LibFinder [39]	IST	2018	GitHub	NSGA-II [20] is used to maximize co-usage of libraries, the similarity with the candidates, and the total number of recommended items	A library L can be useful for a system S if L is commonly used with one or more libraries adopted by S. Evidence of bias is also reported in the paper	✖
CrossRec [36]	JSS	2020	GitHub	CrossRec employs a collaborative-filtering technique to mine TPLs from similar projects	The system is prone to popularity bias as it recommends libraries coming from projects that are similar	✔
Req2Lib [48]	SANER	2020	GitHub	Using the sequence to sequence technique, Req2Lib learns the library linked-usage information and semantic information in natural language	The model is trained with common sequences used by several similar projects, being exposed to popularity bias	✖
LibSeek [25]	TSE	2020	Google Play, GitHub, MVN	LibSeek uses matrix factorization, attempting to neutralize the bias caused by the popularity of TPLs by means of an adaptive weighting mechanism	***Due to its internal design, the system is expected to mitigate the effect of popularity bias*** (We are going to validate this claim in Section IV)	✔
GRec [28]	ESEC/FSE	2021	Google Play	Built on top of graph neural networks, GRec learns to recommend TPLs through app-library interactions	Thanks to the underlying link prediction technique, GRec is supposed to recommend popular libraries	✔

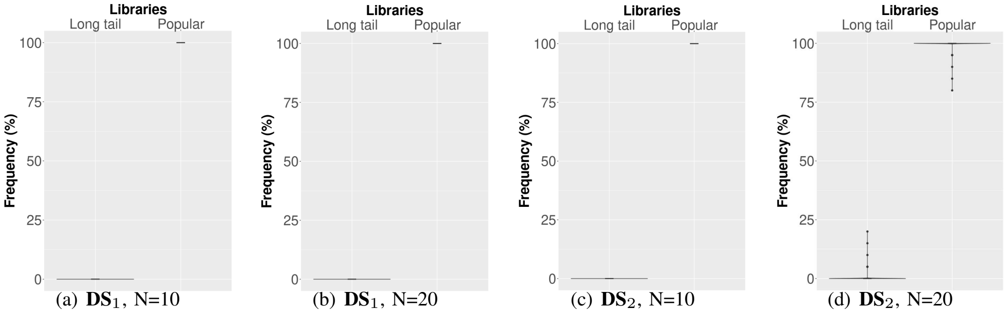
Among the studies that fall under the combination REC and BIF, Zerouali and Mens [56] conducted an empirical study to evaluate the distribution of Java libraries over GitHub projects. Their findings indicated that most of the projects include a limited set of popular libraries that may affect the results of recommender systems. A similar study was carried out on a curated Android dataset to investigate the impact of popular libraries considering several qualitative aspects, i.e., malware detection, application repackaging, and static code analysis [31]. The results showed that common libraries could hamper the performance of repackaging detection tools by introducing both false positives and false negatives. Moreover, the same negative effects were detected in the performance of machine learning-based malware classifiers, i.e., removing the most popular libraries increases the overall accuracy on average. He *et al.* [24] examined the long-tail effects during the migration of TPLs employing a fine-grained commit-level analysis of $19,652$ Java GitHub projects. The rest of the examined studies covers other types of bias, i.e., confirmation bias in testing analysis [45], and fairness in crowd worker recommenders [53]. *This is actually not related to the topic being considered, i.e., popularity bias in TPL recommendation.*

Concerning TPL RSSEs, by combining LIB and REC we ended up with ten papers, however only six of them present TPL recommender systems. Apart from these, we searched the same venues to look for similar systems over the same period, and we came across LibRec [49] which is among the first TPL RSSEs, being used as a baseline for various studies [25], [28], [36]. We included it in our study to further expand the scope of the analysis. We report the seven TPL RSSEs in Table V, paying attention to their working mechanisms as well as the possibility of being exposed to popularity bias.

Overall, the table suggests that diverse underlying techniques are used to recommend libraries. In particular, besides collaborative-filtering based approaches [36], [49], there are those that employ clustering algorithms [44], or NSGA-II [39]. Notably, deep neural networks [25], [28], [48] and matrix factorization [25] also found their application in TPL recommendations. LibRec works on top of a light collaborative-filtering technique and association mining, retrieving libraries that are used by popular projects. LibCUP [44] mines usage patterns using DBSCAN, a hierarchical clustering algorithm. LibFinder [39] makes use of the NSGA-II (Non-dominated Sorting Genetic Algorithm) multi-objective search algorithm to perform recommendations. CrossRec [36] exploits a graph-based structure to recommend relevant TPLs given the developer's context. Req2Lib [48] suggests relevant TPLs starting from the textual description of the requirements to handle the cold-start problem by combining a Sequence-to-Sequence network with a doc2vec pre-trained model. Similarly, GRec [28] encodes mobile apps, TPLs, and their interactions in an app-library graph. Afterward, it uses a graph neural network to distill the relevant features to increase the overall accuracy. Altogether, *all these systems are not conceived to mitigate the effect of popularity bias*.

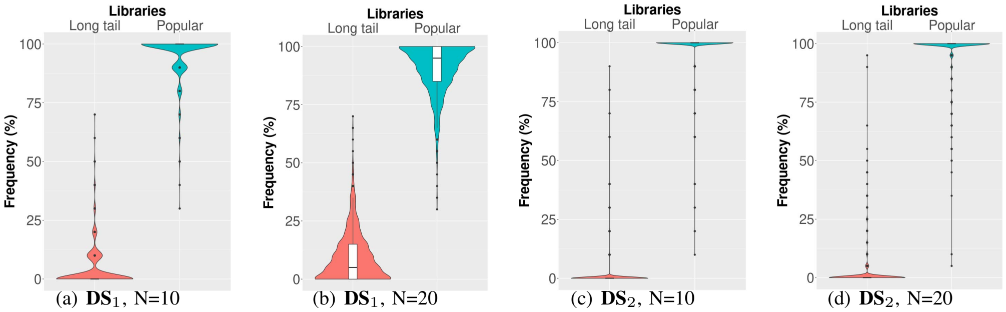
Apart from the seven studies presenting TPL RSSEs previously described, the remaining four of the eleven belonging to the combination of LIB and REC, tackle different issues as reported as follows. Chen *et al.* [15] proposed an unsupervised deep learning approach to embed both usage and description semantics of TPLs to infer mappings at the API level. An approach [14] based on Stack Overflow was proposed to recommend analogical libraries, i.e., a library similar to the ones that developers already use. Nafi *et al.* [34] developed XLibRec, a tool that recommends analogical libraries across different programming languages. Rubei *et al.* [43] investigated the usage of a learning-to-rank mechanism to embody explicit user feedback in TPLs recommenders. In summary, *there is no paper among the ones discussed above copes with popularity bias in RSSEs.*

He *et al.* [25] are, to the best of our knowledge, the first to address the issue of recommending diversified libraries. They found out that the prediction results provided by existing TPL RSSEs tend to favor a small set of libraries. They also highlighted that recommending popular TPLs hampers the usefulness of the recommendations, resulting in the lack of novelty and serendipity, which are deemed to be among the desired features of RSSEs [36]. Thus, He *et al.* proposed



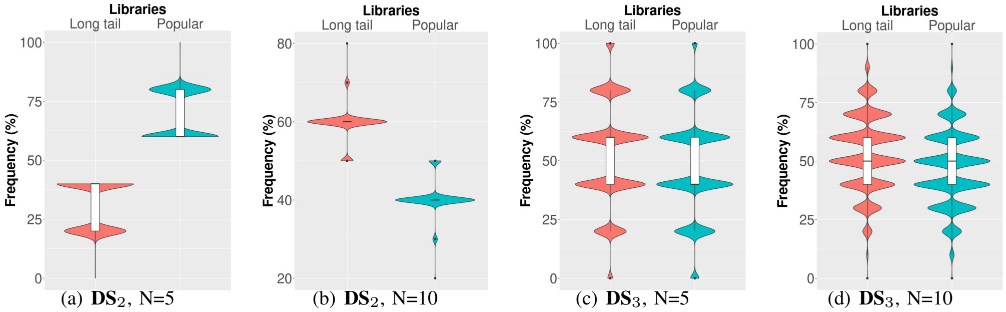
(a) **DS**$_1$, N=10 (b) **DS**$_1$, N=20 (c) **DS**$_2$, N=10 (d) **DS**$_2$, N=20

Fig. 2: LibRec: Recommendation of popular and long tail libraries.



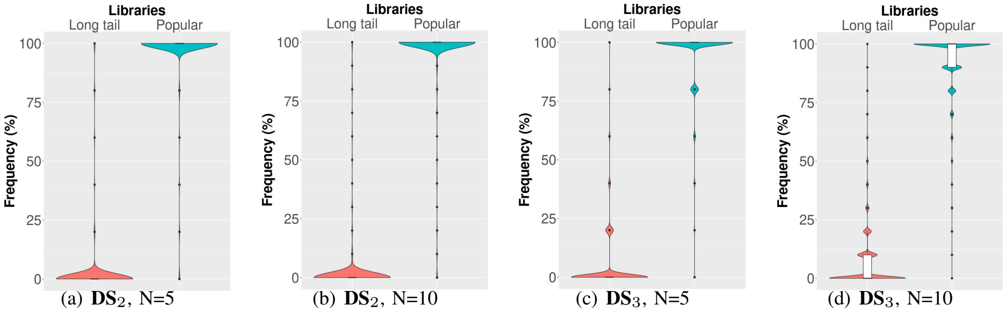
(a) **DS**$_1$, N=10 (b) **DS**$_1$, N=20 (c) **DS**$_2$, N=10 (d) **DS**$_2$, N=20

Fig. 3: CrossRec: Recommendation of popular and long tail libraries.



(a) **DS**$_2$, N=5 (b) **DS**$_2$, N=10 (c) **DS**$_3$, N=5 (d) **DS**$_3$, N=10

Fig. 4: LibSeek: Recommendation of popular and long tail libraries.



(a) **DS**$_2$, N=5 (b) **DS**$_2$, N=10 (c) **DS**$_3$, N=5 (d) **DS**$_3$, N=10

Fig. 5: GRec: Recommendation of popular and long tail libraries.

LibSeek as the first system to provide diversified libraries. The system uses an adaptive weighting mechanism to neutralize popularity bias by assigning higher weights to less popular TPLs. The evaluation conducted on a curated Android dataset confirmed that *LibSeek succeeds in providing a wide range of libraries, thus increasing novelty in the recommendation outcomes*.

> **Answer to RQ$_1$.** So far, dealing with popularity bias in TPL recommender systems has not received adequate attention from the Software Engineering community. Among the considered studies, LibSeek is the sole attempt to improve diversity in the recommendation results.

*B. **RQ$_2$**:* How well do state-of-the-art TPL recommender systems tackle popularity bias?

As explained in Section III-B, four representative systems were chosen, i.e., LibRec [49], CrossRec [36], LibSeek [25], and GRec [28], whose source implementation is available (marked with a ✔ in Table V), allowing us to experiment on real-world datasets according to our needs. Following existing studies [6], [7], we split each list in Fig. 1 into two parts, the first 20% TPLs are determined as popular, and the remaining 80% libraries represent the long tail. For each project, we count in the recommendations the percentage of TPLs that belong to the popular part and the long tail. Moreover, we consider another parameter that affects the final recommendations, i.e., N, the cut-off value of the ranked list. As LibRec and CrossRec return a long list of items, we consider two cut-off values, i.e., N=$\{10, 20\}$. Instead, LibSeek and GRec only provide short ranked lists, and thus only small cut-off values are allowed, i.e., N=$\{5, 10\}$. Besides popularity, we measure the accuracy using precision and recall, two popular metrics for evaluating RSSEs [28], [36]. The evaluation is focused on two aspects, i.e., if the systems *(i)* are subject to popularity bias; and *(ii)* can provide accurate recommendations.

1) Popularity bias: Based on the obtained results, we study how the four systems deal with popularity bias as follows.

▷ **LibRec.** The results obtained with LibRec on DS$_1$ and DS$_2$ are depicted in Fig. 2. It is evident that LibRec is severely exposed to popular bias, i.e., the entire plots concentrate on the 100% level of popularity, implying that all the TPLs belong to the 20% frequent items. Essentially, LibRec barely provides any TPL in the long tail. This holds for both datasets and cut-off values. This happens due to the fact that LibRec works based on association rule mining [9], i.e., *recommending the items widely used by the majority of projects.*

▷ **CrossRec.** Fig. 3 shows that, compared to LibRec, CrossRec can cope with popularity bias better. On DS$_1$, when N=10, CrossRec generally recommends very popular libraries. However, it can provide few libraries in the long tail. The ability to recommend rare TPLs is improved when a longer list is considered as shown in Fig. 3(b), when N=20. The system provides more popular libraries and fewer unpopular libraries by DS$_2$ as we can see in Fig. 3(c) and Fig. 3(d). This means *CrossRec is less successful in defusing popularity bias on datasets such as DS$_2$, where there are more TPLs, as it can be noticed from the long tail.*

▷ **LibSeek.** As shown in Fig. 4(a), LibSeek suffers from popularity bias on DS$_2$ when N=5. Nevertheless, it improves fairness by recommending TPLs in the long tail when a longer ranked list is considered, i.e., N=10. In particular, Fig. 4(b) shows that LibSeek introduces more infrequent items compared to the popular ones. By comparing Fig. 4(a) and Fig. 4(b), we conclude that LibSeek provides long tail TPLs late in the list. The results in Fig. 4(c), and Fig. 4(d) confirm the claim made by the authors of LibSeek [25] (and thus the analysis in Table V), i.e., the tool can handle well the items in the long tail, mitigating the effect of popularity bias. Notably, the ability of LibSeek to provide long tail items is evident only in DS$_3$, which contains a large number of projects, but a small number of TPLs (see Table III). Meanwhile, by DS$_2$, where there are fewer projects but more libraries, the system fails to cope with long tail items as shown in Fig. 4(a). Altogether, this shows that *while being able to cope with popularity bias in general, LibSeek does not succeed in some certain cases. Indeed, the characteristics of a dataset might be a contributing factor in the ability of LibSeek to mitigate such a bias.*

▷ **GRec.** As shown in Fig. 5, the system mainly recommends popular TPLs. By both DS$_2$ (Fig. 5(a) and Fig. 5(b)) and DS$_3$ (Fig. 5(c) and Fig. 5(d)) as well as both values of N, most of the TPLs provided by GRec are frequent, i.e., the plots are concentrated on the maximum level of popularity, only tiny fractions of the libraries reside in the long tail. This means that it provides to developers only libraries that are used by several projects, while ignoring the less popular ones. In a nutshell, the results in Fig. 5 reveal that *GRec cannot deal with popularity bias in the recommendation results.*

TABLE VI: Prediction accuracy.

Systems	**Datasets**	Mean Precision	Mean Recall
LibRec	DS$_1$	0.211	0.217
	DS$_2$	0.272	0.243
CrossRec	DS$_1$	0.289	0.123
	DS$_2$	**0.342**	**0.328**
LibSeek	DS$_2$	0.031	0.090
	DS$_3$	0.211	0.694
GRec	DS$_2$	0.062	0.212
	DS$_3$	0.221	0.713

2) Accuracy: An important feature of any recommender system is its ability to provide accurate items [42]. We analyze the prediction of the considered systems by referring to Table VI. Concerning the results obtained on DS$_2$ (rows marked in gray), it can be noticed that, although LibRec and CrossRec are not very successful at providing diversified TPLs (see Fig. 2 and Fig. 3), they obtain a better prediction accuracy compared to that of LibSeek and GRec. On DS$_2$, CrossRec gets 0.342 and 0.328 as precision and recall, being the best tool concerning prediction accuracy. LibSeek is fairly good at coping with popularity bias; however, this comes at a price: It achieves a very low accuracy compared to the others. The precision and recall scores for DS$_2$ are 0.031 and 0.090, respectively, and for DS$_3$ 0.211 and 0.694, respectively.

Likewise, GRec also fails to provide relevant items, achieving a low accuracy on DS_2, i.e., precision and recall are 0.062 and 0.212, respectively. Only by DS_3, GRec improves its prediction by getting 0.221 and 0.713 as precision and recall.

Answer to RQ_2. Three among the systems recommend very popular third-party libraries, while usually ignoring those that are rarely used by projects. Only LibSeek is able to recommend libraries in the long tail, however, it fails to maintain a trade-off between fairness and precision, resulting in a low accuracy.

C. *RQ_3:* Can the re-ranking mechanism counteract popularity bias in TPL recommender systems?

Based on RQ_2 results, we found that generally, LibSeek recommends a wide range of libraries instead of focusing on only popular items. Thus, the recommendations provided by LibSeek are less prone to popularity bias than those by other systems. Nevertheless, RQ_2 also shows that popularity bias is still an issue for TPL RSSEs, triggering the need for proper countermeasures. Due to these reasons, we do not choose LibSeek for experimenting with the re-ranking technique, which needs a considerably biased recommendation list as input. In turn, GRec returns a short recommendation list, which gives no opportunity for re-ranking by rewarding libraries in the lower part of the ranked list. Thus, from the four systems in RQ_2, we decided to apply re-ranking on LibRec and CrossRec, which usually return a longer ranked list of items. We experimented with the recommendations obtained by the two systems on DS_1 and DS_2. After a first empirical evaluation, we noticed that on DS_1, LibRec always recommends very popular items, even late in the list, and it never provides any rare libraries (see Fig. 2(a) and Fig. 2(b)). Therefore, LibRec could only benefit from re-ranking on DS_2.

The results are depicted in Fig. 6, Fig. 7, and Fig. 8. Overall, we witness a similar pattern in the impact of the proposed ranking mechanism on the two RSSEs according to the three considered metrics. Thus, we analyze the performance by reporting the results according to each metric as follows.

$\triangleright$ **EPC.** Following Equation 1, a larger EPC means better novelty, as shown in Fig. 6(a), Fig. 7(a), and Fig. 8(a) it is evident that by moving rare libraries upper in the ranked list, we are able to improve the EPC scores for the recommendations of both systems, by all the cut-off values (N). Especially for LibRec, there is a sharper increase in EPC compared to CrossRec. For instance, as shown in Fig. 6(a), when N=10 we obtain an EPC of 3.0 after re-ranking, which is greater than EPC=2.0 the corresponding value achieved on the original list.

$\triangleright$ **Coverage.** Fig. 6(b), Fig. 7(b), and Fig. 8(b) show that there is a light improvement in the coverage of the recommendations once the lists have been re-ranked. The gain is more evident for the bottom part of the list. This is understandable as the proposed technique tends to pad rare items at the end of each list. As a consequence, re-ranking increases coverage, recommending a wider set of libraries, thus improving diversity [36].

$\triangleright$ **Precision and Recall.** We investigate how the prediction accuracy changes after the re-ranking process by sketching the precision-recall curves following the cut-off values N in Fig. 6(c), Fig. 7(c), Fig. 8(c). Essentially, a curve close to the right upper corner represents both higher precision and recall, corresponding to more accurate predictions [21]. We can notice how the curve representing the performance after re-ranking stays a bit below the one for the original list. This corresponds to a slight setback in precision and recall, once the re-ranking has been conducted. The results indicate that, by promoting the long tail items upper in the list, we encounter false positives in several projects. This is understandable as promoting rare libraries brings true positives to a handful of projects, and false positives to others.

Answer to RQ_3. On the considered systems, i.e., LibRec and CrossRec, the re-ranking technique helps mitigate popularity bias and increase novelty in the recommendation results. Nevertheless, it introduces a setback in prediction accuracy, necessitating further investigation to solve the popularity bias problem properly.

V. Discussion

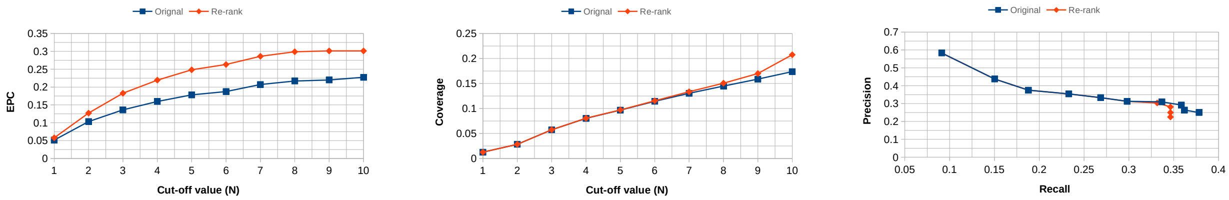
This section discusses possible implications and threats to the validity of our findings.

A. Implications

RQ_1 and RQ_2 reveal that software engineering research has not properly dealt with popularity bias in TPL RSSEs, triggering the need for effective counteracting mechanisms. Looking at other domains, we see that there are three major methods to combat bias in general, i.e., the pre-processing, in-processing, and post-processing paradigms [18]. We assume that they can also be adopted in Software Engineering for the same purpose, as it was also advocated by Chakraborty *et al.* [13], though not directly for RSSEs. Also, as it can be seen from Section IV, He *et al.* [25] conceived LibSeek– the very first TPL RSSE to mitigate the abundance of highly frequent libraries, following the *in-processing* approach. LibSeek attempts to neutralize the bias caused by the popularity of TPLs using an adaptive weighting mechanism. However, while it can diversify the recommendations, LibSeek suffers from a low accuracy (see RQ_2 in Section IV-B). This implies that there is still room for improvement, i.e., conceiving more effective in-processing techniques for RSSEs.

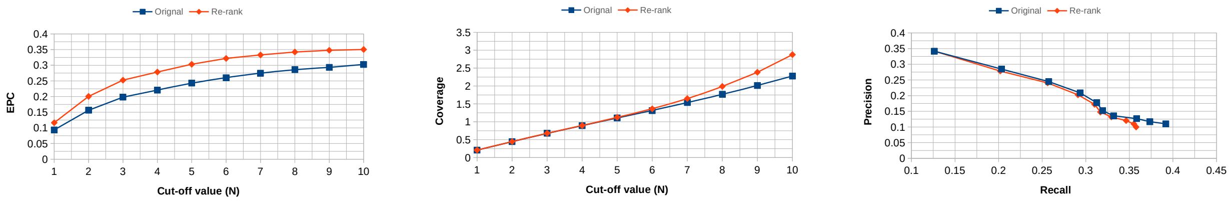
Through RQ_2, we can notice that the characteristics of datasets are a contributing factor in the ability of LibSeek to mitigate popularity bias. In fact, DS_3 contains a considerably large number of projects (56,091), but only a small number of libraries (762). In contrast, compared to DS_3, DS_2 has a lower number of projects (5,200), but its number of libraries is substantially higher (31,817). In this respect, the distribution of the libraries across the projects in DS_3 is much denser than that of DS_2, as the former features projects from the same ecosystem (Android). It is then necessary to perform an in-depth analysis of how the characteristics of a dataset impact on the ability of TPL recommender systems to deal with popularity bias.

In this work, we derived a *post-processing* technique following an algorithm for diversifying Web search results, defusing



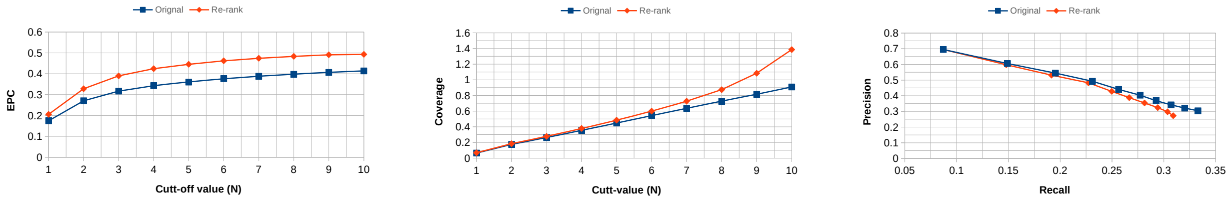
(a) Expected popularity complement (b) Coverage (c) Precision and recall

Fig. 6: Results obtained on the recommendations by running LibRec with DS_2.



(a) Expected popularity complement (b) Coverage (c) Precision and recall

Fig. 7: Results obtained on the recommendations by running CrossRec with DS_1.



(a) Expected popularity complement (b) Coverage (c) Precision and recall

Fig. 8: Results obtained on the recommendations by running CrossRec with DS_2.

popularity bias without touching the internal design of the considered systems. On the one hand, the technique helps improve diversity in the recommendations of two systems. On the other hand, the improvement is rather modest, and there is still a slight decrease in accuracy. Our findings suggest that further research should be conducted to propose effective countermeasures. We suppose that it is crucial to consider additional factors, e.g., the degree of specificity (to certain solutions) of a TPL, when it comes to library recommendation.

Pre-processing techniques can also be useful for TPL RSSEs, e.g., by considering factors related to peculiar, solution-specific aspects of a project that, as pointed out in Section II are neglected, and that would help reward certain libraries. From RQ_1, we see that no approach specifically employs pre-processing to cope with popularity bias in RSSEs.

B. Threats to validity

Threats to *construct validity* are related to the relationship between theory and observation. This concerns the setting used to query the RSSEs. We simulated a basic scheme where given a testing project, half of its libraries are used as a query, and the remaining is ground-truth data. Such a scenario resembles a practical use, in which developers search for suitable TPLs once they have invoked some libraries in their project.

Threats to *internal validity* are the confounding factors internal to our study that might have an impact on the results. For the literature analysis, given a paper that needs to be reviewed, we only classified it by its title and abstract. A possible threat is that popularity bias can be addressed elsewhere in the paper, e.g., approach/evaluation. In this case, we may miss relevant studies. Another threat is related to the considered venues, i.e., we might only cover some of the relevant conferences and journals. As shown in Section III-A, we considered major software engineering venues where state-of-the-art research in RSSEs is presented. We used their original implementations to experiment on the four selected RSSEs.

Threats to *external validity* concern the generalizability of our results. The conclusion drawn from the experiments is valid for the four considered systems, and it may no longer apply to other TPL RSSEs. We anticipate that expanding the evaluation on more systems will help us further validate our hypothesis. To aim for diversity in the training data, we used datasets that cover two major domains, i.e., generic software and Android applications, which exhibit diverse characteristics to simulate real-world data.

VI. Related work

The literature analysis in Section IV-A shows that popularity bias has not received adequate attention from the community. In this section, we further review related work concerning fairness in Software Engineering and tackling bias in recommender systems in other domains.

A. Fairness in Software Engineering

Brun and Meliou [11] investigated the potential impact of biased data in the critical software engineering phases, i.e., requirement specification, system design, testing, and verification. Their investigation indicated that a comprehensive classification of software biases remains an open challenge. Similarly, Verma and Rubin [52] studied fairness in the context of algorithmic classification. Their evaluation with a state-of-the-art dataset and logistic regression classifiers shows that fostering software fairness is challenging since the tested classifier is defined as fair according to the chosen definition. Spoletini *et al.* [47] proposed an initial set of bias-aware guidelines, providing practical instructions when it comes to dealing with bias in software engineering algorithms.

Chakraborty *et al.* [13] advocated for the need for fairness analysis and testing in machine learning software. They propose an approach named Fairway that removes bias during preprocessing (i.e., before training) and in-processing (i.e., during training), and uses multi-objective optimization to avoid that fairness compromises the machine learning performance. In subsequent work, Chakraborty *et al.* [12] proposed a fairness-aware data rebalancing approach, named Fair-SMOTE, that leverages situation testing to balance fair-sensitive labels, outperforming previously-proposed approaches including the previously-proposed Fairway. While Chakraborty *et al.* deal with fairness by handling fair-related attributes, in RQ$_3$ of this work we leverage a technique inspired by Web search [46] to diversify TPL recommendations. Last but not least, multi-objective approaches like the one of Chakraborty *et al.* can also be applied in mitigating popularity bias in TPL RSSEs, however, the goal is different (i.e., reranking rare items that could be useful for some projects instead of coping with fairness-related features).

B. Recommender system bias in other domains

Kowald and Lacic [27] investigated popularity bias in collaborative filtering (CF) multimedia recommender systems. They first selected four datasets composed of users and corresponding media items and then evaluated four CF algorithms in terms of popularity bias. Their results showed that users with a low preference to popular items receive significantly worse recommendations compared to the others, meaning that all the considered systems are prone to popularity bias.

The CPFair framework [35] leverages an optimization-based approach to support the multi-stakeholders fairness requirements in the multimedia domain. Given a list of items provided by an unfair recommender, CPFair exploits a re-ranking post-hoc algorithm to integrate the fairness requirements and remove the bias in the initial set of recommended items.

d'Aloisio *et al.* [19] proposed a data-agnostic Debiaser for Multiple Variables (DEMV) to mitigate biases in multi-class classification problems. The approach alleviates bias using random sampling to remove or duplicate elements until the observed size converges to the expected one. FairSR [29] is a fairness-aware sequential recommender system that exploits knowledge graphs to increase interaction fairness, i.e., users belonging to different groups interact with the items equally.

Li *et al.* [30] proposed a recommender system based on Generative Adversarial Networks, called FairGAN, to handle fairness with implicit user feedback. The tool extracts feedback from user interactions and ranks the items accordingly. Afterward, it generates fairness signals to enforce the exposure of items by satisfying different fairness criteria.

While previous work has studied recommender bias in other domains, ours is the first thorough investigation of popularity bias for RSSEs, and specifically for TPL RSSEs, reviewing the existing literature, by comparing the behavior of four TPL RSSEs, and attempting to mitigate their popularity bias.

VII. Conclusion and future work

In this paper, we performed both a qualitative and quantitative evaluation to study the presence of popularity bias in TPL RSSEs. A literature review on major SE venues reveals that the issue of dealing with popularity bias has not received enough attention from the community. The finding is further confirmed with an empirical evaluation on four TPL RSSEs, i.e., three among the considered systems recommend highly frequent libraries to projects. One system, while being able to undermine the effect of recommending popular libraries, suffers from a low prediction accuracy. Altogether, we see that state-of-the-art research in software engineering overlooks the problem of popularity bias in third-party library recommender systems, leaving a research gap that needs to be properly filled.

For future work, we plan to conceive more effective mechanisms for counteracting popularity bias, increasing fairness in the recommendations, e.g., by means of a multi-objective approach, while keeping high accuracy. More importantly, we assume that it is necessary to study the issue of popularity bias in other RSSEs, for instance, API and code recommender systems as well as pre-trained models for code search.

Acknowledgments

This work has been partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY).

References

[1] "Apache Hadoop." [Online]. Available: https://hadoop.apache.org/

[2] "Eclipse EMF." [Online]. Available: https://www.eclipse.org/modeling/emf/

[3] "Eclipse Equinox." [Online]. Available: https://www.eclipse.org/equinox/

[4] "Eclipse Kapua." [Online]. Available: https://projects.eclipse.org/projects/iot.kapua

[5] "Eclipse Mylyn." [Online]. Available: https://projects.eclipse.org/projects/tools.mylyn

[6] H. Abdollahpouri, R. Burke, and B. Mobasher, "Managing popularity bias in recommender systems with personalized re-ranking," in *Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference, Sarasota, Florida, USA, May 19-22 2019*, R. Barták and K. W. Brawner, Eds. AAAI Press, 2019, pp. 413–418. [Online]. Available: https://aaai.org/ocs/index.php/FLAIRS/FLAIRS19/paper/view/18199

[7] H. Abdollahpouri, M. Mansoury, R. Burke, and B. Mobasher, "The unfairness of popularity bias in recommendation," in *Proceedings of the Workshop on Recommendation in Multi-stakeholder Environments co-located with the 13th ACM Conference on Recommender Systems (RecSys 2019), Copenhagen, Denmark, September 20, 2019*, ser. CEUR Workshop Proceedings, R. Burke, H. Abdollahpouri, E. C. Malthouse, K. P. Thai, and Y. Zhang, Eds., vol. 2440. CEUR-WS.org, 2019. [Online]. Available: http://ceur-ws.org/Vol-2440/paper4.pdf

[8] C. C. Aggarwal, *Recommender Systems - The Textbook*. Springer, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-29659-3

[9] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 207–216. [Online]. Available: https://doi.org/10.1145/170035.170072

[10] C. Anderson, *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[11] Y. Brun and A. Meliou, "Software fairness," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 754–759. [Online]. Available: https://doi.org/10.1145/3236024.3264838

[12] J. Chakraborty, S. Majumder, and T. Menzies, "Bias in machine learning software: why? how? what to do?" in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*. ACM, 2021, pp. 429–440.

[13] J. Chakraborty, S. Majumder, Z. Yu, and T. Menzies, "Fairway: a way to build fair ML software," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*. ACM, 2020, pp. 654–665.

[14] C. Chen, Z. Xing, and Y. Liu, "What's spain's paris? mining analogical libraries from q&a discussions," *Empirical Softw. Engg.*, vol. 24, no. 3, p. 1155–1194, jun 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9657-y

[15] C. Chen, Z. Xing, Y. Liu, and K. O. L. Xiong, "Mining likely analogical apis across third-party libraries via large-scale unsupervised api semantics embedding," *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 432–447, 2021.

[16] J. Chen, H. Dong, X. Wang, F. Feng, M. Wang, and X. He†, "Bias and debias in recommender system: A survey and future directions," *ACM Trans. Inf. Syst.*, oct 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3564284

[17] M. Chouchen, A. Ouni, and M. W. Mkaouer, "Androlib: Third-party software library recommendation for android applications," in *Reuse in Emerging Software Engineering Practices: 19th International Conference on Software and Systems Reuse, ICSR 2020, Hammamet, Tunisia, December 2–4, 2020, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 208–225. [Online]. Available: https://doi.org/10.1007/978-3-030-64694-3_13

[18] B. d'Alessandro, C. O'Neil, and T. LaGatta, "Conscientious classification: A data scientist's guide to discrimination-aware classification," *Big Data*, vol. 5, no. 2, pp. 120–134, 2017, pMID: 28632437. [Online]. Available: https://doi.org/10.1089/big.2016.0048

[19] G. d'Aloisio, A. D'Angelo, A. Di Marco, and G. Stilo, "Debiaser for multiple variables to enhance fairness in classification tasks," *Information Processing & Management*, vol. 60, no. 2, p. 103226, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457322003272

[20] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002. [Online]. Available: https://doi.org/10.1109/4235.996017

[21] J. Di Rocco, D. Di Ruscio, C. Di Sipio, P. T. Nguyen, and R. Rubei, "Development of recommendation systems for software engineering: the CROSSMINER experience," *Empirical Software Engineering*, vol. 26, no. 4, p. 69, 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-09963-7

[22] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

[23] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: Evaluating recommender systems by coverage and serendipity," in *Proceedings of the Fourth ACM Conference on Recommender Systems*, ser. RecSys '10. New York, NY, USA: ACM, 2010, pp. 257–260. [Online]. Available: http://doi.acm.org/10.1145/1864708.1864761

[24] H. He, R. He, H. Gu, and M. Zhou, "A large-scale empirical study on java library migrations: Prevalence, trends, and rationales," in *Proceedings of the 29th ESEC/FSE*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 478–490. [Online]. Available: https://doi.org/10.1145/3468264.3468571

[25] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang, "Diversified third-party library prediction for mobile app development," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[26] B. A. Kitchenham, P. Brereton, Z. Li, D. Budgen, and A. J. Burn, "Repeatability of systematic literature reviews," in *15th International Conference on Evaluation & Assessment in Software Engineering, EASE 2011, Durham, UK, 11-12 April 2011, Proceedings*, 2011, pp. 46–55. [Online]. Available: https://doi.org/10.1049/ic.2011.0006

[27] D. Kowald and E. Lacic, "Popularity Bias in Collaborative Filtering-Based Multimedia Recommender Systems," in *Advances in Bias and Fairness in Information Retrieval*, ser. Communications in Computer and Information Science, L. Boratto, S. Faralli, M. Marras, and G. Stilo, Eds. Cham: Springer International Publishing, 2022, pp. 1–11.

[28] B. Li, Q. He, F. Chen, X. Xia, L. Li, J. Grundy, and Y. Yang, "Embedding app-library graph for neural third party library recommendation," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 466–477. [Online]. Available: https://doi.org/10.1145/3468264.3468552

[29] C.-T. Li, C. Hsu, and Y. Zhang, "FairSR: Fairness-aware Sequential Recommendation through Multi-Task Learning with Preference Graph Embeddings," *ACM Transactions on Intelligent Systems and Technology*, vol. 13, no. 1, pp. 16:1–16:21, Feb. 2022. [Online]. Available: https://doi.org/10.1145/3495163

[30] J. Li, Y. Ren, and K. Deng, "FairGAN: GANs-based Fairness-aware Learning for Recommendations with Implicit Feedback," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. New York, NY, USA: Association for Computing Machinery, Apr. 2022, pp. 297–307. [Online]. Available: https://doi.org/10.1145/3485447.3511958

[31] L. Li, T. Riom, T. F. Bissyandé, H. Wang, J. Klein, and L. T. Yves, "Revisiting the impact of common libraries for android-related investigations," *Journal of Systems and Software*, vol. 154, pp. 157–175, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219301001

[32] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "Libd: Scalable and precise third-party library detection in android markets," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 335–346.

[33] S. G. MacDonell, M. J. Shepperd, B. A. Kitchenham, and E. Mendes, "How reliable are systematic reviews in empirical software engineering?" *IEEE Trans. Software Eng.*, vol. 36, no. 5, pp. 676–687, 2010. [Online]. Available: https://doi.org/10.1109/TSE.2010.28

[34] K. W. Nafi, M. Asaduzzaman, B. Roy, C. K. Roy, and K. A. Schneider, "Mining software information sites to recommend cross-language analogical libraries," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 913–924.

[35] M. Naghiaei, H. A. Rahmani, and Y. Deldjoo, "CPFair: Personalized Consumer and Producer Fairness Re-ranking for Recommender Systems," in *Proceedings of the 45th International*

ACM SIGIR Conference on Research and Development in Information Retrieval. Madrid Spain: ACM, Jul. 2022, pp. 770–779. [Online]. Available: https://dl.acm.org/doi/10.1145/3477495.3531959

[36] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, and M. Di Penta, “CrossRec: Supporting Software Developers by Recommending Third-party Libraries,” *Journal of Systems and Software*, p. 110460, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219302341

[37] P. T. Nguyen, R. Rubei, J. Di Rocco, C. Di Sipio, D. Di Ruscio, and M. Di Penta, “Artifacts: Dealing with Popularity Bias in Recommender Systems for Third-party Libraries: How far Are We?” Zenodo, 2023. [Online]. Available: https://github.com/MDEGroup/BiasInRSSE

[38] K. Niemann and M. Wolpers, “A new collaborative filtering approach for increasing the aggregate diversity of recommender systems,” in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’13. New York, NY, USA: Association for Computing Machinery, 2013, p. 955–963. [Online]. Available: https://doi.org/10.1145/2487575.2487656

[39] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. German, and K. Inoue, “Search-based software library recommendation using multi-objective optimization,” *Inf. Softw. Technol.*, vol. 83, no. C, pp. 55–75, Mar. 2017. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.11.007

[40] Y.-J. Park and A. Tuzhilin, “The long tail of recommender systems and how to leverage it,” in *Proceedings of the 2008 ACM Conference on Recommender Systems*, ser. RecSys ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 11–18. [Online]. Available: https://doi.org/10.1145/1454008.1454012

[41] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds., *Recommender Systems Handbook*. Springer, 2011. [Online]. Available: https://doi.org/10.1007/978-0-387-85820-3

[42] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, Eds., *Recommendation Systems in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, dOI: 10.1007/978-3-642-45135-5. [Online]. Available: http://link.springer.com/10.1007/978-3-642-45135-5

[43] R. Rubei, C. Di Sipio, J. Di Rocco, D. Di Ruscio, and P. T. Nguyen, “Endowing third-party libraries recommender systems with explicit user feedback mechanisms,” in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 817–821.

[44] M. A. Saied, A. Ouni, H. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, “Improving reusability of software libraries through usage pattern mining,” *Journal of Systems and Software*, vol. 145, pp. 164 – 179, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121218301699

[45] I. Salman, B. Turhan, and S. Vegas, “A controlled experiment on time pressure and confirmation bias in functional software testing,” *Empirical Software Engineering*, vol. 24, no. 4, pp. 1727–1761, Aug. 2019. [Online]. Available: https://doi.org/10.1007/s10664-018-9668-8

[46] R. L. Santos, C. Macdonald, and I. Ounis, “Exploiting query reformulations for web search result diversification,” in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW ’10. New York, NY, USA: Association for Computing Machinery, 2010, p. 881–890. [Online]. Available: https://doi.org/10.1145/1772690.1772780

[47] P. Spoletini and R. M. Parizi, “Bias-aware guidelines and fairness-preserving Taxonomy in software engineering education,” in *2018 IEEE Frontiers in Education Conference (FIE)*, Oct. 2018, pp. 1–4, iSSN: 2377-634X.

[48] Z. Sun, Y. Liu, Z. Cheng, C. Yang, and P. Che, “Req2lib: A semantic neural model for software library recommendation,” in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Los Alamitos, CA, USA: IEEE Computer Society, feb 2020, pp. 542–546. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SANER48275.2020.9054865

[49] F. Thung, D. Lo, and J. Lawall, “Automated library recommendation,” in *2013 20th Working Conference on Reverse Engineering (WCRE)*, Oct 2013, pp. 182–191.

[50] S. Vargas and P. Castells, “Rank and relevance in novelty and diversity metrics for recommender systems,” in *Proceedings of the Fifth ACM Conference on Recommender Systems*, ser. RecSys ’11. New York, NY, USA: Association for Computing Machinery, 2011, p. 109–116. [Online]. Available: https://doi.org/10.1145/2043932.2043955

[51] ——, “Improving sales diversity by recommending users to items,” in *Eighth ACM Conference on Recommender Systems, RecSys ’14, Foster City, Silicon Valley, CA, USA - October 06 - 10, 2014*, 2014, pp. 145–152. [Online]. Available: http://doi.acm.org/10.1145/2645710.2645744

[52] S. Verma and J. Rubin, “Fairness Definitions Explained,” in *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, May 2018, pp. 1–7.

[53] J. Wang, Y. Yang, S. Wang, J. Hu, and Q. Wang, “Context- and fairness-aware in-process crowdworker recommendation,” *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 3, mar 2022. [Online]. Available: https://doi.org/10.1145/3487571

[54] C. Wohlin, “Guidelines for snowballing in systematic literature studies and a replication in software engineering,” in *18th International Conference on Evaluation and Assessment in Software Engineering, EASE ’14, London, England, United Kingdom, May 13-14, 2014*, M. J. Shepperd, T. Hall, and I. Myrtveit, Eds. ACM, 2014, pp. 38:1–38:10. [Online]. Available: https://doi.org/10.1145/2601248.2601268

[55] Z. Zaier, R. Godin, and L. Faucher, “Evaluating recommender systems,” in *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*, 2008, pp. 211–217.

[56] A. Zerouali and T. Mens, “Analyzing the evolution of testing library usage in open source java projects,” in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 417–421.

[57] H. Zhang, M. A. Babar, and P. Tell, “Identifying relevant studies in software engineering,” *Information and Software Technology*, vol. 53, no. 6, pp. 625–637, 2011.