Identifying Experts in Software Libraries and Frameworks among GitHub Users

João Eduardo Montandon
Technical College (COLTEC)
Federal University of Minas Gerais
Belo Horizonte, Brazil
joao.montandon@dcc.ufmg.br

Luciana Lourdes Silva
Department of Computer Science
Federal Institute of Minas Gerais
Ouro Branco, Brazil
luciana.lourdes.silva@ifmg.edu.br

Marco Tulio Valente
Department of Computer Science
Federal University of Minas Gerais
Belo Horizonte, Brazil
mtov@dcc.ufmg.br

***Abstract*—Software development increasingly depends on libraries and frameworks to increase productivity and reduce time-to-market. Despite this fact, we still lack techniques to assess developers expertise in widely popular libraries and frameworks. In this paper, we evaluate the performance of unsupervised (based on clustering) and supervised machine learning classifiers (Random Forest and SVM) to identify experts in three popular JavaScript libraries: facebook/react, mongodb/node-mongodb, and socketio/socket.io. First, we collect 13 features about developers activity on GitHub projects, including commits on source code files that depend on these libraries. We also build a ground truth including the expertise of 575 developers on the studied libraries, as self-reported by them in a survey. Based on our findings, we document the challenges of using machine learning classifiers to predict expertise in software libraries, using features extracted from GitHub. Then, we propose a method to identify library experts based on clustering feature data from GitHub; by triangulating the results of this method with information available on Linkedin profiles, we show that it is able to recommend dozens of GitHub users with evidences of being experts in the studied JavaScript libraries. We also provide a public dataset with the expertise of 575 developers on the studied libraries.**

I. Introduction

Modern software development heavily depends on libraries and frameworks to increase productivity and reduce time-to-market [1], [2]. In this context, identifying experts in popular libraries and frameworks—for example, among the members of global open-source software development platforms, like GitHub—has a practical value. For example, open source project managers can use this information to search for potential new contributors to their systems. Private companies can also benefit from this information before hiring developers to their projects. In fact, we manually inspected 1,839 job offers, available on July 2nd, 2018 at Stack Overflow Jobs.[1] We found that 789 jobs (42%) have at least one tag referring to frameworks and libraries, including REACTJS (372 jobs), ANGULARJS (215 jobs), and RUBY ON RAILS (135 jobs). This result suggests that companies, when hiring, often target developers with expertise in specific programming technologies. Furthermore, this information can help to recommend experts to answer questions in Q&A forums [3] or to assist project managers to set up balanced development teams [4].

[1]https://stackoverflow.com/jobs

Previous work on software expertise focused on identifying experts for internal parts of a software project, but not on external components, such as libraries and frameworks. For example, Expertise Browser [5] visually maps parts of a software product (e.g., code or documentation) to the respective experts, using number of changes (commits) as the basic measure of expertise. Fritz et al. [6]–[8] propose the degree-of-knowledge (DOK) metric to identify experts in specific source-code files, which combines both commits and interactions with the code, by means of an IDE. Schuler and Zimmerman [9] advocate that expertise can also be gained by using the component of interest (e.g., by calling its methods). Silva-Junior et al. [10] propose a fine-grained approach to identify expertise in specific source-code elements—methods, classes, or packages. However, these works aim to identify experts that can fix a bug, review or evolve internal parts of an specific software product.

In this paper, we extend existing expertise identification approaches to the context of third-party software components. Our key hypothesis is that when maintaining a piece of code, developers also gain expertise on the frameworks and libraries used by its implementation. We focus on three popular libraries: FACEBOOK/REACT (for building enriched Web interfaces), MONGODB/NODE-MONGODB (for accessing MongoDB databases), and SOCKETIO/SOCKET.IO (for real-time communication). Then, we evaluate the use of unsupervised (based on clustering) and supervised machine learning classifiers to identify experts in these libraries. Both techniques are applied using features about candidate experts in each library, extracted for selected GitHub users. These features include, for example, number of commits on files that import each library and number of client projects a candidate expert has contributed to. We also survey a sample of GitHub users to create a ground truth of developers expertise in the studied libraries. In this survey, the participants declared their expertise (in a scale from 1 to 5) in the libraries. This ground truth provides the expertise of 575 GitHub developers in the studied libraries, including 418 FACEBOOK/REACT developers, 68 MONGODB/NODE-MONGODB developers, and 89 SOCKETIO/SOCKET.IO developers. To validate our hypothesis, we first train and evaluate two machine learning classifiers, based on Random Forest [11] and SVM [12]. Finally, we investigate the use of clustering algorithms to identify library experts.

Research Questions: We ask two research questions:

(RQ.1) How accurate are machine learning classifiers in identifying library experts? For three expertise classes—novices, intermediate, and experts—the maximal F-measure is 0.56 (MONGODB/NODE-MONGODB). We argue that this poor performance is inherent of using GitHub as a full proxy for expertise. For example, there are experts that rarely contribute to public GitHub projects; their expertise comes from working on private projects or projects that are not GitHub-based. low feature values (e.g., commits in library clients), making it challenging to predict the expertise of such developers, by considering their activity on GitHub.

(RQ.2) Which features best distinguish experts in the studied libraries? In this second *RQ*, we first rely on clustering to identify experts that share similar feature values. In FACEBOOK/REACT, we found a cluster where 74% of the developers are experts in the framework; in MONGODB/NODE-MONGODB and SOCKETIO/SOCKET.IO we found clusters with 65% and 75% of experts, respectively. More importantly, we show that the experts in such clusters tend to be active and frequent contributors to library clients on GitHub. Therefore, this finding suggests that GitHub data can be a partial proxy for expertise in libraries and frameworks. By partial proxy, we mean that developers with high feature values (commits, code churn, etc) tend to be experts in the studied libraries; by contrast, the proxy fails in the case of developers with low feature values, who can be both experts and novices, as concluded in *RQ.1*.

Contributions: Our contributions are threefold: (1) based on the findings and lessons learned with *RQ.1*, we document the challenges of using machine learning classifiers to predict expertise in software libraries, using features extracted from GitHub; (2) inspired by the findings of *RQ.2*, we propose an unsupervised method to identify library experts based on clustering feature data from GitHub; by triangulating the results of this method with expertise information available on Linkedin, we show that it is able to recommend dozens of GitHub users with robust evidences of being experts in FACEBOOK/REACT, a popular JavaScript library; (3) we provide a public ground truth with the expertise of 575 developers on three relevant JavaScript libraries; to our knowledge, this is the largest dataset with expertise data on specific software technologies.

Structure: Section II documents the process we followed to collect the data used to answer *RQ.1* and *RQ.2*. Section III describes the techniques used in this work, as well as their setup. Section IV provides answers to the proposed *RQs*. Section V summarizes our findings, lessons learned, and limitations. It also proposes a practical method for identifying library experts and validates its results with Linkedin data. Section VI reports threats to validity and Section VII describes related work. Finally, Section VIII concludes the paper.

II. Data Collection

A. Definitions

Before presenting the data collection process, we define key terms used in this process and also in the rest of this paper:

- *Target Library*: The JavaScript libraries used in this paper; our goal is to identify experts in these libraries based on their activity on GitHub.
- *Client Project (or File)*: A project (or source code file) that depends on a target library.
- *Candidate Expert:* A contributor of a client project whose expertise on a target library is assessed in this paper.
- *Feature:* An attribute of a candidate expert that may act as a predictor of its expertise on a target library.
- *Ground Truth:* A dataset with the expertise of candidate experts in a target library, as self-reported by them.

Table I
TARGET LIBRARIES

Target Library	Stars	Contrib	Commits	Files
FACEBOOK/REACT	91,739	1,171	9,731	797
MONGODB/NODE-MONGODB	6,696	260	4,565	617
SOCKETIO/SOCKET.IO	40,199	149	1,698	83

B. Target Libraries

We evaluate JavaScript libraries due to the importance and popularity of this language in modern software development. We focus on the developers of three JavaScript libraries[2]: FACEBOOK/REACT[3] (a system for building enriched Web interfaces), MONGODB/NODE-MONGODB[4] (the official Node.js driver for MongoDB database server), and SOCKETIO/SOCKET.IO[5] (a library for real-time communication). We selected FACEBOOK/REACT because it is a very popular front-end development library; after making this first selection, we searched for libraries handling important concerns in back-end development and selected MONGODB/NODE-MONGODB, a persistence library; and SOCKETIO/SOCKET.IO, since communication is important both in front-end and back-end programming. Table I shows information about these systems, including number of stars, contributors, commits, and files (on April, 2018). As we can see, they are popular projects (at least 6,696 stars) and actively maintained (at least 149 contributors and 1,698 commits). For brevity, we call them REACT, NODE-MONGODB, and SOCKET.IO in the rest of this paper.

C. Candidate Experts

For each target library $\mathcal{L}$, where $\mathcal{L}$ is REACT, NODE-MONGODB, or SOCKET.IO, we selected a list of candidate experts, as described next. First, we relied on the top-10K most popular JavaScript projects on GitHub, according to their number of stars. We checked out these projects and searched for dependencies to $\mathcal{L}$ in *package.json* and *bower.json* files, which are configuration files used by two popular JavaScript package managers. A candidate expert in $\mathcal{L}$ is a developer who performed at least one change in a source code file (from a client project) that depends on $\mathcal{L}$. In other words, we

[2]In our study, the terms libraries and frameworks are used interchangeably.
[3]https://github.com/facebook/react
[4]https://github.com/mongodb/node-mongodb-native
[5]https://github.com/socketio/socket.io

Table III
FEATURES COLLECTED FOR EACH CANDIDATE EXPERT IN EACH TARGET LIBRARY

Dimension	Feature	Description
Volume	commits	Number of commits in client projects
	commitsClientFiles	Number of commits changing at least one client file
	commitsImportLibrary	Number of commits adding library import statements
	codeChurn	Code churn considering all commits in client projects
	codeChurnClientFiles	Code churn considering only changes in client files
	imports	Number of added library import statements
Frequency	daysSinceFirstImport	Number of days since the first commit where a library import statement was added
	daysSinceLastImport	Number of days since the last commit where a library import statement was added
	daysBetweenImports	Number of days between the first/last commits where a library import statement was added
	avgDaysCommitsClientFiles	Average interval (in days) of the commits changing client files
	avgDaysCommitsImportLibrary	Average interval (in days) of the commits adding library import statements
Breadth	projects	Number of client projects the developer contributed at least once
	projectsImport	Number of client projects where the developer added a library import statement

assume that if a developer changed a file that imports $\mathcal{L}$ he has chances to be an expert in this library. Next, we removed aliases from this initial list of candidate experts, i.e., the same developer, but with distinct e-mails on the considered commits. For this purpose, we used a feature of GitHub API that maps a commit author to its GitHub account. Using this feature, we mapped each developer in the list of candidate experts to his/her GitHub's account. Candidate experts e and e' are the same when they share the same GitHub account. Table II shows for each target library the number of client projects, and the final number of candidate experts after handling aliases. As we can observe, REACT has the highest number of both client projects (1,136) and candidate experts (8,742). Therefore, our dataset includes a popular target library, with thousands of client projects and candidate experts; but it also includes less popular libraries, with just a few hundred candidate experts.

Table II
CLIENT PROJECTS AND CANDIDATE EXPERTS

Library	Clients	Experts
FACEBOOK/REACT	1,136	8,742
MONGODB/NODE-MONGODB	223	454
SOCKETIO/SOCKET.IO	345	608

D. Features

We collected 13 features for each candidate expert selected in the previous step. As documented in Table III, these features cover three dimensions of *changes* performed on client files.[6]

- *Volume of changes*, which includes six features about the quantity of changes performed by candidate experts in client projects, such as number of commits and code churn (e.g., lines added or deleted). We conjecture that by heavily maintaining a file developers gain expertise on libraries used by its implementation.
- *Frequency of changes*, including five features expressing the frequency and time of the changes performed by candidate experts, e.g., number of days since first and last library import. The rationale is that expertise also depends on temporal properties of the changes.
- *Breadth of changes*, which includes two features about the number of client projects the candidate experts worked on. The rationale is that expertise might increase when candidate experts work in different client projects.

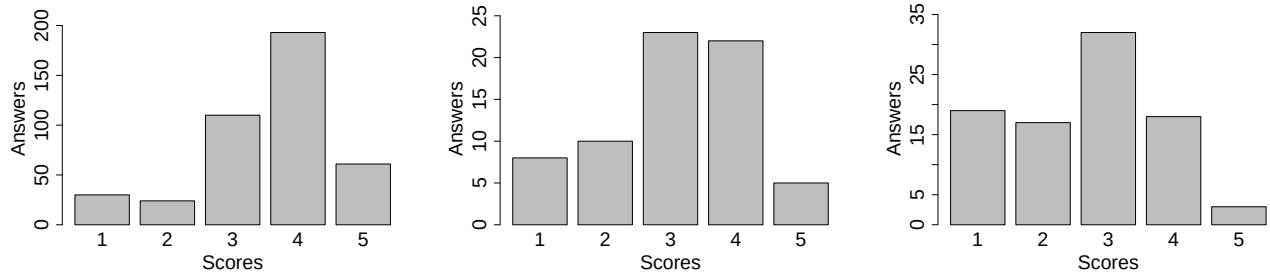
The features are collected from client projects where the candidate experts contributed with at least one commit. In more detailed terms, suppose a candidate expert c; suppose also that $Proj_c$ are the projects where c has made at least one commit (this set is provided by GitHub API). We iterate over $Proj_c$ to create a subset $CliProj_c$ containing only projects that depend on the target libraries. The features collected for c are extracted from $CliProj_c$. After collecting this data, we found that 69% of REACT's candidate experts worked on a single client project; for NODE-MONGODB and SOCKET.IO, this percentage increases to 88% and 87%, respectively. By contrast, we found candidate experts working on 26 projects (REACT), 5 projects (NODE-MONGODB) and 12 projects (SOCKET.IO).

E. Ground Truth

To create a ground truth with developers expertise on each target library, we conducted a survey with the candidate experts identified in Section II-C. For REACT, which has 8,742 candidate experts, we sent the survey to a random sample of 2,185 developers (25%). For NODE-MONGODB and SOCKET.IO, which have less candidates, we sent the survey to *all* candidate experts identified in Section II-C, i.e., to 454 and 608 developers, respectively. For each target library, we e-mailed the candidate experts, describing our research purpose and asking the following single question:

Could you please rank your expertise on [target library] in a scale from 1 (novice) to 5 (expert)?

[6]These dimensions and their features were derived and extended from the literature on developers expertise in open source communities. Volume of changes (particularly, number of commits) is commonly used in related works [5]–[8]. Frequency and breadth of changes have also been considered as proxies to developers expertise [10], [13]–[17]. As an additional criterion, we only use features that can be directly computed from GitHub public API.



(a) FACEBOOK/REACT (b) MONGODB/MONGO (c) SOCKETIO/SOCKET.IO

Figure 1. Survey answers

Table IV
SURVEY NUMBERS

Library	Mails	Answers	Ratio
FACEBOOK/REACT	2,185	418	19%
MONGODB/NODE-MONGODB	454	68	15%
SOCKETIO/SOCKET.IO	608	89	15%

Table IV summarizes the number of e-mails sent, the number of received answers, and the response ratio. The number of answers range from 68 (NODE-MONGODB) to 418 (REACT) and the response ratio ranges from 15% (SOCKET.IO and NODE-MONGODB) to 19% (REACT).

Figure 1 shows the distribution of the survey answers. For REACT, 254 candidates (61%) ranked themselves as experts in the library (scores 4–5); 110 candidates (26%) declared an intermediate expertise (score 3), and 54 candidates (13%) considered themselves as having a limited expertise (scores 1–2). For NODE-MONGODB, the results are 40% (experts), 34% (intermediate expertise), and 26% (limited expertise). For SOCKET.IO, the results are 24%, 36%, and 40%, respectively.

Ground Truth Limitations: The proposed ground truth is based on the developers' perceptions about their expertise in the target libraries. Therefore, it is subjected to imprecisions and noise, since it is not realistic to assume the survey participants ranked themselves according to uniform and objective criteria. For example, some developers might have been more rigorous in judging their expertise, while others may have omitted their lack of experience on the studied libraries (see the Dunning-Kruger Effect [18]). In order to try to reduce these issues, we made it clear to the participants that our interests were strictly academic and that we will never use their answers to commercial purposes. Finally, it is also worth mentioning that previous research has shown that self estimation is a reliable way to measure general programming experience, at least in a student population [19].

F. Final Processing Steps

We performed the following processing steps on the features collected for the developers that answered our survey.

Missing Values: Missing values occur when it is not possible to compute a feature value. In our dataset, there are four features with missing values: *daysSinceFirstImport*, *daysSinceLastImport*, *daysBetweenImports*, and *avgDaysCommitsImportLibrary*. For these features, a missing value appears in candidate experts who have added an insufficient number of import statements to a client project (e.g., $imports = 0$). The percentage of candidate experts with missing values for these four features is relevant, as they appear in 45% of the surveyed developers. To handle such cases, we replaced missing values at *daysSinceFirstImport* and *daysSinceLastImport* by a zero value, because candidate experts without import statements should not be viewed as long time library users. By contrast, missing values at *avgDaysCommitsImportLibrary* were replaced by the maximal observed value, because the respective candidate experts should have the highest values when compared to those who effectively added import statements. Finally, *daysBetweenImports* needs at least two imports to be calculated correctly. Therefore, we assigned a zero value when $imports = 1$, and -1 when $imports = 0$.[7]

Removing Correlated Features: Correlated features may contribute to inaccurate classifications due to their high association degree [20], [21]. To tackle this issue, we first used the *cor*[8] function from R's *stats* package to compute a matrix with Pearson coefficients for each pair of features. Then, we used the *findCorrelation*[9] function from R's *caret* package to identify pairs of features with a correlation greater than 0.7, as previously adopted in the literature [22]; in such cases, we measured the overall correlation of both features and discarded the highest one. Figure 2 shows a heatmap that summarizes this process. Red cells are features discarded due to a high correlation with another feature; gray cells denote features preserved by the correlation analysis, i.e., they are used in the classification process. As we can see, two features are correlated with at least one other feature, regardless the target library: *commitsImportLibrary* and *projectsImport*. As a result of this analysis, six, four, and five features were discarded at REACT, NODE-MONGODB, and SOCKET.IO, respectively.

[7] In fact, we tested different strategies for missing values, such as discarding all fields with missing values, applying different values, etc. However, the results never exceeded the ones based on the values proposed in this paragraph.

[8] https://www.rdocumentation.org/packages/stats/versions/3.4.3/topics/cor

[9] https://www.rdocumentation.org/packages/caret/versions/6.0-79/topics/findCorrelation

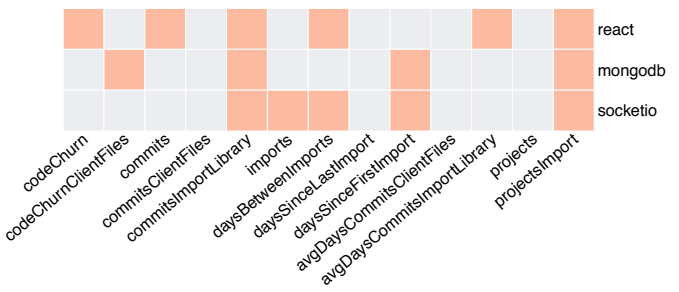


Figure 2. Correlation analysis; red cells are discarded due to high correlation.

Skewed Feature Values: Features with skewed distributions may impact the performance of machine learning classifiers [23], [24]. We assume that skewed feature distributions are the ones where the mean—computed for the candidate experts included in the ground truth of a given target library—is at least four times greater than the median. By following this definition, four, six, and four features have a skewed behavior in REACT, NODE-MONGODB, and SOCKET.IO, respectively. On the values of such features, we applied a log transformation, as in another machine learning study [25].

III. METHODS

In this section, we discuss the setup of the machine learning and clustering models, used on *RQ.1* and *RQ.2,* respectively.

A. Machine Learning Setup and Algorithms

Number of Classes: Machine learning algorithms require a minimal number of samples on each class (or scores, in our terminology) [26]. However, this condition is not followed by our data. For example, for REACT we collected expertise data about 418 developers, but only 24 developers (6%) ranked themselves with score 2. To attenuate this problem, we train and evaluate our models under two scenarios: (1) considering all five classes; (2) by transforming the data into the following ternary classification: *novice* (scores 1–2), intermediate (score 3), and *experts* (scores 4–5). Furthermore, we only evaluate the scenario with five classes for REACT. The reason is because NODE-MONGODB and SOCKET.IO have fewer data points; for example, both libraries have classes with less than 10 samples.

Informed Over Sampling (SMOTE): Besides having few samples for some classes, the ground truth is largely imbalanced, as illustrated in Figure 1. For example, 87% of the REACT developers ranked themselves as having some knowledge on the framework (scores 3-5). It is well-known that machine learning classifiers tend to produce poor results when applied to imbalanced datasets [27]. To tackle this problem, we used a technique called Informed Over Sampling (SMOTE) [28], which balances a dataset by producing and inserting synthetic but similar observations to minority classes (but only in the training part of the dataset). SMOTE was previously used in machine learning approaches to several software engineering problems, including defect prediction [29], mobile apps analysis [30], self-admitted technical debt detection [31], and identification of security issues from commit messages and bug reports [32]. In our problem, we used SMOTE over the minority class, on both scenarios. SMOTE has two parameters: number of the nearest neighbours (KNN) and the percentage of synthetic instances to create. After some initial tests, we set up these parameters to 3 and 30%, respectively. This setup results in a minority class increased by 30%; and the new data points are synthesized by considering 3-nearest neighbours of the existing ones (KNN parameter).

Machine Learning Classifiers: We evaluate two well-known machine learning classifiers: Random Forest [11] and SVM [12]. We compare the results of these classifiers with a ZeroR baseline, which simply predicts the majority class, ignoring all feature values. We do not compare with previous expertise identification approaches (e.g., [5]–[9]) because they are not proposed to measure expertise on libraries and frameworks, but on internal elements of a software project. We use k-fold stratified cross-validation to evaluate the results of these classifiers. Stratified cross-validation is a variant of k-fold cross-validation where folds contain approximately the same proportion of each class. We set k to 5, to avoid testing models in small folds, particularly in small classes, as occur in NODE-MONGODB and SOCKET.IO. Another important step is the tuning of the classifiers parameters. We rely on a grid search strategy for hyper-parameters with cross validation to find the best parameters settings for each classifier [33].

Evaluation Metrics: We evaluate the classifiers using precision, recall, F-measure, and AUC (Area Under the Receiver Operating Characteristic Curve). To compute AUC, we use an implementation recommended for multi-class classifications. This implementation is provided as an R package by Microsoft Azure's data science team.[10] Further, to compute F-measure, we first compute the average precision and recall, considering all classes. The reported F-measure is the harmonic mean of the average precision and average recall. We also report Cohen's kappa, which is also a measure of classifier performance, particularly useful on imbalanced datasets [34].

B. Clustering Setup and Algorithm

We use clustering to investigate more closely the relation of feature values and library expertise (*RQ.2*). To this purpose, we use k-means, which is a widely popular clustering algorithm. In software engineering, k-means was used to support many tasks, including detecting mobile apps with anomalous behavior [35], test case prioritization [36], and to characterize build failures [37]. A key challenge when using k-means is to define the appropriate number k of clusters. There are methods proposed to help on this task, such as the elbow [38] and silhouette methods [39]. However, they also depend on interpretation and subjective decisions [38]. For this reason, we follow an alternative procedure, as described next. We execute k-means multiple times, starting with $k = 2$ and incrementing it after each execution. For each k, we analyze the resulting clusters, searching for clusters dominated by experts. For

[10]https://github.com/Azure/Azure-MachineLearning-DataScience

Table VII
RESULTS FOR 3 CLASSES: NOVICE (SCORES 1-2), INTERMEDIATE (SCORE 3), AND EXPERT (SCORES 4-5)

	FACEBOOK/REACT			MONGODB/NODE-MONGODB			SOCKETIO/SOCKET.IO		
	RForest	**SVM**	**Baseline**	**RForest**	**SVM**	**Baseline**	**RForest**	**SVM**	**Baseline**
Kappa	0.09	0.03	0.00	0.35	0.25	0.00	0.16	0.25	0.00
AUC	0.56	0.51	0.50	0.70	0.56	0.50	0.60	0.71	0.50
Precision (Novice)	0.14	0.60	0.00	0.50	0.47	0.00	0.52	0.54	0.40
Precision (Intermediate)	0.34	0.00	0.00	0.62	0.17	0.00	0.29	0.59	0.00
Precision (Expert)	0.65	0.61	0.61	0.55	0.57	0.40	0.43	0.48	0.00
Recall (Novice)	0.09	0.06	0.00	0.50	0.68	0.00	0.61	0.78	1.00
Recall (Intermediate)	0.18	0.00	0.00	0.57	0.09	0.00	0.19	0.19	0.00
Recall (Expert)	0.83	1.00	1.00	0.63	0.75	1.00	0.56	0.56	0.00
F-measure	0.36	0.29	0.25	0.56	0.44	0.19	0.42	0.46	0.19

REACT, we search for clusters with at least 70% of experts (since REACT has a higher percentage of experts in the ground truth, close to 61%); for NODE-MONGODB and SOCKET.IO—which have less experts, 40% and 24%, respectively—we search for clusters with at least 60% of experts. We stop after finding at least one cluster attending the proposed thresholds. Table V shows data on each execution; for each k, it shows the percentage of experts of the cluster with the highest percentage of experts. For REACT, we select 3 clusters, since it leads to a cluster with 74% of experts. For NODE-MONGODB, we also select 3 clusters, including a cluster with 65% of experts. For SOCKET.IO, there are 5 clusters and one has 75% of experts.

Table V
CLUSTER WITH THE HIGHEST PERCENTAGE OF EXPERTS (VALUES IN BOLD DEFINE THE SELECTED NUMBER OF CLUSTERS)

Library	k			
	2	3	4	5
REACT	66	**74**	-	-
NODE-MONGODB	57	**65**	-	-
SOCKET.IO	39	44	44	**75**

IV. RESULTS

(RQ.1) How accurate are machine learning classifiers when used to identify library experts?

Table VI presents the results of the machine learning classifiers for five classes. The results are provided only for REACT, since NODE-MONGODB and SOCKET.IO do not have sufficient samples to perform a classification using five classes, as explained in Section III-A. For almost all performance metrics and classifiers, the results are not good. For example, kappa is 0.09 and AUC is 0.56 for Random Forest. Precision ranges from 0.00 (Novice 2, SVM) to 0.50 (Expert 4, Random Forest). F-measure is 0.24 (Random Forest) and 0.15 (SVM), against 0.13 with the ZeroR baseline.

Table VII presents the results for three classes (scores 1-2, score 3, scores 4-5). First, we discuss the results of Random Forest. For this classifier, kappa varies from 0.09 (REACT) to 0.35 (NODE-MONGODB); AUC ranges from 0.56 (REACT) to 0.70 (NODE-MONGODB). Precision results are greater for experts than for novices, both for REACT (0.65 vs 0.14) and NODE-MONGODB (0.61 vs 0.60), while SOCKET.IO has the highest precision for novices (0.52). Recall ranges from 0.09 (REACT, novices) to 0.83 (REACT, experts). F-measure is 0.36 (REACT), 0.56 (NODE-MONGODB), and 0.42 (SOCKET.IO). By contrast, the baseline results for F-measure are 0.25 (REACT) and 0.19 (NODE-MONGODB and SOCKET.IO). In the same scenario, SVM results are in 13 out of 27 combinations of metrics and libraries lower than the ones of Random Forest; they are also just slightly greater than ZeroR.

Table VI
MACHINE LEARNING RESULTS FOR 5 CLASSES (FACEBOOK/REACT)

	RForest	**SVM**	**Baseline**
Kappa	0.09	0.05	0.00
AUC	0.52	0.53	0.50
Precision (Novice 1)	0.25	0.00	0.00
Precision (Novice 2)	0.07	0.00	0.00
Precision (Intermediate)	0.35	0.23	0.00
Precision (Expert 4)	0.50	0.48	0.46
Precision (Expert 5)	0.29	0.00	0.00
Recall (Novice 1)	0.07	0.00	0.00
Recall (Novice 2)	0.04	0.00	0.00
Recall (Intermediate)	0.27	0.10	0.00
Recall (Expert 4)	0.77	0.98	1.00
Recall (Expert 5)	0.10	0.00	0.00
F-measure	0.24	0.15	0.13

> For five classes, machine learning classifiers have a maximal F-measure of 0.24 (REACT). For three classes, F-measure reaches 0.56 (NODE-MONGODB) and precision on identifying experts reaches 0.65 (REACT, experts).

(RQ.2) Which features best distinguish library experts?

First, Table VIII shows the percentage of novices (scores 1-2), intermediate (score 3), and experts (scores 4-5) in the clusters of each library. The table also shows the number of developers in each cluster. As defined in Section III-B, for REACT and NODE-MONGODB, we have 3 clusters; for SOCKET.IO, we have 5 clusters. In Table VIII, the clusters are sorted by percentage of experts. Therefore, Cluster 1 is the experts' cluster in each library. In REACT, 74% of the developers in this cluster ranked themselves as experts and only 3% as novices. For NODE-MONGODB and SOCKET.IO,

Table VIII
CLUSTERING RESULTS (CLUSTER 1 HAS THE HIGHEST % OF EXPERTS)

Cluster	% Novices	% Intermediate	% Experts	# Devs
FACEBOOK/REACT				
C1	0.03	0.23	0.74	97
C2	0.12	0.28	0.60	129
C3	0.18	0.27	0.55	192
MONGODB/NODE-MONGODB				
C1	0.12	0.24	0.65	17
C2	0.21	0.43	0.36	14
C3	0.35	0.35	0.30	37
SOCKETIO/SOCKET.IO				
C1	0.00	0.25	0.75	4
C2	0.29	0.36	0.36	28
C3	0.33	0.33	0.33	15
C4	0.50	0.40	0.10	30
C5	0.67	0.33	0.00	12

Cluster 1 includes 65% and 75% of experts, respectively. By contrast, it has only 12% and 0% of novices, respectively. The number of developers in the experts' cluster ranges from 4 (SOCKET.IO) to 97 developers (REACT). However, the ground truth has also more REACT experts (254 vs 21 developers, respectively). Interestingly, in SOCKET.IO, Cluster 5 should be viewed as a novice's clusters; 67% of its members are novices and the cluster does not include any expert.

> In the three studied libraries, there are clusters dominated by experts. These clusters have 74% (REACT), 65% (NODE-MONGODB), and 75% (SOCKET.IO) of experts.

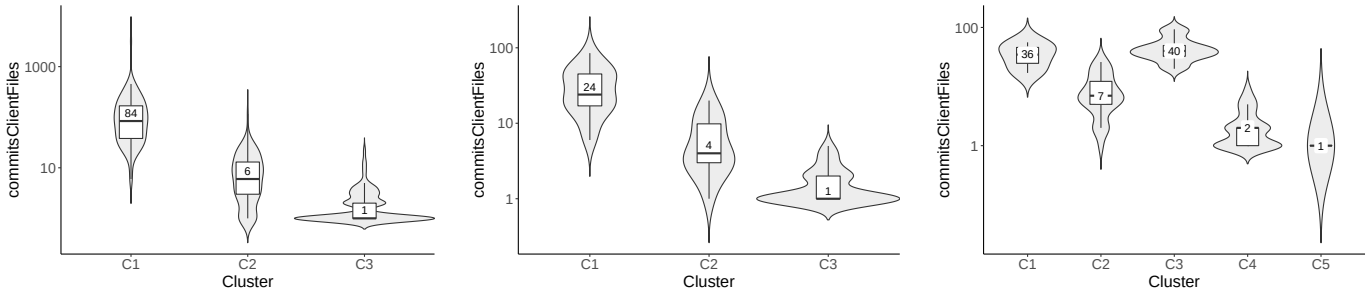
We also compare the distributions of feature values, for the developers in each cluster. For each feature F, we compare F's distribution in Cluster 1 (experts) with the cluster whose median of F's distribution is closest to the one of Cluster 1. In other words, this cluster tends to be the most similar to Cluster 1, among the remaining clusters; our goal is to assess the magnitude (effect size) and direction of this similarity. First, we use a Mann-Whitney test to confirm that the distributions of F's values in both clusters are statistically distinct, assuming a p-value of 0.05. Furthermore, and more interestingly, we measure the magnitude and direction of the difference, using Cliff's delta. As in other works [40]–[43], we interpret Cliff's delta as negligible for $d < 0.147$, small for $0.147 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$.

Table IX shows the results. For REACT, there is a *large* difference for the distributions of all features in Cluster 1, with the exception of *daysSinceFirstImport*, which has a *medium* effect size. The direction is mostly positive (+), i.e., developers in Cluster 1 have higher feature values than the ones in the second most similar cluster (in summary, they are more active on client files). The exception regards the distributions of *avgDaysCommitsClientFiles*, i.e., experts tend to commit more frequently to REACT client files—in lower time intervals—than developers of the second cluster. In general, the results for NODE-MONGODB follow the same patterns observed for REACT; the main exception is that a *medium* difference is observed for *daysSinceLastImport*. However, in the case of SOCKETIO/SOCKET.IO there is a major change in the statistical tests. First, Cliff's delta reports a *large* difference for a single feature: number of projects the developers have committed to (*projects*). According to Mann-Whitney tests, the remaining feature distributions are statistically indistinct. To visually illustrate these results, Figure 3 shows violin plots with the distribution on each cluster of *commitsClientFiles*, for the three studied libraries. We can see a *large* difference between the distributions of Cluster 1 and Cluster 2, both for REACT and NODE-MONGODB. By contrast, for SOCKET.IO, there is no clear difference between the distributions of Cluster 1 and Cluster 3 (cluster with the median closest to Cluster 1). Finally, Figure 4 shows boxplots with *projects* distribution for SOCKET.IO. In this case, we can see a clear difference between Cluster 1 (1st quartile is 8 projects; median is 8.5 projects) and Cluster 3 (1st quartile is one project; median is two projects).

Table IX
COMPARING FEATURE DISTRIBUTIONS USING CLIFF'S DELTA: EXPERTS VS CLUSTER WITH THE CLOSEST MEDIAN (○ MEANS SIMILAR DISTRIBUTIONS, ACCORDING TO MANN-WHITNEY, p-VALUE= 0.05)

Feature	Effect size	Relationship
FACEBOOK/REACT		
codeChurnClientFiles	large	+
commitsClientFiles	large	+
imports	large	+
daysSinceLastImport	large	+
daysSinceFirstImport	medium	+
avgDaysCommitsClientFiles	large	−
projects	large	+
MONGODB/NODE-MONGODB		
codeChurn	large	+
commits	large	+
commitsClientFiles	large	+
imports	large	+
daysBetweenImports	large	+
daysSinceLastImport	medium	+
avgDaysCommitsClientFiles	large	−
avgDaysCommitsImportLibrary	large	−
projects	large	+
SOCKETIO/SOCKET.IO		
codeChurn	○	○
codeChurnClientFiles	○	○
commits	○	○
commitsClientFiles	○	○
daysSinceLastImport	○	○
avgDaysCommitsClientFiles	○	○
avgDaysCommitsImportLibrary	○	○
projects	large	+

> For REACT and NODE-MONGODB, developers in the experts cluster are more active on GitHub than developers in other clusters, regarding most features. However, for SOCKET.IO, experts are only distinguished by the number of projects they worked on.



(a) FACEBOOK/REACT (b) MONGODB/NODE-MONGODB (c) SOCKETIO/SOCKET.IO

Figure 3. Distributions of *commitsClientFiles* values for each cluster/library. Cluster 1 (experts) has higher values than other clusters, except for SOCKET.IO.

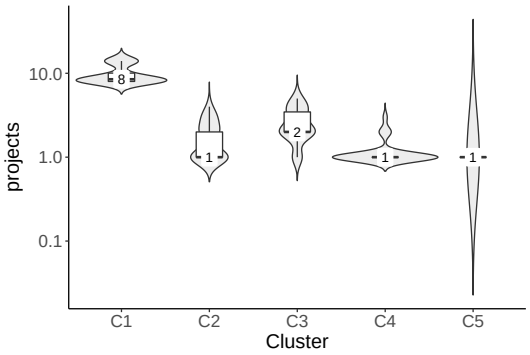


Figure 4. Distributions of *projects* values for SOCKET.IO clusters. Cluster 1 (experts) has higher values than other clusters.

To conclude, it is important to mention that the feature values are different for experts in each library. For example, experts in FACEBOOK/REACT (Cluster 1) perform 84 commits at client files, against 24 commits for NODE-MONGODB's experts (median values, see Figure 3). Our hypothesis is that REACT is a more complex framework than NODE-MONGODB, besides targeting a different domain. As a result, it is no trivial to define feature thresholds to classify experts; furthermore, these thresholds should not be reused across libraries.

V. Discussion and Practical Usage

A. Relevance and Key Findings

In the survey to create the ground truth, we only asked for a score (in a 5-point scale). Despite that, we received some comments about the relevance of approaches to predict developers expertise in specific programming technologies, as in the following answers:

What you are doing sounds very interesting and worthwhile to the developer's community at large. (P021)

Technical recruiting seems to be an extremely valid use-case for accurately assess the skills of devs based on their GitHub contributions, which could lead to a profitable product. (P183)

We associate the high number of responses received in the survey (575 answers) to the relevance and potential practical value of the problem we proposed to investigate, which was rapidly viewed in this way by the surveyed GitHub users.

As mentioned in one of the previous answers, the main interest of companies is on accurately identifying experts in a given programming technology. In this particular context, precision is more important than recall, since companies do not need to identify all skilled engineers in a given technology, but only a few of them. When approaching the problem using machine learning classifiers, we achieved a maximal precision of 65% for the experts class (scores 4-5, Random Forest, REACT). In the same scenario, the baseline precision is 0.61. Therefore, this result casts doubts on the practical value of using machine learning in this problem. By contrast, when using unsupervised techniques, based on clustering (k-means), we were able to identify clusters with 74% (REACT), 65% (NODE-MONGODB), and 75% (SOCKET.IO) of experts. If we consider that predicting expertise on programming technologies is a relevant but challenging problem, we claim that precision values close to 70%—across multiple libraries—can sustain the practical adoption of automatic classifiers based on features extracted from GitHub activity. Even so, unsupervised techniques should be carefully used, as their gains may vary according to the library (see REACT clusters). It is also worth mentioning that such classifiers do not replace but complement traditional mechanisms for assessing developers expertise, like interviews and curriculum analysis.

B. Practical Usage

Suppose a library $\mathcal{L}$ with developers grouped in clusters $\mathcal{C}_1, \ldots, \mathcal{C}_n$, after following the methodology proposed in this paper. Suppose that $\mathcal{C}_1$ groups the experts in $\mathcal{L}$. Given these clusters, suppose we want to assess the expertise of a new developer d on $\mathcal{L}$, e.g., we are part of a company that heavily depends on $\mathcal{L}$ and we want to assess the expertise of d in this library, before hiring her. In this case, we should retrieve the feature vector $\mathcal{F}_d$ for d, based on her activities on GitHub. Then, we compute the Euclidean distance between $\mathcal{F}_d$ and the centroid of each cluster $\mathcal{C}_i$, for $i = 1, \ldots, n$. If the smallest distance is found between $\mathcal{F}_d$ and C_1's centroid, we can assume that d is more similar to the experts in $\mathcal{L}$ and therefore she has high chances of also being an expert in this library. Otherwise, our method fails to predict d's expertise in $\mathcal{L}$, i.e., she can be or not an expert. It is also straightforward to identify expertise in multiple libraries. In this case, we only need to compute the intersection of experts in each library.

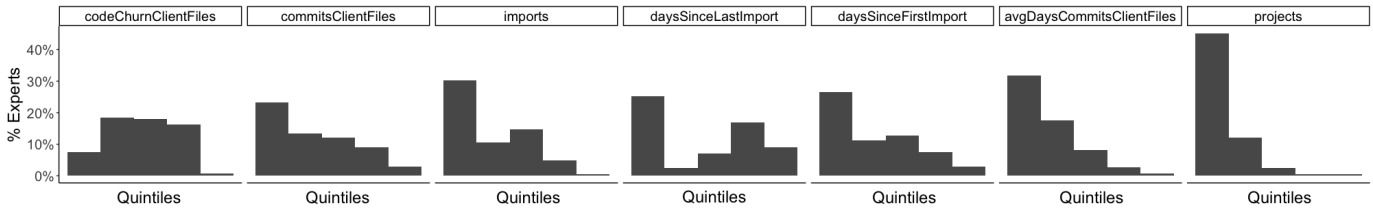


Figure 6. Percentage of REACT experts by quintiles of feature distributions. For most features, there is an important proportion of experts in lower quintiles.

C. Triangulation with Linkedin Profiles

To provide preliminary evidence on the value of the procedure described in the previous section to identify experts, we triangulated its results with expertise information available on Linkedin, starting with REACT experts. First, we mapped each REACT developer who did not answer our survey—and therefore was not considered at all in *RQ.1* and *RQ.2*—to one of the clusters produced for REACT, as discussed before. 263 (out of 2,129 developers, 12%) were mapped to the experts cluster. After that, the first author of this paper manually searched for the Linkedin page of these developers, looking for their names and possibly e-mails on Linkedin (when available, he also compared the profile photos, at Linkedin and GitHub). He was able to find the Linkedin profile of 160 developers (61%). Finally, he manually examined these profiles, searching for evidences of expertise on REACT. 115 developers (72%) explicitly refer to REACT on their Linkedin short bios, on the description of the projects they worked on, or in the list of programming technologies they have skills on. The first paper's author also assessed the experience of these developers as Web developers, by calculating the number of years on jobs directly related to Web programming. Figure 5 shows a violin plot with the results. As we can see, 50% of the developers predicted as experts have more than four years of experience on Web-related jobs.

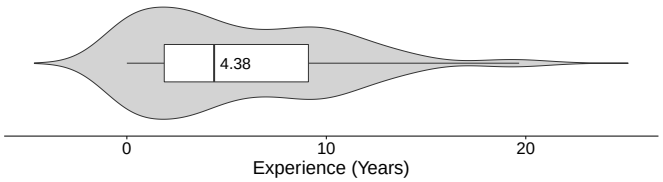


Figure 5. Years of experience on REACT of developers predicted as experts

We reproduced this analysis with NODE-MONGODB and SOCKET.IO. For NODE-MONGODB, 44 out of 58 developers predicted as experts by the proposed method have pages on Linkedin; for SOCKET.IO, this happens with 5 out of 10 experts. Furthermore, 28 of such experts (64%) explicitly mention MONGODB on their Linkedin pages; and one developer (20%) refer to SOCKET.IO. Therefore, both proportions are lower than the one we reported for REACT. We claim this happens because NODE-MONGODB and SOCKET.IO are simple and less complex libraries, when compared with REACT. For this reason, developers usually do not cite them on Linkedin. For example, one of the experts in SOCKET.IO declare on his GitHub profile that he is one of the library's core developers; but this information is not available on his Linkedin profile. Due to this reason, we also do not evaluate the years of experience of Linkedin users on SOCKET.IO and NODE-MONGODB.

Altogether, this triangulation with Linkedin shows that the proposed clustering-based method was able in most cases to find several GitHub developers with evidences of having experience on the studied libraries. However, before concluding, it is also important to acknowledge that expertise and experience are distinct concepts; indeed, experience is normally viewed as a necessary condition to achieve expertise [44], [45].

D. Limitations

Certainly, developers can gain expertise on libraries and frameworks by working on private projects or in projects that are not on GitHub, as highlighted by these developers:

None of my projects are publicly on GitHub. (P037, score 4)

My work on GitHub isn't my strongest. My much larger projects are at work and aren't open source. (P503, score 4)

Thus, the lack of public activity on GitHub is a major obstacle for achieving high recall using approaches like the one proposed in this paper. However, as mentioned before, precision tends to be more important in practical settings than recall. If we focus on precision, the proposed clustering approach is effective on identifying experts among GitHub users that frequently contribute to client projects.

To illustrate this discussion, Figure 6 shows histograms with the percentage of REACT experts in each quintile of the feature distributions (0%-19%, 20%-39%, etc). We can observe an important concentration of experts in the first and second quintiles, for features like *codeChurnClientFiles* (26%), *commitsClientFiles* (37%), and *projects* (57%). In other words, the histograms confirm the comments of the survey participants, showing that it is common to have experts with sparse activity on GitHub. Indeed, this behavior explains the poor performance of machine learning supervised classifiers in our context, as observed in *RQ.1*. By construction, these classifiers predict the expertise of all developers in the ground truth. Therefore, the presence of experts at both ends of the distributions showed in Figure 6 is a major challenge to their performance. Typically, these classifiers are not able to provide an *unknown* answer, as we discussed in Section V-B.

VI. Threats to Validity

Target Libraries: We mined experts in three popular JavaScript libraries. Thus, it is not possible to fully generalize our findings to experts of other libraries and frameworks.

Candidate Experts: Our list of candidate experts was extracted from an initial list with the top-10K most starred GitHub projects (see Section II-C). We acknowledge that our results might be impacted if we expand or reduce this initial list.

Alias Handling: The method used for detecting aliases in the initial list of candidate experts (see Section II-C) do not distinguish developers that have multiple GitHub accounts, i.e., they are considered distinct developers. Therefore, further analysis is required to quantify the incidence of such accounts.

Ground Truth: Another threat is related to mislabeled classes, due to personal opinions of the surveyed developers, as discussed in Section II-E. However, we surveyed 575 developers and some level of mislabeling would not interfere in our results, since the selected algorithms are robust to label noises. Furthermore, to tackle the imbalanced behavior of our ground truth, we used a technique called SMOTE, commonly used on several software engineering problems [29]–[32]. But we acknowledge that there are other techniques , such as oversampling and cost-sensitive methods [46], [47].

VII. Related Work

CVExplorer [48] is a tool to extract and visualize developers' skills data from GitHub, including skills on programming languages, libraries, and frameworks. The extracted data is presented in the form of a "tag cloud" interface, where the tags denote programming technologies (e.g., web development), libraries and frameworks (e.g., React) or programming languages (e.g., JavaScript). Tags are mined from the project's READMEs and from commit messages. CPDScorer [49] is another tool that scores developers' skills, but by correlating developers' activity on Stack Overflow and GitHub. The tool assumes that developers with high quality Stack Overflow answers (measured by number of upvotes) are more likely to be experts in specific programming technologies; the same is assumed for developers who contributed to high quality projects, as measured using source code metrics. Constantinou and Kapitsaki [50] also propose a repository-mining approach for assessing developer's skills in specific programming languages. Essentially, the aforementioned works differ from the approach described in this paper regarding their methods and goals. CVExplorer considers only commit messages, while we consider the specific files and import statements modified in a commit. CPDScorer works at the level of projects, i.e., the skills acquired by developers on individual commits are not considered. Finally, the approach proposed by Constantinou and Kapitsaki identifies experts in programming languages; by contrast, we target expertise in frameworks and libraries.

Hauff and Gousios [51] rely on natural language processing to match job advertisements to GitHub users. First, they extract concept vectors from the text of job advertisement and from README's files on GitHub. Then, cosine similarity is used to compare and match these vectors. SCSMiner [52] also relies on a vector space model and cosine similarity to calculate the semantic similarity between a project's README and a given query, which can be the name of a programming language or framework or even a more generic skill, such as "game development".

There are also works that rely on machine learning to predict other characteristics and events on software developers life. Wang et. al [53] and Mao et al. [54] investigate the problem of recommending skilled developers to work on programming tasks posted on the TopCoder crowdsourcing platform. Bao et. al. [22] investigate the use of machine learning to predict developers turn over in two private software companies.

Lastly, we also identified previous works that approached developers expertise in a more conceptual level. Siegmund et. al. [19], [55] asked students a set of questions about their programming experience and then, by means of a controlled experiment, contrasted their answers with the performance of the respondents in program comprehension tasks. They report a strong correlation between the number of tasks successfully concluded and the self-estimates. Baltes and Diehl [45] propose a conceptual framework—obtained from a set of mixed-methods—that maps the main traits around software developers expertise. Their framework reinforces that developers expertise depends on deliberate practice to be enhanced.

VIII. Conclusion

Companies often hire based on expertise in libraries and frameworks, as we found in the tags of Stack Overflow jobs. In this paper, we investigated the usage of clustering and machine learning algorithms to identify library experts, using public GitHub data. First, we found that standard machine learning classifiers (e.g., Random Forest and SVM) do not have a good performance in this problem, at least when they are trained with all developers from a sample of GitHub users. The main reason is that not all experts have a strong presence on GitHub. By contrast, we can use clustering techniques to identify experts with high activity on GitHub projects that depend on particular libraries and frameworks. Particularly, we found clusters with 74% (REACT), 65% (NODE-MONGODB), and 75% (SOCKET.IO) of experts. Supported by these results, we proposed a method to identify library experts based on their similarity (in terms of feature data) to a cluster previously labeled as including a high proportion of experts.

As future work, we recommend to (1) investigate other target libraries and frameworks; (2) investigate the use of features from other platforms, such as Stack Overflow and TopCoder; and (3) investigate the accuracy of the proposed method with other developers, including developers of less popular projects. As a final note, our data—in a fully anonymized format—and scripts are publicly available at: https://doi.org/10.5281/zenodo.1484498.

Acknowledgments

We thank the 575 GitHub users who kindly answered our survey. This research is supported by CNPq and FAPEMIG.

REFERENCES

[1] I. J. M. Ruiz, B. Adams, M. Nagappan, S. Dienst, T. Berger, and A. E. Hassan, "A large-scale empirical study on software reuse in mobile apps," *IEEE Software*, vol. 31, no. 2, pp. 78–86, 2014.

[2] A. A. Sawant and A. Bacchelli, "fine-GRAPE: fine-grained API usage extractor - an approach and dataset to investigate API usage," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1348–1371, 2017.

[3] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (NIER track)," in *International Conference on Software Engineering (ICSE)*, 2011, pp. 804–807.

[4] K. Siau, X. Tan, and H. Sheng, "Important characteristics of software development team members: an empirical investigation using repertory grid," *Information Systems Journal*, vol. 20, no. 6, pp. 563–580, 2010.

[5] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *International Conference on Software Engineering (ICSE)*, 2002, pp. 503–512.

[6] T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *Foundations of Software Engineering (FSE)*, 2007, pp. 341–350.

[7] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *International Conference on Software Engineering (ICSE)*, 2010, pp. 385–394.

[8] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: modeling a developer's knowledge of code," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, pp. 14:1–14:42, 2014.

[9] D. Schuler and T. Zimmermann, "Mining usage expertise from version archives," in *International Working Conference on Mining Software Repositories (MSR)*, 2008, pp. 121–124.

[10] J. R. Da Silva, E. Clua, L. Murta, and A. Sarma, "Niche vs. breadth: Calculating expertise over time through a fine-grained analysis," in *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015, pp. 409–418.

[11] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[12] J. Weston and C. Watkins, "Multi-Class Support Vector Machines," University of London, Tech. Rep., 1998.

[13] L. A. Dabbish, H. C. Stuart, J. Tsay, and J. D. Herbsleb, "Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository," in *ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW)*, 2012, pp. 1277–1286.

[14] G. Avelino, L. Passos, F. Petrillo, and M. T. Valente, "Who can maintain this code? assessing the effectiveness of repository-mining techniques for identifying software maintainers," *IEEE Software*, vol. 1, no. 1, pp. 1–15, 2019.

[15] G. Avelino, L. Passos, A. Hora, and M. T. Valente, "A novel approach for estimating truck factors," in *24th International Conference on Program Comprehension (ICPC)*, 2016, pp. 1–10.

[16] L. Singer, F. Figueira Filho, B. Cleary, C. Treude, M.-A. Storey, and K. Schneider, "Mutual Assessment in the Social Programmer Ecosystem: An Empirical Investigation of Developer Profile Aggregators," in *ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW)*, 2013, pp. 103–116.

[17] J. Marlow and L. Dabbish, "Activity Traces and Signals in Software Developer Recruitment and Hiring," in *ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW)*, 2013, pp. 145–156.

[18] J. Kruger and D. Dunning, "Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments." *Journal of personality and social psychology*, vol. 77, no. 6, p. 1121, 1999.

[19] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding Understanding Source Code with Functional Magnetic Resonance Imaging," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 378–389.

[20] L. Yu and H. Liu, "Feature Selection for High-dimensional Data: A Fast Correlation-based Filter Solution," in *International Conference on Machine Learning (ICML)*, 2003, pp. 856–863.

[21] Z. Chen, T. Menzies, D. Port, and B. Boehm, "Finding the Right Data for Software Cost Modeling," *IEEE Software*, vol. 22, no. 6, pp. 38–46, 2005.

[22] L. Bao, Z. Xing, X. Xia, D. Lo, and S. Li, "Who Will Leave the Company? A Large-Scale Industry Study of Developer Turnover by Mining Monthly Work Report," in *International Conference on Mining Software Repositories (MSR)*, 2017, pp. 170–181.

[23] N. Zumel, J. Mount, and J. Porzak, *Practical Data Science with R*, 1st ed. Manning, 2014.

[24] M. Kuhn and K. Johnson, *Applied Predictive Modeling*, 1st ed. Springer, 2013.

[25] S. Chulani, B. W. Boehm, and B. Steece, "Bayesian Analysis of Empirical Software Engineering Cost Models," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 573–583, 1999.

[26] S. J. Raudys and A. K. Jain, "Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 252–264, 1991.

[27] N. Japkowicz and S. Stephen, "The Class Imbalance Problem: A Systematic Study," *Intelligent Data Analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[28] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, no. 1, pp. 321–357, 2002.

[29] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online Defect Prediction for Imbalanced Data," in *International Conference on Software Engineering (ICSE)*, 2015, pp. 99–108.

[30] L. Li, T. F. Bissyandé, D. Octeau, and J. Klein, "Reflection-aware Static Analysis of Android Apps," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 756–761.

[31] F. Zampetti, C. Noiseux, G. Antoniol, F. Khomh, and M. D. Penta, "Recommending when Design Technical Debt Should be Self-Admitted," in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 216–226.

[32] Y. Zhou and A. Sharma, "Automated Identification of Security Issues from Commit Messages and Bug Reports," in *Foundations of Software Engineering (ESEC/FSE)*, 2017, pp. 914–919.

[33] M. Claesen and B. D. Moor, "Hyperparameter Search in Machine Learning," *Metaheuristics International Conference (MIC)*, pp. 1–5, 2015.

[34] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[35] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *International Conference on Software Engineering (ICSE)*, 2014, pp. 1025–1035.

[36] M. J. Arafeen and H. Do, "Test case prioritization using requirements-based clustering," in *International Conference on Software Testing, Verification and Validation (ICST)*, 2013, pp. 312–321.

[37] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella, "A tale of CI build failures: An open source and a financial organization perspective," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 183–193.

[38] A. Ng, "Machine Learning Course (Stanford CS229 Lecture notes)," 2000.

[39] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, no. C, pp. 53–65, 1987.

[40] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*. Lawrence Erlbaum, 2005.

[41] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys," in *Annual Meeting of the Florida Association of Institutional Research*, 2006, pp. 1–33.

[42] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "API change and fault proneness: a threat to the success of Android apps," in *Foundations of Software Engineering (FSE)*, 2013, pp. 477–487.

[43] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free android applications," in *International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 301–310.

[44] K. A. Ericsson, *The Cambridge Handbook of Expertise and Expert Performance*, 2006, ch. 38, pp. 683–704.

[45] S. Baltes and S. Diehl, "Towards a Theory of Software Development Expertise," in *Foundations of Software Engineering (FSE)*, 2018, pp. 1–14.

[46] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[47] D. Chicco, "Ten Quick Tips for Machine Learning in Computational Biology," *BioData Mining*, vol. 10, no. 1, pp. 1–35, 2017.

[48] G. J. Greene and B. Fischer, "CVExplorer: Identifying Candidate Developers by Mining and Exploring Their Open Source Contributions," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2016, pp. 804–809.

[49] W. Huang, W. Mo, B. Shen, Y. Yang, and N. Li, "CPDScorer: Modeling and Evaluating Developer Programming Ability across Software Communities," in *Software Engineering and Knowledge Engineering Conference (SEKE)*, 2016, pp. 01–06.

[50] E. Constantinou and G. M. Kapitsaki, "Identifying Developers' Expertise in Social Coding Platforms," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016, pp. 63–67.

[51] C. Hauff and G. Gousios, "Matching GitHub Developer Profiles to Job Advertisements," *Working Conference on Mining Software Repositories (MSR)*, pp. 362–366, 2015.

[52] Y. Wan, L. Chen, G. Xu, Z. Zhao, J. Tang, and J. Wu, "SCSMiner: Mining Social Coding Sites for Software Developer Recommendation with Relevance Propagation," *World Wide Web*, pp. 1–21, 2018.

[53] Z. Wang, H. Sun, Y. Fu, and L. Ye, "Recommending Crowdsourced Software Developers in Consideration of Skill Improvement," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 717–722.

[54] K. Mao, Y. Yang, Q. Wang, Y. Jia, and M. Harman, "Developer Recommendation for Crowdsourced Software Development Tasks," in *IEEE Symposium on Service-Oriented System Engineering (SOSE)*, 2015, pp. 347–356.

[55] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1299–1334, 2014.