How Well Do You Know This Library? Mining Experts from Source Code Analysis

Johnatan Oliveira[1], Markos Viggiato[2], Eduardo Figueiredo[1]
[1]Dept. of Computer Science, Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil
[2]Dept. of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada
johnatan.si@dcc.ufmg.br,viggiato@ualberta.ca,figueiredo@dcc.ufmg.br

ABSTRACT

Third-party libraries have been widely adopted in modern software projects due to several benefits, such as code reuse and software quality. Software development is increasingly complex and requires specialists with knowledge in several technologies, such as the nowadays libraries. Such complexity turns it extremely challenging to deliver quality software given the time pressure. For this purpose, it is necessary to identify and hire qualified developers, to obtain a good team, both in open source and proprietary systems. For these reasons, enterprise and open source projects try to build teams composed of highly skilled developers in specific libraries. Developers with expertise in specific libraries may reduce the time spent on software development tasks and improve the quality of the final product. However, their identification may not be trivial. In this paper, we first argue that source code activities can be used to identify library experts. We then evaluate a mining-based strategy to identify library experts. To achieve our goal, we selected the 9 most popular Java libraries and identified the top-10 experts in each library by analyzing commits in 16,703 Java projects on GitHub. We validated the results by applying a survey with 137 library expert candidates and observed, on average, 88% of precision for the applied strategy.

CCS CONCEPTS

• **Software and its engineering → Software design engineering**; **Software defect analysis**; **Software evolution**; **Maintaining software**.

KEYWORDS

Library Experts, Software Skills, Expert Identification, Mining Software Repositories

ACM Reference Format:
Johnatan Oliveira[1], Markos Viggiato[2], Eduardo Figueiredo[1]. 2019. How Well Do You Know This Library? Mining Experts from Source Code Analysis. In *XVIII Brazilian Symposium on Software Quality (SBQS'19), October 28-November 1, 2019, Fortaleza, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3364641.3364648

1 INTRODUCTION

Software development has become increasingly complex, both in open source and proprietary systems [5]. Such complexity makes it extremely challenging to delivery software with quality in time, as well as may hinder the participation of developers in worldwide repositories of source code, such as GitHub [25]. In order to find developers to contribute to open source projects or to hire developers (in the case of a company), identifying the developer with the right (and required) skills to obtain a good team is a difficult task [8, 13]. In addition, in many cases, project managers must build teams of skilled developers in relevant libraries. However, decisions taken at the hiring process are a well-known decisive factor to the success of a software project [24]. Providing a more reliable way of identifying developers' skills can support project managers to take the right decision when hiring or attracting the right developers for an open source project. The task of finding experts in specific technologies is especially complex, despite the existence of business-oriented social networks, such as LinkedIn, where developers write about their attributes and qualifications. In fact, this type of platform is commonly used for online recruitment of professionals. However, developers may inflate their skills or, conversely, omit some skills.

Most currently used strategies to find experts have their limitations [3, 24]. For instance, the analysis of the curriculum from LinkedIn or in paper format can omit desirable skills. In addition, developers may have difficulty to express their qualifications [24]. Sometimes, the developer has a specific ability but considers it irrelevant. In another situation, the developer cites many skills, but does not have expertise in the technologies mentioned [3]. Even large companies may rely on traditional curriculum analysis, and this type of analysis may have inaccurate or outdated information. Besides, even talent recruiters may incorrectly identify the developer skills or identify other skills that are not the focus of the organization. Hiring lowly skilled software developers can lead to additional costs, efforts, and resources for training them, or expending more time and resources hiring others [3, 8, 22].

Software developers have used social coding platforms, such as GitHub and BitBucket, to showcase their work in the hope that this may help them being hired for a better job. Developers use these social coding platforms to demonstrate their skills and to create an online profile about their projects and technologies [3]. Some contributors are even using the social aspects of these platforms to infer project popularity trends and promote themselves more efficiently through specific projects and collaborations in other open source projects [3, 4]. In some cases, profiles derived from accounts of social platforms, such as GitHub, are considered even more reliable than a curriculum from LinkedIn, concerning the technical qualifications of a job candidate [3]. Therefore, the exploitation of data from coding platforms is a promising way for potential employers when assessing candidates [2].

GitHub has been widely used in several works mainly because it provides several user-based summary statistics, such as the number of contributions in the last year, the number of forked projects, and the number of followers. For instance, some works have used this platform to identify appropriate maintainers for the source code [1] and collaborations between projects [4]. Different approaches have been used to investigate the skills of developers from GitHub [9, 14, 20]. For instance, a prior work conducted interviews with members of GitHub to understand the hiring process [12]. Our study builds on the works of [19] and [15] which proposed a strategy to identify library experts from GitHub.

In this paper, we evaluate the feasibility of identifying library experts from source code analysis. We rely on GitHub data to identify the skills of developers based on the actual contributions they made. From each type of developer contribution, we can identify essential developers skills. We evaluate the applicability and accuracy of the strategy. In the applicability evaluation, we performed a mining study with the top-9 most popular Java libraries from GitHub, aiming to identify library experts in these libraries. In total, we analyzed more than 16 thousand projects. In the accuracy evaluation, we designed and sent a survey with more than 1 thousand top developers identified for these libraries. We received answers from 137 developers. We selected developers with the top-20% highest values in at least two metrics. As a result, we observe that it is possible to identify experts from source code with high precision (in average 88%). We also note that the strategy provides meaningful information to recruiters, such as, the history of write lines of code (LOC) for each library. These details about the developers can improve the selection of candidates. Our key contributions are threefold: (1) we developed a tool to support identification library experts from source code analysis; (2) we empirically evaluate the applicability and accuracy of identifying library experts; and (3) we identify 1,045 experts in 9 libraries.

The remainder of this paper is organized as follows. In Section 2, we describe our analysis by detailing the strategy to identify library experts, our research questions and survey design. Section 3 presents the results of the applicability evaluation to identify library experts. Section 4 shows the results to accuracy evaluation from a survey with experts candidates. Section 5 presents and discusses threats to validity. Related work is discussed in Section 6. Finally, Section 7 discusses the concluding remarks and future work.

2 STUDY SETTINGS

This section describes the protocol to evaluate the identification of library experts through an empirical study. Section 2.1 presents the aims of our study and the research questions we address. Section 2.2 shows the steps performed to evaluate the experts. Section 2.3 describes the used dataset. Section 2.4 presents our strategy by explaining its steps to identify library experts. Section 2.5 presents the supporting tool and how it applies the strategy. Section 2.6 presents the settings of the survey with developers from GitHub.

2.1 Goal and Research Questions

The primary goal of this study is to evaluate the applicability and accuracy of a strategy to identify library experts from source code analysis using software repositories. We are interested in whether the strategy can precisely identify experts in a specific library. We are also concerned with assessing the relevance of the results provided by the strategy. For this purpose, we select the 10 most popular and standard Java libraries among the GitHub developers. To achieve this goal, we use the Goal-Question-Metric method to select measurements of source code. The GQM method proposes a top-down approach to define measurement; goals lead to questions, which are then answered with metrics [10].

Table 1 shows the GQM with the research questions and metrics investigated in this study. As mentioned, the goal of this paper is to identify library experts from source code. Therefore, from this goal, we check if it is feasible to analyze the source code in order to identify library experts. Through RQ1, we are interested in investigating the efficiency of the number of commits (metric) to indicate the level of activity of a developer in a specific library. In other words, we aim to analyze the number of commits involving a specific library performed by a developer to compute his level of activity in a library.

With RQ2, we aim at assessing the knowledge intensity based on the number of imports to a specific library. That is, from all imports made by a developer at the source code, we investigate the number related to the specific library. Finally, the last research question (RQ3) analyzes the knowledge extension of the developers from the number of LOC related to the library (metric). In this last question, we aim to evaluate the amount of LOC implemented by a developer using a specific library. For this purpose, we evaluate the relation of total LOC and LOC related to a specific library.

Table 1: The Metrics Analysis as GQM method

Questions	Metrics
RQ1– How to evaluate the level of activity of a developer in a library?	Number of commits
RQ2– How to evaluate the knowledge intensity of a developer in a library?	Number of imports
RQ3– How to evaluate the knowledge extension of a developer in a library?	Lines of Code

2.2 Evaluation Steps

This section describes the steps to evaluate the identification of library experts from source code. To answer the research questions presented in Section 2.1, we designed a mixed-methods study composed of four steps: 1) *Library Selection*, 2) *Dataset Collection*, 3) *Expert Identification*, and 4) *Survey Application*. Figure 1 presents the steps of our research, which are discussed next. For *Library Selection* (Section 2.3), we selected the top-10 most popular libraries in the Java programming language to identify library experts. In the *Dataset Collection* step (Section 2.3), we clone the projects that contain these libraries from GitHub. For *Identification of Library Experts* (Section 2.4), we compute the skills of developers based on three metrics: *Number of Commits*, *Number of Imports*, and *Lines of*

Code. These metrics are presented in Section 2.4. Finally, we performed a *survey study*. This survey was conducted to identify the accuracy of the strategy according to the responses of developers. Section 2.6 presents details about the survey.

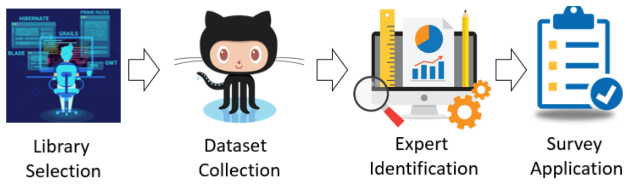


Figure 1: Study Steps

2.3 Dataset

To create our dataset, we select the 10 most popular and common Java libraries among the GitHub developers: Hibernate, Selenium, Hadoop, Spark, Struts, GWT, Vaadin, Primefaces, Apache Wicket, and JavaServer Faces. The selection was made based on a survey provided by Stack Overflow[1] in 2018 with answers of over 100,000 developers around the world. Table 2 summarizes the definitions of each library. All definitions of the libraries were retrieved from Stack Overflow and their Web pages. We selected Java because it is one of the most popular programming languages [2] and there are many Java projects available on GitHub. The projects that compose our dataset were retrieved in August 2018.

Figure 2 illustrates the criteria for defining our dataset. To achieve more realistic results for software development, we apply the following exclusion criteria. 1) We excluded systems with less than 1 KLOC because we considered them toy examples or early stage software projects. 2) We removed projects with no commit in the last 3 years because the developer may forget his code [11]. Finally, in the last exclusion criteria (3), we removed projects which did not contain imports related to the selected libraries. Besides the 3 mentioned criteria, we excluded all official projects of these libraries, because we assume all developers of a library project are experts in the corresponding library. We also removed libraries with less than 100 projects (the case of JavaServer Faces). Therefore, we end up analyzing 9 libraries in this study.

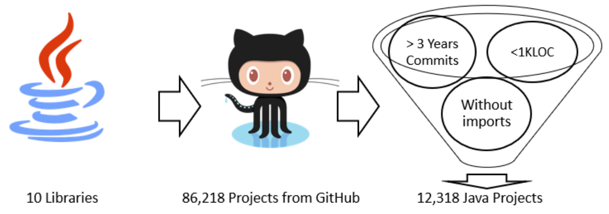


Figure 2: Steps for Collecting Software Projects from GitHub

Table 3 shows the number of remained projects after each step in our filtering process. The column *Projects* presents the number of projects initially selected. Next, the column *Filtered* shows the number of projects removed through filtering step. Finally, the column *Remained* presents the number of projects analyzed for each library.

[1]https://insights.stackoverflow.com/survey/2018#most-popular-technologies
[2]https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

Table 2: Library Descriptions

Library	Description
Hibernate	Hibernate is a library of object-relational mapping to object-oriented.
Selenium	Selenium is a test suite specifically for automating Web.
Hadoop	Hadoop is a library that facilitates the use of the network from many computers to solve problems involving massive amounts of data [23, 27].
Spark	Spark is a general-purpose distributed computing engine used for processing and analyzing a large amount of data
Struts	Struts help in developing Web-based applications.
GWT	Google Web Toolkit (GWT) allows Web developers to develop and maintain complex JavaScript front-end applications in Java.
Vaadin	Vaadin includes a set of Web components, a Java Web library, and a set of tools and application starters. It also allows the implementation of HTML5 web user interfaces using the Java.
PrimeFaces	PrimeFaces is a library for JavaServer Faces featuring over 100 components.
Apache Wicket	Wicket provides a library for creating reusable components and offers an object-oriented methodology to Web development while requiring only Java and HTML.
JavaServer Faces	JavaServer Faces is a Java view library running on the server machine which allows you to write template text in client-side languages (like HTML, CSS, JavaScript, etc.).

Table 3: Projects Selected for Analysis

Library	#Projects	Filtered	Reimaned
Hibernate	31,134	26,020	5,114
Selenium	19,062	17,648	1,414
Hadoop	11,715	10,778	937
Spark	9,144	7,650	1,494
Struts	4,741	4,127	614
GWT	4,086	2,635	1,451
Vaadin	3,240	2,625	615
PrimeFaces	1,881	1,401	480
Apache Wicket	1,095	896	199
JavaServer Faces	120	120	0
TOTAL	86,218	73,900	12,318

2.4 Identification of Library Experts

This evaluation performs three steps described as follows.
Step 1: Extract data from source code – In this step, we obtain data of the classes created by developers from a Git repository. All data, such as added or removed LOC, written imports, commits, date, email, and the name of developers are stored locally. **Step 2: Search for imports** – From the previous step, we search for specific "imports" related to the chosen library. The idea is to explore all files that import the name of the target library. **Step 3: Calculate skills** – In this last step, we compute the skills for each developer. We rely on three metrics to identify the library experts. Table 4 shows these metrics and explains them in more details. Each metric is computed in relation to the amount of commits to a specific library. That is, when a commit to a library is identified to library, the metrics were calculated.

Table 4: Proposed Metrics

Metric	Description
Number of Commits	This metric calculates the activity of each developer through the number of commits using a particular library. Through this metric, it is possible to measure the amount of use of the library in a project that a specific developer works.
Number of Imports	This metric presents the intensity of use of a particular library. For this metric, we count all imports to the library written by a developer. Repeated imports are included.
Lines of Code	To compute this metric, we developed a heuristic to count the amount of LOC related to a specific library, as follows. First, we obtain the ratio of changed LOC by the number of all imports in the file. Then, we multiply the ratio by the number of imports related to the library.

2.5 Tool Support

The identification of library experts is supported by a prototype tool, named JExpert. We developed JExpert in Java programming language. JExpert only works with Java projects, but the tool can be easily adapted to identify library experts in other programming languages. Figure 3 presents the simplified architecture design of JExpert. The tool expects a set of projects from a Git repository and keywords to represent each chosen library. It processes each commit, extracting three pieces of information: (i) the file path; (ii) the developer who performed the change; and (iii) the type of the change. After that, JExpert analyzes the projects to find the library experts. It then computes three metrics (Table 4) and returns a ".xls" file with the sorted list of experts for each library.

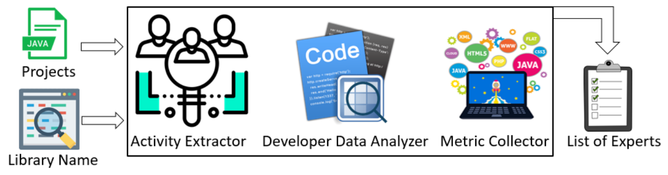


Figure 3: JExpert Architecture Overview

2.6 Survey Design

This section describes the survey applied to developers from GitHub to evaluate the accuracy of JExpert. According to Easterbrook et al. [7], survey studies are used to identify characteristics of a population and are usually associated with the application of questionnaires. In addition, surveys are meant to collect data to describe, compare or explain knowledges, attitudes, and behaviors [18]. We select the library experts with the best values in the evaluated metrics to validate them through a survey. We designed and applied a survey with the top developers identified by our strategy. We selected developers with the top-20% highest values in at least two (out of three) metrics.

We created a questionnaire on Google Forms[3] with two parts: the first one was composed of 5 questions about the background of the expert candidates; the second part also had 5 questions about the knowledge of the expert candidates regarding the evaluated libraries. Table 5 shows the list of questions in the first part of our survey and Table 6 summarizes the questions of the second part. This table contains the tag <*libray name*> meaning a specific library, for instance, Hadoop. The background questions were named BQ1 to BQ5, while the questions of the second part of our survey were named SQ1 to SQ5. Tables 5 and 6 also describe the possible answers for each question.

Table 5: Survey Questions on the Participant Background

ID	Questions
BQ1	What is your highest level of education? ( ) Technical formation ( ) Bachelor's ( ) Post-graduate ( ) Master's ( ) PhD
BQ2	In which course are you graduated? ( ) Computer Science ( ) Information Systems ( ) Software Engineering ( ) Others
BQ3	How many years have you dedicated to software development? ( ) Less than 1 year ( ) 1 to 5 years ( ) 5 to 10 years ( ) More than 10 years
BQ4	How often do you use Git? ( ) Most of the time (all code) ( ) Sometimes (not all code) ( ) Rarely (little code)
BQ5	Which role do you usually play when developing applications? ( ) Back-end developer ( ) Front-end developer ( ) Mobile developer ( ) For learning

Table 6: Survey Questions on the Use of the Libraries

ID	Questions
SQ1	How do you assess your knowledge in <*libray name*>? ( ) 1 ( ) 2 ( ) 3 ( ) 4 ( ) 5
SQ2	How many projects have you worked with <*libray name*>? ( ) 1 to 5 ( ) 6 to 10 ( ) 11 to 20 ( ) More than 20 projects
SQ3	How many packages of <library name> have you used? ( ) A few ( ) A lot
SQ4	How often do your commits include <*libray name*>? ( ) A few ( ) A lot
SQ5	How much of your code is related to <*libray name*>? ( ) Few of my code is related to <*libray name*> ( ) My code is partially related to <*libray name*> ( ) Most of my code contains <*libray name*>

To obtain the email that the developer used to perform the commits in the source code, we used the Git-Blame[4] tool. The emails

[3]https://www.google.com/forms/
[4]https://git-scm.com/docs/git-blame

were collected in order to send the survey. We send an email to each developer asking him/her to assess his/her knowledge on each library. The developers are invited, for instance, to rank their knowledge (Table 6, SQ1) using a scale from 1 (one) to 5 (five), where (1) means no knowledge about the library; and (5) means extensive knowledge about the library. Questions are not mandatory because they may require knowledge on exceptional features of the library. Therefore, participants are not forced to provide an answer when they did not remember a specific element of the library, such as, time of development using the library and the approximate frequency of commits that contains the library. The survey remained open for three weeks in January 2019.

3 APPLICABILITY EVALUATION

In this section, we present the results of the applicability evaluation of the study aiming to verify the feasibility of library expert identification. We analyzed 16,703 software systems mined from GitHub and 9 libraries: Hibernate, Selenium, Hadoop, Spark, Struts, GWT, Vaadin, Primefaces, and Apache Wicket. In addition, we analyzed data from more than 1.6 million developers who have contributed to these projects in our dataset. Table 7 shows the results of top-10 library experts in 9 bar charts. The black columns show the results for the *Number of Commits* metric; the grey columns refer to the *Number of Imports* metric; finally, the white columns show the results regarding the *Lines of Code* metric.

The values of data presented in Table 7 are normalized between 0 to 1, and we identify each developer by the start name of library followed by an identifier (e.g., HAD-1 means the first developer of Hadoop). Due to privacy concerns, we are omitting the real name of the identified developers. To obtain the main experts in each library, we select the top-10 developers to present and discuss the results in this section.

In general, it is possible to see from Table 7 that the metrics *Number of Imports* and *Lines of Code* remained high (up to 60%) for libraries GWT (Table 7–a), Hadoop (Table 7–b), Hibernate (Table 7–c), Spark (Table 7–f), and Struts (Table 7–g). More specifically, developers GWT-1, GWT-2, HAD-1, HAD-2, HAD-3, SPA-1, SPA-2, and STR-1 to STR-8 achieved significant results for these 2 metrics. This result suggests that these developers frequently make many imports and write many LOC related to the particular library.

Regarding PrimeFaces (Table 7–d), Selenium (Table 7–e), and Spark (Table 7-f) libraries, the *Number of Commits* metric (black columns) stayed high in all cases (above 60%). For instance, the *Number of Commits* metric remained at high levels (60%), for the following developers and libraries: PRI-1 to PRI-7 from PrimeFaces (Table 7–d), SEL-1 to SEL-6 from Selenium (Table 7–e) and SPA-3 to SPA-9 from Spark (Table 7–f). Consequently, projects that contain these libraries own developers who are more active. However, based only on the values of these metrics, we cannot state whether developers are expert or not as discussed in the next section. Therefore, we must combine *Number of Commits* with other metrics to identify library experts, to promote more accurate results.

When we analyze the metric related to the number of commits together with other metrics, we are able to identify library experts with more precision. For instance, metrics *Number of Commits*, *Number of Imports*, and *Lines of Code* combined show that developers SPA-1, SPA-2, and SPA-3 from Spark (Table 7–f), VAA-1 to VAA-3 from Vaadin (Table 7–h), and WIC-1 from Wicket (Table 7–i) possibly are very skilled since all metrics have high values for them. Due to space constraints, we focus on the more relevant results. However, the complete raw data are available. online [5]

4 ACCURACY EVALUATION

In this section, we present the results of the accuracy evaluation based on a survey with expert candidates in each library. The goal of this evaluation is to verify the precision of the library expert identification. We empirically selected 1,045 developers among the top-20% values in at least two metrics. The questionnaire was sent January 2019. After a period of 15 days, we obtained 137 responses resulting in a response rate of about 15%. We asked the 137 developers about their software development experience in general (background), and the use of the specific libraries investigated in this paper. Each RQ is presented in specific sections. Section 4.1 discusses the general goal. Section 4.2 shows the results for RQ1. Section 4.3 presents the results of RQ2. Finally, Section 4.4 reports the results to RQ3.

Table 8 presents the setup of the experts' candidates contacted to answer our survey. This table has the following structure. The first column (library) indicates the name of the analyzed library. The second column (emails sent) shows the number of emails collected and sent to expert candidates. The third column (email invalid) presents the number of emails invalid which returned by the server. In the fourth column (remaining emails) indicates the number of valid emails. The fifth column shows the number of answers we obtained for each library. Finally, in the last column, we show the response rate of each library.

Table 9 reveals the participants' background. It is worth highlighting that in some cases the percentage presented in Half of the respondents (50% – first column) are graduated in Computer Science, 42% have Masters degree and 7% Ph.D. degree (second column). Note that more than half (51% – third column) of the respondents are back-end developer. That is, a programmer who creates the logical back-end and core computational logic of a website, software or information system. Most of our respondents (71%) have use the architecture Git (fourth column). Concerning time dedicated to software development, 47% have more than 10 years of experience, and less than 3% have less than 1 year of experience (fifth column). Therefore, we can conclude that, in general, the participants are not novices.

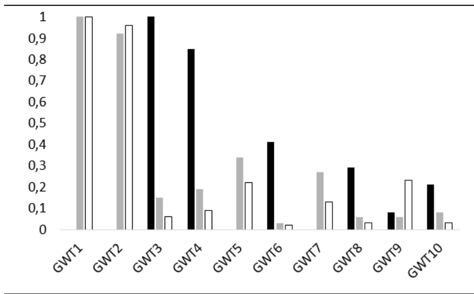
4.1 Overview

In this section, we present a overview of some relevant findings.

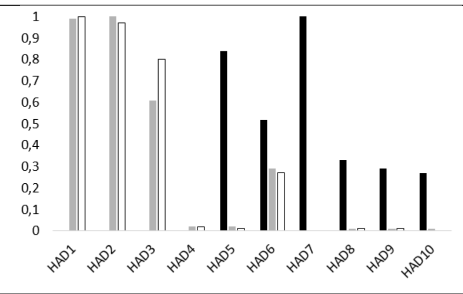
Our study shows that a significant amount of expert candidates makes commits, when writing code related to a specific library, performs many imports of particular libraries, and writes lines of code in sequence when making an import of the library. Table 10 shows the results about knowledge that surveyed developers claim to have in each library. The developers were invited to rank their knowledge using a scale from 1 (one) to 5 (five), where (1) means no knowledge about the library; and (5) means extensive knowledge

[5] https://tinyurl.com/scam2019

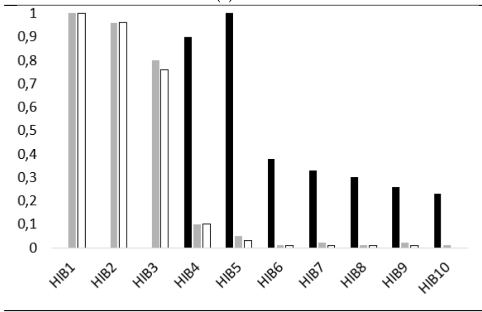
Table 7: Top 20% from Library Experts



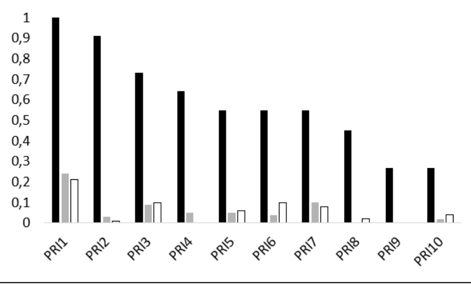
(a) GWT



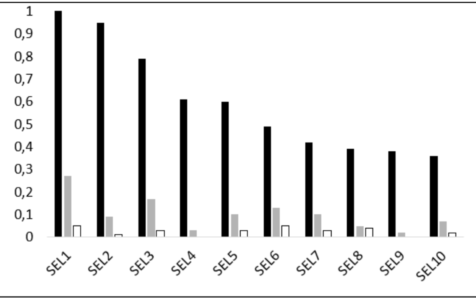
(b) Hadoop



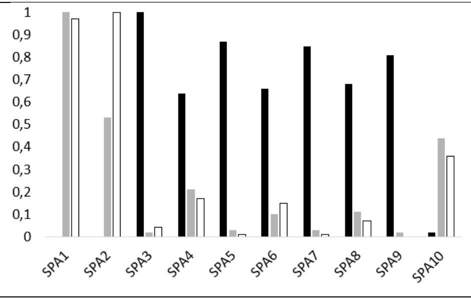
(c) Hibernate



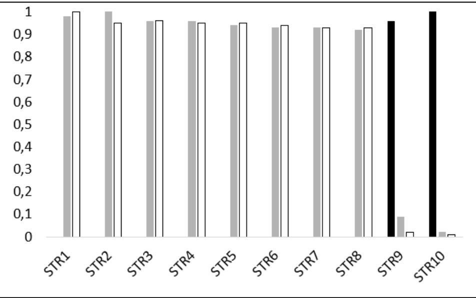
(d) PrimeFaces



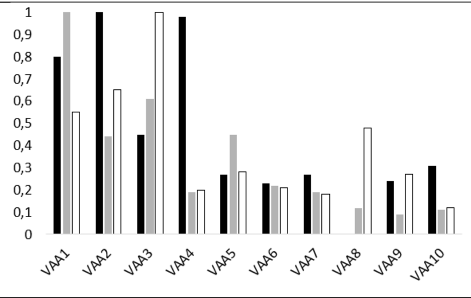
(e) Selenium



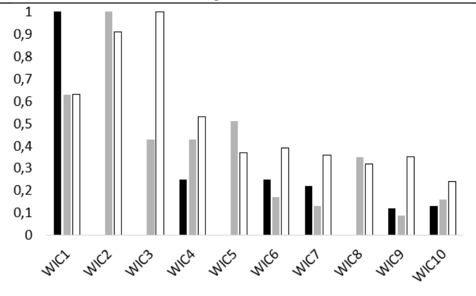
(f) Spark



(g) Struts



(h) Vaadin



(i) Wicket

■ Number of Commits
▨ Number of Imports
□ LOC

(j) Legend

Table 8: Top 20% from Library Experts Selected to Answer the Survey

Library	Emails sent	Invalid email	Remaining email	# Answers	%
GWT	160	18	142	31	0,22
Hadoop	181	33	148	11	0,07
Hibernate	155	10	145	16	0,11
Spark	138	19	119	11	0,09
Struts	42	2	40	9	0,23
Vaadin	107	18	89	15	0,17
PrimeFaces	30	1	29	9	0,31
Wicket	23	2	21	8	0,38
Selenium	209	31	178	27	0,15
TOTAL	1,045	134	911	137	0,15

about the library. If we analyze the data about the precision of the strategy from the sum of levels 3, 4 and 5 of Likert-type scale, we obtain on average 88,49% of accuracy in relation the knowledge of the developers, i.e., identification is correct in more than 88% of the cases. On the other hand, although a score three may represent an acceptable knowledge, if we followed a more conservative criterion, only classifying as library experts the developers that informed a higher (≥ 4) knowledge on the libraries, we obtain, on average, 63,31% of precision. This way, we conclude that most of the identified expert candidates identified by the strategy contain high knowledge about the evaluated libraries. In contrast, to a level of knowledge < 3, we achieved only 11,51% of the developers, i.e., possibly the strategy fails by selecting these developers.

More than 85% of the library experts who answered the survey have a high knowledge about the evaluated libraries.

4.2 Number of Commits

In this section, we answer the first research question.
RQ1– *How to evaluate the level of activity of a developer in a library?*

To answer this research question, we asked the library experts the following question. "How often are your commits related to the <*libray name*> library"? Table 11 shows the results to this question in the first column. For most libraries, the majority of the participants answered they made "few" commits using the evaluated libraries. This way, if we evaluated the results obtained for this label, it is possible to see that from 137 experts, 54% made "few" commits. For instance, in the library Hibernate, 87% of developers said they made few commits related to this library. Another library that deserves special attention is Struts. In this library, 88% of the developers responded that they made few commits. Regarding the label "a lot", only 39% of experts polled said they performed many commits. GWT was the library with a higher rate of answers to this label (62%). Therefore, the numbers indicate that the metric *Number of Commits* needs to be combined with other metrics to achieved conclusive results about the skill from developers and even develop other metrics to identify the level of activity ability.

Answer to RQ1. A large proportion of library experts make "few" commits using the library. Therefore, we concluded that the solo use of number of commits cannot identify library experts.

4.3 Number of Imports in General of Expert Candidates

In this section, we answer the second research question.
RQ2– *How to evaluate the knowledge intensity of a developer in a library?*

Regarding the number of imports to indicate a library expert, we ask the developers the following question: "How often do you include an import of <*libray name*> library in your commits?". Table 11 shows the results in the second column to this question. We analyze the number of imports performed by developers. The main reason for this analysis is to evaluate the feasibility of inferring the skills of the developers from the types of imports performed. In general, the label "few" and "a lot" are tied or with little difference between them. For example, Hibernate, Spark, and PrimeFaces have practically tied. These libraries did not show significant differences; in some cases, the difference was only of 1 absolute point. In only three cases, the label "a lot" remained significantly higher: GWT (83%), Vaadin (67%) and Selenium (78%).

From 137 experts, 68% said that they made "a lot of imports". However, the number informed by the experts indicates that this metric requires a combination with other metrics to achieve better results, because 32% of experts said they made few imports to libraries evaluated. Therefore, from the survey results, the metric *Number of Imports*, as well as the metric *Number of Commits*, are not able to identify library experts, when we apply one at a time.

Answer to RQ2. The metric *Number of Imports* is not able to identify library experts, when we use it alone. The imports achieved lower overall results in most cases.

4.4 Number of Lines of Code

In order to evaluate the metric *Lines of Code*, we present the third research question as follows.
RQ3– *How to evaluate the knowledge extension of a developer in a library?*

In this research question, we analyze the developers skill from the number of LOC related to library. We evaluate the number LOC implemented by a developer to specific library. For this purpose, we asked the library experts from the survey the following question. "How much of your code is related to the <*libray name*> library when you perform a commit?". Table 11 shows the results from third column to this question. The libraries GWT, Wicket, Selenium, and Hadoop for instance, obtained 74%, 71%, 70%, and 64% respectively to label "a lot".

We noted, however, the label "a few" also remained at a high level in some cases, for instance, the libraries Struts (88%) and Spark (55%). In fact, the library Hibernate remained tied to labels "a few" and "a lot". In general, from 137 experts, 39% said they write "a few" LOC and 61% write "a lot" LOC with respect to libraries. Therefore, it is possible to infer that the metric *Lines of Code* alone also does not provide indications about developer skills, although this metric achieved better precision then the metric *Number of Commits*.

Table 9: Respondents' Background

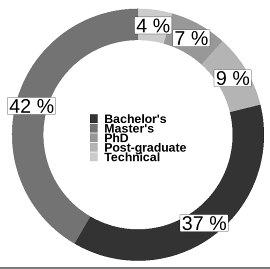
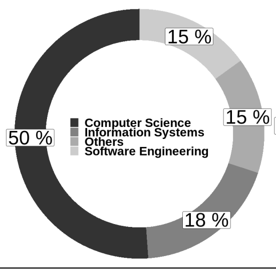
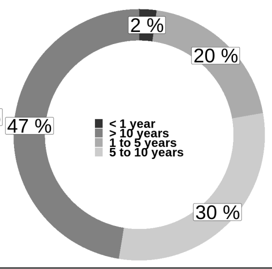
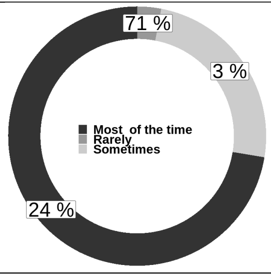
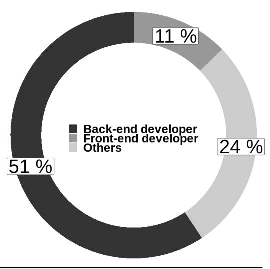
What is your highest level of education?	In which course are you graduated?	How many years have you dedicated to the software development?	How often do you use Git?	Which role do you usually play when developing applications?
				

Table 10: Level of Knowledge in Each Library

Library	Likert scale					Total	4-5	3-4-5
	1	2	3	4	5			
GWT	1	1	4	9	16	31	81%	94%
Hadoop	0	1	3	4	3	11	64%	91%
Hibernate	1	3	8	3	3	18	33%	78%
Spark	0	1	4	2	4	11	55%	91%
Struts	2	2	1	4	0	9	44%	56%
Vaadin	0	2	5	3	5	15	53%	87%
PrimeFaces	0	0	4	4	1	9	56%	100%
Wicket	1	0	2	4	1	8	63%	88%
Selenium	0	1	4	13	9	27	81%	96%

Answer to RQ3. According to our analysis, the metric *Lines of Code* alone cannot reliably provide indications about developers' skills. However, it achieved in general, results better than the metric *Number of Commits*.

5 THREATS TO VALIDITY

We based our study on related work to support evaluation of a strategy to identify library experts. Regarding the evaluation, we conducted a careful empirical study to assess efficiency of the strategy from software systems hosted from GitHub. The strategy evaluated is able to analyze source code from platforms that follows the Git architecture. However, some threats to validity may affect our research findings. The main threats and respective treatments are discussed below based on the proposed categories of Wohlin et al. [26].

Construct Validity. This validity is related to whether measurements in the study reflect real-world situations [26]. Before running the strategy, we conducted a careful filtering of software systems from GitHub repositories. However, some threats may affect the correct filtering of systems, such as human factors that wrongly lead to the discard of a valid system to be evaluated. Considering that the exclusion criteria to system selection were applied in a manual process, we may have discarded interesting systems that we identified as non-Java, for instance.

Internal Validity. The validity is related to uncontrolled aspects that may affect the strategy results [26]. The strategy may be affected by some threats. To treat this possible problem, we selected a sample of 5 software systems that contains the library Hadoop from our dataset, with a diversified number of LOC. Then, we manually identified the number of commits from the GitHub repository, the number of imports and the number of LOC codified to the specific library. We compared our manual results with the results provided by the tool and observed a loss of 5% in metrics terms computed through the automated process. We believe that this error rate do not invalidate our main conclusions.

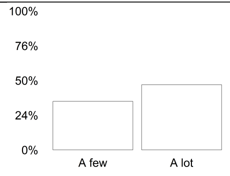
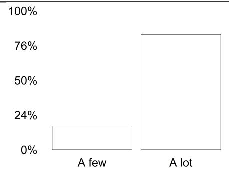
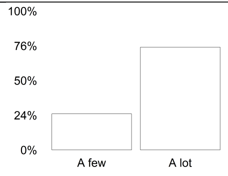
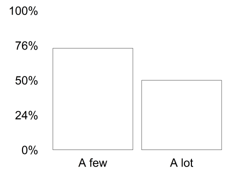
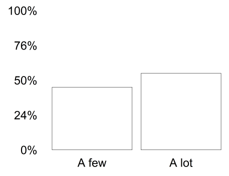
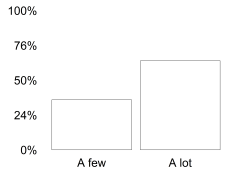
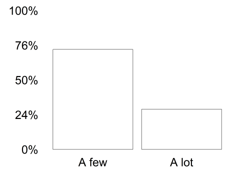
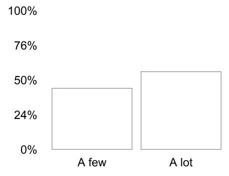
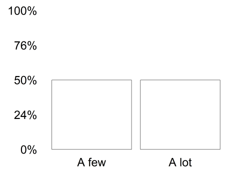
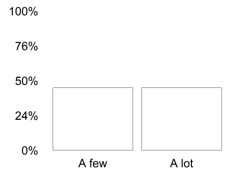
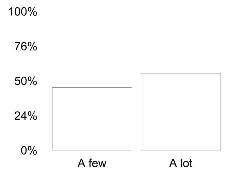
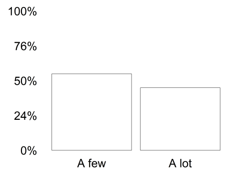
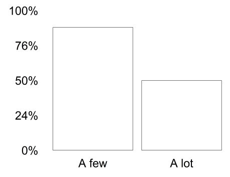
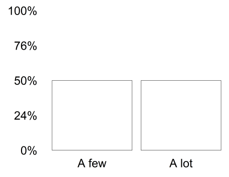
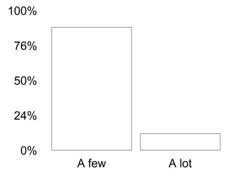
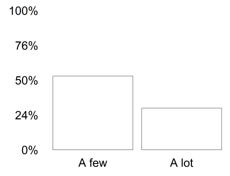
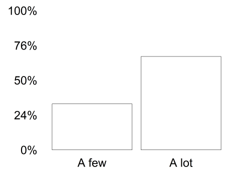
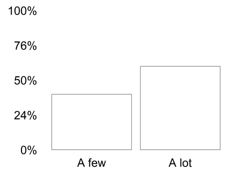
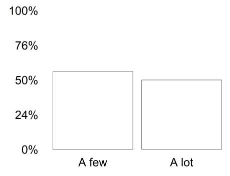
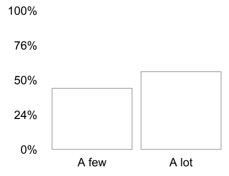
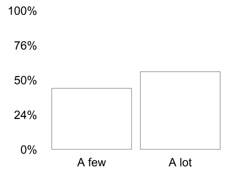
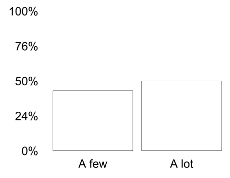
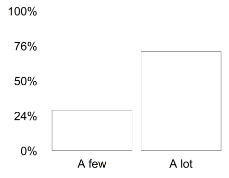
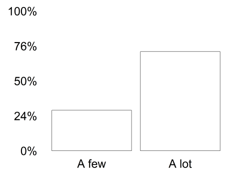
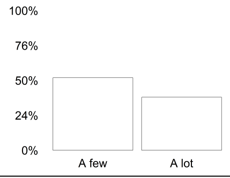
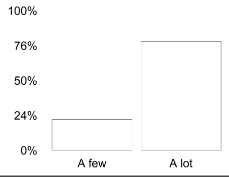
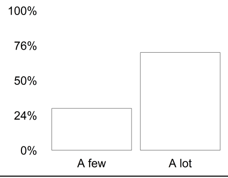
External Validity. This validity is related to the possibility to generalize our results [26]. We evaluated the strategy with a set of 16,703 software projects from GitHub. Considering that these systems may not include all existing libraries, our findings may not be generalized. Furthermore, we evaluated the strategy with an online survey with only 137 developers that implemented projects with the investigated libraries. We analyzed the data with only 9 Java libraries. However, we chose the top libraries from the survey reported by StackOverflow in 2018 with over 100,000 responses developers around the world. This way, we believe these libraries can represent the reasonable option to evaluate the strategy.

6 RELATED WORK

The use of data from GitHub to understand how software developers work and collaborate has become recurrent in software engineering studies [6, 9, 16, 21]. Some studies seek to understand the behavior of developers concerning interaction with their peers [16]. For example, a few studies [16, 17] tried to understand who are the developers with peaceful behavior and those with aggressive behavior and if these developers coexist productively in software development projects [17]. Similar studies also tried to understand if there is a relationship between bug resolution time and behavior of developers [16]. Also, some studies investigated developers manners [6] and seek to understand the emotional behavior of software developers [17].

In a close related work, Greene and Fischer [9] have developed an approach to extract technical information from GitHub developers.

Table 11: Survey Results

Library	Number of Commits (RQ1)	Number of Imports (RQ2)	Lines of Code (RQ3)
GWT			
Hadoop			
Hibernate			
Spark			
Struts			
Vaadin			
PrimeFaces			
Wicket			
Selenium			

The work of these researchers also does not differentiate developers from their level of knowledge of technical skills, since a recruiter has several candidates for the same job position. In addition, such work only shows the profile of the users in GitHub, and it does not extract other characteristics of their knowledge and skills. The other limitation is that they do not provide actual data about the developer's knowledge production and neither presented a survey in order to evaluate the results. Singer et al. [21] investigated the use of profile aggregators in the evaluation of developer skills by developers and recruiters. However, these aggregators only gather skills for individual developers, and it is not clear how they support the identification of relevant developers from a large dataset.

We believe that the strategy evaluated in our study is complementary to the described related work, providing a different approach focusing on the identification of possible experts. To the best of our effort, we did not find a similar large scale study that evaluates some strategy able to identify library experts. Hence, we cannot compare the strategy evaluated to other studies.

7 CONCLUSION

In this paper, we evaluated a strategy to identify library experts in software systems. We also presented a prototype tool that implements the strategy. The strategy evaluated is composed of three metrics: Number of Commits, Number of Imports and Lines of Code. We evaluated the strategy in two dimensions, applicability and accuracy. First, Applicability Evaluation analyzed the feasibility of identifying library experts candidates. Second, Accuracy Evaluation compared the results provided by a strategy with developers from a survey on GitHub. In total, we analyzed 16k software systems mined from GitHub, 9 libraries and a survey with 1,045 developers. Our findings point that the strategy was able to identify library experts in different libraries from the set of input software systems with precision of 88% in average.

There are many possible extensions for this work. For instance, we did not consider all the available data in our analysis, such as the number of forks, number of projects belonging to the developer that have received stars, the number of followers, number of methods, source code quality, and contributions in project discussions. Besides, we did not consider the number of lines of code added and removed between versions. Future work can also extend our research to evaluate the strategy to other languages and libraries.

8 ACKNOWLEDGMENTS

This research was partially supported by Brazilian funding agencies: CAPES, CNPq (424340/2016-0), and FAPEMIG (PPM-00651-17).

REFERENCES

[1] 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*, 1(1):1–15, 2018.

[2] A. Capiluppi, A. Serebrenik, and L. Singer. Assessing technical candidates on the social web. *IEEE software*, 30(1):45–51, 2013.

[3] E. Constantinou and G. M. Kapitsaki. Identifying developers' expertise in social coding platforms. In *42th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA)*, pages 63–67, Limassol,Cyprus, 2016. IEEE.

[4] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb. Social coding in github: Transparency and collaboration in an open software repository. In *12th Proc. of the Conf. on Computer Supported Cooperative Work (CSCW)*, pages 1277–1286, Seattle, Washington, USA, 2012.

[5] V. Damasiotis, P. Fitsilis, P. Considine, and J. O'Kane. Analysis of software project complexity factors. In *Proc. of the 2017 International Conf. on Management Engineering, Software Engineering and Service Sciences*, ICMSS '17, pages 54–58, New York, NY, USA, 2017. ACM.

[6] G. Destefanis, M. Ortu, S. Counsell, S. Swift, M. Marchesi, and R. Tonelli. Software development: do good manners matter? *PeerJ Computer Science*, 2(2):1–10, 2016.

[7] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. 2008.

[8] V. C. Garcia, D. Lucrédio, A. Alvaro, E. S. D. Almeida, R. P. de Mattos Fortes, and S. R. de Lemos Meira. Towards a maturity model for a reuse incremental adoption. In *Proc. of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*, SBCARS '07, pages 61–74, Recife,PE, Brazil, 2007. ACM.

[9] G. J. Greene and B. Fischer. Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In *Proc. of the 31st IEEE/ACM Int. Conf. on Automated Software Engineering*, ASE 2016, pages 804–809, New York, NY, USA, 2016. ACM.

[10] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. The emerging role of data scientists on software development teams. In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 96–107, New York, NY, USA, 2016. ACM.

[11] J. Krüger, J. Wiemann, W. Fenske, G. Saake, and T. Leich. Do you remember this source code? In *40th Proc. of the International Conf. on Software Engineering (ICSE)*, pages 764–775, Gothenburg, Sweden, USA, 2018.

[12] J. Marlow and L. Dabbish. Activity traces and signals in software developer recruitment and hiring. In *16th Proc. of the 2013 Conf. on Computer supported cooperative work (CSCW)*, pages 145–156, San Antonio, Texas, USA, 2013.

[13] P. McCuller. *How to recruit and hire great software engineers: building a crack development team.* Apress, 2012.

[14] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *24rd Proc. of the International Conf. on Software Engineering (ICSE)*, pages 503–512, Orlando, FL, USA, 2002.

[15] J. E. Montandon, L. L. Silva, and M. T. Valente. Identifying experts in software libraries and frameworks among GitHub users. In *16th International Conference on Mining Software Repositories (MSR)*, pages 1–12, 2019.

[16] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli. Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. In *12th Proc. of the Working Conf. on Mining Software Repositories (MSR)*, pages 303–313, Florence, Italy, 2015.

[17] M. Ortu, G. Destefanis, S. Counsell, S. Swift, R. Tonelli, and M. Marchesi. Arsonists or firefighters? affectiveness in agile software development. In *18th International Conf. on Agile Software Development (XP)*, pages 144–155, Porto, Portugal, 2016.

[18] S. L. Pfleeger and B. A. Kitchenham. Principles of survey research: Part 1: Turning lemons into lemonade. *SIGSOFT Softw. Eng. Notes*, 26(6):16–18, Nov. 2001.

[19] A. Santos, M. Souza, J. Oliveira, and E. Figueiredo. Mining software repositories to identify library experts. In *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*, SBCARS '18, pages 83–91, New York, NY, USA, 2018. ACM.

[20] R. Saxena and N. Pedanekar. I know what you coded last summer: Mining candidate expertise from GitHub repositories. In *17th Companion of the Conf. on Computer Supported Cooperative Work and Social Computing (CSCW)*, pages 299–302, New York, NY, USA, 2017. ACM.

[21] L. Singer, F. F. 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 *13th Proc. of the Conf. on Computer supported cooperative work (CSCW)*, pages 103–116, San Antonio, Texas, USA, 2013.

[22] I. Sommerville. *Software Engineering.* Pearson, 2015.

[23] J. Tong, L. Ying, T. Hongyan, and W. Zhonghai. Can we use programmer's knowledge? fixing parameter configuration errors in hadoop through analyzing q amp;a sites. In *5th IEEE Int. Congress on Big Data (BigData Congress)*, pages 478–484, San Francisco, CA, USA, 2016.

[24] F. Tsui, O. Karam, and B. Bernal. *Essentials of software engineering.* Jones & Bartlett Learning, 2016.

[25] M. Viggiato, J. Oliveira, E. Figueiredo, P. Jamshidi, and C. Kästner. Understanding similarities and differences in software development practices across domains. In *Proceedings of the 14th International Conference on Global Software Engineering*, ICGSE '19, pages 74–84, Piscataway, NJ, USA, 2019. IEEE Press.

[26] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering.* Springer Publishing Company, Incorporated, 2012.

[27] C. Ye. Research on the key technology of big data service in university library. In *13th Int. Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 2573–2578, Guilin, China, 2017.