MINERAÇÃO DE HABILIDADES TÉCNICAS DE DESENVOLVEDORES DE PROJETOS DE CÓDIGO ABERTO

JOÃO EDUARDO MONTANDON DE ARAUJO FILHO

MINERAÇÃO DE HABILIDADES TÉCNICAS DE DESENVOLVEDORES DE PROJETOS DE CÓDIGO ABERTO

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Marco Túlio de Oliveira Valente

Belo Horizonte

Janeiro de 2021

JOÃO EDUARDO MONTANDON DE ARAUJO FILHO

MINING THE TECHNICAL SKILLS OF OPEN SOURCE DEVELOPERS

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Marco Túlio de Oliveira Valente

Belo Horizonte

January 2021

© 2021, João Eduardo Montandon de Araujo Filho.
Todos os direitos reservados

Araujo Filho, João Eduardo Montandon de.

A663m Mining the technical skills of open source developers [manuscrito] / João Eduardo Montandon de Araujo Filho.- 2021.
xix, 113 f. il.

Orientador: Marco Túlio de Oliveira Valente.
Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f.97-111
.
1. Computação – Teses. 2. Software – Desenvolvedores – Teses. 3. Mineração de repositórios de software – Teses. 4. GitHub – Teses I. Valente, Marco Túlio de Oliveira . II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III.Título.

CDU 519.6*32(043)

Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa
CRB 6ª Região nº 1510

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Mining the Technical Skills of Open Source Developers

JOÃO EDUARDO MONTANDON DE ARAUJO FILHO

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. MARCO TÚLIO DE OLIVEIRA VALENTE - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ANDRÉ CAVALCANTE HORA
Departamento de Ciência da Computação - UFMG

PROF. EDUARDO MAGNO LAGES FIGUEIREDO
Departamento de Ciência da Computação - UFMG

PROF. MARCELO DE ALMEIDA MAIA
Faculdade de Computação - UFU

PROF. LEONARDO GRESTA PAULINO MURTA
Instituto de Computação - UFF

Belo Horizonte, 5 de Fevereiro de 2021.

À todos que, de alguma forma, me inspiram a ser mais do que sou.

To those who inspire me to be a better version of myself.

Agradecimentos

Este manuscrito é o resultado de um período de intensa dedicação, profundo aprendizado, e muita luta. O caminho rumo ao desconhecido por algumas vezes se mostra adverso, incerto, e solitário. O que aparentava ser uma longa e desgastante jornada se revela ao final um caminho de realizações, conquistas, e superação. Nada disso seria possível sem o apoio incondicional daqueles que, realmente, nos querem bem.

Em primeiro lugar agradeço a Fé em Deus por ser a força motriz que me manteve sempre em movimento.

A minha querida Míriam Cristina, agradeço por ser a luz a me guiar em todos os momentos. Seu brilho iluminou meu caminho nos momentos de maior dúvida e insegurança.

Agradeço também às minhas duas famílias pelo constante apoio nesta caminhada. Aos meus pais, Marilene e João, obrigado pela compreensão em momentos difíceis; ao meu irmão Pedro, agradeço as inúmeras conversas e reflexões que me trouxeram grande sabedoria. A Gilberto, Isabel, Ludmila e Thales, agradeço pelos ótimos momentos que passamos juntos. Todos vocês foram minha fonte revigorante neste trajeto.

Meu muito obrigado ao Colégio Técnico de Minas Gerais (COLTEC) e a seus profissionais por estarem ao meu lado desde o início desta caminhada, propiciando o ambiente para a realização desta qualificação. Em especial, agradeço ao Núcleo de Tecnologia da Informação (NTI) pelo sacrifício realizado durante minha ausência.

Aos integrantes do aSERG, fica meu agradecimento pela inestimável companhia do dia-a-dia, pelas discussões que certamente contribuíram para o trabalho, e pelos encontros semanais que realizamos neste período. Em especial, registro meu agradecimento a profa. Luciana Silva pela parceria nos trabalhos que realizamos em conjunto.

Agradeço também aos profs. Yann-Gaël Guéhéneuc e Fábio Petrillo, e ao Cristiano Politowski por me receberem de braços abertos na Universidade de Concordia. Apesar do curto período, as discussões e convivência que tivemos seguramente contribuíram para meu desenvolvimento como pesquisador.

Por fim, gostaria de expressar a minha mais sincera gratidão ao prof. Marco Túlio Valente que, durante estes 10 anos, me guiou, aconselhou, e me ensinou sábias lições com as quais levarei comigo em minha vida e carreira.

Acknowledgments

This manuscript represents a period of intense dedication, learning, and struggle. Walking into the unknown sometimes is to face the adverse, the uncertain, and the loneliness. The path that appeared to be long and exhausting turned out to be a journey of pleasant surprises and personal achievements. None of this would be possible without the unconditional support of those who wish me the best.

First and foremost, I thank my faith in God for being a major driving force that continuously kept me moving.

To my beloved Míriam Cristina, thanks for being the guiding light at every moment in this journey. Your brightness enlightened my path at the most crucial moments.

I also acknowledge my both families for the never-ending support in this period. To my parents, Marilene and João, thanks for supporting me at the difficult times; to my brother, Pedro, I appreciate our discussions and the wisdom they brought to me. To Gilberto, Isabel, Ludmila, and Thales, thank you for providing such refreshing moments.

Many thanks to the Colégio Técnico de Minas Gerais (COLTEC) and its professionals for supporting me since the beginning. A special thanks to my colleagues at NTI who willingly took my place during my absence.

To my fellows at the Applied Software Engineering Group (aSERG), your day-by-day company meant a lot to me: thank you very much! Particularly, I thank professor Luciana Silva for the partnership we established during this research.

A special thanks to professors Yann-Gaël Guéhéneuc and Fábio Petrillo, and to

Cristiano Politowski for the warmed welcome at the Concordia University. I'm grateful for the fruitful discussions we had, and the opportunities we worked together, which certainly contributed to my growth as a researcher.

Finally, I would like to express my most sincere gratitude to my mentor professor Marco Túlio Valente, for wisely guiding, advising, and teaching me insightful lessons during these 10 years. I will surely take them with me in my life and career.

"It is better to be lucky. But I would rather be exact.
Then when luck comes, you are ready."
(Ernest Hemingway)

Resumo

Atualmente, software está "devorando o mundo" a medida em que surgem novas empresas nas quais o modelo de negócios é totalmente centralizado em um sistema computacional. O sucesso da implantação de tais sistemas depende, em grande medida, da qualidade e competência dos desenvolvedores responsáveis por sua implementação. Em virtude disso, empresas de TI tem empregado um esforço contínuo na contratação de novos profissionais para atuar em seus projetos. Em paralelo, o crescimento de comunidades digitais de desenvolvimento—tais como GitHub e Stack Overflow—tem contribuído com o crescimento de uma nova geração de desenvolvedores. Essas plataformas disponibilizam publicamente informações de seus usuários, frequentemente utilizadas por recrutadores durante a busca de novos talentos. Todavia, o volume e formato dos dados limita esta análise apenas a informações básicas e superficiais dos desenvolvedores. Neste contexto, propõe-se nesta tese uma ampla investigação dos métodos para identificar habilidades técnicas de desenvolvedores de software. Esta pesquisa está organizada em três grandes trabalhos. O primeiro investiga as habilidades técnicas e comportamentais mais demandadas dos desenvolvedores na visão das empresas de TI. Em seguida, analisa-se a efetividade das abordagens orientadas a dados na identificação das habilidades técnicas dos desenvolvedores em duas perspectivas: (a) profundidade, usando técnicas supervisionadas e não-supervisionadas para determinar o nível de conhecimento de desenvolvedores em bibliotecas de software; e (b) largura, aplicando métodos supervisionados para detectar a proficiência de desenvolvedores em seis funções de trabalho. A pesquisa obteve resultados promissores ao adotar um método de clusterização na classificação do nível de conhecimento dos desenvolvedores; identificaram-se grupos nos quais a concentração de desenvolvedores especialistas variou entre 65% e 75%. Em relação às funções de trabalho, o modelo proposto reportou resultados com eficácia entre 69% (revocação) e 89% (AUC).

Palavras-chave: Mineração de Repositórios de Software, Desenvolvedores de Software, Habilidades Técnicas, GitHub.

Abstract

Software is "eating the world" as we witness the rise of companies whose business model is totally centered on software. The successful implementation of these systems heavily depends on the quality and expertise of their software development teams. Indeed, IT-based companies are making an increasing effort to hire new professionals to fulfill their open positions. At the same time, the emergence of Social Coding Platforms (SCPs)—e.g., GitHub, Stack Overflow, etc—is contributing to nurturing a new generation of software developers. On one hand, these platforms favor technical recruiters by providing interesting information about software developers when prospecting a new workforce to their companies. On the other, the large volume of data available limits recruiters to only assess superficial information of their candidates. In order to contribute to this problem, we describe in this thesis an extensive investigation of methods and techniques that effectively identify the technical skills of software developers based on their activity in SCPs. We organize the research in three major working units. We start by studying in more detail the most demanded technical and soft skills of software developers under the eyes of IT companies. Next, we analyze the effectiveness of data-driven methods to assess developers' technical skills from two perspectives: (a) deep, where we evaluate supervised and unsupervised methods to identify the expertise level of software developers in three popular JavaScript libraries; and (b) broad, where we apply supervised methods to detect developers' proficiency in six widely mentioned technical roles. Overall, we obtained promising results in using an unsupervised technique to classify developers' expertise level. For example, we were able to produce clusters where the number of experts ranges from 65% to 75%. With respect to technical roles, the proposed model reported higher outcomes for precision (88%) and AUC (89%).

Palavras-chave: Mining Software Repositories, Software Developers Expertise, Technical Skills, GitHub.

List of Figures

1.1 T-shaped professional skill model. 4
1.2 Jobs opportunities list 7

2.1 Software development expertise conceptual framework, extracted from Baltes and Diehl [2018]. 12
2.2 Conceptual diagram summarizing the contribution of each Background section to the study of software developers' expertise. 24

3.1 An example of a post in Stack Overflow Jobs portal. Hard skills are annotated in blue, while soft skills are in yellow. 29
3.2 Distribution of job posts collected in this study. 30
3.3 Heatmap indicating how hard skills are distributed among developers roles. 31
3.4 Soft skills word cloud. 33

4.1 Survey answers 45
4.2 Correlation analysis; red cells are discarded due to high correlation. 46
4.3 Distributions of *commitsClientFiles* values for each cluster/library. Cluster 1 (experts) has higher values than other clusters, except for SOCKET.IO. . . 54
4.4 Distributions of *projects* values for SOCKET.IO clusters. Cluster 1 (experts) has higher values than other clusters. 55
4.5 Years of experience on REACT of developers predicted as experts 58
4.6 Percentage of REACT experts by quintiles of feature distributions. For most features, there is an important proportion of experts in lower quintiles. . . 59

5.1 Survey responses from the recruiters that use GitHub in their hiring process. 65
5.2 Example of Stack Overflow developer profile. The data used in the ground truth creation is highlighted. 69

5.3 A developer profile from GitHub. Appart from *Projects' Dependencies*, the extracted data is highlighted according to each category: *Programming Languages* in blue, *Short Bio* in green, *Projects' Names* in purple, *Projects' Topics* in grey, and *Projects' Descriptions* in orange. 71
5.4 Multi-label classification using Classifier Chains. 76
5.5 Most relevant features for each technical role. Colors and shapes represent each feature category: developers bio ((*Bio*), green circle), projects' descriptions ((*desc.*), blue square), projects' names ((*name*), pink diamond), and programming languages ((*rate*, *author*, and *total*), orange triangle) 79
5.6 Overall results using Classifier Chain, reported in relation to the *RQ.1* baseline (i.e., dashed line). 82

List of Tables

3.1 High-level hard skills created after open card sorting 31

4.1 Target Libraries 41
4.2 Client Projects and Candidate Experts 42
4.3 Features collected for each candidate expert in each target library 43
4.4 Survey Numbers 44
4.5 Cluster with the highest percentage of experts (values in bold define the selected number of clusters) 49
4.6 Machine learning results for 5 classes (FACEBOOK/REACT) 50
4.7 Results for 3 classes: novice (scores 1–2), intermediate (score 3), and expert (scores 4–5) 51
4.8 Clustering results (cluster 1 has the highest % of experts) 52
4.9 Comparing feature distributions using Cliff's delta: Experts vs Cluster with the closest median ($\circ$ means similar distributions, according to Mann-Whitney, p-value$= 0.05$) 53

5.1 Regular expressions used to identify technical roles. 69
5.2 Distribution of developers among the analyzed roles. 70
5.3 Dataset with five target labels. 74
5.4 Binary Relevance overall results. 77
5.5 Binary Relevance results for each role. 78
5.6 Binary Relevance results after including FULLSTACK, Random Forest only. 83

Contents

Agradecimentos **xiii**

Acknowledgments **xv**

Resumo **xix**

Abstract **xxi**

List of Figures **xxiii**

List of Tables **xxv**

1 Introduction **1**

1.1 Problem and Motivation 1

1.1.1 The Literature Perspective 3

1.1.2 The Industry Perspective 4

1.2 Goals and Contributions 5

1.2.1 Analyzing the Skills Profile Required by IT Companies 5

1.2.2 Library Expertise Assessment 6

1.2.3 Mining Developers' Technical Roles 7

1.3 Publications 8

1.4 Thesis Outline 9

2 Background **11**

2.1 Expertise in Software Development 11

2.1.1 Domain Expertise 12

2.1.2 General Expertise 14

2.2 Mining Software Repositories 16

2.3 Machine Learning in Software Engineering 17

2.4 Related Work 18

2.4.1 Technical Expertise with Low-Level Features 19
2.4.2 Technical Expertise with High-Level Features 21
2.4.3 Technical Expertise Towards Specific Tasks 23
2.5 Final Remarks 24

3 What Skills do IT Companies look for in New Developers? A Study with Stack Overflow Jobs 27
3.1 Introduction 27
3.2 The Anatomy of a Job Opportunity 28
3.3 Data Collection 28
3.4 Analyzing Hard Skills 30
3.5 Analyzing Soft Skills 32
3.6 How do Skills Change Over Time? 33
3.7 Threats to Validity 34
3.8 Final Remarks 34

4 Identifying Experts in Software Libraries and Frameworks among GitHub Users 37
4.1 Introduction 37
4.2 Data Collection 40
4.2.1 Definitions 40
4.2.2 Target Libraries 40
4.2.3 Candidate Experts 41
4.2.4 Features 42
4.2.5 Ground Truth 44
4.2.6 Final Processing Steps 45
4.3 Methods 47
4.3.1 Machine Learning Setup and Algorithms 47
4.3.2 Clustering Setup and Algorithm 48
4.4 Results 49
4.5 Discussion and Practical Usage 56
4.5.1 Relevance and Key Findings 56
4.5.2 Practical Usage 57
4.5.3 Triangulation with Linkedin Profiles 57
4.5.4 Limitations 58
4.6 Threats to Validity 60
4.7 Final Remarks 61

5 Mining the Technical Roles of GitHub Users **63**
5.1 Introduction . . . 63
5.2 Study Design . . . 67
5.2.1 Technical Roles . . . 67
5.2.2 Ground Truth . . . 68
5.2.3 Data Collection . . . 70
5.2.4 Selected Features . . . 72
5.2.5 Machine Learning Setup . . . 74
5.3 Results . . . 77
5.4 Understanding the FullStack Role . . . 82
5.5 Discussion . . . 84
5.5.1 Implications . . . 84
5.5.2 A Note on Precision and Recall . . . 85
5.5.3 Manual Analysis . . . 85
5.6 Threats to Validity . . . 88
5.7 Final Remarks . . . 89

6 Conclusion **91**
6.1 Thesis Recapitulation . . . 91
6.2 Contributions . . . 92
6.3 Future Work . . . 93

Bibliography **97**

A Library Expertise Survey **113**

Chapter 1

Introduction

We start this chapter by introducing the problem that motivates the research reported in this thesis (Section 1.1). Then, we highlight both the major goals and contributions we obtained around this work (Section 1.2). Finally, we present the list of publications we achieved so far (Section 1.3), as well as the outline of the remaining of this thesis (Section 1.4).

1.1 Problem and Motivation

Software development is a human-centric activity, which makes developers the most important asset of software companies [DeMarco and Lister, 1999]. Moreover, after 25 years of the invention of modern Internet technologies, software is "eating the world" and we everyday observe the rise of companies totally centered on software. Among the examples, we can mention companies that recently relied on software to disrupt traditional markets. For instance, the world's largest taxi company (Uber) owns no taxies, but an innovative software. The same is happening with hotels (due to AirBnB), telecommunication companies (due to Skype, Whatsapp, and Zoom), and media companies (due to Google and Facebook).

During this period, software systems have become more complex artifacts, which increasingly demands new levels of specialization of their development teams. For example, current software teams include experts in different areas, such as databases, security, human-computer interface (or front-end design), core features (or back-end design), mobile development, etc. Besides, software companies require their developers to master several specific technologies so they can perform their daily work. Accordingly to the Bureau of Labor Statistics[1]—a US government agency that provides

[1] `https://www.bls.gov/ooh/computer-and-information-technology/software-`

statistics about the labor market—the "employment of software developers is projected to grow 24% from 2016 to 2026, much faster than the average for all occupations". As a consequence, we are observing a worldwide shortage of skilled software engineers.

Consequently, IT-based companies are giving high importance when it comes to hiring new professionals. For instance, Mark Zuckerberg has publicly stated that "our [Facebook's] policy is literally to hire as many talented engineers as we can find".[2] As a second example, Valve Corporation—a well-known video game developer company—have even highlighted the importance of hiring people in an internal handbook distributed to newcomers:

> *Hiring well is the most important thing in the universe. Everything else in our world is subordinate to finding great people and keeping the bar high.* (Valve Corporation [2012])

At the same time, we have noticed the rise of global repositories and communities of open source developers, known as Social Coding Platforms (SCPs). For example, GitHub—the world's largest collection of open source projects—contains approximately 56 million users and more than 60 million repositories were created in the platform in the last year.[3] Similarly, Stack Overflow—the most accessed Q&A platform for software developers—has almost 14 million users responsible for producing more than 51 million questions and answers.[4] Such platforms are contributing to nurture and train a new generation of software developers, who have the opportunity to participate in the development of a variety of popular and complex software systems [Eghbal, 2016; Dabbish et al., 2012; Begel et al., 2013].

Therefore, it is not a surprise that software companies are using SCPs to prospect potential candidates to fulfill their open positions. As mentioned by Marlow and Dabbish [2013], employers believe that *GitHub profiles can provide relevant insights into developers' technical abilities and personal qualities in a more reliable way than personal resumes.* On one hand, employers can use GitHub to access not only developers' code but also their development history, online presence, social interactions, community engagement, etc. Furthermore, the amount of information available in these platforms allow technical recruiters to select candidates that fit their specific needs. On the other hand, employers usually limited themselves to analyze easily verifiable signals—e.g., the number of projects owned, the programming languages used, etc—since an in-

`developers.htm`, accessed in April 2018

[2] `https://code.org/quotes`, accessed on November 2020

[3] `https://octoverse.github.com/2020`, accessed on November 2020.

[4] `https://data.stackexchange.com/`, accessed on November 2020.

depth analysis of GitHub profiles requires significant manual effort. As a consequence, employers do not take full advantage of the data provided by these platforms.

1.1.1 The Literature Perspective

We observe an increasing number of studies relying on data provided by SCPs to conduct empirical research and large-scale analysis in the software engineering field [Cosentino et al., 2017]. These papers cover a range of research topics in software development, including software maintenance and evolution issues [Brito et al., 2018; Uddin et al., 2020; Hora, 2021], source code documentation [Menezes et al., 2018; Campos et al., 2016; Souza et al., 2019], and open-source projects' characterization [Silva and Valente, 2018; Politowski et al., 2021; Treude and Robillard, 2017; Coelho and Valente, 2017].

When it comes to the state-of-the-art concerning expertise in software development, the existing literature unveils developers' abilities in several scenarios. Some studies focus on discerning developers' abilities according to a particular domain they are involved. Da Silva et al. [2015] and Honsel et al. [2016] proposed to identify experts for source code elements—such as methods, classes, and packages—of specific projects. Constantinou and Kapitsaki [2017] overcame with a model to detect the contribution roles of developers in open source projects, e.g., bug fixer, triager, core developer, etc. Avelino et al. [2019a] used the truck-factor metric to identify core developers—those who have more knowledge of the system's structure—in popular projects. The approaches we just described are intrinsically limited to the scope of the problem defined by them, i.e., they are not centered on the general expertise of developers.

Other works aimed at leveraging developers' skills independently of the context they are working on. Teyton et al. [2013] performed a syntactical analysis over developers' commits to measure their expertise in third-party libraries. Likewise, Saxena and Pedanekar [2017] instrumented the profile of GitHub users by annotating them with Stack Overflow tags. In both cases, relying on abstract syntax trees to obtain expertise information may reduce the performance of the solution in a larger dataset. Oliveira et al. [2019] relied on more simplified information—activity-based features such as the number of commits—to detect experts in Java libraries, but restricted the expertise granularity level and the universe of developers considered in their analysis.

We also identify studies that reveal general abilities of software developers—e.g., programming languages, operating systems, IDEs, etc—but they do so having specific situations in mind. For instance, Hauff and Gousios [2015] mined GitHub profiles and job advertisements to match each other automatically. Mao et al. [2015] and Wang

et al. [2017] aimed at pointing the most suitable developers for performing tasks in crowdsourcing platforms.

Notwithstanding the acknowledged effort in identifying the abilities of software developers, the aforementioned research papers do not completely meet the demands reported by industry players., i.e., automatically leverage skills profiles that—based on developers' general activity in SCPs like GitHub—can assist recruiters and employers during the hiring process. Therefore, *in this thesis we investigate methods and techniques that can better recognize the technical skills of software developers based on their activity in open source projects.*

1.1.2 The Industry Perspective

The ideal large-scale hiring process should enable technical recruiters to visualize a large number of developers' profiles containing different levels of technical expertise, such as their principal hard skills[5] and professional roles. Put differently, companies are interested in candidates who have deep knowledge in their main area but also have a broad understanding of the software development cycle as a whole.

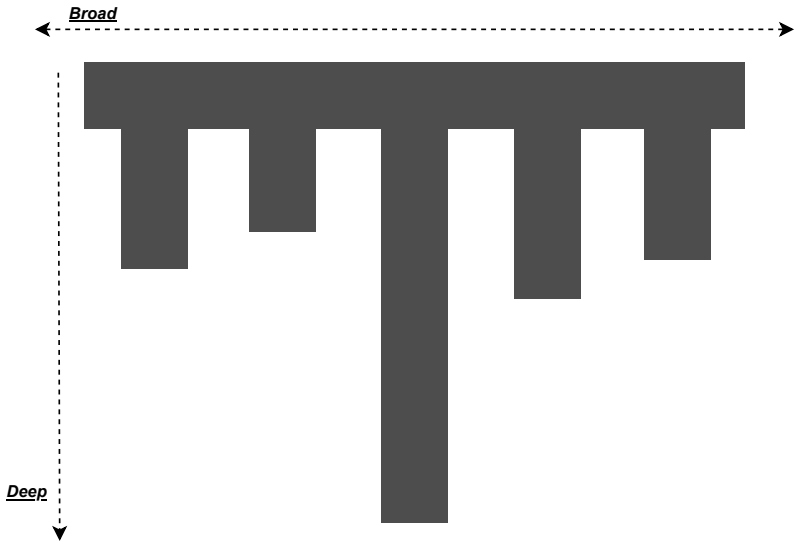


Figure 1.1: T-shaped professional skill model.

We visually depicted the relationship between these two axes in Figure 1.1. People carrying such a profile are known as T-shaped professionals, and pick up the characteristics of both generalists (broad) and specialists (deep). These professionals are highly-valued due to their capacity of solving problems in a multidisciplinary environment [IfM and IBM, 2007]. In the context of software development, professionals fitting

[5]In this thesis, the terms *hard skills* and *technical skills* are used interchangeably.

this profile are capable of interacting with other fields in order to build an innovative product [Valve Corporation, 2012]. Thereby, *in this thesis we shed light on the abilities that IT companies require when hiring new collaborators, as well as study techniques to identify developers' technical skills in both broad and deep perspectives.*

1.2 Goals and Contributions

As early stated, the existing methods for assessing software developers' expertise are project-oriented, i.e., they aim at identifying experts to work at specific parts of a given project [Fritz et al., 2014; Bhattacharya et al., 2014; Da Silva et al., 2015; Constantinou and Kapitsaki, 2017]. The ones that propose a domain-independent developers' skills analysis contain restrictions either regarding the performance or the scope of their evaluation [Teyton et al., 2013; Saxena and Pedanekar, 2017; Oliveira et al., 2019]. On the other hand, technical recruiters are more interested in skills profiles that can reveal developers' abilities in both broad and deep perspectives [IfM and IBM, 2007; Valve Corporation, 2012]. Hence, our main goal in this thesis is described as follows:

> We aim at providing a comprehensive investigation around methods and techniques to effectively identify the technical skills of software developers given their activity in Social Coding Platforms.

To make this research possible, we conducted three major studies. First, we study in more detail the side view of the skills required by IT companies when they are looking for new professionals. In the second study, we investigate the use of data-driven methods to mine developers' expertise in a deep perspective. More specifically, we assess the expertise level of software developers in third-party libraries. The third study also relied on data-driven techniques but to evaluate the expertise of software developers in a broad perspective, where we proposed to automatically identify their technical roles. We summarize each study and highlight their contributions in the remainder of this section.

1.2.1 Analyzing the Skills Profile Required by IT Companies

Due to the increasing complexity of software development, IT companies are continuously in demand for more professionals to work on their projects. Despite this fact, we still lack information on how these companies deal with this issue when looking for developers to fulfill their open positions. Therefore, in this thesis we initially report the

results of a quantitative study designed to reveal the skills required by IT companies when selecting new employees. In this study, we present the following contributions:

- We performed a large-scale analysis of 20,000 job opportunities available in Stack Overflow Jobs portal, a platform that allows companies to publish new opportunities for IT professionals.

- We leveraged the characteristics of 14 IT professional roles—e.g., Backend, Frontend, Mobile, etc—and revealed which kind of technical skills are demanded in each one. For example, we observed that programming languages are largely required even in management-based positions. Furthermore, experience in third-party components—i.e., libraries and frameworks—is frequently mentioned in developer-based ones.

- We revealed which soft skills are mostly required by IT companies when selecting new professionals. Particularly, we reinforced the importance of communication, collaboration, and problem-solving skills to software developers.

1.2.2 Library Expertise Assessment

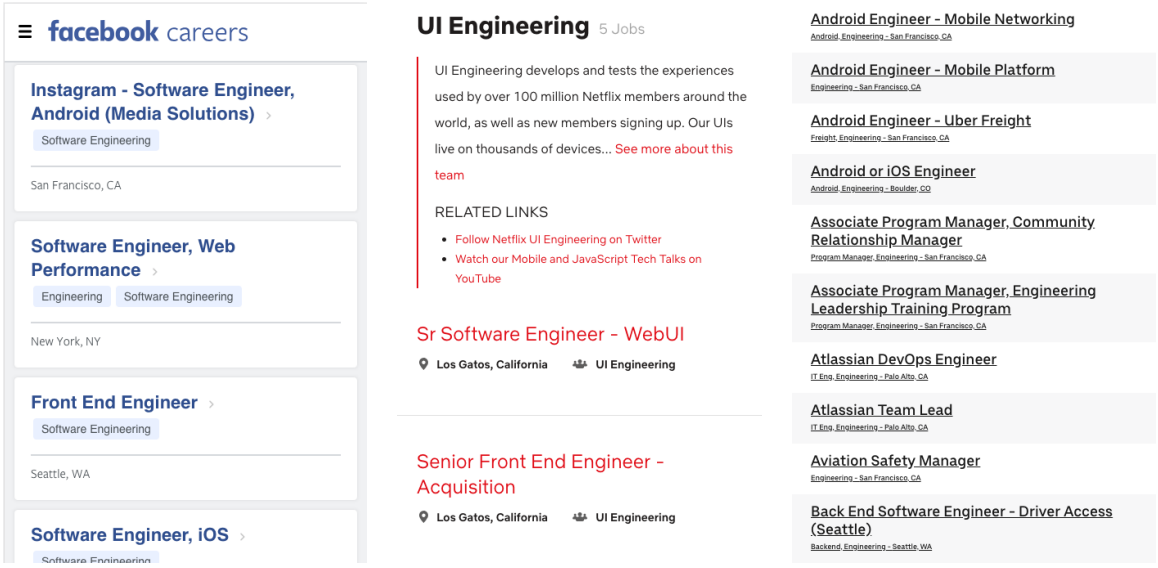
Modern software development heavily depends on libraries and frameworks to increase productivity and reduce time-to-market [Ruiz et al., 2014; Sawant and Bacchelli, 2017]. In this context, identifying experts on popular libraries and frameworks has key importance. For example, open-source project managers can use this information to select contributors to their systems. Private companies can also consider this information before hiring developers for their projects. Therefore, in this second study, we evaluate the use of machine learning and data-mining methods in order to identify developers' expertise level on popular libraries and frameworks. Our analysis was based in the following hypothesis: *When maintaining a piece of code, developers also gain expertise on the frameworks and libraries used by its implementation.* In other words, we claim that low-level data—based on source code—can help such methods in this detection task. We emphasize the following contributions of this work:

- We surveyed 575 developers and collected their expertise level on three largely used JavaScript libraries: FACEBOOK/REACT (for building enriched Web interfaces), MONGODB/NODE-MONGODB (for accessing MongoDB databases), and SOCKETIO/SOCKET.IO (for real-time communication). We then build a public dataset with the activity-based information collected from their GitHub profiles.

- We evaluated the performance of two supervised machine learning algorithms to predict developers' expertise level. We documented the difficulties and challenges of using this method to classify expertise levels on widely popular third-party components.

- We propose an unsupervised method to identify library experts based on clustering methods, this time achieving promising results. For example, we were able to produce clusters where the number of experts ranges from 65% to 75%. We also triangulated our results with information available on LinkedIn. Indeed, 72% of the experts in FACEBOOK/REACT explicitly cite this framework in their LinkedIn profiles.

1.2.3 Mining Developers' Technical Roles



(a) Facebook (b) Netflix (c) Uber

Figure 1.2: Jobs opportunities list

As mentioned in Section 1.1, software development teams include experts in different areas, e.g., back-end, front-end, mobile development, etc. This composition is a consequence of how current software systems are implemented; indeed, most companies rely on these roles when hiring new developers. For example, Figure 1.2 depicts the jobs opportunities list for three widely known software companies: Facebook[6], Uber[7], and

[6] `https://www.facebook.com/careers/jobs`, accessed on November, 2018
[7] `https://www.uber.com/careers`, accessed on November 2018

Netflix[8]. As we can observe, software engineering jobs are commonly labeled according to major expertise areas, such as mobile development, machine learning engineering, front-end development, back-end engineering, etc. This is also observed for other software companies. Therefore, in this third working unit, we study three machine learning methods to identify the technical roles of software developers. To perform this investigation, we assume the following hypothesis: *the technologies that developers master define their technical roles.* This time we rely on high-level data—e.g., textual information, GitHub metadata, etc—as we understand they represent a better source to capture the information of such technologies. This work has contributed to the following:

- By relying on the information provided by developers on Stack Overflow, we leveraged a ground truth containing the technical roles of 2,284 users who also have a GitHub profile.
- We evaluated the effectiveness of three machine learning strategies to classify the qualification of software developers in the following technical roles: *Backend*, *Frontend*, *FullStack*, *Mobile*, *DataScience*, and *DevOps*. These models presented competitive results with respect to precision (0.88) and AUC (0.89) when identifying all six roles. We also show the relevance of programming languages when predicting the aforementioned roles.
- We performed a qualitative investigation on some developers' profiles to understand in which scenarios these classifiers make correct and incorrect predictions. We analyzed these errors and discussed their impact on the performance of our classifiers. For instance, we found out that a developer correctly classified as *mobile* frequently indicate the use of mobile-specific technologies, such as the presence of *Swift* source code. On the other hand, the misclassified developers we analyzed did not present role-specific information enough; e.g., a *DevOps* developer not using devops-specific technologies.

1.3 Publications

This thesis encompasses the contributions contained in the following publications:

MSR '19 João Eduardo Montandon, Luciana L. Silva, Marco Tulio Valente. *Identifying Experts in Software Libraries and Frameworks among GitHub Users.* In 16th

[8] `https://jobs.netflix.com/teams/engineering`, accessed on November, 2018

International Conference on Mining Software Repositories (MSR), pages 276-287, 2019. (**Chapter 4**).

IST '21 João Eduardo Montandon, Cristiano Politowski, Luciana Lourdes Silva, Marco Tulio Valente, Fabio Petrillo, Yann-Gaël Guéhéneuc. *What Skills do IT Companies look for in New Developers? A Study with Stack Overflow Jobs.* Information and Software Technology, vol. 129, pages 1-6, 2021. (**Chapter 3**).

IST '21 João Eduardo Montandon, Marco Tulio Valente, Luciana Lourdes Silva. *Mining the Technical Roles of GitHub Users.* Information and Software Technology, vol. 131, pages 1-19, 2021. (**Chapter 5**).

Furthermore, we also contributed to the following work during this Ph.D. research:

JSS '21 Cristiano Politowski, Fabio Petrillo, João Eduardo Montandon, Marco Tulio Valente, Yann-Gaël Guéhéneuc. *Are Game Engines Software Frameworks? A Three-perspective Study.* Journal of Systems and Software, vol. 171, pages 1-22, 2021.

1.4 Thesis Outline

We organize this thesis as the following:

Chapter 2 covers in detail background information to support this thesis. Basically, we describe the theory about expertise in software development along with state-of-the-art research regarding the use of machine learning methods in the context of mining software repositories.

Chapter 3 describes our motivational study where we analyze which hard and soft skills are more required by IT companies. We report which types of hard skills are more requested for 14 predefined IT roles, as well as the most mentioned soft skills, overall.

Chapter 4 describes our first study where we evaluated the performance of machine learning methods in identifying developers who are experts in specific technologies, i.e., libraries and frameworks. By relying on low-level activity data—such as the number of changed lines, the number of commits, and the time interval between them—we apply both unsupervised and supervised models to predict

the expertise level on three popular JavaScript libraries. We document the challenges of using such techniques and propose a novel method to classify library experts based on clustering analysis.

Chapter 5 presents a second study we have done to investigate the use of three machine learning strategies to detect the technical roles of open source developers. This time, we rely on more coarse-grained information—e.g., source code projects' dependencies, commits by each programming language—to classify developers into six popular technical roles. We report in this chapter the effectiveness of applying such models to reveal these technical roles, as well as discuss which data points are the most relevant to identify the aforementioned roles.

Chapter 6 summarizes the conclusions we leveraged throughout this thesis and outlines some ideas we find interesting to investigate in the future.

Chapter 2

Background

We start this chapter by defining expertise in the context of software development (Section 2.1). Then, we discuss in Section 2.2 the most common techniques used to mine software repositories. Similarly, we also listed some papers that applied some Machine Learning methods to analyze the information mined from these repositories (Section 2.3). In Section 2.4, we present the works that are directly related to this thesis. Finally, we report our final remarks in Section 2.5.

2.1 Expertise in Software Development

Accordingly to Ericsson [2012], **expertise** "refers to the characteristics, skills, and knowledge that distinguish experts from novices and less experienced people". Someone is considered an expert when he/she is "widely recognized as a reliable source of knowledge, technique, or skill whose judgment is accorded authority and status by the public or his or her peers." Experts stand out for presenting superior performance when doing representative tasks in their field. Sometimes, this performance can be mapped into objective criteria. For instance, it is quite simple to determine if someone is an expert at running; we just need to analyze his running time. On the other hand, it becomes difficult to identify experts in other fields as the criteria become more subjective. We observe such characteristics when analyzing software developers since their performance can be assessed in several ways.

To provide more concrete guidelines on how to measure software developers' expertise, Baltes and Diehl [2018] leveraged a conceptual expertise framework specifically for software development, illustrated in Figure 2.1. The framework was built using a mixed-methods approach, where the authors surveyed software developers about which characteristics do experts have and studied the existing literature on expertise and ex-

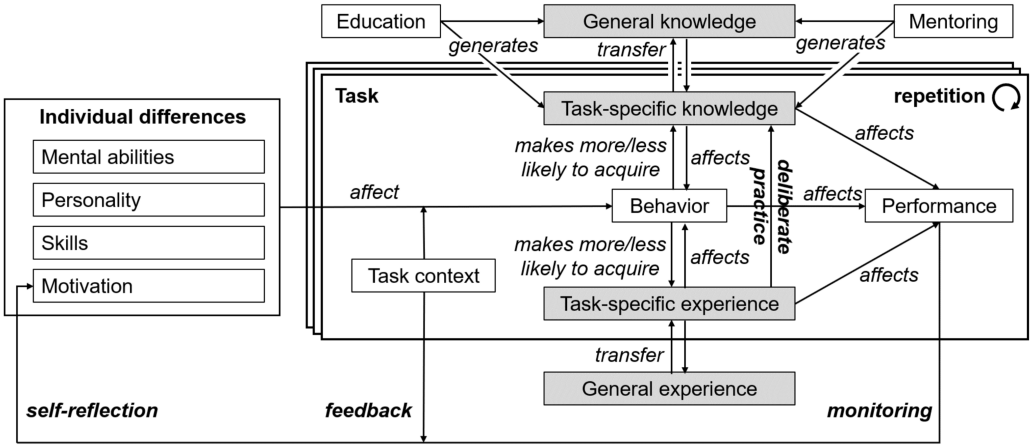


Figure 2.1: Software development expertise conceptual framework, extracted from Baltes and Diehl [2018].

pert performance. We immediately notice that knowledge and experience play a central role when determining software developers' expertise in this framework. In fact, we intuitively associate developers' expertise with a certain degree of knowledge together with a certain amount of experience [Siegmund et al., 2014]. Knowledge and experience are improved after completing a sequence of tasks. First, both are acquired at a more specific level, related to a given task, e.g., a developer learns how to use a new framework when working on a given task. Over time, this gain is transferred to a broader and general level, e.g., the developer has mastered the framework and now can use it in other tasks. As a result, this process directly influences the performance of software developers, i.e., how they write their code.

In the following sections, we describe expertise in two major perspectives. Section 2.1.1 discusses the existing work regarding *domain expertise*, i.e., expertise in the context of specific projects. Section 2.1.2 presents some studies that investigate developers' *general expertise*, by analyzing their behavior in distinct environments.

2.1.1 Domain Expertise

Works in this category focus on researching developers' ability to perform specific tasks. Most of them aim at improving the allocation of software developers to perform maintenance tasks in a particular context, i.e., the projects they often contribute to. We identified research centered on topics such as source code authorship [Alonso et al., 2008; Fritz et al., 2010; Da Silva et al., 2015; Avelino et al., 2019b], and identification of organizational and contribution roles [Yu and Ramaswamy, 2007; Bhattacharya et al., 2014].

Alonso et al. [2008] started investigating how Version Control Systems (VCS) can help to identify domain experts. For this, the authors build a rule-based model to identify the core developers of one open source project. This model classifies source code files into a set of categories and then derive developers' expertise based on the frequency of their commits on these files. The resulting output is made available in a word cloud tool, which they used to identify the ones that most contribute to the project. The authors concluded that a few developers are responsible for most project's source code.

Similarly, Kagdi et al. [2008] introduced an approach to recommend a ranked list of developers who likely master the content of a particular file in a given project. To leverage this ranking, the authors rely on a set of heuristics extracted from the commits performed on a CVS-based project, including the number of contributions, the most recent contribution, and the number of days they have contributed to the project. The authors then implemented the resulting model as a tool for the Eclipse IDE and evaluated its performance in 8 projects.

Fritz et al. [2010, 2014] propose a new mixed metric to identify experts in specific source code files. This metric, known as degree-of-knowledge (DOK), was based on the information collected from both commits and real-time interactions between developers and source code files. Recently, Avelino et al. [2019b] partially extended this metric to assess the effectiveness of repository-mining techniques for identifying software maintainers. More specifically, the authors evaluate distinct techniques, including (a) the number of source code file changes; (b) the number of lines owned by a developer in the most updated file; and (c) a linear regression based on DOA metric. After analyzing these techniques against 10 systems, the authors observed better results for the latter approach. In another work, the same authors relied on developers commit activity, where they estimate truck-factors—i.e., a source code authorship metric—against 119 open source systems to identify their core developers [Avelino et al., 2019a].

Da Silva et al. [2015] introduced a refined technique to identify domain expertise at a finer granularity level. The proposed solution has three particularities when compared to other ones. First, the technique provides expertise at several granularity levels, e.g., project, package, file, class, or method. Second, developers' expertise can also be obtained at different time intervals for the same project. Third, the technique was implemented upon a GPU-based solution, allowing them to mine software repository data more efficiently. The authors evaluated their approach by mining expertise information at the Apache Derby project, and find out that granularity and time constraints do impact the identification of domain experts. Honsel et al. [2016] aggregated different sources of data—such as commits, bug fixes, and mailing list activity—to mea-

sure developers' level of involvement in a given project. The authors then used Hidden Markov Models to predict the contribution behavior of the analyzed developers. As a result, they were able to classify developers into three categories: minor, major, and core developers.

Other works investigated domain expertise in the context of organizational roles [Yu and Ramaswamy, 2007; Bhattacharya et al., 2014]. Yu and Ramaswamy [2007] presented a clustering-based model to derive developer roles based on the interaction among them. The interaction frequency was calculated using information from e-mail correspondences, task sharing, co-editing actions, etc. The clustering model showed two distinct groups: *core members*, who scored higher interaction rates; and *associate members*, who reached lower interaction rates. Bhattacharya et al. [2014] studied the use of graph models to classify developers on seven organizational roles: *patch tester*, *assist*, *triager*, *bug analyst*, *core developer*, *bug fixer*, and *patch-quality improver*. For this, the authors first leveraged, for each developer, two distinct expertise profiles: (a) bug-fixing profile, which contains the activity information on bug tracking systems for each developer in each project; and (b) source-code profile, which includes the activity data each developer has for each project source code repository. The roles were then obtained after applying a customized hierarchical graph model over both profiles.

As we can observe, works in this category focus on identifying expertise in software development for specific projects' artifacts, i.e., source code files' owners, developers for specific bug fixing tasks, etc. That is, the expertise approaches proposed in these studies are restricted, by definition, to the scope they previously defined, e.g., source code files, issues, etc. In this thesis, we chose to investigate a method that is more centered on the developers themselves, i.e., more robust to the different contexts they work on.

2.1.2 General Expertise

Differently, the research described in this section aims at studying developers' skills independent from the context they work on. Overall, the literature distinct general expertise into two major groups: **soft skills** and **technical skills**. While the former includes skills, abilities, and traits related to people's personality and behavior, the latter comprises the technical skills needed to accomplish their work in several distinct contexts. This section reports studies that investigated general expertise from a qualitative perspective.

Marlow and Dabbish [2013] interviewed GitHub users to understand how do they use the information available in GitHub profiles to assess people during the recruitment and hiring process for software development positions. In total, they conducted a semi-structured interview with 13 GitHub users; seven recruiters and six job seekers. The authors analyzed the transcript and extracted relevant statements that both groups bring up during the interview. Most recruiters indicated they rely on the users' project and community management activity to infer their traits. Job seekers, on the other hand, highlighted they mostly look for signals indicating developers' technical expertise and their commitment to the open source community. In a second study, the authors interviewed 18 GitHub users to understand how users in these platforms form "first impressions" on their peers [Marlow et al., 2013]. The authors identified three categories used by the interviewees to analyze other profiles: (a) general coding ability, formed after examining users' coding activity, such as number of commits, number of projects owned vs forked, etc; (b) project-relevant skills, formed after probing the type of the contributions, e.g., commits vs issues/comments; and (c) personality and interaction style, formed after analyzing past discussions.

Similarly, Singer et al. [2013] investigated developer profile aggregators—webpages that leverage developer profiles from the available data at Social Coding Platforms (SCP)—to understand the influence these tools have in assessing developers' abilities. Specifically, the authors surveyed 14 developers and 12 recruiters of two popular profile aggregators systems—Coderwall[1] and Masterbranch[2]—about the reasons they use such platforms. For developers, the most important reason for enjoying the community was the recognition of others. Furthermore, they frequently mentioned the possibility to showcase their work as another reason to join this ecosystem. Recruiters, on the other hand, highlighted they mainly use these systems to better assess developers' abilities and find better candidates to hire. For this, they generally rely on some features provided by these platforms, such as endorsements, badges, and skills-based filtering mechanisms (e.g., programming languages, file extensions, etc).

Sarma et al. [2016] relied on the aforementioned studies to create a tool that summarises developers' both soft and technical skills information into one single profile. The tool, called Visual Resume, aggregates data from GitHub and Stack Overflow and quantifies them into five indicators: (a) coding competency and quality work for technical skills; and (b) collaboration, project management, and motivation for soft skills. In total, nine participants with experience in hiring software developers evaluated the proposed tool. The majority of the subjects used commits and projects-

[1] `http://coderwall.com`
[2] `http://masterbranch.com`

based metrics to analyze developers' technical skills, and endorsement-based metrics—e.g., reputation scores, number of followers, etc—to assess developers' soft skills.

We centered our thesis on studying techniques to automatically identify developers' technical skills. Therefore, research that address expertise in a quantitative context are discussed in detail in Section 2.4.

2.2 Mining Software Repositories

Research in the Mining Software Repositories field has intensified in the last decade due to the rising and popularization of Social Coding Platforms like GitHub and Stack Overflow. Indeed, more than 56 million developers have a profile on GitHub according to its 2020 survey; these developers have created approximately 60 million repositories in the last year, resulting in more than 1.9 billion contributions in the same period.[3] Likewise, Stack Overflow has become the most popular Q&A platform for software developers. At the time we write this thesis, almost 14 million users were responsible for providing 31 million answers to more than 20 million questions.[4] Indeed, the amount of data provided by these platforms made it possible to conduct empirical research on the most varied topics in software engineering.

Some of the aforementioned works explored such data to investigate API maintenance and evolution issues. For instance, Brito et al. [2018] relied on GitHub to gather information about the usage of Java libraries and frameworks, and used this information to study the motivations behind APIs' breaking changes. Later, the authors expanded their initial study and analyzed the information provided by Stack Overflow as well [Brito et al., 2020]. Similarly, Hora [2021] also mined the source code of GitHub projects, this time looking for examples of API usage. In order to generate APIs documentation automatically, Uddin et al. [2020] identified API usage examples in Stack Overflow posts and then used the textual description present in these posts to leverage APIs usage scenarios. Campos et al. [2016] make use of the existing "crowd knowledge" in these platforms to recommend context-sensitive information to software developers during their activities. Later, the authors based on this work to build a solution that semi-automatically creates APIs cookbooks from the existing "crowd knowledge" on Stack Overflow [Souza et al., 2019].

Other works studied more specific maintenance issues. For instance, Menezes et al. [2018] analyzed the merge conflicts present in almost 3K Java projects and uncovered some strategies to assist in merge-conflict resolution. Pimentel et al. [2019]

[3] `https://octoverse.github.com/`. Last access: Dec. 02, 2020.
[4] `https://data.stackexchange.com/`. Last access: Dec. 02, 2020.

analyzed 1.4M Jupyter notebooks from GitHub and reported the main factors that impact their reproducibility. Silva et al. [2016] monitored Java projects hosted on GitHub and surveyed developers who applied refactorings on these projects to understand why they do so. Such projects were later used as an oracle to evaluate automated refactoring tools [Tsantalis et al., 2018; Silva et al., 2020].

Finally, we also identified research focused on studying other aspects in both GitHub and Stack Overflow ecosystems. Politowski et al. [2021] relied on the GitHub ecosystem to describe the differences between video game engines and software frameworks. Silva and Valente [2018] investigated the reasons that make open source projects popular on GitHub and revealed this popularity in four different scenarios. In another work, the authors conducted an exploratory study to understand the usage and impact of GitHub reactions [Borges et al., 2019]. Zhou et al. [2020] reported their findings of adopting hard forks in GitHub projects. Earlier, Vasilescu et al. [2014] studied the usage of continuous integration techniques in this ecosystem. Treude and Robillard [2017] performed an exploratory survey with 321 users to unveil the information that developers need to understand Stack Overflow code fragments. Thiselton and Treude [2019] implemented a plugin that extends error messages from the Python Compiler with answers from Stack Overflow.

As in the studies we just described, this thesis centered its investigation in the context of Social Coding Platforms, i.e., we studied methods and techniques to mine developers' expertise based on the data available in the GitHub ecosystem (see Chapters 4 and 5). We also used Stack Overflow as a complementary data source for the research we accomplished in Chapters 3 and 5.

2.3 Machine Learning in Software Engineering

Machine Learning (ML) refers to the process of applying statistical modeling techniques to reveal patterns in a large amount of data [Foster Provost, 2015; Amershi et al., 2019]. Recently, the volume of data provided by Social Coding Platforms allowed software engineering researchers to make use of such techniques in their work. Since then, an increasing number of studies relied on ML techniques to investigate various topics of the software development lifecycle [Ferreira et al., 2019; Shafiq et al., 2020].

Many studies use these models to tackle quality assurance issues, like bug prediction, test case prioritization, vulnerability analysis, etc. For example, Machalica et al. [2019] developed a predictive model to select tests to be executed after changes are committed in a continuous integration system. To train this classifier, they have

had access to a large dataset containing changes and their associated tests. Peters et al. [2013] proposed a customized metric that helps to collect relevant data in other systems in a way that users can build cross-company defect prediction models. Abebe et al. [2012] measured the quality of source code identifiers and verified whether this information can improve traditional fault prediction strategies. Both studies relied on the data provided by open source repositories to evaluate their approach.

Others applied such techniques to assist developers in software maintenance tasks. Hora et al. [2016] implemented a classification model that, based on basic interface usage features, recommends the right moment to promote internal APIs to public ones; the model was evaluated in five widely adopted Java systems. Coelho et al. [2018] proposed a survival model to identify unmaintained projects on GitHub, which was later packaged as a free open source tool [Coelho et al., 2020]. For this, the authors derived a set of activity-based features related to the project, its contributors, and its owner. Dias et al. [2015] worked on a solution to untangle commits containing unrelated changes—e.g., bug fix and refactoring—based on fine-grained modifications such as the instant these corrections are made to each file, methods invocation, variable access, etc. Alencar da Costa et al. [2014] and Bernardo et al. [2018] leveraged ML models to estimate the time to integrate pull requests; the former analyzed this strategy in the context of issues already addressed, while the latter analyzed this integration time in continuous integration environments.

Some papers also relied on ML to handle architecture and design issues. Mendes et al. [2016] overcome with a classification strategy to identify utility functions implemented in open source systems; the idea is to verify if the identified functions are in the correct module (`utils`). Bajammal et al. [2018] proposed an approach to automatically generate reusable web components from HTML Mockups, where it first normalizes the Mockup into a picture containing the interface elements and then applied an unsupervised strategy to derive its components; the approach was evaluated on five real-world mockups. Thaller et al. [2019] developed an algorithm to automatically identify the use of four design patterns in source code.

Finally, other specific research problems in software engineering were tackled in an analogous format. Bao et al. [2017] analyzed the use of ML techniques to predict developers turn over in two private software companies. Tian et al. [2015] applied Random Forest—a specific ML technique—to investigate the most influential factors when classifying high-rated apps in the Android ecosystem. Kuhn et al. [2007] implemented a clustering technique to group source code artifacts based on the semantic vocabulary contained in them, like comments, identifiers names, etc.

Although the goals of the aforementioned studies vary greatly, they are similar

with respect to the methodology used by them. Such similarities come from the way that ML algorithms are applied in these studies; they are mostly used as an empirical method to assist researchers in validating their hypothesis through large-scale data analysis [Shafiq et al., 2020]. Likewise, we also relied on these techniques to support the hypothesis we leveraged in Chapters 4 and 5.

2.4 Related Work

The studies which approach technical expertise centered on software developers present several particularities. Besides, these works partially overlap the methods and goals they set up; for instance, some works use the same feature set to investigate different aspects of software expertise [Wan et al., 2018; Constantinou and Kapitsaki, 2016]. As a consequence, these works can be categorized in several aspects, ranging from the distinct features used in their models (e.g., commit activity, GitHub metadata, Q&A data, etc) to the type of expertise that was investigated (e.g., programming languages, third-party libraries, etc). To better align with the way this thesis is structured, we decided to organize related work as follows. Section 2.4.1 presents research that used low level features—i.e., source code—to study developers expertise. In turn, Section 2.4.2 describes studies which relied on high level features, such as developers' metadata. Lastly, Section 2.4.3 depicts studies centered on analyzing technical expertise in more specific tasks, like matching job opportunities.

2.4.1 Technical Expertise with Low-Level Features

As previously stated, the following studies rely on low-level features to mold developers' technical expertise. Therefore, we opted to compare these works with the one described in Chapter 4 of this thesis, as it is based on a similar feature set.

Teyton et al. [2013] structured a search engine solution that, based on mining software repositories techniques, lists experts in third-party libraries. To identify library usage, the model performs a syntactical analysis over each commit looking for source code lines that introduce any symbol of the library under analysis. Next, these symbols are extracted and assigned to the author of the commit. Finally, the library expertise grade is defined by the ratio between the number of symbols used by the developer and the ones available at the library. Instead of reporting this information in a profile format, the authors developed a DSL for querying experts in a given library. This tool was used in two case studies with developers of the Apache HBase project: to assess developers' knowledge in the libraries used at HBase, and to recommend devel-

opers to perform software development tasks. The authors identified that 8 out of 29 libraries have only one expert; so more developers should be trained to master them. Furthermore, only three developers (out of 33) have expertise in more than half of the necessary libraries, which illustrates their importance to the project. Later, the authors fine-tuned the aforementioned approach to include more abstract features, such as file extensions [Teyton et al., 2014]. Whilst this work focus on identifying developers who are experts in a given library, our method considers developers' expertise level as well. Furthermore, the authors restricted the criteria to identify experts to only one feature (amount of different symbols); on the other hand, we mined other contribution aspects, such as the frequency, volume, and time between the contributions. Finally, this solution has a major constraint with respect to its extension, since it is necessary to reimplement the syntactical parser for every new programming languages.

Constantinou and Kapitsaki [2016] focused on extracting developers' expertise specifically for programming languages based on GitHub data. For each developer in each programming language, the authors mined the commits from projects he/she participated, and then calculated three commit-based features for each project: (a) Commit Activity, corresponding to the percentage of commits he/she has; (b) Commit Files Activity, representing the percentage of distinct files modified by the developer; and (c) Changed LOC Activity, responsible for measuring the percentage of lines changed by the developer. Furthermore, a boosting factor was introduced for each feature to increase the weight of his/her previous contributions. Lastly, the authors aggregated, averaged, and normalized the values obtained from each feature into a single one, indicating the developer's expertise level in each programming language. Similar to other studies, a subset of 150 GitHub users with active profiles in Stack Overflow was selected to compare the expertise rank generated by the proposed approach against the users' answering activity in Stack Overflow. In general, their extraction method presented interesting results when identifying users' main programming language, as their main language appears, on average, at the top 20% most recommended ones. Differently, we studied techniques to identify developers' expertise in the context of third-party libraries. Besides, we also included more specific features to support our technique—e.g., *number of imports*—due to the nature of technical skills we investigate (libraries and frameworks).

Saxena and Pedanekar [2017] presented a method that extends the profile of GitHub users by annotating them with Stack Overflow tags. To perform this match, the proposed method parses the import statements of each source code file that a user had contributed to, and extracts the class and method names attached to each package. Next, these names are used as input to Stack Overflow search queries, where the authors

extract the tags associated with the top-25 questions retrieved for each class/method combination. Finally, these tags are weighted based on their frequency, and clustered into distinct technological areas; the resulting skill profile is composed, for each user, by its top-100 tags. An instance of this method was implemented for the Java language and executed for 66 users. After qualitatively comparing the skill profiles with the LinkedIn endorsements provided by these users, the authors observed their solution provided a more detailed, yet accurate, list of technical skills. For instance, while some resumes describe experience with `Android`, `Java`, and `Web Development`, the proposed approach identified more specific techonologies like `swing`, `opengl`, `jboss`, etc. As in Teyton et al. [2013] work, this approach is restricted to the parser's implementation (Java, in this case). It is also important to note that the same features are used to assess expertise in distinct topics, such as languages, platforms, and libraries. By contrast, we investigate in this thesis different strategies to mine distinct expertise topics.

Oliveira et al. [2019] proposed another strategy to identify experts, this time considering frequency, depth, and breadth aspects when using third-party libraries. By relying on three activity-based features—the number of commits, number of imports, and number of LOC—the authors investigated the effectiveness of these features in identifying experts in nine Java libraries. For this, they first collected the data of 12K GitHub projects that used one or more selected libraries and then calculated the values of the features for each developer who contributed to these projects. Next, the developers located in the top quintile of at least two features were requested to provide an expertise rate in the evaluated libraries. After cross-checking the received responses, the authors find out that a high precision rate can be achieved as long as the features are combined. Although the authors follow a similar strategy to ours to identify expertise in software libraries, we worked further on this problem and proposed a solution to classify developers' expertise level on these libraries. To make this possible, we also have to overcome with an expanded feature set which takes into account volume and time aspects. Moreover, we also conducted a wider evaluation where we analyzed the performance of our method for developers not classified as experts.

2.4.2 Technical Expertise with High-Level Features

The studies reported below consolidate their features on a higher level of information, not directly related to developers' programming activity. We analyze these works from the Chapter 5 perspective, where we also employed similar features in our study.

Wan et al. [2018] developed a recommendation system to rank developers coding skills based on information extracted from GitHub. The coding skills, mapped

as a list of technology topics, are calculated by a probabilistic model, which takes into account the textual metadata of GitHub projects (e.g., README files), and the existing network information among users and repositories. The goal is to, given a specific technology keyword (e.g., *linux*), lists the most relevant developers related to it. An instance of this tool was implemented using Stack Overflow's data as ground truth and evaluated in two different scenarios. In the first one, the authors selected developers with existing profiles in both platforms and then verified if their proposed solution returns similar technology topics to the ones present in Stack Overflow. The second one was built manually for eight popular randomly selected topics, where four independent subjects graded the returned developers according to each topic. In both scenarios the tool presented better results, performing better for 5 out of 8 metrics in the first scenario, and presenting the best ones for all 7 metrics in the second. We also relied on Stack Overflow to leverage our ground truth in Chapter 5. However, we focused our efforts on classifying developers into more abstract clusters, like technical roles (e.g., backend, frontend, mobile, etc). We also discarded any kind of prioritization between software developers since these features provide insufficient data to perform such evaluation.

Previously, Greene and Fischer [2016] implemented a tool—called CVExplorer—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). These tags—mined from the project's README files and commits information—are extracted according to a previously defined white list containing popular programming languages, web frameworks, third-party libraries, etc. The authors then used CVExplorer to recommend candidates for open positions in two private companies and then evaluated their feedback. Overall, both companies returned positive reports; the first identified all 12 candidates as interesting options, and the second flagged two out of five as suitable ones. Differently from our method, the authors considered existing information on README files to analyze different expertise characteristics, i.e., languages, libraries, etc. On the other hand, we collected data from more diverse sources—developers' bio, projects' metadata, commits activity, etc—and also analyzed developers from a higher perspective (i.e., technical roles).

Huang et al. [2016] developed an approach that scores the programming ability of software developers by considering data from Q&A and Open Source Software (OSS) communities. The proposed approach is composed of four steps. First, an Identity Linkage procedure is used to link developers' profiles between the platforms into a single

one. Next, a list of programming ability terms is extracted according to each platform; answers information from Q&A and basic information from existing projects in OSS platforms. These features are then used to feed independent expertise models that will rate developers' ability in a list of programming topics, e.g., libraries, frameworks, programming languages, etc. Lastly, these scores are merged and averaged into a single rate per topic, per developer. The authors experimented with their solution using the information provided by Stack Overflow and GitHub platforms, where they evaluated the effectiveness of their tool in identifying the top-10 developers in 305 topics in four different models. Considering the best one, their technique had an average precision of 60%; showing better results for programming languages (80%). Although the authors gathered data from multiple data sources, they restricted themselves to collect only tagged information, i.e., answers tags from Stack Overflow and projects tags from GitHub.

Agrawal et al. [2016] studied how to automatically reveal collaborators' roles in VCS based projects. The proposed solution consists of mining and analyzing the list of commits each collaborator has performed in a given project. The authors start by applying the bag-of-words technique to extract keywords from the commits' metadata—e.g., commit messages, file extensions, etc—and tag each keyword into one or more predefined labels representing distinct technical roles, such as `Test`, `Web`, `Backend`, etc. This label is propagated to each commit that refers to the respective keywords, resulting in a list of labeled commits. Lastly, the commits are aggregated by each collaborator together with their labels, which are merged and averaged based on their number of occurrences. As a result, the authors reveal which activities—and for how much time—are performed in each role. This approach was tested in three private projects, where the authors used the label's distribution among each developer to better understand the particularities of the technical roles they belong to. For instance, they found out that a tester spends 43% of its time with tests, 18% with maintenance, and 16% with web-based activities. It is important to note that, unlikely our study, the authors unveil the activities performed in each role.

2.4.3 Technical Expertise Towards Specific Tasks

On the one hand, the research described in this section also mined the developers' activity to extract their technical abilities. On the other, these works did so having specific purposes in mind, like job ads recommendation. As a consequence, it might be more difficult to extend the proposed solutions to other situations. The work described in this thesis focuses on providing an overview of developers' technical skills,

i.e., independently of any specific task.

Hauff and Gousios [2015] proposed a pipeline that mines GitHub profiles and job advertisements to match each other automatically. To make this pipeline possible, the authors structured their solution into three steps. Firstly, they relied on NLP techniques to extract expertise concepts from both job advertisements (job description) and developers' profiles (README files). Secondly, the extracted concepts are vectorized and weighted using TF-IDF [Foster Provost, 2015] technique. Thirdly, they use cosine similarity to match developers' vectors with the ones extracted for the job ads.

Wang et al. [2017] aimed at recommending developers for software development tasks in crowdsourcing platforms—e.g., TopCoder[5]—by considering the skill improvement of software developers. To design such a tool, the authors first selected 74 developers with high commitment at TopCoder and extracted four features to map the difficulty of the tasks they worked on. Based on this difficulty score, they leveraged a prediction model that infers the performance rate of the developers for new crowdsourcing tasks. Mao et al. [2015] also proposed a content-based recommendation technique to automatically match tasks and developers in TopCoder. In particular, their solution focuses on more descriptive attributes, such as tasks' title, overview description, etc. These features are used as input for a multi-class machine learning model that predicts, based on developers' past solved tasks, which one is most skilled for solving a given task.

Venkataramani et al. [2013] implemented a model that mine developers' activity on GitHub to capture their technical expertise to recommend them to answer questions in Stack Overflow. For this, the model first extracts a list of technical terms from the source code files modified by each developer. Next, each term is mapped into its respective Stack Overflow tags. Developers' expertise familiarity is then measured according to the frequency each tag appears for each developer. The authors evaluated their approach by verifying contributors from 20 Java projects in GitHub who answered questions tagged as `java` in Stack Overflow. For a sample of 15 developers, 7 have answered questions containing tags that were also predicted by their approach.

2.5 Final Remarks

Figure 2.2 summarizes our view on how the background sections are related to each other given the circumstances of this thesis. We started by defining what is expertise in the context of software development, as well as which characteristics should we look at

[5] `https://www.topcoder.com/`

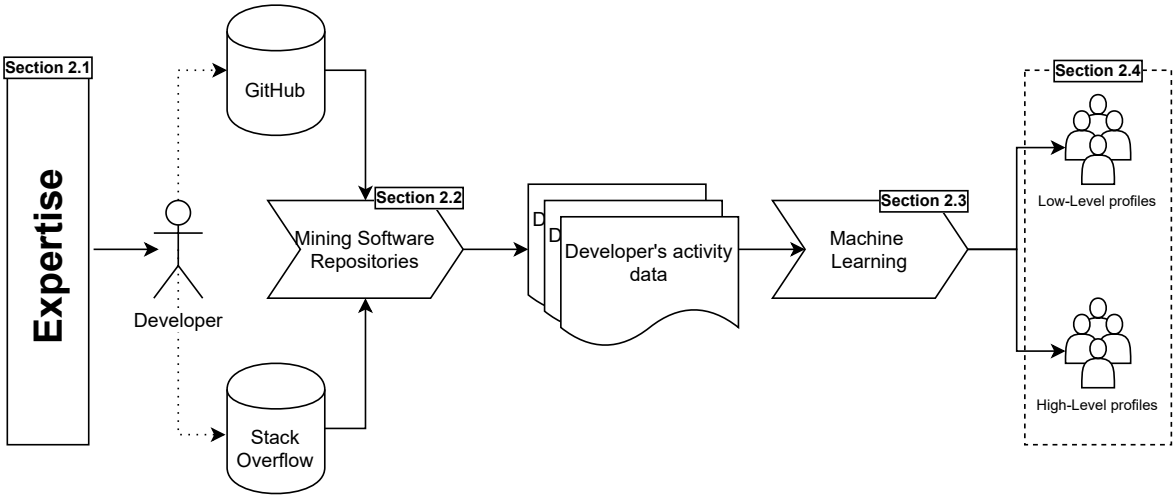


Figure 2.2: Conceptual diagram summarizing the contribution of each Background section to the study of software developers' expertise.

when analyzing software developers (Section 2.1). Moreover, we described some studies that qualitatively analyzed the activity information available for software developers, and mapped the data points that can be used to help in attesting their expertise. Next, we listed other papers that specifically use Mining Software Repositories (MSR) methods in these Social Coding Platforms to investigate important issues, such as APIs maintenance and evolution, the popularity of open source projects, etc (Section 2.2). In Section 2.3, we described relevant works that adopted Machine Learning (ML) techniques to empirically investigate several topics in the software engineering field, ranging from quality assurance to architecture and design problems. Finally, we bring in Section 2.4 the state-of-the-art that applied MSR methods to collect information about software developers, and employed ML techniques to shed light on which data can be used to discern expertise elements.

Chapter 3

What Skills do IT Companies look for in New Developers? A Study with Stack Overflow Jobs

We describe in this chapter the results of a large-scale exploratory study intended to **reveal the technical and soft skills do IT companies look for when hiring new developers**. This study is structured into the following sections. We start by introducing the relevance of conducting this investigation in Section 3.1. Sections 3.2 and 3.3 describe in-detail the data we used to conduct this research. We report our analysis of technical skills in Section 3.4. Likewise, our findings regarding soft skills are described in Section 3.5. We compare our results with other works in Section 3.6. Section 3.7 lists the threats to validity concerning this work. Lastly, we conclude this chapter in Section 3.8.

3.1 Introduction

There is a growing demand for information on how modern companies deal with the skills they need when looking for developers for their open positions. To find out what are these skills, we report in this work the first results of an analysis of more than 20,000 job opportunities available in Stack Overflow Jobs portal, a platform that allows companies to publish new opportunities for IT professionals. In particular, we leveraged the main characteristics of 14 IT roles, such as Backend, Frontend, and Mobile developers. We also analyze which soft skills are mostly requested by IT companies when selecting new candidates. Finally, we discuss the implications of this analysis for

both recruiters and developers.

3.2 The Anatomy of a Job Opportunity

A job opportunity is a declaration of expectations. It describes what a company expects from its candidates, as well as what the candidates should expect from the company. For this, a job opportunity includes important company details, such as mission, culture, benefits, etc. At the same time, the company should provide enough details so candidates can decide whether they are qualified or not for the position. Lastly, an opportunity should list the job's responsibilities, so that candidates can be aware of their duties.

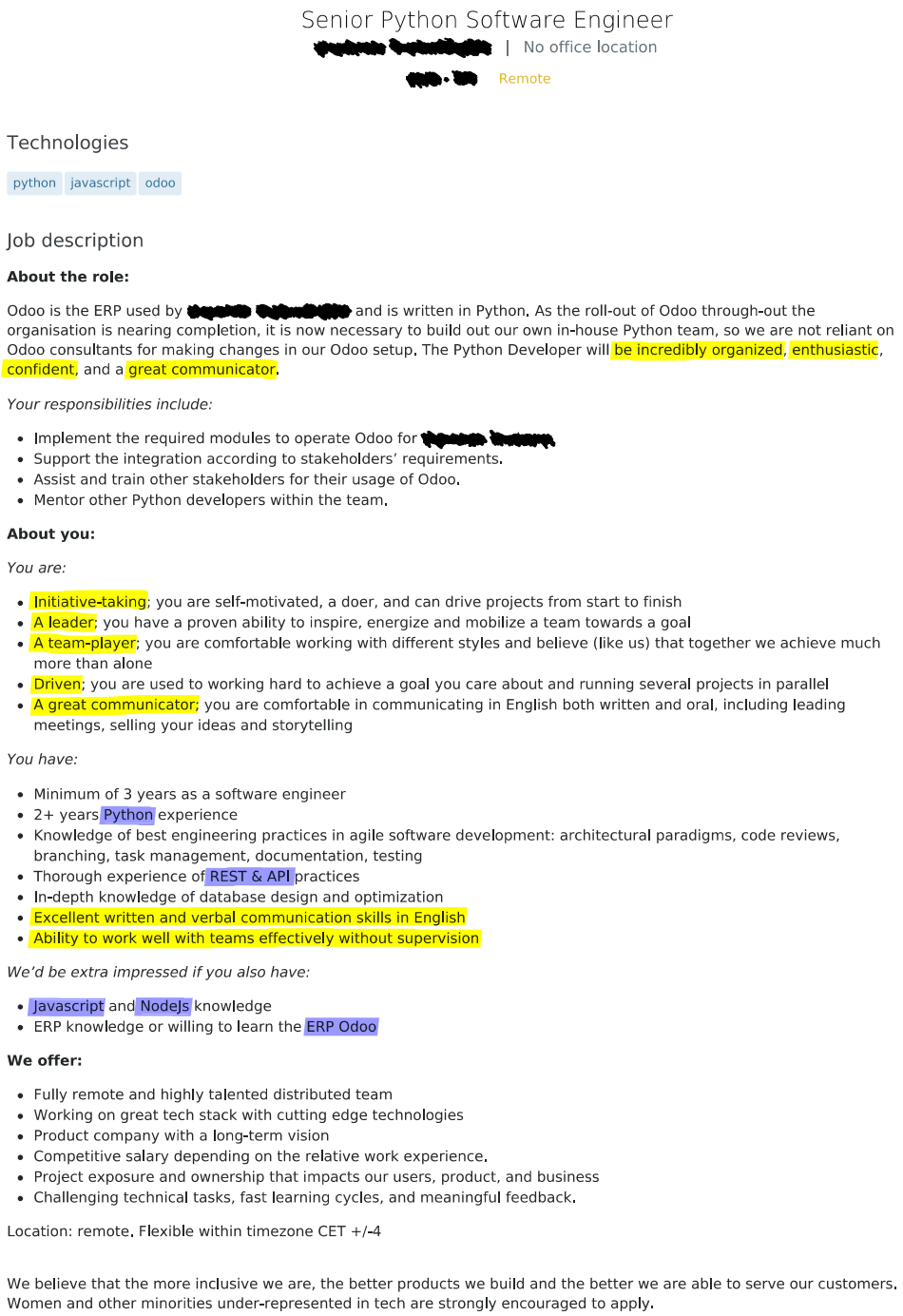
Figure 3.1 shows an example of a job opportunity posted on Stack Overflow Jobs portal. As we can observe, some skills are technically-driven, e.g., *Python*, *REST & API*, *JavaScript*, etc. Skills in this group are known as hard skills [Xia et al., 2019]. By contrast, other skills denote behavioral characteristics of the candidates, such as *verbal communication*, *team player*, *leadership*, etc. They are known as soft skills [Sayfullina et al., 2018]. In this investigation, we analyze both hard and soft skills required by IT job opportunities.

3.3 Data Collection

We investigated the jobs posts available on Stack Overflow Jobs portal.[1] We collected the posts visible in the platform for three months, from March 25th to June 28th, 2019 by downloading the posts available in every weekday during this period. As a result, we retrieved a total of 20,968 job posts, including their titles, description, and tags. Furthermore, each post in Stack Overflow is automatically associated with at least one out of 14 predefined roles provided by the platform, such as BackendDeveloper, FrontendDeveloper, MobileDeveloper, etc. We also collected this information to perform a fine-grained analysis of each post.

Figure 3.2 shows the distribution of opportunities for each role. FullStack-Developer, BackendDeveloper, and SystemAdministrator are the most demanded positions, with at least 3,600 posts each, representing 52.8% of all job posts. By contrast, six roles have less than 1,000 posts: DesktopDeveloper, DataScientist, EmbeddedDeveloper, ProductManager, GameDeveloper, and Designer. Together they represent 10.2% of the analyzed posts.

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



Senior Python Software Engineer

[redacted] | No office location

[redacted] Remote

Technologies

python javascript odoo

Job description

About the role:

Odoo is the ERP used by [redacted] and is written in Python. As the roll-out of Odoo through-out the organisation is nearing completion, it is now necessary to build out our own in-house Python team, so we are not reliant on Odoo consultants for making changes in our Odoo setup. The Python Developer will be incredibly organized, enthusiastic, confident, and a great communicator.

Your responsibilities include:

- Implement the required modules to operate Odoo for [redacted]
- Support the integration according to stakeholders' requirements.
- Assist and train other stakeholders for their usage of Odoo.
- Mentor other Python developers within the team.

About you:

You are:

- Initiative-taking; you are self-motivated, a doer, and can drive projects from start to finish
- A leader; you have a proven ability to inspire, energize and mobilize a team towards a goal
- A team-player; you are comfortable working with different styles and believe (like us) that together we achieve much more than alone
- Driven; you are used to working hard to achieve a goal you care about and running several projects in parallel
- A great communicator; you are comfortable in communicating in English both written and oral, including leading meetings, selling your ideas and storytelling

You have:

- Minimum of 3 years as a software engineer
- 2+ years Python experience
- Knowledge of best engineering practices in agile software development: architectural paradigms, code reviews, branching, task management, documentation, testing
- Thorough experience of REST & API practices
- In-depth knowledge of database design and optimization
- Excellent written and verbal communication skills in English
- Ability to work well with teams effectively without supervision

We'd be extra impressed if you also have:

- Javascript and NodeJs knowledge
- ERP knowledge or willing to learn the ERP Odoo

We offer:

- Fully remote and highly talented distributed team
- Working on great tech stack with cutting edge technologies
- Product company with a long-term vision
- Competitive salary depending on the relative work experience.
- Project exposure and ownership that impacts our users, product, and business
- Challenging technical tasks, fast learning cycles, and meaningful feedback.

Location: remote. Flexible within timezone CET +/-4

We believe that the more inclusive we are, the better products we build and the better we are able to serve our customers. Women and other minorities under-represented in tech are strongly encouraged to apply.

Figure 3.1: An example of a post in Stack Overflow Jobs portal. Hard skills are annotated in blue, while soft skills are in yellow.

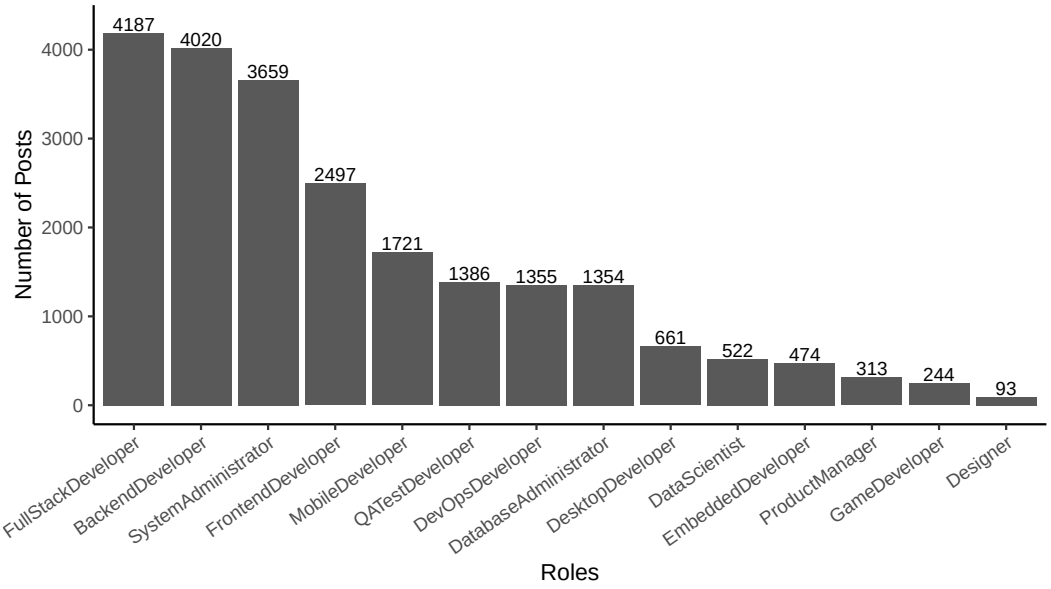


Figure 3.2: Distribution of job posts collected in this study.

3.4 Analyzing Hard Skills

On Stack Overflow Jobs, companies describe the hard skills required for a given position using the same tagging mechanism provided by Stack Overflow's main platform. Tags play a central role in this ecosystem, as they are used to identify Q&A topics [Saha et al., 2013]. Therefore, we use the tags added to each post as an indication of the hard skills demanded by each one. For instance, the tags associated to the job post in Figure 3.1 (top) indicate that candidates should master *python*, *javascript*, and *odoo*.

In total, we identified 1,916 hard skills (i.e., tags) mentioned 70,680 times in the collected posts. The frequency of these skills follows a long tail shape, where few elements are generally responsible for almost all occurrences. For this reason, we removed the hard skills with less than 10 occurrences. We ended up with 282 hard skills, representing 67,062 occurrences (95%). Next, two authors used open card sorting [Spencer, 2009] to group tags into abstract categories. After analyzing together a group of 50 tags, they leveraged six abstract hard skills: *Languages*, *Libs & Frameworks*, *OS & Infrastructure*, *Process & Methods*, *Data Systems*, and *Development Tools*. Then, they independently annotated the remaining 232 tags into these categories (i.e., each tag was evaluated by two authors). The inter-rater agreement, calculated using Cohen's Kappa [Warrens, 2015], was 0.82. Lastly, both authors discussed the conflicting cases to reach a common ground. Table 3.1 presents a sample of ten hard skills tags collected during this procedure along with the category assigned to them.

Figure 3.3 shows a heatmap with the distribution of high-level hard skills (rows)

Table 3.1: High-level hard skills created after open card sorting

Category	Hard Skills Examples
Data Systems	sql-server, oracle, mysql, hadoop, database, postgresql, mongodb, elasticsearch, bigdata, machine-learning.
Development Tools	continuous-integration, selenium, jenkins, git, jira, puppet, cucumber, maven, xcode, eclipse.
Languages	java, javascript, sql, python, c#, c++, java-ee, css, php, c.
Libs & Frameworks	reactjs, .net, spring, node.js, angular.js, asp.net, angular, ruby-on-rails, spring-mvc, api.
OS & Infrastructure	amazon-web-services, cloud, linux, windows, web-services, android, security, ios, docker, azure.
Process & Methods	testing, agile, qa, design, project-management, user-interface, etl, automated-tests, user-experience, scrum.

per developer roles (columns). With this heatmap, we intend to provide a technology-agnostic analysis, i.e., one that focuses on high-level hard skills (e.g., *Languages*) instead of current technologies (e.g., *Java*). In this way, we also intend to increase the validity of our results against technological changes.

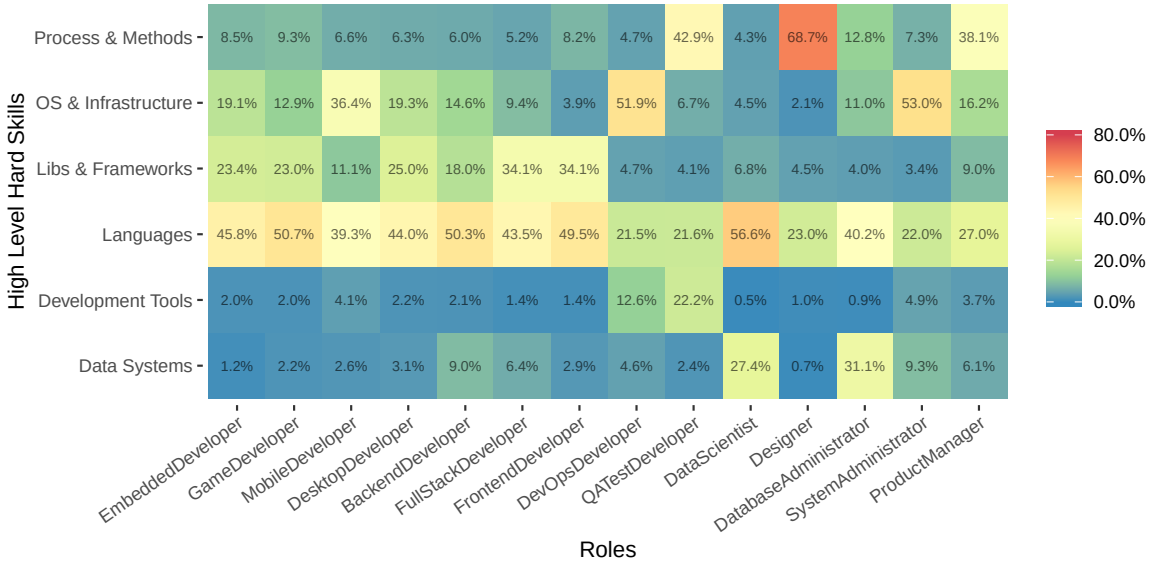


Figure 3.3: Heatmap indicating how hard skills are distributed among developers roles.

First, *Languages*-based skills are relevant for all roles, ranging from 21.5% (DevOpsDeveloper) to 56.6% (DataScientist). In fact, *Languages* is the most required hard skill for 9 out of 14 technical roles analyzed. Even designers must master some sort of languages to better provide their prototypes, such as *css*, *html*, and *javascript*. However, the concentration of *Languages* is higher for development-based roles, e.g., MobileDeveloper (39.3%), GameDeveloper (50.7%), FullStack-

Developer (43.5%), and FrontendDeveloper (49.5%). These characteristics make *Languages* the only skills that are significantly mentioned in all roles.

Likewise, development-based roles also demand skills on *Libs & Frameworks*, which generally have more than 20% of participation in this group. They are specially required for FullStackDeveloper and FrontendDeveloper (34.1%, both).

Regarding *OS & Infrastructure*, two roles stand out with more than 50%: SystemAdministrator (53.0%) and DevOpsDeveloper (51.9%). Interestingly, MobileDeveloper appears next with 36.4%. This result is explained by the fact that such developers must master mobile operating systems, such as *ios* and *android*.

For *Process & Methods*, Designer stands out with 68.7% of the skills coming from this group. Indeed, the most mentioned Designer skills are related to user interfaces, e.g., *user-experience*, *user-interface*, and *design*. Next, QATestDeveloper and ProductManager follow up with 42.9% and 38.1%, respectively. Differently, these roles require hard skills associated with software quality assurance. For instance, most QATestDeveloper opportunities require knowledge on automated testing methods. ProductManager's candidates should have experience in agile development processes such as *scrum*. By contrast, *Process & Methods*-based skills are not highly demanded for development-based jobs.

Skills on *Data Systems* are more required for DatabaseAdministrator and DataScientist, with 31.1% and 27.4%, respectively. Surprisingly, *Development Tools*-based skills are considerably mentioned only in QATestDeveloper (22.2%) and DevOpsDeveloper (12.6%). In fact, tools like *selenium* and *jenkins* are in the top-5 most mentioned ones for this category.

3.5 Analyzing Soft Skills

Unlike hard skills, soft skills are only available on jobs' textual descriptions, i.e., Stack Overflow does not include tags for such skills. This limitation makes the act of collecting soft skills more complex, as we have to extract them from unstructured text. We handled this issue by first randomly selecting a sample of 376 opportunities from a subset of 17,756 posts.[2] We discarded the remaining 3,212 posts as they were not written in English and, therefore, do not fit into our analysis. Then, three authors manually annotated sentences that refer to soft skills, such as "good communication skills", "ability to work independently", etc. In total, we extracted 1,530 sentences from 315 opportunities. Next, we generated a word cloud to identify the most common terms

[2]The sample size was determined after specifying a limit of 95% confidence level and 5% confidence interval in a well-known sample size calculator, available at `https://surveysystem.com/sscalc.htm`.

among these sentences. As some soft skills can be described using multiple words (e.g., "verbal communication"), we adapted the word cloud to consider bigrams as well.

Figure 3.4 depicts the word cloud for the top-100 most frequent terms. We can observe that *communication* plays a central role among the most required soft skills. Our sample allows us to conclude that $32\% \pm 5\%$ of the posts in our population (i.e., about one in three posts) mention this skill. *Collaboration*-based skills also feature a special position in this ranking. For instance, the word "team" appears in at least 22% of the jobs posts (i.e., $27\% \pm 5\%$). Lastly, some companies also require experience in *problem-solving* skills, as the words "analytical", "problem solving", or "deliver" are present in at least one out of ten posts (i.e., $15\% \pm 5\%$).

Figure 3.4: Soft skills word cloud.

3.6 How do Skills Change Over Time?

Litecky et al. [2010] leveraged the most demanded hard and soft skills in the IT industry between July 2007 and April 2008 through an analysis of more than 200,000 online job opportunities. When we compare their results with ours, we observe some changes regarding hard skills. For instance, languages such as *C/C++*, *SQL*, and *Java* were frequently mentioned in 2008, as they were listed as major ones in 14 out of 20 positions (70%). Likewise, in our study *Languages* is the most requested high-level hard skill in 9 out of 14 developer roles (64%). Interestingly, no library or framework appeared among the popular hard skills in 2008. This represents a contrast with our study, as *Libs & Frameworks* are the second most demanded hard skill for six roles.

On the other hand, we observe minor changes in terms of soft skills. Litecky et al. [2010] also show that *leadership* is mentioned in 20% of the opportunities. Similarly, terms like *leadership* (7.6%), *lead* (4.0%), and *leading* (3.5%) are also featured among

the most popular soft skills in our study. More recently, Sayfullina et al. [2018]—which proposed a model to automatically extract soft skills from job ads' description texts—also reported similar popular keywords, e.g., *responsibility*, *independent*, *flexible*, etc. Such alignment persists even in earlier works. For example, *problem-solving* and *communication*-based skills appeared in 77% and 65% of the ads in the early 2000s, respectively [Lee and Han, 2008]; they are still highly requested nowadays.

3.7 Threats to Validity

We identified the following issues as possible threats to our work:

Data Collection: Our results are limited by the platform we use to collect the data for our analysis, i.e., the Stack Overflow Jobs portal. However, this study is focused at analyzing the most demanded skills in the IT industry, and therefore should not be generalized to other areas. Furthermore, the number of job posts analyzed in this study reduces the bias introduced by considering the information from one platform only.

High-level Hard Skills: We manually leveraged high-level hard skills to provide a more robust analysis. We applied open card sorting with two researchers to reduce the personal influence on the generated categories. We also calculated Cohen's Kappa to verify the inter-rater agreement between the classification of both researchers.

Soft Skills Sample Size: We relied on a subset of job posts to perform our soft skills analysis. To ensure the reliability of our results, the size of this sample was determined considering well known statistical parameters, i.e., random sampling method with 95% confidence level and 5% confidence interval.

Soft Skills Annotations: Three researchers manually annotated the sentences we used to analyze the most popular soft skills. Although we did not rely on Cohen's Kappa to verify the agreement between the annotations, the researchers previously discussed examples of soft skills sentences in other job opportunities.

3.8 Final Remarks

With respect to candidates for IT jobs, our study reveals the distribution of the most demanded hard skills for 14 contemporary developer roles. For example, people who are

applying to a frontend position should describe the *programming languages* and *libraries & frameworks* they master since 49.5% and 34.1% of the tags used by companies when hiring such developers refer to these hard skills. Furthermore, the study confirms the importance companies give to soft skills. The most requested ones are related to communication, collaboration, and problem-solving.

Replication Package: The data used in this study is available at `https://doi.org/10.5281/zenodo.3906955`.

Chapter 4

Identifying Experts in Software Libraries and Frameworks among GitHub Users

In this chapter, **we propose a method to identify developers' expertise level on popular third-party software components**, particularly libraries and frameworks. We organize it into seven major sections. First, we highlight the relevance of studying expertise in this context and present the key hypothesis behind this investigation (Section 4.1). Next, Section 4.2 documents the process we followed to collect the data used to answer the research questions leveraged previously. Section 4.3 describes the techniques we used in this work, as well as their setup. Section 4.4 provides the answers we obtained from this study. We discuss in Section 4.5 our findings, lessons learned, and limitations. This section also proposes a practical method for identifying library experts and validates its results with Linkedin data. Finally, Section 4.6 reports threats to validity and Section 4.7 concludes this chapter.

4.1 Introduction

Modern software development heavily depends on libraries and frameworks to increase productivity and reduce time-to-market [Ruiz et al., 2014; Sawant and Bacchelli, 2017]. 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 com-

panies can also benefit from this information before hiring developers for 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 [Treude et al., 2011] or to assist project managers to set up balanced development teams [Siau et al., 2010].

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 [Mockus and Herbsleb, 2002] visually maps parts of a software product (e.g., code or documentation) to the respective experts, using the number of changes (commits) as the basic measure of expertise. Fritz et al. [2007, 2010, 2014] propose the degree-of-knowledge (DOK) metric to identify experts in specific source-code files, which combines both commits and interactions with the code through an IDE. Schuler and Zimmermann [2008] advocate that expertise can also be gained by using the component of interest (e.g., by calling its methods). Da Silva et al. [2015] 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 a specific software product.

In this chapter, 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, the number of commits on files that import each library and the 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

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

(on 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 [Breiman, 2001] and SVM (Support Vector Machine) [Weston and Watkins, 1998]. Finally, we investigate the use of clustering algorithms to identify library experts. 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 in using GitHub as a full proxy for expertise. For example, some experts rarely contribute to public GitHub projects; their expertise comes from working on private projects or projects that are not GitHub-based. For this reason, it is common to have both experts and novices with 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*.

Our 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 pieces of evidence of being experts in facebook/react, a popular JavaScript library; (3) we provide a public ground truth with the expertise of 575 developers that can be used as a baseline to evaluate other solutions; indeed, we already observe a few works relying on this dataset to conduct their research [Eke, 2020; Vadlamani and Baysal, 2020; Dey et al., 2021].

4.2 Data Collection

4.2.1 Definitions

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

Target Library: The JavaScript libraries used in this study; 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 study.

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.

4.2.2 Target Libraries

We evaluate JavaScript libraries due to the importance and popularity of this language in modern software development. Particularly, JavaScript is the most popular language on GitHub; around 34% of the most popular projects on GitHub are implemented in JavaScript [Avelino et al., 2016]. We focus on the developers of three JavaScript libraries:[2]

1. facebook/react: A system for building enriched Web interfaces. The project source code can be found at `https://github.com/facebook/react`.

[2]In our study, the terms libraries and frameworks are used interchangeably.

2. mongodb/node-mongodb: the official Node.js driver for MongoDB database server. The project is available at `https://github.com/mongodb/node-mongodb-native`.

3. socketio/socket.io: a library for real-time communication. The source code of the project is available at `https://github.com/socketio/socket.io`.

We start by selecting 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. Finally, we choose socketio/socket.io since communication is important both in front-end and back-end programming.

Table 4.1: Target Libraries

Target Library	**Stars**	**Contributions**	**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

Table 4.1 shows information about these systems, including the number of stars, contributors, commits, and files (in 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 chapter.

4.2.3 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 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's 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 4.2: Client Projects and Candidate Experts

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

Table 4.2 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.

4.2.4 Features

We collected 13 features for each candidate expert who was selected in the previous step. As documented in Table 4.3, these features cover three dimensions of *changes* performed on client files: *Volume*, *Frequency*, and *Breadth*. These dimensions and their features were derived and extended from the literature on developers' expertise in open source communities. For instance, the volume of changes is commonly used in related works [Mockus and Herbsleb, 2002; Fritz et al., 2007, 2010, 2014]. Likewise, frequency and breadth of changes have also been considered as proxies to developers' expertise [Dabbish et al., 2012; Singer et al., 2013; Da Silva et al., 2015; Sarma et al., 2016; Avelino et al., 2019b]. As an additional criterion, we only use features that can be directly computed from the GitHub public API.

Volume of changes: Includes six features about the number of changes performed by candidate experts in client projects, such as 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: Encompass five features expressing the frequency and time of the changes performed by candidate experts, e.g., the number of days since first and last library import. The rationale is that expertise also depends on the temporal properties of the changes.

Table 4.3: 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

Breadth of changes: 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 on 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).

4.2.5 Ground Truth

To create a ground truth with developers' expertise on each target library, we surveyed the candidate experts identified in Section 4.2.3. 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 4.2.3, 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)?*

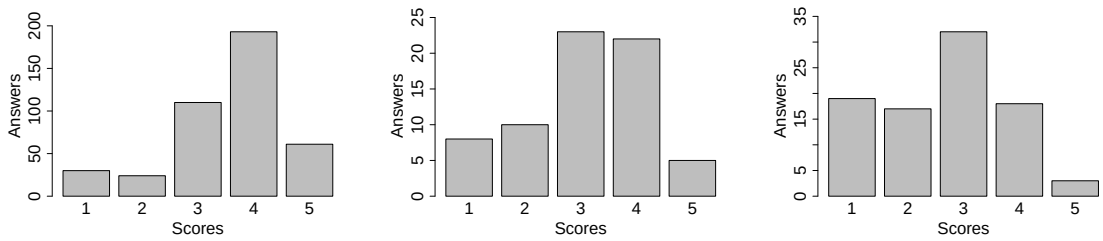
Table 4.4 summarizes the number of e-mails sent, the number of received answers, and the response ratio. The number of answers range from 68 (NODE-MONGODB) to 418 (REACT) and the response ratio ranges from 15% (SOCKET.IO and NODE-MONGODB) to 19% (REACT).

Table 4.4: Survey Numbers

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

Figure 4.1 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



(a) facebook/react (b) mongodb/mongo (c) socketio/socket.io

Figure 4.1: Survey answers

have omitted their lack of experience on the studied libraries (see the Dunning-Kruger Effect [Kruger and Dunning, 1999]). 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 for 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 [Siegmund et al., 2014].

4.2.6 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 $imports = 0$. Actually, 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.

Removing Correlated Features: Correlated features may contribute to inaccurate classifications due to their high association degree [Yu and Liu, 2003; Chen et al., 2005]. To tackle this issue, we first used the *cor*[3] function from R's *stats* package to compute a matrix with Pearson[4] coefficients for each pair of features. Then, we used the *findCorrelation*[5] function from R's *caret* package to identify pairs of features with a correlation greater than 0.7, as previously adopted in the literature [Bao et al., 2017]. In such cases, we discarded the feature that has the highest correlation rate when compared to the other ones. Figure 4.2 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.

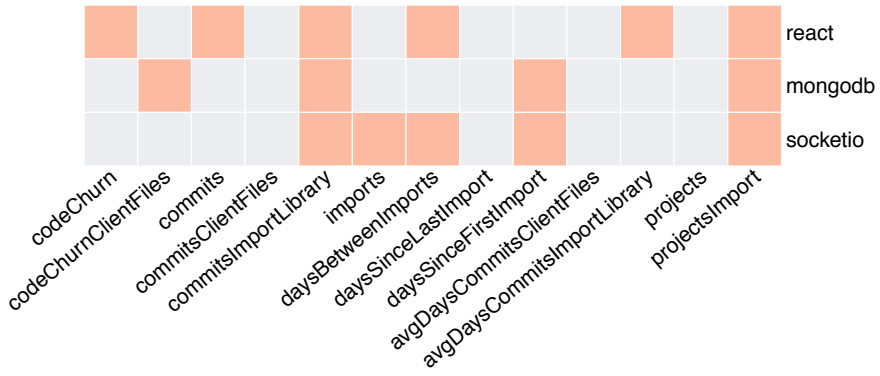


Figure 4.2: Correlation analysis; red cells are discarded due to high correlation.

Skewed Feature Values Features with skewed distributions may impact the performance of machine learning classifiers [Kuhn and Johnson, 2013; Zumel et al., 2014]. We assume that skewed feature distributions are the ones where the mean—computed for the candidate experts included in the ground truth of a

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

[4] We executed our model applying both Pearson and Spearman coefficients, achieving better results with the former.

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

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 [Chulani et al., 1999].

4.3 Methods

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

4.3.1 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) [Raudys and Jain, 1991]. 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 a score of 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 that 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 a few samples for some classes, the ground truth is largely imbalanced, as illustrated in Figure 4.1. 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 [Japkowicz and Stephen, 2002]. To tackle this problem, we used a technique called Informed Over Sampling (SMOTE) [Chawla et al., 2002], 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 [Tan et al., 2015], mobile apps analysis [Li et al., 2016], self-admitted tech-

nical debt detection [Zampetti et al., 2017], and identification of security issues from commit messages and bug reports [Zhou and Sharma, 2017]. In our problem, we used SMOTE over the minority class, in both scenarios. SMOTE has two parameters: the number of the nearest neighbors (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 neighbors of the existing ones (KNN parameter).

Machine Learning Classifiers

We evaluate two well-known machine learning classifiers: Random Forest [Breiman, 2001] and SVM [Weston and Watkins, 1998]. 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 methods (e.g., [Mockus and Herbsleb, 2002; Fritz et al., 2007, 2010, 2014; Schuler and Zimmermann, 2008]) 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 classifier's parameters. We rely on a grid search strategy for hyper-parameters with cross-validation to find the best parameter settings for each classifier [Claesen and Moor, 2015].

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.[6] 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 [Landis and Koch, 1977].

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

4.3.2 Clustering Setup and Algorithm

We use clustering to investigate more closely the relation of feature values and library expertise (*RQ.2*). For 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 [Gorla et al., 2014], test case prioritization [Arafeen and Do, 2013], and to characterize build failures [Vassallo et al., 2017]. 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 [Ng, 2000] and silhouette methods [Rousseeuw, 1987]. However, they also depend on interpretation and subjective decisions [Ng, 2000].

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 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 fewer 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 4.5 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 4.5: 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**

4.4 Results

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

Table 4.6 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 4.3.1. 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 4.6: 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

Table 4.7 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.

Table 4.7: Results for 3 classes: novice (scores 1–2), intermediate (score 3), and expert (scores 4–5)

	RForest	SVM	Baseline
facebook/react			
Kappa	0.09	0.02	0.00
AUC	0.55	0.53	0.50
Precision (Novice)	0.35	0.10	0.00
Precision (Intermediate)	0.28	0.00	0.00
Precision (Expert)	0.65	0.61	0.61
Recall (Novice)	0.16	0.04	0.00
Recall (Intermediate)	0.16	0.00	0.00
Recall (Expert)	0.82	1.00	1.00
F-measure	0.38	0.27	0.25
mongodb/node-mongodb			
Kappa	0.32	0.21	0.00
AUC	0.67	0.55	0.50
Precision (Novice)	0.58	0.37	0.00
Precision (Intermediate)	0.47	0.35	0.00
Precision (Expert)	0.59	0.55	0.40
Recall (Novice)	0.50	0.38	0.00
Recall (Intermediate)	0.52	0.21	0.00
Recall (Expert)	0.63	0.79	1.00
F-measure	0.53	0.41	0.19
socketio/socket.io			
Kappa	0.16	0.24	0.00
AUC	0.62	0.70	0.50
Precision (Novice)	0.48	0.52	0.40
Precision (Intermediate)	0.36	0.59	0.00
Precision (Expert)	0.45	0.49	0.00
Recall (Novice)	0.53	0.61	1.00
Recall (Intermediate)	0.28	0.38	0.00
Recall (Expert)	0.58	0.52	0.00
F-measure	0.44	0.47	0.19

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 4.8 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 4.3.2, for REACT and NODE-MONGODB, we have 3 clusters; for SOCKET.IO, we have 5 clusters. In Table 4.8, 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, 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 cluster; 67% of its members are novices and the cluster does not include any expert.

Table 4.8: 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

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 [Grissom and Kim, 2005; Romano et al., 2006; Linares-Vásquez et al., 2013; Tian et al., 2015], 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 4.9 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.

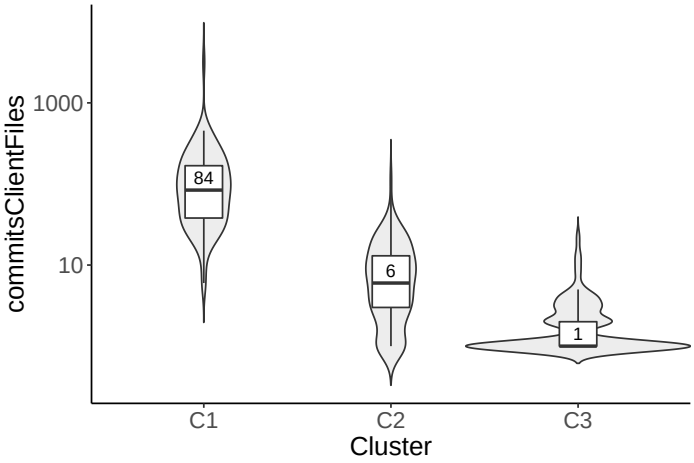
To visually illustrate these results, Figure 4.3 shows violin plots with the distribution on each cluster of *commitsClientFiles*, for the three studied libraries. We can see a *large* difference between the distributions of Cluster 1 and Cluster 2, both for REACT and NODE-MONGODB. By contrast, for SOCKET.IO, there is no clear difference between the distributions of Cluster 1 and Cluster 3 (cluster with the median closest to Cluster 1). Finally, Figure 4.4 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; the median is 8.5 projects) and Cluster 3 (1st quartile is one project; the median is two projects).

Table 4.9: 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)

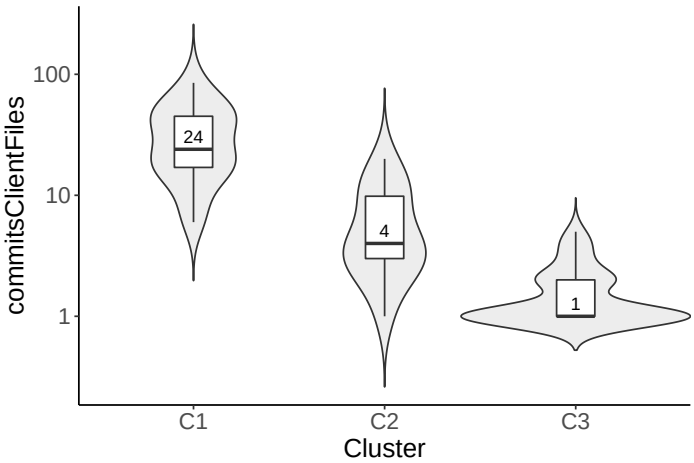
Feature	**Effect size**	**Relationship**
FACEBOOK/REACT		
codeChurnClientFiles	large	+
commitsClientFiles	large	+
imports	large	+
daysSinceLastImport	large	+
daysSinceFirstImport	medium	+
avgDaysCommitsClientFiles	large	−
projects	large	+
MONGODB/NODE-MONGODB		
codeChurn	large	+
commits	large	+
commitsClientFiles	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.

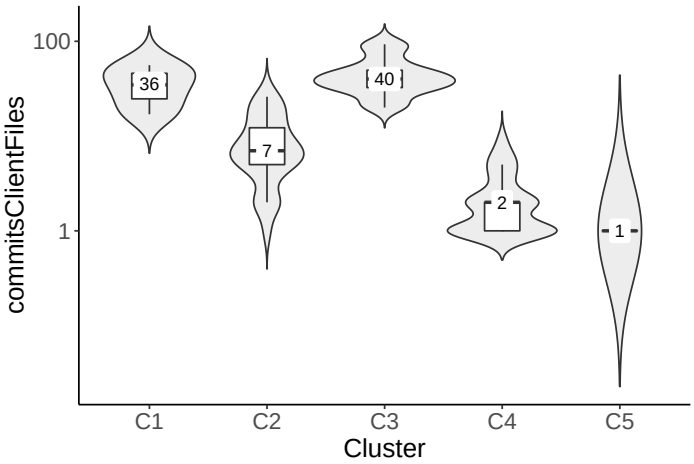
To conclude, it is important to mention that the feature values are different for experts in each library. For example, experts in FACEBOOK/REACT (Cluster 1) perform 84 commits at client files, against 24 commits for NODE-MONGODB's experts (median



(a) FACEBOOK/REACT



(b) MONGODB/NODE-MONGODB



(c) SOCKETIO/SOCKET.IO

Figure 4.3: Distributions of *commitsClientFiles* values for each cluster/library. Cluster 1 (experts) has higher values than other clusters, except for SOCKET.IO.

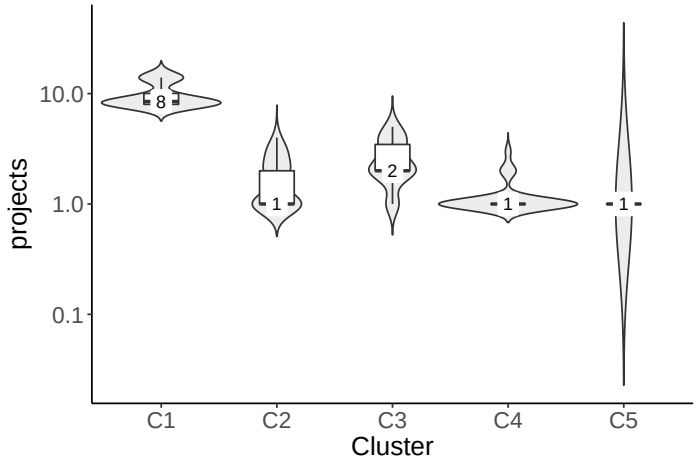


Figure 4.4: Distributions of *projects* values for SOCKET.IO clusters. Cluster 1 (experts) has higher values than other clusters.

values, see Figure 4.3). We hypothesize 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.

4.5 Discussion and Practical Usage

In this section, we summarize our key findings; we also discuss the practical usage and limitations of the machine learning and clustering methods investigated in this work.

4.5.1 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 methods 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.

4.5.2 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 study. 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.

4.5.3 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, we manually searched for the Linkedin page of these developers, looking for their names and possibly e-mails on Linkedin (when available, we also compared the profile photos, at Linkedin and GitHub). We were able to find the Linkedin profile of 160 developers (61%). Finally, we manually examined these profiles, searching for shreds of evidence of expertise on REACT. 115 developers (72%) explicitly refer to REACT on their Linkedin short bio, on the description of the projects they worked on, or in the list of programming technologies they have skills on. We also assessed the experience of these developers as Web developers, by calculating the number of years on jobs directly related to Web programming. Figure 4.5 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 in Web-related jobs.

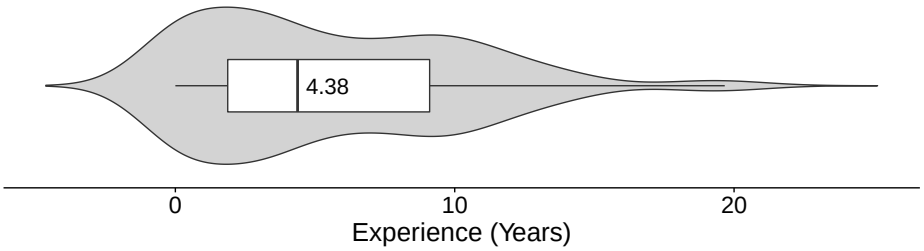


Figure 4.5: Years of experience on REACT of developers predicted as experts

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 evidence of having experience in the studied libraries. However, before concluding, it is also important to acknowledge that expertise and experience are distinct concepts; indeed, the experience is normally viewed as a necessary condition to achieve expertise [Ericsson, 2006; Baltes and Diehl, 2018].

4.5.4 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 GitHub is not very representative of my skills ... Most of my work is done privately.* (P374, score 2)

> *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 methods like the one proposed in this work. However, as mentioned before, precision tends to be more important in practical settings than recall. If we focus on precision, the proposed clustering method is effective on identifying experts among GitHub users that frequently contribute to client projects.

To illustrate this discussion, Figure 4.6 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 4.6 is a major challenge to their performance.

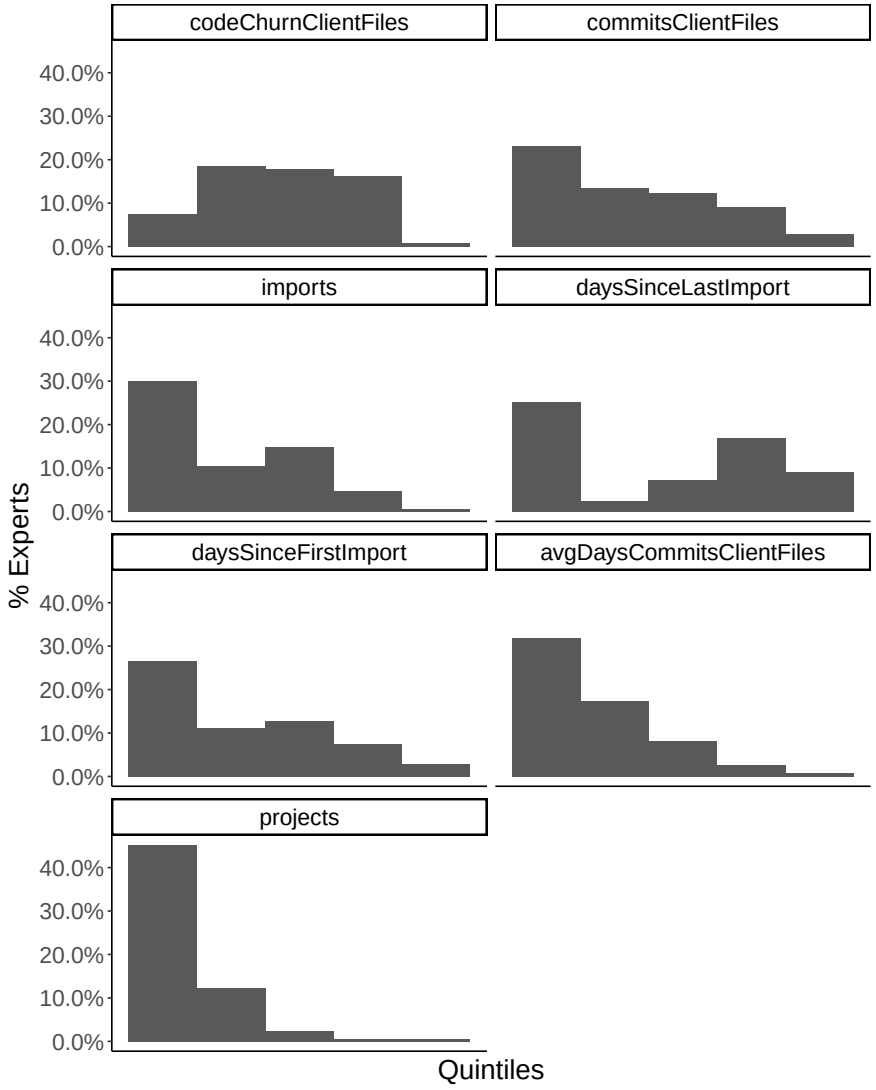


Figure 4.6: Percentage of REACT experts by quintiles of feature distributions. For most features, there is an important proportion of experts in lower quintiles.

Typically, these classifiers are not able to provide an *unknown* answer, as we discussed in Section 4.5.2.

How to improve the current precision results: We envision two main works that can improve the current results, particularly the precision of the experts class. First, one promising work is to expand the ground truth, which can contribute to reducing its imbalanced behavior. Particularly, according to the methodology followed in the study, the candidate experts are retrieved from the developers of the top-10K most popular GitHub projects, by their number of stars. Therefore, a natural extension is to consider candidate experts from a large base of GitHub projects (e.g., top-100K projects). Second, we can also include new features in the current setup, possibly

extracted from other data sources besides GitHub. For example, we can consider features extracted from Stack Overflow or TopCoder (a software development crowdsourcing platform with an online community of over 1M crowd software workers [Saremi et al., 2017]).

4.6 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 4.2.3). 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 4.2.3) 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 4.2.5. 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 [Tan et al., 2015; Li et al., 2016; Zampetti et al., 2017; Zhou and Sharma, 2017]. But we acknowledge that there are other techniques , such as over-sampling and cost-sensitive methods [He and Garcia, 2009; Chicco, 2017].

Number of Clusters: The selection of the optimal number of clusters is a central issue in partitioning clustering (e.g., k-means). This task is somehow subjective and its result depends on which method is used to measure similarities among objects (as discussed in Section 4.3.2). For this reason, we follow an expert-driven measure to detect clusters dominated by experts.

Triangulation with Linkedin: This analysis fundamentally depends on the accuracy of the information provided by Linkedin users. Moreover, Linkedin data

tends to reveal experience, but not expertise in programming technologies. Despite that, we claim it can be used as a first proxy to the task of identifying experts on software libraries and frameworks.

Machine Learning Models: To attenuate the bias of our results, we rely on 5-fold cross-validation and compute the average performance. In addition, we use grid search with cross-validation to tune hyper-parameters, both for Random Forest and SVM models.

Evaluation Metrics: Standard performance metrics (recall, precision, F-measure, and AUC) can provide misleading information on imbalanced datasets with multiple classes. For this reason, we showed results for each class, separately. Furthermore, we evaluated the models using Cohen's kappa, which is a metric particularly useful on imbalanced datasets.

4.7 Final Remarks

Companies often hire based on expertise in libraries and frameworks, as we found in the tags of Stack Overflow jobs. In this chapter, 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.

Replication Package: Our data—in a fully anonymized format—and scripts are publicly available at: `https://doi.org/10.5281/zenodo.1484498`.

Chapter 5

Mining the Technical Roles of GitHub Users

In this chapter, we aim at mining software developers' technical skills at a higher level. More specifically, **we report the findings we obtained at evaluating three distinct machine learning methods to identify developers' technical roles.** This investigation is divided into seven sections. In Section 5.1, we argue the importance of investigating this topic and present the research questions leveraged to perform such work. Section 5.2 documents the data and methods used in the study, including the ground truth creation, the features extracted from GitHub, and the machine learning setup. Section 5.3 presents the results for the first three research questions. Section 5.4 describes the exploratory study conducted on *full-stack* developers. In Section 5.5.3, we manually analyze some developers classified by our method to qualitatively interpret the classification results. In Section 5.5, we discuss the implications to both practitioners and software engineering community. Section 5.6 discusses threats to validity. Finally, Section 5.7 concludes this chapter.

5.1 Introduction

Software systems are complex engineering artifacts, which demand high levels of technical specialization in different areas [Brooks Jr, 1995]. These conditions make IT companies focus on creating cross-functional teams with experts in several positions, such as databases, security, frontend design, backend design, mobile apps, etc.

Previous works have proposed solutions to use the data of Social Coding Platforms to discover some skills of these experts. They mainly focus on identifying which technologies do software developers most work on, such as libraries, frameworks, and

languages [Singer et al., 2013; Teyton et al., 2013; Venkataramani et al., 2013; Teyton et al., 2014; Huang et al., 2016; Constantinou and Kapitsaki, 2016; Wang et al., 2017; Wan et al., 2018]. A few others have dedicated to discerning developers' soft skills, including communication, teamwork, responsibility, etc [Singer et al., 2013; Sarma et al., 2016; Ahmed, 2018]. Lastly, some studies aimed at identifying the roles of developers in open source projects, like bug fixer, bug triager, core developer, and tester [Ye and Kishida, 2003; Robles et al., 2009; Bhattacharya et al., 2014; Da Silva et al., 2015; Honsel et al., 2016; Agrawal et al., 2016; Joblin et al., 2017; Constantinou and Kapitsaki, 2017].

As observed in Chapter 3, IT companies organize their development teams accordingly to the technology the developers master, e.g., *frontend*, *backend*, *mobile*, and others. For instance, *frontend* developers are specialized on the application's interface, as opposed to *backend* who are responsible for core features. In this context, we currently lack approaches for inferring developers' technical roles. Therefore, our key goal in this chapter is to identify the technical roles played by developers using information available in open source platforms.

Does the Identification of Technical Roles Matter?

We define technical roles as the ones derived from expertise in particular programming technologies (e.g., programming languages) and/or architectural components (e.g., frontend frameworks). In this sense, distinguishing candidates' technical roles are important as they represent a good proxy for the technologies and techniques each one master. For instance, data scientists might have a deeper understanding of data analyzing techniques. By contrast, frontend developers should master Web APIs concepts, such as RESTFul APIs, AJAX, etc. In fact, source code hosting platforms—e.g., GitHub—are currently looking for alternatives to make their users profile richer by better expressing their skills, accomplishments, and interests.[1]

Indeed, technical roles are one of the first aspects considered by recruiters when hiring developers for a specific position. To provide evidence of this claim, we inspected the jobs listed by Stack Overflow Jobs[2]—the part of the Q&A forum where companies post job offers—on October 25, 2018. Our key finding can be summarized as follows:

> 3,234 out of 5,027 Stack Overflow job posts (64%) include in their description at least one of the technical roles investigated in this study.

[1] `https://twitter.com/natfriedman/status/1133700043695886336`
[2] `https://stackoverflow.com/jobs`

Additionally, we conducted a preliminary survey to better understand how GitHub is used when hiring software developers. For this, we contacted all Stack Overflow users self-described as technical recruiters who were active in the platform since 2018, and that publicly provided their e-mail for contact. We emailed these users with a brief survey where we asked:

> *Do you often use GitHub in your hiring process? If you do, please select the options that best describe your usage (you can mark multiple options).*

We provided four distinct options for the second question, which were formulated after consulting the literature about software expertise [Marlow and Dabbish, 2013; Sarma et al., 2016; Baltes and Diehl, 2018]. To avoid possible bias introduced by the order of the options, we configured our survey system to present them in random order for each contact. In total, we contacted 30 recruiters and received 7 answers (23% response rate). Six developers confirmed they use GitHub for hiring purposes. The survey results are summarized in Figure 5.1.

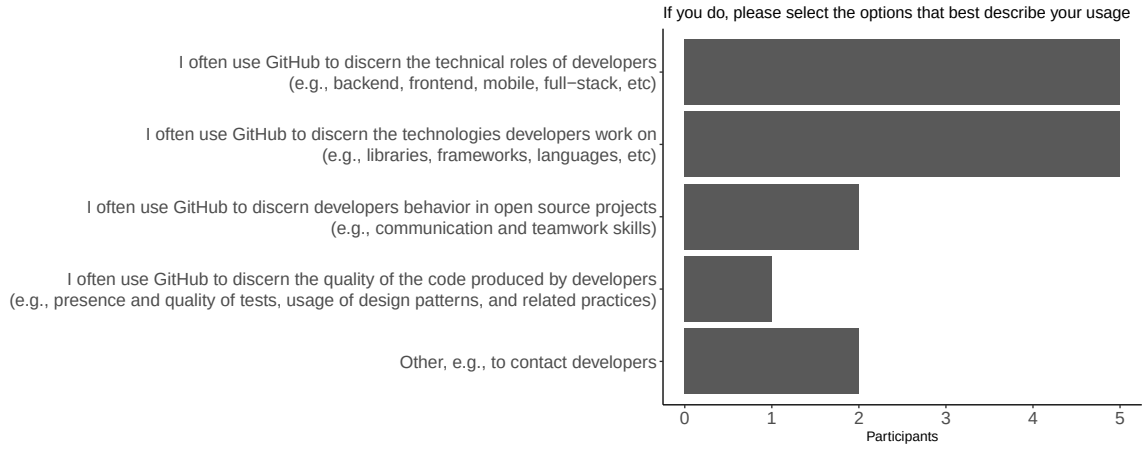


Figure 5.1: Survey responses from the recruiters that use GitHub in their hiring process.

Considering our intention in this chapter, the main finding of this survey can be summarized as follows:

> Five out of six technical recruiters use GitHub to discern the *technical roles* of the candidates, which is exactly our central goal in this investigation.

Although not directly linked with our goals here, another frequent usage of GitHub includes identifying which technologies do developers work on (also with 5 answers) and discerning developers' behavior in open source projects (2 answers).

Finally, we opted for focusing this work specifically on GitHub due to its importance in relation to other OSS platforms. Indeed, GitHub contains a larger userbase when compared to its peers; the platform has registered a total of 56M users in 2020, four times larger than Stack Overflow (14M). Furthermore, GitHub provides more diverse and detailed data sources to map developers' profiles, such as commits messages, source code lines added/removed, issue trackers messages, repositories metadata, etc.

Proposed Study

In this study, we investigate a method centered on supervised machine-learning techniques to identify developers' technical roles by considering their contributions to public GitHub projects. More specifically, we infer developers' technical roles by using features extracted from their public GitHub profiles (e.g., programming languages, short bio, etc) and their GitHub projects. We first build a ground truth with the technical roles of 2,284 GitHub developers. Then, we executed the proposed technique to identify developers in six popular technical roles:[3] *backend*, *frontend*, *full-stack*, *mobile*, *devops*, and *data science*. Lastly, we evaluated our method by answering four research questions.

RQ.1 How accurate are machine learning classifiers on identifying developers' technical roles? Our best model scored meaningful results for precision (0.75) and AUC (0.70), while it reported lower ones for recall and F1 (0.49 and 0.59, respectively). When considered independently, our models provide the best results for *data science* and *frontend* roles (0.86 and 0.77; precision). By contrast, we observed lower results for *backend* (0.62; precision).

RQ.2 What are the most relevant features to distinguish technical roles? Features associated with programming languages are relevant for all roles. Individually, *data science*'s top-10 most relevant features have the largest representativeness rate: 33.2%. By contrast, *backend* top-10 features scores only 6.8% of the total.

RQ.3 Do technical roles influence each other during classification? In this RQ, we investigate the gains achieved with the use of Classifiers Chains in our problem. Suppose that a developer is predicted as having a role R_1. After making this prediction, R_1 is used as input to the classifiers of R_2, R_3, R_4, and R_5 roles (assuming we analyze five roles, as in our first three RQs). Specifically, we check

[3]Accordingly to `https://insights.stackoverflow.com/survey/2018`

the improvements achieved with Classifier Chains by setting up and training 120 different classification models (i.e., covering all possible permutations of five roles). Overall, such models present minor improvements in recall (+0.05) at the cost of precision scores (−0.05) when compared to the ones in *RQ.1*.

RQ.4 How effectively can we identify full-stack developers? Differently from the technical roles analyzed in the first three RQs, the *full-stack* role is derived from the combined expertise in *backend* and *frontend* technologies. Therefore, we conducted a separated study to identify *full-stack* developers and verify how they impact the classification of *backend* and *frontend* developers. The proposed classifier performed very well when identifying FullStack, achieving 0.99 for precision and 0.71 for recall. Moreover, the addition of FullStack developers significantly improved the identification of both Backend and Frontend—0.87 and 0.86 for precision, respectively—after specific adjustments.

Our Contributions

We show that—based on high-level features retrieved from developers' public profiles and projects in GitHub—it is possible to infer major technical roles commonly used in industry to recognize software developers. Secondly, we make publicly available our ground truth with the roles of 2,284 GitHub developers. This ground truth can motivate and support further research in the area.

5.2 Study Design

5.2.1 Technical Roles

The focus of this work is to propose a method to unveil the technical roles of GitHub developers. We used the Annual Developer Survey conducted by Stack Overflow in 2018 to select the roles considered in our study. More than 100,000 developers from 183 countries answered this 30-minute survey where, among other questions, they answered which technical roles they associate with the tasks they normally perform. In this work, we study the top-4 most popular roles, according to this survey: Backend (58%), FullStack (48%), Frontend (38%), and Mobile (20%). Moreover, we included DevOps (10.4%) and DataScience (7.7%) roles in our analysis as both are also frequently featured in the top trending IT positions. Specifically, we first focus on five roles: Backend, Frontend, DevOps, DataScience and Mobile (*RQ.1*, *RQ.2*,

and *RQ.3*). Due to its particularity, we investigate the FULLSTACK role separately (*RQ.4*).

5.2.2 Ground Truth

As the proposed machine learning model relies on features extracted from GitHub to classify developers' technical roles, we fully avoided using GitHub's data to generate our ground truth. Instead, we relied exclusively on Stack Overflow's data to build this ground truth, since this data would not be used further in our study. Other works use data from Stack Overflow as a reliable source to assess developers' expertise [Venkataramani et al., 2013; Saxena and Pedanekar, 2017; Ahmed, 2018]. On the other hand, after building the ground truth, our goal is to identify the technical roles of GitHub users, i.e., by only considering GitHub data. We followed a three-step process to build the ground truth, as follows.

Stack Overflow Data Gathering

We used Stack Exchange Data Explorer (SEDE)[4] to collect Stack Overflow users with GitHub profiles. SEDE is a publicly available tool that allows querying data available in the Stack Exchange platform, using a web interface. On June 2nd, 2020, we queried SEDE for Stack Overflow users who have a link to GitHub. This query returned 27,051 developers. We then extracted—through regular expressions—the GitHub username from the provided links, and used GitHub GraphQL API[5] to retrieve GitHub data (as described in Section 5.2.3) for developers with valid usernames. During this procedure, we discarded developers with URLs pointing to invalid GitHub usernames. We ended up with a dataset composed of 24,889 developers.

Labeling

In this step, we analyzed the profile information provided by Stack Overflow to label developers' roles. Figure 5.2 depicts an example of Stack Overflow's user profile. Among the information available in his profile, this developer described himself as a "Lead Front End Developer". This information is endorsed by his top tags, e.g., *angularjs*, *javascript*, *jquery*, etc. Therefore, we relied on the description text of developers in Stack Overflow to determine their roles.

More specifically, we elaborated five distinct regular expressions to identify each role, as described in Table 5.1. These regular expressions were configured to consider

[4] `https://data.stackexchange.com/`
[5] `https://developer.github.com/v4/`

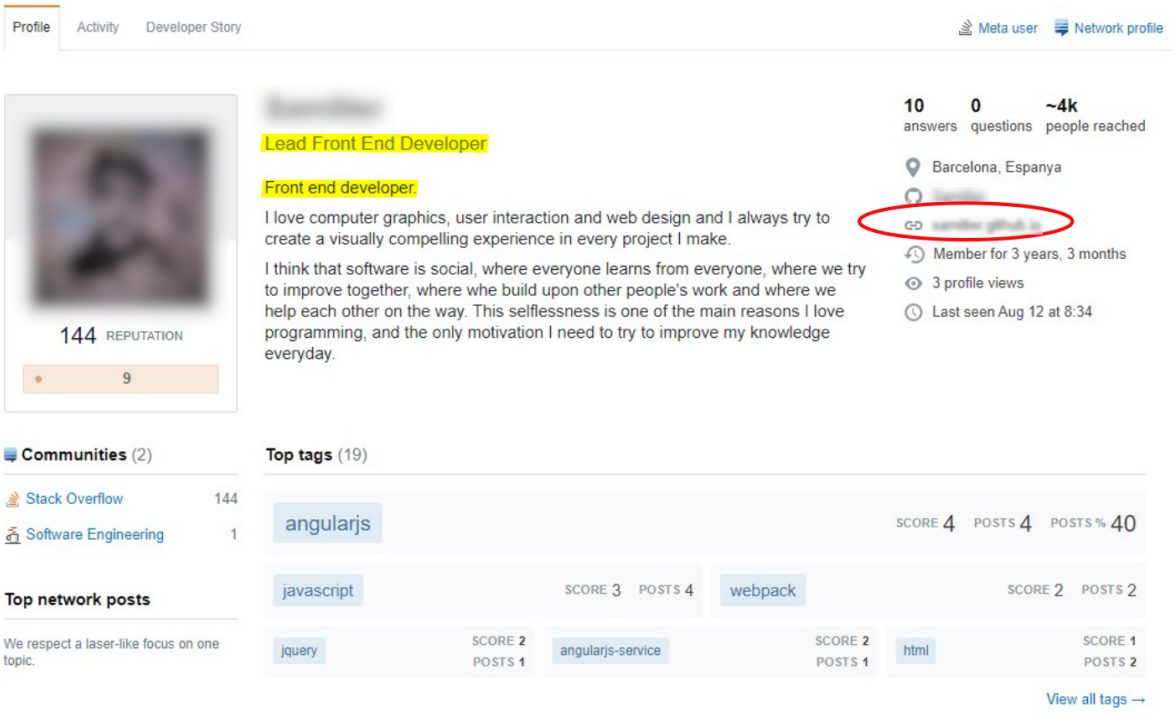


Figure 5.2: Example of Stack Overflow developer profile. The data used in the ground truth creation is highlighted.

Table 5.1: Regular expressions used to identify technical roles.

Role	Regular Expression	Examples
Backend	`/\bback.{0,1}end\b/i`	*Back end*, *backend*
Frontend	`/\bfront.{0,1}end\b/i`	*Frontend*, *front-end*
DevOps	`/\bdev.{0,1}ops\b/i`	*DevOps*, *dev-ops*
DataScience	`/\bdata.{0,1}scientist\b/i`	*Data Scientist*, *data scientist*
Mobile	`/\bmobile\b/i`	*mobile*, *Mobile*

spaces, e.g., *Back-end*, *Front end*) and to ignore case (e.g., *Mobile* and *mobile*. Considering the profile in Figure 5.2, our labeling process classify him as Frontend ("*Lead Front End Developer*" and "*Front end developer.*").

Selected Developers

By following the aforementioned steps, 1,802 developers were labeled with at least one technical role. We discarded 140 (7.7%) developers with less than five GitHub public repositories. The reason is that our models depend on developers being active on GitHub to infer their roles. Lastly, 1,662 developers were included in our ground truth, containing 2,022 role assignments.

Frontend is the role with more developers (820), followed by Mobile (453)

Table 5.2: Distribution of developers among the analyzed roles.

# Roles	Role					# Devs.
	Backend	Frontend	Mobile	DevOps	DataScience	
One	✓					178
		✓				545
			✓			352
				✓		119
					✓	146
Two	✓	✓				198
	✓		✓			29
	✓			✓		9
	✓				✓	1
		✓	✓			39
		✓		✓		4
			✓	✓		5
			✓		✓	1
				✓	✓	1
Three	✓	✓	✓			23
	✓	✓		✓		7
	✓	✓			✓	1
	✓		✓	✓		1
Four	✓	✓	✓	✓		3

and BACKEND (450). Table 5.2 shows how these labels are distributed. As we can observe, 1,340 developers (80%) are associated with just one role. Most of them are FRONTEND (545; 33%), or MOBILE (352; 21%). Considering the 287 developers who have two or more roles, the majority are both BACKEND and FRONTEND (198; 69%). By contrast, DATASCIENCE is the one with the smallest intersection with other roles, i.e., one developer in all situations. 32 developers were labeled in three roles, most of them in BACKEND, FRONTEND, and MOBILE (23; 72%). Only three developers were assigned in four roles. Finally, we found no developers who self-labeled themselves in all five roles.

5.2.3 Data Collection

After using Stack Overflow's data to leverage our ground truth, we collected GitHub's data for each developer in this golden set. This data will be later transformed into a list of features to feed our prediction models (see Section 5.2.4). To provide a code agnostic solution (and therefore analyze repositories in multiple programming languages), we

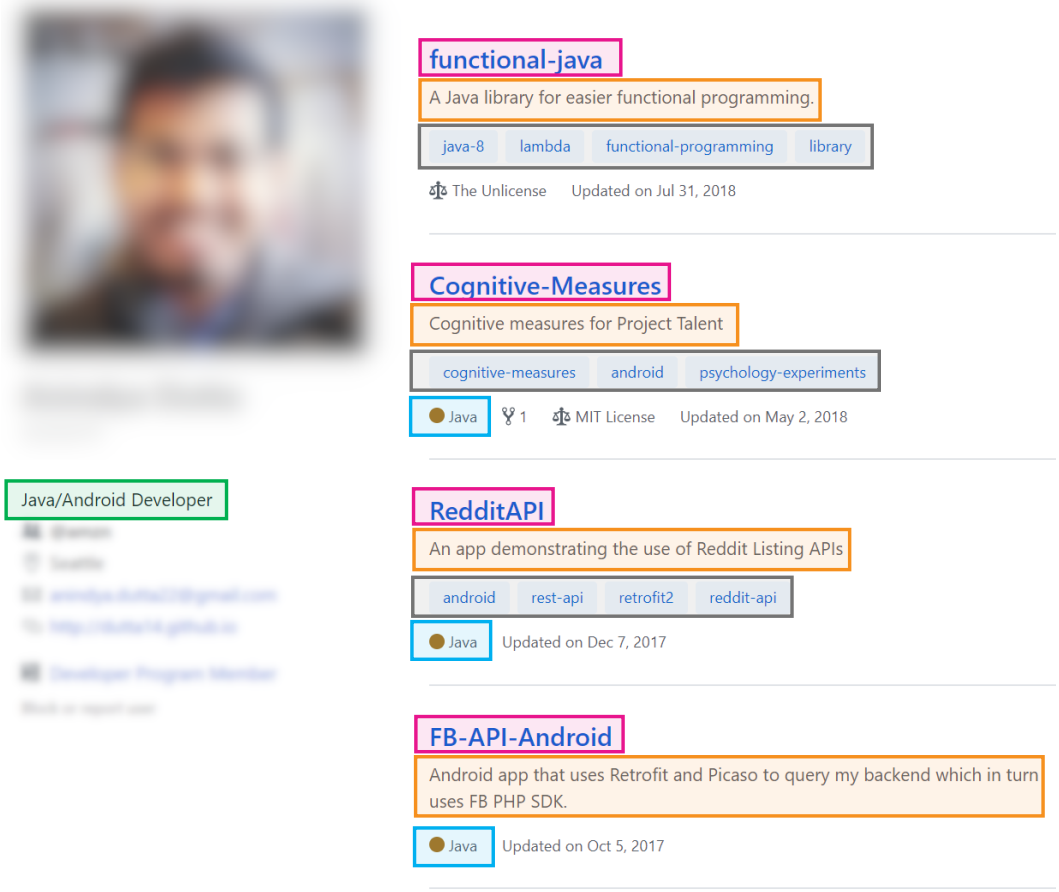


Figure 5.3: A developer profile from GitHub. Appart from *Projects' Dependencies*, the extracted data is highlighted according to each category: *Programming Languages* in blue, *Short Bio* in green, *Projects' Names* in purple, *Projects' Topics* in grey, and *Projects' Descriptions* in orange.

collected mostly textual data about the developers' profiles and the projects they own.[6] Figure 5.3 illustrates the data we collected for a given profile. The data we collected fits into six distinct categories:

Programming Language: Previous works report that programming languages are a reliable proxy to assess developers' expertise area [Marlow and Dabbish, 2013; Marlow et al., 2013; Sarma et al., 2016; Ford et al., 2017; Baltes and Diehl, 2018]. For each developer, we first retrieved her list of projects, and then extracted both *commits* and the *main programming language* of each one. Next, we derived the commits information into three dimensions: (a) the total number of commits; (b) the number of commits performed by the developer (i.e., she is the author); and (c) the rate between both of them. Lastly, we grouped the projects accord-

[6]It is important to note that we have discarded forked projects in our analysis.

ingly to the programming language collected earlier and averaged the resulting values. We ended up with the following features for each developer, in each language: *Language (total)*, *Language (author)*, and *Language (rate)*. This group is highlighted in blue in Figure 5.3.

Short Bio: Short description of the developers' main activities, manually provided by each one. We decided to include this category in our analysis as some developers use this information to describe their field of activity, e.g., "I develop iOS/Android apps with Swift/Kotlin language". We highlighted this group in green in Figure 5.3.

Projects' Names: These names may contain keywords referencing specific technologies used in the projects, e.g., *android-data-binding-1*, *docker-example*, etc. As observed in previous works, these technologies are also indicators of expertise [Teyton et al., 2014; Greene and Fischer, 2016; Saxena and Pedanekar, 2017; Montandon et al., 2019]. Therefore, for each developer, we collected the name of her GitHub projects and merged them into a single data point. This group is colored in purple in Figure 5.3.

Projects' Topics: For the same reason as above, topics are generally used by GitHub developers to declare technologies and tools used by their projects, e.g., *ionic*, *gulp*, and *webpack*. We also collected the GitHub topics of the developers' projects and merged them into a single data point as well. This group is highlighted in grey in Figure 5.3.

Projects' Descriptions: This information can also provide clues about the technologies used by GitHub projects [Hauff and Gousios, 2015]. For instance, the description "a React.js contact manager" indicates that *ReactJS* is used in the project. For this reason, we collected the short description of each developer's project and merged them into a single data point. This group is highlighted in orange in Figure 5.3.

Projects' Dependencies: We decided to include the list of dependencies since third-party libraries are largely used in modern software systems [Teyton et al., 2013; Hauff and Gousios, 2015; Huang et al., 2016; Montandon et al., 2019]. For each project, we extracted its list of dependencies using a GitHub GraphQL endpoint.[7] Next, we assembled these features by summing up the number of times each dependency was referenced in each developer's projects. We initially collected

[7] `https://developer.github.com/v4/object/dependencygraphmanifest/`

data on 17,556 dependencies, but we restricted ourselves to the top-1,000 most used ones as they are present in 81% of all analyzed projects.

5.2.4 Selected Features

To transform the data described in Section 5.2.3 into a list of features, which can be used as input for a machine learning algorithm, we applied the following transformation steps:

Correlation Analysis

Initially, we ended up with 477 and 1,000 features for *Programming Languages* and *Projects' Dependencies* categories, respectively. Due to this high number, we followed a correlation analysis procedure to remove the ones with high correlations in each category. For this, we used *corr* function from *pandas*[8] Python library to generate a Spearman correlation matrix. Then, we identified pairs of features with a correlation greater than 0.7, as previously adopted in the literature [Bao et al., 2017]. In such cases, we discarded the feature that has the highest correlation rate when compared to the other ones. In total, we discarded 260 features for *Programming Languages*. Generally these correlations happened with features belonged to the same language, but covering different domains, e.g., *C (author)* and *C (rate)*, *CSS (author)* and *CSS (rate)*, etc. In most cases, we maintain the *rate*-based ones since they usually have the lowest correlation rate with the others. As for *Projects' Dependencies*, we discarded 202. Differently, the discarded dependencies are more diverse, including specific (*ionic-native-statusbar*, a specific plugin for ionic framework) and general (*unicorn*, an http webserver) purpose libraries.

Bag-of-Words

Machine learning algorithms do not deal with text directly. For this reason, we used a bag-of-words to transform each textual category into a list of features. Bag-of-words is a simple and efficient technique to extract features from text [Goldberg, 2017]. In summary, each word is mapped to a feature that describes its frequency in a document. As in other studies [Shirabad et al., 2003; Kim et al., 2008; Beyer et al., 2018], we performed the following steps to apply this technique on *Short Bio*, *Projects' Names*, *Projects' Topics*, and *Projects' Descriptions*. First, we manually stripped out HTML tags, punctuation, and numbers through regular expressions. Then, we removed

[8] `https://pandas.pydata.org/`

stop words (e.g., of, him, the, and, etc.) using the default English list provided by *sklearn.text* module. Finally, we used *TfidfVectorizer*[9] class from *sklearn* library to apply bag-of-words using the TF-IDF outcome as feature values. Following common guidelines for text processing [Foster Provost, 2015], we considered only words in a certain document frequency range: [0.04, 0.15] for *Projects' Descriptions*, [0.03, 0.25] for *Projects' Names*, [0.01, 0.25] for *Projects' Topics*, and [0.01, 0.20] for *Short Bio*. These limits were defined after executing Random Forest classifier with 100 different bag-of-words configurations by randomly selecting different document frequency values in each configuration. We selected the configuration which presents improvements in most evaluated metrics. The result of each configuration is available in our replication package for further reference.

Number of Selected Features: After these steps (i.e., *Correlation Analysis* and *Bag-of-Words*), we ended up with 1,471 features, including 798 from *Projects' Dependencies* category, 217 from *Programming Languages*, 169 from *Projects' Descriptions*, 155 from *Projects' Names*, 69 for *Short Bio*, and 63 from *Projects' Topics*.

5.2.5 Machine Learning Setup

Multi-label Problem

Usually, classification problems rely on a single-label, i.e., each instance is associated with a single label. On the other hand, a multi-label classification problem can associate more than one label to each instance [Tsoumakas et al., 2011; Beyer et al., 2018]. Particularly, our dataset is a five-label classification problem since there are five distinct roles each developer can be an expert on. Table 5.3 illustrates this dataset. In this example, developer D_1 is associated with BACKEND and FRONTEND roles. Likewise, developer D_2 is tied with BACKEND and DEVOPS. On the other hand, D_3 is associated just with DATASCIENCE.

Problem Transformation

We can approach a multi-label classification problem in different ways. For instance, we can transform a multi-label dataset into a single-label one [Read et al., 2011; Luaces et al., 2012]. Alternatively, some algorithms work directly with multi-label

[9] `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer`

Table 5.3: Dataset with five target labels.

Developer		Backend	Frontend	Mobile	DevOps	DataScience
D_1	$\cdots$	1	1	0	0	0
D_2	$\cdots$	1	0	0	1	0
D_3	$\cdots$	0	0	0	0	1
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
D_n	$\cdots$	0	0	1	0	1

datasets [Tsoumakas et al., 2011]. For this investigation, we use two common transformation techniques for dealing with multi-label datasets:

Binary Relevance (BR): This technique splits the original dataset into multiple binary classification problems, i.e., one binary classification problem for each label [Tsoumakas et al., 2010; Luaces et al., 2012]. Then, such models are fitted independently. The results can be shown either independently (one result for each model) or aggregated (weighted average of all models). In this study, we calculated the aggregated results using a *micro-average* strategy, as it is recommended for multilabel problems.[10]

Classifier Chains (CC): This technique links the binary classifiers along a chain in a way that each classifier uses results from earlier ones in its predictions [Read et al., 2011]. The goal is to take advantage of eventual dependencies that might exist among target labels. Figure 5.4 illustrates a classification scenario using *CC* considering five technical roles: Mobile, Frontend, Backend, DevOps, and DataScience. Each model includes previous predictions as features and propagates its results along the chain. Figure 5.4a presents the first model in the chain (Model I). As we can see, this model makes its predictions using only the available features. Model II (Figure 5.4b) predicts its values using all features plus Model I predicted labels. The process is propagated to Model III, using all features plus both Models I and II predicted labels (Figure 5.4c). As shown in Figure 5.4d, the results of Models I, II, and III are forwarded to Model IV along with the already available features. Finally, this propagation ends up at Model V, which receives the predictions from all previous models plus the initial features as input (Figure 5.4e). In this way, *CC* tackles the label independence problem faced by *BR* [Madjarov et al., 2012; Zhang and Zhou, 2014].

[10] `https://scikit-learn.org/stable/modules/model_evaluation.html#from-binary-to-multiclass-and-multilabel`

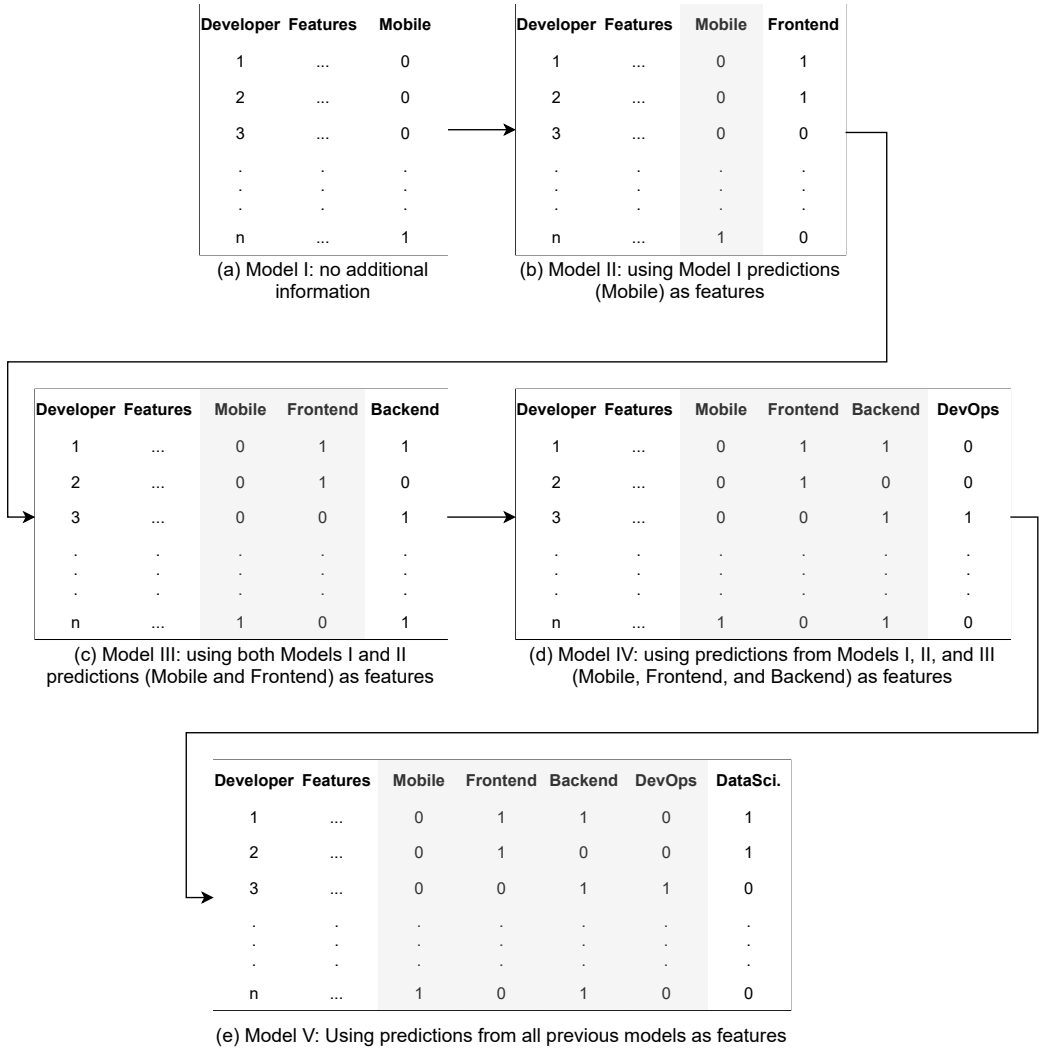


Figure 5.4: Multi-label classification using Classifier Chains.

Machine Learning Classifier

We initially decided to use Random Forest [Breiman, 2001] and Naive Bayes [Maron, 1961] classifiers to train and test our models for identifying the technical roles of GitHub users. We selected Random Forest due to its robustness to noise and outliers [Tian et al., 2015]. Moreover, Random Forest was successfully used in many application areas, including software engineering problems [Menzies et al., 2013; Peters et al., 2013; Provost and Fawcett, 2001; Coelho et al., 2018; Bao et al., 2017; Beyer et al., 2018]. As we are dealing mostly with textual information, we also decided to include Naive Bayes in our study as this classifier generally presents good results in textual scenarios, such as spam filtering [Metsis et al., 2006] and news classification [McCallum and Nigam, 1998]. We used the implementation provided by *scikit-learn* [Pedregosa et al., 2011] for both classifiers along with 10-fold cross-validation to select the best

model. Basically, cross-validation splits the data into k folds (in our case, $k = 10$), where $k - 1$ folds are used to fit the model and the remaining one is used to test the predictions.

Evaluation Metrics

We use six metrics to evaluate the quality of the model's predictions: precision, recall, F1-score, AUC (Area Under the Curve), Jaccard Coefficient, and Hamming Loss. Precision measures the correctness, while recall measures the completeness of the model predictions. F1-score is the harmonic mean of precision and recall. AUC represents the area under the Precision-Recall (PR) curve. Unlike ROC, PR curves are more recommended for assessing unbalanced datasets [Foster Provost, 2015]. Jaccard Coefficient—also known as multi-label accuracy [Boutell et al., 2004]—is the division of the number of correctly predicted labels by all true labels. Hamming Loss is the ratio of wrong labels by the number of labels [Dembczyński et al., 2012]. As it is a loss measure, the lower the value, the better the model. Finally, we compared all metrics against a Stratified baseline. This baseline considers the dataset distribution to perform its predictions. For instance, in a dataset where 10% of the samples are positive labels, this baseline limits its positive predictions to 10% as well.

5.3 Results

(RQ.1) How accurate are machine learning classifiers in identifying technical roles?

Table 5.4: Binary Relevance overall results.

Metric	**Stratified Baseline**	**Random Forest**	**Naive Bayes**
Precision	0.33	0.77	0.51
Recall	0.32	0.49	0.62
F1	0.33	0.59	0.56
AUC	0.41	0.71	0.59
Jaccard Coeff.	0.20	0.42	0.39
Hamm. Loss	0.32	0.16	0.24

Table 5.4 presents the general results for the first three roles using Binary Relevance (BR). The Random Forest classifier presented the best results overall, scoring

0.77 for precision and 0.71 for AUC. Moreover, apart from recall, Random Forest outperformed Naive Bayes for all evaluated metrics. Finally, both classifiers performed significantly better than the Stratified Baseline. The baseline results are, for example, only 0.33 for precision.

Table 5.5 presents the classification results for each role, separately. Overall, Random Forest presented better results for precision when compared to Naive Bayes, and both outperformed the Stratified Baseline. Regarding Random Forest, the classifier presented high precision rates—i.e., above 0.7—for 4 out of 5 roles: DATASCIENCE (0.86), MOBILE (0.78), FRONTEND (0.77) and DEVOPS (0.70). On the other hand, only FRONTEND achieved results as good as for recall (0.78) and F1 (0.77). Further, the classifier scored poor results especially for BACKEND role (recall and F1 equal to 0.12 and 0.18, respectively).

Table 5.5: Binary Relevance results for each role.

Role	Precision	Recall	F1
Stratified Baseline			
BACKEND	0.28	0.28	0.28
FRONTEND	0.48	0.46	0.47
MOBILE	0.28	0.28	0.28
DEVOPS	0.08	0.06	0.07
DATASCIENCE	0.09	0.09	0.09
Random Forest			
BACKEND	0.62	0.12	0.18
FRONTEND	0.77	0.78	0.77
MOBILE	0.78	0.38	0.51
DEVOPS	0.70	0.13	0.20
DATASCIENCE	0.86	0.66	0.74
Naive Bayes			
BACKEND	0.33	0.40	0.36
FRONTEND	0.69	0.82	0.75
MOBILE	0.51	0.46	0.47
DEVOPS	0.24	0.47	0.32
DATASCIENCE	0.45	0.85	0.58

When it comes to Naive Bayes, we did not identify any score above 0.7 for precision; FRONTEND has the highest one (0.69), followed by MOBILE (0.51) and DATASCIENCE (0.45). In fact, we observed such good results in three scenarios only: FRONTEND and DATASCIENCE for recall (0.82 and 0.85, respectively), and FRONTEND for F1 (0.75). Even though, Naive Bayes was superior to the baseline in all scenarios.

> Random Forest presented the best results overall (e.g., 0.77, precision; 0.71, AUC), outperforming both Naive Bayes and the baseline. When analyzed individually, DataScience and Frontend scored the best results for most metrics (e.g., 0.86 and 0.77; precision), while Backend showed the worse ones (e.g., 0.62; precision).

(RQ.2) What are the most relevant features to distinguish technical roles?

Figure 5.5 shows the top-10 most relevant features by technical role. To identify these features, we executed the Random Forest classifier for each role independently, using the same parameters as in *RQ.1*. We then selected the top-10 features with the highest value from the feature relevance ranking provided by each model. In Figure 5.5, the colors and shapes identify the category of each feature (programming languages, projects' names, projects' descriptions, projects' topics, projects' dependencies, or short bio). Furthermore, we annotate the category associated with each feature in its description. Finally, we normalized the ranking with respect to the feature with the highest value.

Features associated with *programming languages* are largely predominant for all five roles, representing 38 out of 50 features: 7 for DataScience, DevOps, and Mobile; 8 for Frontend, and 9 for Backend. From these, 19 (50%) are rate-based, 13 (34%) consider the total number of commits, and 6 (16%) include only the commits performed by the author. In other words, 66% of the relevant *programming languages* features do take into account actual developers' contributions.

Next, *short bio* appears as the second most frequent with 6 occurrences, followed by *projects' descriptions* and *projects' names* (3 and 2 occurrences, respectively). Nevertheless, *short bio* shows up as the most important feature in three roles. Most of these features directly describes the role that is analyzed: *scientist* for DataScience, *devops* for DevOps, *mobile* for Mobile, and *backend* for Backend. Surprisingly, *projects' topics* and *projects' dependencies* are not present in any ranking position.

When we analyze the results for each role, we see that DataScience has the highest relevant features. Six features stand out with more than 3% of relevance rate: *scientist (Bio)* (6.5%), *Jupyter Notebook (total)* (5.4%), *Jupyter Notebook (rate)* (5.1%), *data (Bio)* (5.0%), *R (total)* (4.4%), and *R (rate)* (3.5%). Although these values may sound low, our model includes 1,662 features. Also, these features are rather specific to the DataScience field; from them, only *Jupyter Notebook (total)* is also present in other roles (Frontend).

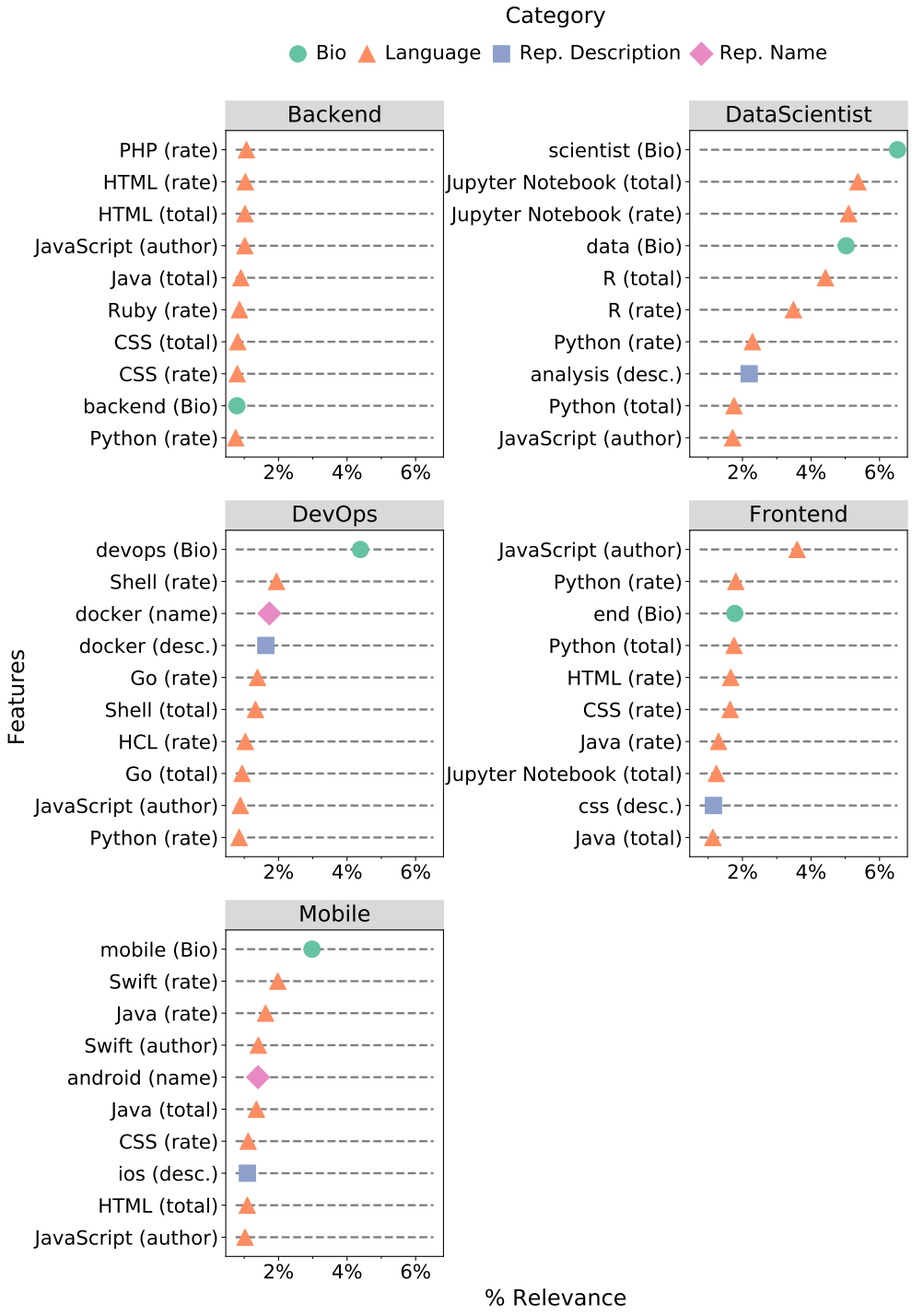


Figure 5.5: Most relevant features for each technical role. Colors and shapes represent each feature category: developers bio ((*Bio*), green circle), projects' descriptions ((*desc.*), blue square), projects' names ((*name*), pink diamond), and programming languages ((*rate*, *author*, and *total*), orange triangle)

For Mobile, *mobile (Bio)* stands out with 3.0% of relevance rate, followed by *Swift (rate)* (2.0%) and *Java (rate)* (1.6%). More specifically, three out of 10 features are directly associated to the iOS development platform (*Swift (rate)*, *Swift (author)*, and *ios (desc.)*). Likewise, other three features are linked to the Android platform: *Java (rate)*, *android (name)*, and *Java (total)*. The features related to iOS and Android development represent 4.5% and 4.3% of the top-10 listed features, respectively.

Regarding DevOps, we observe that *devops (Bio)* plays a prominent position in the ranking, with 4.4%. The next ones are *Shell (rate)* (1.9%), *docker (name)* (1.7%), and *docker (desc.)* (1.6%). Frontend ranking presents a similar distribution, where *JavaScript (author)* is the prominent feature with 3.6%, whereas the next five are valued between 1.8% (*Python (rate)*) and 1.6% (*CSS (rate)*).

Finally, we observe that no feature stands out in Backend ranking. The most relevant feature is *PHP (rate)*, with 1.1%, while *Python (rate)* is the 10th feature in the ranking, with 0.8%. Even though, 5 out of the top-10 features are linked with backend development: *PHP (rate)*, *Java (total)*, *Ruby (rate)*, *backend (Bio)*, and *Python (rate)*.

> Features related to *programming languages* are predominant for all five roles. In DataScience role, six features stand out: *scientist (Bio)*, *Jupyter Notebook (total)*, *Jupyter Notebook (rate)*, *data (Bio)*, *R (total)*, and *R (rate)*. On the other hand, fewer features are presented such importance in other roles: *devops (Bio)* for DevOps, *mobile (Bio)* for Mobile, *JavaScript (author)* for Frontend. Lastly, no feature stands out from the others for Backend.

(RQ.3) Do technical roles influence each other during classification?

As explained in Section 5.2.5, Classifier Chains (*CC*) is a technique to deal with multi-label classification problems. When using this technique, Binary Classifiers are ordered in such a way that each model uses previous predictions as input. The rationale is that, given two labels A and B, if A influences B then we might want to use A to improve B's predictions.

Particularly, we executed the *CC* technique with Random Forest for all possible role combinations. In total, the classifier was executed for 5! combinations, i.e., 120 different combinations. Figure 5.6 presents the results in six different graphics, one for each metric considered in this study. For comparison reasons, the results are reported in relation to *RQ.1*, which is used as a baseline in this RQ and it is represented by a

dashed line in the figure. The horizontal axis represents each *CC* configuration and the vertical axis the number of points each metric has above or below the baseline. This means that the *CC* technique performed better than the baseline when its results are above the dashed line, or worse otherwise.

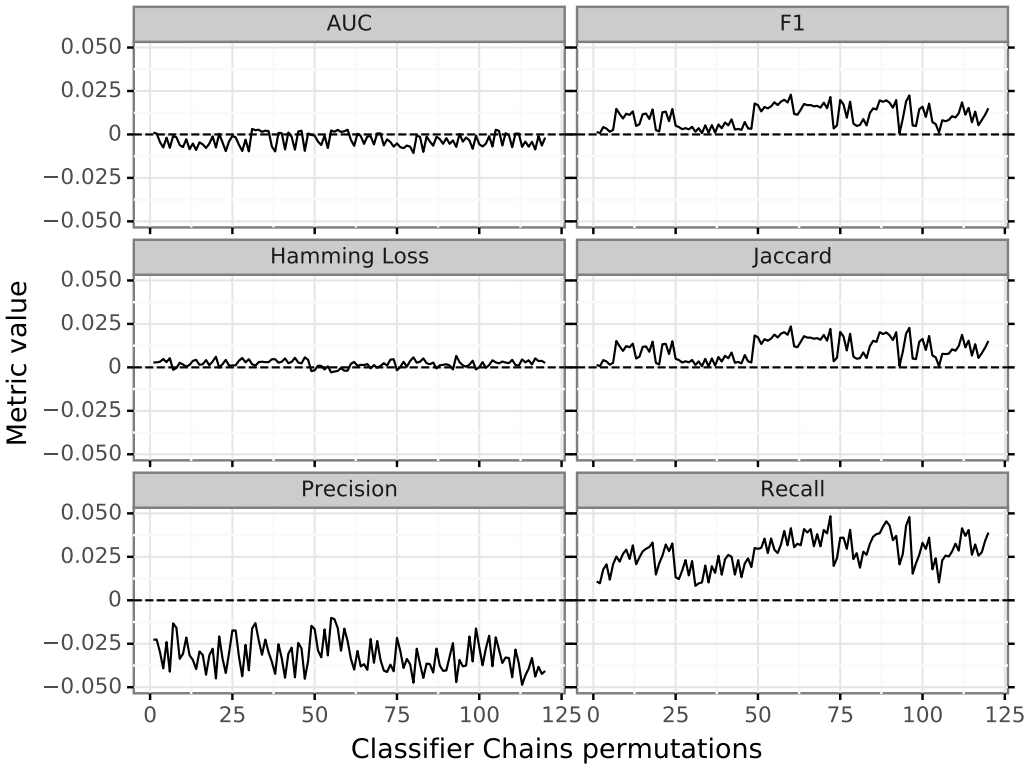


Figure 5.6: Overall results using Classifier Chain, reported in relation to the *RQ.1* baseline (i.e., dashed line).

Overall, all models show a minor increase of recall up to almost +0.05 (configurations 72, 89, and 96), at a cost of lower precision (−0.05; configurations 47, 80, 93, and 113). We also observe minor improvements for both F1 and Jaccard Coefficient (+0.02, configurations 60, 72, and 96; both), while AUC presented a minimal decrease (−0.01, configurations 80 and 83). By contrast, we did not find any relevant change to the Hamming Loss score. Interestingly, the highest improvements for recall, F1, and Jaccard—and also some of the minor setbacks for precision—happened between configurations 49 and 60, when MOBILE is classified first, followed by BACKEND or FRONTEND.

> The results reveal that including technical roles predictions to the classifying process does not bring significant improvements. Overall, we obtained minor improve-

ments for recall, Jaccard, and F1 on the cost of precision. These results are more consistent when Mobile is classified first, followed by Backend or Frontend.

5.4 Understanding the FullStack Role

Differently from the previously analyzed roles, FullStack developers are defined by their ability to work through the whole application stack. In fact, the industry sees a FullStack developer as someone who can perform both Frontend and Backend tasks. The following quote, retrieved from a popular Medium post,[11] illustrates this point:

> *Being a Full-Stack developer [...] means that you are able to work on both sides and understand what is going on when building an application.*

Due to this particularity, we decided to investigate the FullStack role in a specific section. First, we extended the ground truth described in Section 5.2.2 to include developers self-identified as FullStack on Stack Overflow.[12] The new ground truth is composed by 2,284 developers; 853 of them are FullStack. From these, 783 were not labeled either as Backend or Frontend, which contradicts the FullStack's definition. To fix this inconsistency, we adapted the labeling process to consider every FullStack developer as an expert in both Backend and Frontend roles.

After applying the same process described in Sections 5.2.3 and 5.2.4, this extended dataset ends up with 1,567 features: 819 from *Projects' Dependencies* category, 219 from *Programming Languages*, 212 from *Projects' Descriptions*, 146 from *Projects' Topics*, 101 from *Projects' Names*, and 70 from *Short Bio*.

(RQ.4) How effectively can we identify full-stack developers?

Table 5.6 presents the classification results for Random Forest using the Binary Relevance approach (*RQ.1*) and including FullStack. Besides, we included the results for each of the five previous roles, along with the difference for the one described in RQ.1.

Overall, all metrics reported significantly better results after including FullStack developers in the ground truth. Precision, recall and F1 increase to 0.88

[11] `https://medium.com/coderbyte/a-guide-to-becoming-a-full-stack-developer-in-2017-5c3c08a1600c`, Accessed at 2019-04-04.

[12] We used the following regex to label developers as FullStack: `/\bfull.{0,1}stack\b/i`.

[13] Overall value is calculated using the same strategy as described in Section 5.2.5.

Table 5.6: Binary Relevance results after including FULLSTACK, Random Forest only.

Role	Precision	Recall	F1
FULLSTACK	0.99	0.71	0.83
BACKEND	0.87 (+0.25)	0.63 (+0.51)	0.73 (+0.55)
FRONTEND	0.86 (+0.09)	0.89 (+0.11)	0.87 (+0.10)
MOBILE	0.80 (+0.03)	0.34 (−0.04)	0.47 (−0.04)
DEVOPS	0.75 (+0.05)	0.06 (−0.07)	0.11 (−0.09)
DATASCIENCE	0.86 (+0.00)	0.62 (−0.04)	0.71 (−0.03)
Overall[13]	0.88 (+0.11)	0.69 (+0.20)	0.77 (+0.18)

(+0.11), 0.69 (+0.20), and 0.77 (+0.18), respectively. Likewise the new AUC, Jaccard Coefficient, and Hamming Loss reach 0.89 (+0.18), 0.69 (+0.10), and 0.13 (−0.03), respectively.

Considering each role, we observe that FULLSTACK presented very high results: its precision is 0.99, recall is 0.71, and F1 is 0.83. When it comes to the other roles, we noticed an interesting improvement for both FRONTEND and BACKEND. FRONTEND's value for F1 increased from 0.77 to 0.87 (+0.10), mainly due to recall improvement (from 0.78 to 0.89; +0.11). Most important, BACKEND gained 55 points in F1 when compared to the one in *RQ.1* (from 0.18 to 0.73). Recall is also largely responsible for this difference as it scored 0.63 against 0.12 before; an increase of 51 points. By contrast, we observe a minor negative impact in F1 for MOBILE (−0.04), DEVOPS (−0.09), and DATASCIENCE (−0.03). Even so, the precision score for these roles has either increased (MOBILE, +0.03; DEVOPS, +0.05) or stayed the same (DATASCIENCE).

We accredit this side effect as a consequence of expanding the dataset with the new FULLSTACK developers. Particularly, in our second dataset, we found several FULLSTACK developers with missing BACKEND or FRONTEND labels (more precisely, 783 developers, as we mentioned before). In other words, once they are FULLSTACK, for these developers it is implicit that they should be also viewed as BACKEND or FRONTEND. Consequently, after labeling FULLSTACK developers as BACKEND and FRONTEND we provided new information to the classifiers, which allowed them to improve their predictions.

> The proposed classifier performed very well when identifying FULLSTACK developers (*precision* = 0.99). Moreover, FULLSTACK developers have an interesting collateral effect, since they contributed to improving results of both BACKEND (+25%) and FRONTEND (+9%).

5.5 Discussion

In this section, we start discussing the implications of our work to both academia and practitioners (Section 5.5.1). Then, we argue about how the results for precision and recall should be interpreted in the context of this study (Section 5.5.2). Lastly, we analyze the performance of our method by inspecting developers profiles in three different scenarios (Section 5.5.3).

5.5.1 Implications

We shed light on the importance of unveiling technical roles; an expertise topic not yet extensively investigated by the software engineering community. More specifically, we performed two preliminary studies to demonstrate the importance of this topic. First, we inspected posts from the Stack Overflow Jobs platform and find out that 64% of 5,027 offers are for one of the six technical roles studied in this study. Second, we surveyed technical recruiters to understand which characteristics do they look for in GitHub profiles. Five out of six recruiters indicated they search for clues to discern the *technical roles* of the candidates.

Besides, the proposed machine learning method has most of its usage in hiring processes. In this context, when hiring software developers, technical recruiters can benefit from the technical roles inferred by the proposed models. This usage might occur in two main ways. First, by *proactively* identifying developers with the skills expected by existing job positions in the company. Secondly, by *reactively* evaluating the profile of candidates who have already applied to existing job positions, to assure they have the expected skills. In both cases, the inferred technical roles should be used as a piece of additional information during the hiring process, which certainly includes other selection instruments, such as technical interviews, reference letters, etc.

5.5.2 A Note on Precision and Recall

In general, our method showed its effectiveness in revealing the technical roles of software developers given their GitHub profiles. For instance, our model scored 0.88 for precision and 0.89 for AUC when considering all six roles analyzed in this study (see Table 5.6); by contrast, we achieved a lower result for recall (0.69). When analyzed individually, we observe that the precision for each role ranges from 0.75 (DevOps) to 0.99 (FullStack). In fact, except for DevOps all other roles achieved at least 80% of precision. On the other hand, only Frontend reached a similar value for recall (0.89); the other ones remained between 0.06 (DevOps) and 0.71 (FullStack).

Essentially, precision answers the following question *"What proportion of positive identifications was actually correct?"*, whereas recall seeks to answer *"What proportion of actual positives was identified correctly?"*.[14] In the context of technical recruiters, we advocate that precision is more important than recall as they normally need to identify a small set of candidates for the existing positions; i.e., they do not need to locate all developers matching their job positions in the GitHub universe. Furthermore, companies avoid at most a false positive (a bad hiring), as it is more expensive [McDowell, 2015; Behroozi et al., 2019].

5.5.3 Manual Analysis

We manually inspected developers' profiles to analyze their predictions so we can better understand the proposed classification results. Specifically, we analyze developers in three distinct scenarios: correct classifications (True Positive); wrongly classified in a given role (False Positive); and not classified in a given role, despite being so (False Negative).

We select three developers and manually analyze their predictions to better understand the proposed classification results. Specifically, we analyze D_{844}, correctly classified as MOBILE developer (true positive); D_{1341}, wrongly classified as DEVOPS (false positive); and D_{68}, who is not classified as FRONTEND developer, despite being labeled as one (false negative).

True Positive Scenario

Developers in this group have had their roles correctly predicted by our method. In this scenario, we inspected developer D_{844}, classified as MOBILE Developer. D_{844} owns 22 public GitHub projects, most of them related to mobile development; four projects have *Swift* as their main programming language, and other five have the word *Android* in their description. Moreover, another four projects are implemented in Java and contain experiments about chart animations in Android. In total, 19 out of 22 projects are directly related to mobile development. In his personal website, D_{844} describes himself as "Android & iOS Engineer | App Maker". Through D_{844}'s GitHub page, we can also reach his LinkedIn profile and find out he has been working with mobile development since 2014.

[14] Questions retrieved from `https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall`

False Positive Scenario

This group contains developers wrongly predicted as experts in any of the analyzed roles. We inspected the profile of D_{1341}, which is not a DevOps developer, but was predicted as such. In his LinkedIn profile, D_{1341} identifies himself as a "Data Engineer", also mentioning he works in this position since 2016. D_{1341} received most endorsements in *Python* (19), *Data Science*, and *Machine Learning* (18, both). By contrast, D_{1341} has few endorsements in DevOps-specific technologies: one for *AWS* and two for *Linux*. He also obtained certifications in "Machine Learning" and "Scalable Microservices with Kubernetes". D_{1341} is very active on GitHub, performing 711 contributions in the last year. This developer owns 30 projects in 13 different languages, most of them in *Python* (five projects). Further, *Docker* appears in the description of two other projects described as an assistant tool for data analysis. Therefore, due to his limited experience with DevOps-based tools and languages, D_{1341} should not be classified as a DevOps.

False Negative Scenario

Developers are considered False Negative when they are not classified as experts in a given role, despite being so. For this scenario, we examined D_{68}, who is a Frontend developer, but was not identified as such. In his short bio, D_{68} describes himself as a "Frontend Engineer". On GitHub, this developer has performed 1,018 contributions over the last year, where 51% of them were commits. Even though, D_{68} is the owner of only 12 projects; five of them are directly related to frontend technologies (e.g., *JavaScript*, *vue.js*, etc). The remaining projects are implemented in other languages, such as *Java*, *PHP*, and *Python*. Moreover, his projects' descriptions do not mention the use of Frontend technologies. In fact, five projects do not contain any description at all. In other words, the *programming languages* and *short bio* used by D_{68} are the only features directly related to Frontend, but their presence was not enough to label him correctly.

False negative predictions gather our attention as they directly impact the recall's performance. As our classifiers scored lower values for this metric in Mobile and DevOps roles, we decided to extend this analysis by inspecting 10 new randomly selected developers, five in each role. Overall, we observe these developers do not hold specific information to classify them in such roles. For instance, the five selected DevOps developers maintain 70 projects in total. However, only one is mainly implemented using *Shell Script*, which is the second most relevant feature for this role (see Figure 5.5). One developer mentioned the *devops* keyword in his bio, but this was not reinforced by

the other relevant features; only one of his 18 projects is implemented using a relevant language (*Python*). Similar behavior is noted when analyzing MOBILE developers. The majority of the projects analyzed are not written in MOBILE-based languages, such as *Swift* and *Java*. Interestingly, most of them are implemented with FRONTEND-based ones, such as *JavaScript* and *TypeScript*. We analyzed this characteristic further and found out that some of these projects rely on cross-platform MOBILE frameworks—e.g., *cordova*, *ionic*, etc—which are not in the top-10 most relevant features. In fact, two out of the five developers explicitly stated they are FRONTEND developers with experience in building mobile web apps. Lastly, relevant keywords like *android*, *ios*, and *mobile*, are indeed mentioned in the repositories of one developer but in a format that prevents their extraction by our bag-of-words tokenization process, e.g., *iOSUIAutomation*, *MyFirstAndroid*, etc.

5.6 Threats to Validity

The following issues are possible threats to our results:

Target Roles: We analyzed six technical roles in this research. Although they represent a restricted number of roles, we selected the ones among the most popular, based on a recent large-scale survey conducted by Stack Overflow. Moreover, they cover 64% of the jobs recently listed at Stack Overflow jobs, as mentioned in the Introduction.

Ground Truth: Our dataset is restricted to developers who have profiles on both Stack Overflow and GitHub. Besides, the ground truth is based on developers' public activities on GitHub. Evidently, they represent a subset of all activities of several developers. This limitation may increase false negatives in particular scenarios. For example, a developer is *mobile* but the publicly available data characterizes him/her as *backend* developer. A potential direction for tackling this problem could be the usage of developers' activities from additional data sources. Furthermore, we followed an automated process to label the 2,284 developers in the ground truth. This process is subjected to bias since it relies on the description provided by the developer on Stack Overflow. Further work should consider a semi-automatic labeling strategy [Vajda et al., 2015] or semi-supervised learning techniques [Chapelle et al., 2006] to evaluate the accuracy of our labeling process.

Multi-label Classification: Other classification techniques can be applied in multi-label problems, e.g., Label Powerset [Herrera et al., 2016]. However, we rely on the two most used techniques to handle multi-label classification. Moreover, we used two well-known classification algorithms—Random Forest and Naive Bayes. Despite that, further work should consider other classification algorithms, such as XGBoost [Chen and Guestrin, 2016].

Thresholds: As usual in machine learning studies, we acknowledge that our results depend on different thresholds, which are used for example to discard correlated features (Spearman ≥ 0.7), to limit the number of features used in projects' descriptions ([0.04, 0.15]), projects' names ([0.01, 0.25]), projects' topics ([0.01, 0.25]), and short bio ([0.01, 0.20]), as reported in Section 5.2.4. As a general guideline, we always use conservative thresholds. Furthermore, we experimented with other threshold values in all cases and selected the ones that presented the best results.

5.7 Final Remarks

The increasing complexity and relevance of modern software systems are fostering the specialization of software developers in particular components and technologies. As a result, when hiring developers, companies usually do not look for developers with a broad range of skills, but for ones who can work with specific technologies and in specific tasks. Motivated by this context, in this chapter we described a method centered on supervised machine learning to predict six widely popular technical roles of developers nowadays: Backend, Frontend, Mobile, FullStack, DataScience, and DevOps. Using features extracted from the public profiles of GitHub users, we obtained great results for identifying all six roles in terms of precision (0.88) and AUC (0.89). By contrast, we observed lower results for DevOps and Mobile regarding recall, e.g., 0.06 and 0.34, respectively. Even so, we believe that this study can offer good assistance to technical recruiters as, in their context, identifying correct candidates (precision) is more relevant.

Replication Package: Our data and our scripts—in a Jupyter Notebook format—are publicly available at the following URL: `https://doi.org/10.5281/zenodo.3986172`.

Chapter 6

Conclusion

In this chapter, we briefly describe the research we conducted throughout this thesis in Section 6.1. Next, we list the main outcomes of this work in Section 6.2. Finally, we wrap-up this manuscript by outlining further work in Section 6.3.

6.1 Thesis Recapitulation

In recent years, developers have been playing an increasingly preponderant role in the software development process. Indeed, IT-based companies are giving importance to hiring new professionals at unprecedented levels. When it comes to the state-of-art in this field, most of them investigate developers' expertise in particular domains, i.e., towards specific open-source projects. By contrast, the industry is more interested in obtaining software developers' information from a more general perspective. We report in this thesis a set of three major studies where we extensively analyzed data-driven methods and techniques to leverage software developers' profiles, considering their activity in Social Coding Platforms.

We started by defining what expertise is and the difficulties in interpreting it in the context of software developers (Chapter 2). We also discussed the most common techniques proposed in the literature to mine activity information from software repositories and to apply machine learning methods in the software engineering domain. Finally, we described the state-of-the-art concerning the application of mining software repositories techniques in Social Coding Platforms to unveil the skill of software developers.

Next, we reported in Chapter 3 a large-scale investigation with more than 20,000 job advertisements to understand which soft and technical skills do IT companies look for when hiring new employees. To analyze technical skills, we applied open card sorting

to perform a high-level analysis on which type of skill is more requested by 14 distinct professional roles. As for soft skills, we conducted a statistical analysis to reveal the most mentioned ones. Overall, we observed that development-based roles frequently require software developers to master both programming languages and third-party libraries relevant to their activity field. Further, we reinforced the importance that communication and teamwork-based soft skills have to IT companies.

In Chapter 4, we analyzed the performance of machine learning and data mining techniques to identify developers' expertise level in three widely popular JavaScript libraries: *ReactJS*, *socket.io*, and *mongodb*. For this, we first surveyed 575 software developers with GitHub profiles and asked them to provide their expertise in one of the mentioned libraries. Next, we derived 13 low-level features from their GitHub activity—e.g., the number of commits, time between first and last commit, etc—and evaluated the performance of both supervised and unsupervised methods. We documented the challenges we faced during this study and evaluated the effectiveness of a clustering strategy by triangulating its results with the information available in developers' LinkedIn profiles.

Finally, we investigated the use of machine learning methods to automatically identify the technical roles of open source developers in Chapter 5. We built a ground truth with 2,284 developers labeled in six different roles: *backend*, *frontend*, *full-stack*, *mobile*, *devops*, and *data science*. By relying on high-level features—such as projects' names, programming languages used, etc—we proposed three machine learning strategies to identify these roles. In general, we obtained very good results when identifying all six roles. Furthermore, we showed the relevance that programming languages have in predicting the investigated roles.

6.2 Contributions

In the context of the research conducted in this thesis, we highlight the following contributions:

- We mapped which technical skills are more required from the industry perspective according to 14 different IT professional roles (Chapter 3). Further, we listed the most referred soft skills among the opportunities we analyzed. We argue that the information revealed in this investigation can direct researchers towards relevant research topics related to software developers' expertise.
- We build a ground-truth containing the expertise information of 575 developers with respect to three widely popular JavaScript libraries: *ReactJS*, *socket.io*, and

mongodb (Chapter 4). We claim that this dataset can be used as a baseline to evaluate the performance of data-driven solutions targeted at identifying the expertise of software developers. Indeed, we observed a few works relying on this dataset to conduct their research [Eke, 2020; Vadlamani and Baysal, 2020; Dey et al., 2021].

- We proposed an unsupervised method that, based on low-level source code features provided by GitHub, was able to generate clusters containing experts in third-party libraries and frameworks. We confirmed the effectiveness of this method by qualitatively comparing the obtained results with real LinkedIn profiles; for example, 72% of the investigated *ReactJS* experts do cite this library explicitly in their profiles. We claim that both the proposed method and collected features can guide researchers and practitioners towards further fine-tuned solutions.

- We showed that given a list of coarse-grained features (e.g., projects' metadata, programming languages used, biography text, etc), supervised machine-learning methods can effectively infer the technical roles of software developers. This claim is substantially supported by the results we obtained for precision (0.88) and AUC (0.89) metrics. Essentially, this contribution shows that we can rely on GitHub's activity information to infer major characteristics (i.e., technical roles) of software developers as well.

- We reveal and analyzed the issues we found when adopting data-driven strategies to determine the technical skills of software developers. Specifically, we qualitatively analyzed the wrong predictions in our studies and proposed a list of measures to be tackled further, such as the adoption of multiple data sources. Furthermore, we also discussed different interpretations for both precision and recall metrics in the context of our problem.

6.3 Future Work

During the research on this subject, we identified some unexplored questions that can result in relevant works if developed properly. We enumerated these issues in the following topics:

Expand Soft Skills Analysis: On one hand, this thesis centered its research on investigating the technical skills of software developers. On the other, soft skills

also represent an essential element to the success of a project [Bakar and Choo-Yee Ting, 2011]. As described in Chapter 3, IT companies frequently look for professional with developed communication and teamwork skills. However, we lack pieces of evidence concerning the importance software developers give to these soft skills. Hence, one can conduct a similar study to understand the developers' perspective regarding soft skills and then compare the results we report in this manuscript. Alternatively, we can also expand our currently soft skills inspection by adopting an automated method to mine soft skills from job advertisements [Sayfullina et al., 2018], and perform a stratified analysis to verify the most demanded soft skills for each IT role (as we did with hard skills).

Technical Expertise in Programming Languages: Our study in Chapter 3 also showed the relevance that programming languages have to select new candidates. Although we relied on programming language's data to assist in identifying developers' technical skills (see Chapter 5), we did not conduct a specific study to extract the expertise they have in this topic. Therefore, we understand it is important to propose data-driven methods aimed at identifying developers' expertise in programming languages.

Third-party Libraries Assessment in Other Ecosystems: In Chapter 4, we proposed to identify the expertise level in three widely popular JavaScript libraries. Despite the increasing popularity of the JavaScript universe in recent years [Borges et al., 2016], we acknowledge that the scope of this study is rather strict when compared with the software development ecosystem as a whole. Due to this fact, we argue that extending this evaluation to other languages is very important to ensure the applicability of our method. Indeed, we already started a new investigation in this direction by analyzing the effectiveness of the proposed unsupervised technique to identify experts in eight well-known third-party libraries implemented in four programming languages: *Java*, *Python*, *PHP*, and *C#*.

Include Multiple Data Sources in Expertise Prediction: In this thesis, we relied on the information available at GitHub to identify the expertise elements investigated in Chapters 4 and 5. We opted for doing so because we would like to depend mostly on the developers' programming activity. On the other side, we realized that adopting just one data source limit the variety of information that can be used to perform such predictions [Sarma et al., 2016; Eke, 2020; Vadlamani and Baysal, 2020]. For example, we showed in Section 4.5.4 that the

lack of pertinent information represents a relevant limitation to the performance of data-driven methods. So, we claim that future work may include new data sources to improve the prediction results.

Develop Plug-in Tools to Identify Expertise in the Wild: Although we developed two concrete solutions to detect software developers' expertise, both implementations remain at the early-research stage. In this line, a further effort can be employed in order to implement publicly available plug-in tools for both methods [Brito and Valente, 2020; Coelho et al., 2020].

Bibliography

Abebe, S. L., Arnaoudova, V., Tonella, P., Antoniol, G., and Guéhéneuc, Y.-G. (2012). Can Lexicon Bad Smells Improve Fault Prediction? In *Working Conference on Reverse Engineering (WCRE)*, pages 235--244.

Agrawal, K., Aschauer, M., Thonhofer, T., Bala, S., Rogge-Solti, A., and Tomsich, N. (2016). Resource Classification from Version Control System Logs. In *International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 1–10.

Ahmed, I. (2018). What Makes a Good Developer ? An Empirical Study of Developers ' Technical and Social Competencies. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 319--321.

Alencar da Costa, D., Abebe, S. L., McIntosh, S., Kulesza, U., and Hassan, A. E. (2014). An Empirical Study of Delays in the Integration of Addressed Issues. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 281--290.

Alonso, O., Devanbu, P. T., and Gertz, M. (2008). Expertise Identification and Visualization from CVS. In *International Working Conference on Mining Software Repositories (MSR)*, pages 125–128.

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., and Zimmermann, T. (2019). Software engineering for machine learning: A case study. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 291--300.

Arafeen, M. J. and Do, H. (2013). Test case prioritization using requirements-based clustering. In *International Conference on Software Testing, Verification and Validation (ICST)*, pages 312--321.

Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2016). A novel approach for estimating truck factors. In *24th International Conference on Program Comprehension (ICPC)*, pages 1--10.

Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2019a). Measuring and analyzing code authorship in 1+118 open source projects. *Science of Computer Programming*, 1(1):1–34.

Avelino, G., Passos, L., Petrillo, F., and Valente, M. T. (2019b). Who can maintain this code? assessing the effectiveness of repository-mining techniques for identifying software maintainers. *IEEE Software*, 1(1):1--15.

Bajammal, M., Mazinanian, D., and Mesbah, A. (2018). Generating reusable web components from mockups. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 601--611.

Bakar, A. A. and Choo-Yee Ting (2011). Soft skills recommendation systems for IT jobs: A Bayesian network approach. In *Conference on Data Mining and Optimization (DMO)*, pages 82--87.

Baltes, S. and Diehl, S. (2018). Towards a Theory of Software Development Expertise. In *Foundations of Software Engineering (FSE)*, pages 1--14.

Bao, L., Xing, Z., Xia, X., Lo, D., and Li, S. (2017). 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)*, pages 170--181.

Begel, A., Bosch, J., and Storey, M. D. (2013). Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. *IEEE Software*, 30(1):52--66.

Behroozi, M., Parnin, C., and Barik, T. (2019). Hiring is Broken: What Do Developers Say About Technical Interviews? In *Visual Languages and Human Centric Computing (VLHCC)*, pages 1--9.

Bernardo, J. H., da Costa, D. A., and Kulesza, U. (2018). Studying the impact of adopting continuous integration on the delivery time of pull requests. In *International Conference on Mining Software Repositories (MSR)*, pages 131--141.

Beyer, S., Macho, C., Pinzger, M., and Di Penta, M. (2018). Automatically classifying posts into question categories on stack overflow. In *International Conference on Program Comprehension (ICPC)*, pages 211–221.

Bhattacharya, P., Neamtiu, I., and Faloutsos, M. (2014). Determining Developers' Expertise and Role: A Graph Hierarchy-Based Approach. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 11–20.

Borges, H., Brito, R., and Valente, M. T. (2019). Beyond textual issues: Understanding the usage and impact of github reactions. In *XXXIII Brazilian Symposium on Software Engineering (SBES)*, pages 397--406.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the Factors that Impact the Popularity of GitHub Repositories. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334--344.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning Multi-label Scene Classification. *Pattern Recognition*, 37(9):1757 – 1771.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5--32.

Brito, A., Valente, M. T., Xavier, L., and Hora, A. (2020). You broke my code: Understanding the motivations for breaking changes in APIs. *Empirical Software Engineering*, 25:1458—1492.

Brito, A., Xavier, L., Hora, A., and Valente, M. T. (2018). Why and how Java developers break APIs. In *25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 255--265.

Brito, R. and Valente, M. T. (2020). RefDiff4Go: Detecting refactorings in go. In *Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)*, pages 1--10.

Brooks Jr, F. P. (1995). *The mythical man-month*. Addison-Wesley.

Campos, E. C., de Souza, L. B., and Maia, M. d. A. (2016). Searching crowd knowledge to recommend solutions for api usage tasks. *Journal of Software: Evolution and Process*, 28:863--892.

Chapelle, O., Schlkopf, B., and Zien, A. (2006). *Semi-Supervised Learning*. Cambridge, MIT Press.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16(1):321--357.

Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 785–794.

Chen, Z., Menzies, T., Port, D., and Boehm, B. (2005). Finding the Right Data for Software Cost Modeling. *IEEE Software*, 22(6):38--46.

Chicco, D. (2017). Ten Quick Tips for Machine Learning in Computational Biology. *BioData Mining*, 10(1):1--35.

Chulani, S., Boehm, B. W., and Steece, B. (1999). Bayesian Analysis of Empirical Software Engineering Cost Models. *IEEE Transactions on Software Engineering*, 25(4):573--583.

Claesen, M. and Moor, B. D. (2015). Hyperparameter Search in Machine Learning. *Metaheuristics International Conference (MIC)*, pages 1--5.

Coelho, J. and Valente, M. T. (2017). Why modern open source projects fail. In *11th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 186--196.

Coelho, J., Valente, M. T., Milen, L., and Silva, L. L. (2020). Is this GitHub project maintained? measuring the level of maintenance activity of open-source projects. *Information and Software Technology*, 1:1--35.

Coelho, J., Valente, M. T., Silva, L. L., and Shihab, E. (2018). Identifying Unmaintained Projects in GitHub. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 14–21.

Constantinou, E. and Kapitsaki, G. M. (2016). Identifying Developers' Expertise in Social Coding Platforms. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 63--67.

Constantinou, E. and Kapitsaki, G. M. (2017). Developers expertise and roles on software technologies. In *Asia-Pacific Software Engineering Conference (APSEC)*, pages 365–368.

Cosentino, V., Canovas Izquierdo, J. L., and Cabot, J. (2017). A Systematic Mapping Study of Software Development With GitHub. *IEEE Access*, 5:7173–7192.

Da Silva, J. R., Clua, E., Murta, L., and Sarma, A. (2015). Niche vs. Breadth: Calculating Expertise Over Time Through a Fine-grained Analysis. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 409–418.

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

DeMarco, T. and Lister, T. R. (1999). *Peopleware: Productive Projects and Teams*. Addison-Wesley Professional. ISBN 9780932633439.

Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On Label Dependence and Loss Minimization in Multi-label Classification. *Machine Learning*, 88(1):5--45.

Dey, T., Karnauch, A., and Mockus, A. (2021). Representation of developer expertise in open source software. In *International Conference on Software Engineering (ICSE)*, pages 1--11.

Dias, M., Bacchelli, A., Gousios, G., Cassou, D., and Ducasse, S. (2015). Untangling Fine-Grained Code Changes. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 341--350.

Eghbal, N. (2016). Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure. Technical report, Ford Foundation.

Eke, N. (2020). *Cross-Platform Software Developer Expertise Learning*. PhD dissertation, Carleton University, Ottawa, Ontario.

Ericsson, K. A. (2006). *The Cambridge Handbook of Expertise and Expert Performance*, chapter 38, pages 683–704.

Ericsson, K. A. (2012). An Introduction to The Cambridge Handbook of Expertise and Expert Performance: Its Development, Organization, and Content. In *The Cambridge Handbook of Expertise and Expert Performance*, pages 3–20. Cambridge University Press.

Ferreira, F., Silva, L. L., and Valente, M. T. (2019). Software engineering meets deep learning: A literature review. *arXiv*.

Ford, D., Barik, T., Rand-Pickett, L., and Parnin, C. (2017). The tech-talk balance: what technical interviewers expect from technical candidates. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 43--48.

Foster Provost, T. F. (2015). *Data Science for Business: What you need to know about data mining and data-analytic thinking*. O'Reilly UK Ltd. ISBN 1449361323.

Fritz, T., Murphy, G. C., and Hill, E. (2007). Does a programmer's activity indicate knowledge of code? In *Foundations of Software Engineering (FSE)*, pages 341--350.

Fritz, T., Murphy, G. C., Murphy-Hill, E., Ou, J., and Hill, E. (2014). Degree-of-knowledge: modeling a developer's knowledge of code. *ACM Transactions on Software Engineering and Methodology*, 23(2):14:1--14:42.

Fritz, T., Ou, J., Murphy, G. C., and Murphy-Hill, E. (2010). A degree-of-knowledge model to capture source code familiarity. In *International Conference on Software Engineering (ICSE)*, pages 385--394.

Goldberg, Y. (2017). *Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies)*. Morgan & Claypool Publishers, 1th edition.

Gorla, A., Tavecchia, I., Gross, F., and Zeller, A. (2014). Checking app behavior against app descriptions. In *International Conference on Software Engineering (ICSE)*, pages 1025--1035.

Greene, G. J. and Fischer, B. (2016). CVExplorer: identifying candidate developers by mining and exploring their open source contributions. In *International Conference on Automated Software Engineering (ASE)*, pages 804–809.

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

Hauff, C. and Gousios, G. (2015). Matching GitHub Developer Profiles to Job Advertisements. *Working Conference on Mining Software Repositories (MSR)*, pages 362--366.

He, H. and Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263--1284.

Herrera, F., Charte, F., Rivera, A. J., and del Jesus, M. J. (2016). *Multilabel Classification: Problem Analysis, Metrics and Techniques*. Springer.

Honsel, V., Herbold, S., and Grabowski, J. (2016). Hidden Markov Models for the Prediction of Developer Involvement Dynamics and Workload. In *International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, pages 1–10.

Hora, A. (2021). Apisonar: Mining api usage examples. *Software: Practice and Experience*, 1:1--34.

Hora, A., Valente, M. T., Robbes, R., and Anquetil, N. (2016). When should internal interfaces be promoted to public? In *Foundations of Software Engineering (FSE)*, pages 280--291.

Huang, W., Mo, W., Shen, B., Yang, Y., and Li, N. (2016). CPDScorer: Modeling and Evaluating Developer Programming Ability across Software Communities. In *Software Engineering and Knowledge Engineering Conference (SEKE)*, pages 01--06.

IfM and IBM (2007). *Succeeding through service innovation.* Number December.

Japkowicz, N. and Stephen, S. (2002). The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analysis*, 6(5):429--449.

Joblin, M., Apel, S., Hunsen, C., and Mauerer, W. (2017). Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics. In *International Conference on Software Engineering (ICSE)*, pages 164–174.

Kagdi, H., Hammad, M., and Maletic, J. I. (2008). Who can Help me with this Source Code Change? In *International Conference on Software Maintenance (ICSM)*, pages 157–166.

Kim, S., E. James Whitehead, J., and Zhang, Y. (2008). Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, 34(2):181–196.

Kruger, J. and Dunning, D. (1999). Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of personality and social psychology*, 77(6):1121.

Kuhn, A., Ducasse, S., and Girba, T. (2007). Semantic clustering: Identifying topics in source code. *Information and Software Technology*, pages 230--243.

Kuhn, M. and Johnson, K. (2013). *Applied Predictive Modeling.* Springer, 1st edition.

Landis, J. R. and Koch, G. G. (1977). The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159--174.

Lee, C. K. and Han, H.-J. (2008). Analysis of skills requirement for entry-level programmer/analysts in Fortune 500 corporations. *Journal of Information Systems Education*, 19(1):17.

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

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

Litecky, C., Aken, A., Ahmad, A., and Nelson, H. J. (2010). Mining for Computing Jobs. *IEEE Software*, 27(1):78--85.

Luaces, O., Díez, J., Barranquero, J., del Coz, J. J., and Bahamonde, A. (2012). Binary relevance efficacy for multilabel classification. *Progress in Artificial Intelligence*, 1(4):303–313.

Machalica, M., Samylkin, A., Porth, M., and Chandra, S. (2019). Predictive test selection. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 91--100.

Madjarov, G., Kocev, D., Gjorgjevikj, D., and Deroski, S. (2012). An Extensive Experimental Comparison of Methods for Multi-label Learning. *Pattern Recognition*, 45(9):3084–3104.

Mao, K., Yang, Y., Wang, Q., Jia, Y., and Harman, M. (2015). Developer Recommendation for Crowdsourced Software Development Tasks. In *IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 347--356.

Marlow, J. and Dabbish, L. (2013). Activity Traces and Signals in Software Developer Recruitment and Hiring. In *Conference on Computer Supported Cooperative Work (CSCW)*, pages 1–11. ACM Press.

Marlow, J., Dabbish, L., and Herbsleb, J. (2013). Impression Formation in Online Peer Production : Activity Traces and Personal Profiles in GitHub. In *Conference on Computer Supported Cooperative Work (CSCW)*, pages 117–128.

Maron, M. E. (1961). Automatic indexing: An experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417.

McCallum, A. and Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *International Conference on Machine Learning*, pages 41–48.

McDowell, G. (2015). *Cracking the Coding Interview: 189 Programming Questions and Solutions*. CareerCup, LLC. ISBN 9780984782857.

Mendes, T., Valente, M. T., Hora, A., and Serebrenik, A. (2016). Identifying utility functions using random forests. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 614--618.

Menezes, G. G. L., Murta, L. G. P., Barros, M. O., and Van Der Hoek, A. (2018). On the nature of merge conflicts: a study of 2,731 open source java projects hosted by github. *IEEE Transactions on Software Engineering*.

Menzies, T., Butcher, A., Cok, D., Marcus, A., Layman, L., Shull, F., Turhan, B., and Zimmermann, T. (2013). Local Versus Global Lessons for Defect Prediction and Effort Estimation . *IEEE Transactions on Software Engineering*, 39(6):822–834.

Metsis, V., Androutsopoulos, I., and Paliouras, G. (2006). Spam filtering with naive bayes – which naive bayes? In *Conference on Email and Anti-Spam (CEAS)*, pages 1–9.

Mockus, A. and Herbsleb, J. D. (2002). Expertise browser: a quantitative approach to identifying expertise. In *International Conference on Software Engineering (ICSE)*, pages 503--512.

Montandon, J. E., Silva, L. L., and Valente, M. T. (2019). Identifying Experts in Software Libraries and Frameworks among GitHub Users. In *International Conference on Mining Software Repositories (MSR)*, pages 1–12.

Ng, A. (2000). Machine Learning Course (Stanford CS229 Lecture notes).

Oliveira, J., Viggiato, M., and Figueiredo, E. (2019). How well do you know this library? Mining experts from source code analysis. In *Brazilian Symposium on Software Quality (SBQS)*, pages 1--10.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825--2830.

Peters, F., Menzies, T., and Marcus, A. (2013). Better Cross Company Defect Prediction. In *Working Conference on Mining Software Repositories (MSR)*, pages 409--418.

Pimentel, J. F., Murta, L., Braganholo, V., and Freire, J. (2019). A large-scale study about quality and reproducibility of jupyter notebooks. In *16th International Conference on Mining Software Repositories (MSR)*, pages 507--517.

Politowski, C., Petrillo, F., Montandon, J. E., Valente, M. T., and Guéhéneuc, Y. (2021). Are game engines software frameworks? A three-perspective study. *Journal of Systems and Software*, 171:110846.

Provost, F. and Fawcett, T. (2001). Robust Classification for Imprecise Environments. *Machine learning*, 42(3):203–231.

Raudys, S. J. and Jain, A. K. (1991). Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):252--264.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier Chains for Multi-label Classification. *Machine Learning*, 85(3):333–359.

Robles, G., Gonzalez-Barahona, J. M., and Herraiz, I. (2009). Evolution of the core team of developers in libre software projects. In *Working Conference on Mining Software Repositories (MSR)*, pages 167–170.

Romano, J., Kromrey, J. D., Coraggio, J., and Skowronek, J. (2006). 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*, pages 1--33.

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(C):53--65.

Ruiz, I. J. M., Adams, B., Nagappan, M., Dienst, S., Berger, T., and Hassan, A. E. (2014). A large-scale empirical study on software reuse in mobile apps. *IEEE Software*, 31(2):78--86.

Saha, A. K., Saha, R. K., and Schneider, K. A. (2013). A Discriminative Model Approach for Suggesting Tags Automatically for Stack Overflow Questions. In *Working Conference on Mining Software Repositories (MSR)*, pages 73--76.

Saremi, R. L., Yang, Y., Ruhe, G., and Messinger, D. (2017). Leveraging Crowdsourcing for Team Elasticity: An Empirical Evaluation at TopCoder. In *Interna-*

tional Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pages 103--112.

Sarma, A., Chen, X., Kuttal, S., Dabbish, L., and Wang, Z. (2016). Hiring in the global stage: Profiles of online contributions. In *International Conference on Global Software Engineering (ICGSE)*, pages 1–10.

Sawant, A. A. and Bacchelli, A. (2017). fine-GRAPE: fine-grained API usage extractor - an approach and dataset to investigate API usage. *Empirical Software Engineering*, 22(3):1348--1371.

Saxena, R. and Pedanekar, N. (2017). I Know What You Coded Last Summer: Mining Candidate Expertise from GitHub Repositories. In *ACM Conference on Computer-Supported Cooperative Work and Social Computing (CSCW)*, pages 299--302.

Sayfullina, L., Malmi, E., and Kannala, J. (2018). Learning Representations for Soft Skill Matching. In *Lecture Notes in Computer Science*, pages 141--152. Springer International Publishing.

Schuler, D. and Zimmermann, T. (2008). Mining usage expertise from version archives. In *International Working Conference on Mining Software Repositories (MSR)*, pages 121--124.

Shafiq, S., Mashkoor, A., Mayr-Dorn, C., and Egyed, A. (2020). Machine learning for software engineering: A systematic mapping. *arXiv*.

Shirabad, J. S., Lethbridge, T. C., and Matwin, S. (2003). Mining the maintenance history of a legacy software system. In *International Conference on Software Maintenance (ICSM)*, pages 95–104.

Siau, K., Tan, X., and Sheng, H. (2010). Important characteristics of software development team members: an empirical investigation using repertory grid. *Information Systems Journal*, 20(6):563--580.

Siegmund, J., Kästner, C., Liebig, J., Apel, S., and Hanenberg, S. (2014). Measuring and modeling programming experience. *Empirical Software Engineering*, 19(5):1299--1334.

Silva, D., da Silva, J. P., Santos, G., Terra, R., and Valente, M. T. (2020). Refdiff 2.0: A multi-language refactoring detection tool. *IEEE Transactions on Software Engineering*, 1:1--17.

Silva, D., Tsantalis, N., and Valente, M. T. (2016). Why we refactor? confessions of github contributors. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, pages 858--870.

Silva, H. and Valente, M. T. (2018). What's in a github star? understanding repository starring practices in a social coding platform. *Journal of Systems and Software*.

Singer, L., Figueira Filho, F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K. (2013). 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)*, pages 103--116.

Souza, L. B., Campos, E. C., Madeiral, F., Paixão, K., Rocha, A. M., and de Almeida Maia, M. (2019). Bootstrapping cookbooks for apis from crowd knowledge on stack overflow. *Information and Software Technology*, 111:37--49.

Spencer, D. (2009). *Card sorting: Designing usable categories*. Rosenfeld Media.

Tan, M., Tan, L., Dara, S., and Mayeux, C. (2015). Online Defect Prediction for Imbalanced Data. In *International Conference on Software Engineering (ICSE)*, pages 99--108.

Teyton, C., Falleri, J.-R., Morandat, F., and Blanc, X. (2013). Find your Library Experts. In *Working Conference on Reverse Engineering (WCRE)*, pages 202--211.

Teyton, C., Palyart, M., Falleri, J.-R., Morandat, F., and Blanc, X. (2014). Automatic Extraction of Developer Expertise Categories and Subject Descriptors. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10.

Thaller, H., Linsbauer, L., and Egyed, A. (2019). Feature maps: A comprehensible software representation for design pattern detection. In *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*, pages 207--217. IEEE.

Thiselton, E. and Treude, C. (2019). Enhancing python compiler error messages via stack. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1--12.

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

Treude, C., Barzilay, O., and Storey, M.-A. (2011). How do programmers ask and answer questions on the web? (NIER track). In *International Conference on Software Engineering (ICSE)*, pages 804--807.

Treude, C. and Robillard, M. P. (2017). Understanding stack overflow code fragments. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 509--513.

Tsantalis, N., Mansouri, M., Eshkevari, L., Mazinanian, D., and Dig, D. (2018). Accurate and efficient refactoring detection in commit history. In *40th International Conference on Software Engineering (ICSE)*, pages 483--494.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2010). *Mining Multi-label Data*, pages 667–685. Springer.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2011). Random k-labelsets for multilabel classification. *IEEE Transactions on Knowledge and Data Engineering*, 23(7):1079–1089.

Uddin, G., Khomh, F., and Roy, C. K. (2020). Mining api usage scenarios from stack overflow. *Information and Software Technology*, 122:106277.

Vadlamani, S. and Baysal, O. (2020). Studying Software Developer Expertise and Contributions in Stack Overflow and GitHub. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 1--12.

Vajda, S., Rangoni, Y., and Cecotti, H. (2015). Semi-automatic ground truth generation using unsupervised clustering and limited manual labeling: Application to handwritten character recognition. *Pattern Recognition Letters*, 58:23–28.

Valve Corporation (2012). *Handbook for new employees*. Valve Press, 1st edition.

Vasilescu, B., Van Schuylenburg, S., Wulms, J., Serebrenik, A., and van den Brand, M. G. (2014). Continuous integration in a social-coding world: Empirical evidence from github. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 401--405. IEEE.

Vassallo, C., Schermann, G., Zampetti, F., Romano, D., Leitner, P., Zaidman, A., Penta, M. D., and Panichella, S. (2017). A Tale of CI Build Failures: An Open Source and a Financial Organization Perspective. In *International Conference on Software Maintenance and Evolution (ICSME)*, pages 183–193.

Venkataramani, R., Gupta, A., Asadullah, A. M., Muddu, B., and Bhat, V. D. (2013). Discovery of Technical Expertise from Open Source Code Repositories. In *International Conference on World Wide Web (WWW)*, pages 97--98.

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

Wang, Z., Sun, H., Fu, Y., and Ye, L. (2017). Recommending Crowdsourced Software Developers in Consideration of Skill Improvement. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 717--722.

Warrens, M. (2015). Five Ways to Look at Cohen's Kappa. *Journal of Psychology & Psychotherapy*, 05:1--4.

Weston, J. and Watkins, C. (1998). Multi-Class Support Vector Machines. Technical report, University of London.

Xia, X., Wan, Z., Kochhar, P. S., and Lo, D. (2019). How Practitioners Perceive Coding Proficiency. In *International Conference on Software Engineering (ICSE)*, pages 1--12.

Ye, Y. and Kishida, K. (2003). Toward an understanding of the motivation open source software developers. In *International Conference on Software Engineering (ICSE)*, pages 419–429.

Yu, L. and Liu, H. (2003). Feature Selection for High-dimensional Data: A Fast Correlation-based Filter Solution. In *International Conference on Machine Learning (ICML)*, pages 856--863.

Yu, L. and Ramaswamy, S. (2007). Mining CVS Repositories to Understand Open-Source Project Developer Roles. In *International Workshop on Mining Software Repositories (MSR)*, pages 1–10.

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

Zhang, M.-L. and Zhou, Z.-H. (2014). A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.

Zhou, S., Vasilescu, B., and Kästner, C. (2020). How has forking changed in the last 20 years? a study of hard forks on github. In *42nd International Conference on Software Engineering (ICSE)*, pages 445--456.

Zhou, Y. and Sharma, A. (2017). Automated Identification of Security Issues from Commit Messages and Bug Reports. In *Foundations of Software Engineering (ESEC/FSE)*, pages 914--919.

Zumel, N., Mount, J., and Porzak, J. (2014). *Practical Data Science with R*. Manning, 1th edition.

Appendix A

Library Expertise Survey

Hi `[developer]`,

I am a PhD student at UFMG, Brazil, working with techniques to assess the skills of developers by mining their activities on git repositories.

To calibrate my current approach, I need data about the skills of developers in a set of frameworks.

For example, I found that you have been contributing to GitHub projects that use `[target library]` JavaScript library.

So, could please rank your expertise on `[target library]` JavaScript library in a scale from 1 (novice) to 5 (expert)?

(important: I will use your answer exclusively in my research and only in aggregated and anonymised format).

Best,

João Eduardo Montandon
DCC, UFMG, Brazil
Applied Software Engineering Research Group
`http://aserg.labsoft.dcc.ufmg.br`