

UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Johnatan Alves de Oliveira

**Identifying and Evaluating Hard Skills of Software Developers from Source
Code Analysis**

Belo Horizonte
2023

Johnatan Alves de Oliveira

**Identifying and Evaluating Hard Skills of Software Developers from Source
Code Analysis**

Final Version

Thesis 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: Eduardo Magno Lages Figueiredo
Co-Advisor: Maurício Ronny de Almeida Souza

Belo Horizonte
2023

2023, Johnatan Alves de Oliveira.
Todos os direitos reservados

Oliveira, Johnatan Alves de.

O48i

Identifying and evaluating hard skills of software developers from source code analysis [recurso eletrônico] : / Johnatan Alves de Oliveira – 2023.

1 recurso online (105 f. il., color.) : pdf.

Orientador: Eduardo Magno Lages Figueiredo

Coorientador: Maurício Ronny de Almeida Souza

Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.

Referências: f.95-105

1. Computação – Teses. 2. Engenharia de Software – Teses. 3. Engenheiros de software - Recrutamento - Teses. 4. Git (Arquivo de computador) – Teses. I. Figueiredo, Eduardo Magno Lages. II. Souza, Maurício Ronny de Almeida. III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. IV. Título.

CDU 519.6*32(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEX



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

FOLHA DE APROVAÇÃO

IDENTIFYING AND EVALUATING HARD SKILLS OF SOFTWARE DEVELOPERS FROM SOURCE CODE ANALYSIS

JOHNATAN ALVES DE OLIVEIRA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores(a):

Prof. Eduardo Magno Lages Figueiredo - Orientador
Departamento de Ciência da Computação - UFMG

Prof. Maurício Ronny de Almeida Souza - Coorientador
Departamento de Computação Aplicada - UFLA

Prof. Marco Túlio de Oliveira Valente
Departamento de Ciência da Computação - UFMG

Prof. André Cavalcante Hora
Departamento de Ciência da Computação - UFMG

Prof. Fischer Jonatas Ferreira
Departamento de Computação - UFC

Prof. Uira Kulesza
Departamento de Informática e Matemática Aplicada - UFRN

Profa. Juliana Alves Pereira
Departamento de Informática - PUC-Rio

Belo Horizonte, 10 de julho de 2023.



Documento assinado eletronicamente por **Eduardo Magno Lages Figueiredo, Professor do Magistério Superior**, em 09/08/2023, às 16:50, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Andre Cavalcante Hora, Professor do Magistério Superior**, em 29/08/2023, às 14:58, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Marco Tulio de Oliveira Valente, Professor do Magistério Superior**, em 29/08/2023, às 16:09, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Maurício Ronny de Almeida Souza, Usuário Externo**, em 30/08/2023, às 16:30, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Fischer Jônatas Ferreira, Usuário Externo**, em 05/09/2023, às 12:02, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Uira Kulesza, Usuário Externo**, em 11/09/2023, às 15:47, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Juliana Alves Pereira, Usuária Externa**, em 12/09/2023, às 19:18, conforme horário oficial de Brasília, com fundamento no art. 5º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



A autenticidade deste documento pode ser conferida no site https://sei.ufmg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2454594** e o código CRC **AA6E5911**.

Acknowledgments

Essa tese de doutorado é o resultado de uma jornada de muito aprendizado, esforço e dedicação. Gostaria de expressar meu profundo agradecimento a todos que contribuíram para o sucesso da minha tese de doutorado. Neste momento especial, gostaria de agradecer em particular:

Inicialmente, gostaria de expressar a minha gratidão a Deus por ter me concedido o conhecimento, as oportunidades e as conquistas ao longo dessa jornada de pesquisa. Sou grato pela inspiração, força e orientação divina que recebi em todos os momentos. Ao meu professor orientador, Eduardo Figueiredo, que me acolheu desde o início da minha jornada científica e me guiou com sabedoria, paciência e inspiração ao longo de todo o processo de pesquisa. Sua orientação foi essencial para o desenvolvimento deste trabalho e para o meu crescimento como pesquisador. Sua experiência e conhecimento contribuíram significativamente para a qualidade e profundidade desta tese. Ao professor coorientador, Maurício Souza, pelo apoio, orientação e valiosos insights ao longo do meu percurso acadêmico. Aos funcionários da UFMG, em especial aos funcionários do PPGCC, pela disponibilidade, eficiência e apoio durante toda a minha trajetória acadêmica. O trabalho de vocês nos bastidores é fundamental para o bom funcionamento do programa e para o sucesso dos alunos. Aos meus colegas de laboratório, pela colaboração, troca de ideias e amizade ao longo desses anos. Suas contribuições foram valiosas para o amadurecimento das minhas pesquisas e para o ambiente estimulante em que pude desenvolver meu trabalho. A minha mãe, Eliana de Fátima, e ao meu pai, João Matildes, por todo o amor, apoio incondicional e pelos ensinamentos que moldaram minha personalidade e me incentivaram a buscar sempre o melhor em todas as etapas da minha vida acadêmica. Sou imensamente grato pela presença de vocês ao meu lado. À banca de avaliação, pela disponibilidade em dedicar seu tempo para analisar e comentar minha tese. Suas observações e ideias proporcionaram valiosas contribuições para aprimorar e enriquecer este trabalho. Às agências de fomento, como a CAPES e a FAPEMIG, pelo apoio financeiro concedido, por meio de bolsas e projetos de pesquisa. A contribuição de vocês foi fundamental para viabilizar o desenvolvimento deste trabalho e para o avanço da ciência e tecnologia em nosso país. A minha irmã, Daiany Oliveira, pelos ensinamentos, desabafos e longas conversas motivadoras. Sua presença e apoio constante foram essenciais para meu crescimento pessoal e para superar os desafios ao longo desta jornada acadêmica. Ao meu irmão, Aldivon Ribeiro (em memória), que, mesmo que indiretamente, me ensinou os desafios que a vida pode causar.

Por fim, gostaria de expressar minha gratidão a todos aqueles que, de alguma forma, contribuíram para o meu crescimento acadêmico e pessoal ao longo dessa jornada. Sou verdadeiramente grato por todo o apoio, incentivo e confiança depositados em mim.

“Success is not final, failure is not fatal: it is the courage to continue that counts.”
(Winston Churchill)

Resumo

O desenvolvimento de software é uma atividade cujo sucesso depende da qualidade e experiência dos profissionais envolvidos. No entanto, avaliar de forma precisa a experiência em engenharia de software pode ser um desafio, uma vez que é difícil mensurar as habilidades técnicas dos desenvolvedores de forma objetiva. Portanto, é fundamental contar com frameworks, métricas, métodos e ferramentas avançadas para avaliar essas habilidades. Neste contexto, propõe-se um framework para avaliar as habilidades de programação a partir do código-fonte. O objetivo deste framework é fornecer uma representação curricular dos desenvolvedores, contendo informações sobre as linguagens de programação utilizadas, especializações em back-end e front-end, testes de unidade, dados adicionais sobre o desenvolvedor e bibliotecas utilizadas. O framework foi avaliado em três perspectivas: o perfil dos desenvolvedores, especialistas em bibliotecas e suporte aos recrutadores. Os resultados demonstraram que o framework é capaz de avaliar as habilidades de programação dos desenvolvedores e auxiliá-los na autoavaliação de suas competências. Adicionalmente, o framework pode ajudar os recrutadores a selecionar o profissional mais adequado para projetos de software, com base nas habilidades necessárias para o projeto em questão. Em resumo, o framework proposto oferece uma solução confiável, adaptável e objetiva para avaliar as habilidades técnicas dos desenvolvedores de software. Com a sua utilização, é possível aprimorar a seleção e o desempenho dos profissionais envolvidos no desenvolvimento de projetos de software.

Palavras-chave: Desafios de Recrutamento, Engenheiros de Software, GitHub, Hard Skills

Abstract

Software development is an activity that heavily relies on its development workforce, making the quality and experience of professionals a crucial factor for project success. However, evaluating software engineering experience can be challenging as accurately measuring developers' hard skills is difficult. For this reason, it is essential to have frameworks, metrics, methods, and advanced tools to assess developers' abilities. In this context, we propose a framework for computing programming skills from source code. This framework aims to provide a curriculum representation for developers, with information about programming languages, back-end and front-end profiles, unit testing, additional developer data, and used libraries. Our framework was evaluated from three perspectives: developer profiles, library experts, and support for recruiters. The results demonstrate that the framework can compute developers' programming skills and assist them in self-assessing their abilities. Moreover, the framework can aid recruiters in selecting the most suitable developer for software projects based on the required skills. In summary, our framework offers a reliable, adaptable, and objective solution for assessing software developers' hard skills. With its help, it is possible to enhance the selection and performance of professionals involved in software development projects.

Keywords: Recruitment Challenges, Software Engineers, GitHub, Hard Skills

List of Figures

1.1	Step by Step	17
2.1	GitHub Profile Example	23
3.1	An Overview of the Framework	29
3.2	Programming Skills Profiles Instance	30
3.3	Library Instance	31
3.4	Tech Skills Instance	33
3.5	Tool Architecture Overview	34
3.6	Tool to Compute Programming Skills	35
4.1	Study Steps to Identifying Developer Profiles	39
4.2	Changed Files and the Respective Number of Changed Lines of Code for Fictitious Developers	40
4.3	Steps to Select Developers from GitHub	42
4.4	Overview	46
4.5	Programming Language Skills	48
4.6	Back-end & Front-end Skill	49
4.7	Test Skill	50
5.1	Study Steps to Identifying Library Experts	57
5.2	Steps for Collecting Software Projects from GitHub	59
5.3	Number of Developers by Library	60
5.4	Number of Commits per Library	60
5.5	Number of Imports per Library	61
5.6	Number of LOC per Library	61
5.7	Tool to compute programming skills	65
6.1	Study steps with Software Recruiters	71

List of Tables

2.1	Metrics and Hard Skills by Authors	25
4.1	Distribution of Programming Languages Based on Changed Files	40
4.2	Distribution of Programming Languages Based on Changed Lines of Code . .	41
4.3	Survey Questions	43
4.4	Feedback Categories	52
5.1	The Metrics Analysis as GQM method	56
5.2	Library Descriptions	59
5.3	Projects Selected for Analysis	60
5.4	Survey Questions on the Use of the Libraries	62
5.5	Top 20% from Library Experts Selected to Answer the Survey	63
5.6	Level of Knowledge in Each Library	64
6.1	Interview Questions	74
6.2	Participant demographics	76
6.3	Applicability of the framework Evaluated	78
6.4	Category Visualization	82
6.5	Category Lacking information	86

Contents

1	Introduction	14
1.1	Problem and Motivation	14
1.2	Goals	15
1.3	Study Steps	16
1.4	Results	17
1.5	Publications	18
1.6	Outline of the Thesis	19
2	Background and Related Work	20
2.1	Hard Skills and Soft Skills	20
2.2	Recruitment Process	21
2.3	Using GitHub Data to Understand Software Developers	22
2.4	Related Work	24
2.5	Concluding Remarks	27
3	A Framework to Identify Programming Skills	28
3.1	Framework Overview	28
3.2	Profile Instance	30
3.3	Library Instance	31
3.4	Tech Skills Instance	32
3.5	A Tool to Support Programming Skill Identification	33
3.6	Concluding Remarks	36
4	Identifying Developer Profiles	37
4.1	Goal and Research Questions	38
4.2	Evaluation Steps	39
4.3	Metrics used to Profile Instance	40
4.4	Dataset	41
4.5	Survey Design	42
4.6	Data Analysis	44
4.7	Overview Results	46
4.8	Results for Programming Language Skills	47
4.9	Results for Back-end & Front-end Profiles	48
4.10	Results Test Development	49

4.11	Feedback from Developers	50
4.12	Threats to Validity	52
4.13	Concluding Remarks	53
5	Identifying Library Experts	55
5.1	Goal and Research Questions	56
5.2	Evaluation Steps	57
5.3	Metrics used to Library Instance	57
5.4	Dataset	58
5.5	Survey Design	61
5.6	Overview	62
5.7	Level of Activity	64
5.8	Knowledge Intensity	66
5.9	Knowledge Extension	66
5.10	Threats to Validity	67
5.11	Concluding Remarks	68
6	A Study with Software Recruiters	69
6.1	Goal and Research Questions	69
6.2	Evaluation Steps	70
6.3	Metrics used to Tech Skills Instance	71
6.4	Instrumentation	72
6.5	Selection Subjects	73
6.6	Interview with Recruiters	73
6.7	Data Analysis	74
6.8	Results for Participants' Demographic Information	75
6.9	Results for Recruitment Channels	76
6.10	Results for Applicability	77
6.11	Alternative Visualization	80
6.12	Miss Information	84
6.13	Threats to Validity	86
6.14	Concluding Remarks	87
7	Conclusion	89
7.1	Thesis Recapitulation	90
7.2	Contributions	91
7.3	Future Work	92
	Bibliography	95

Chapter 1

Introduction

Software development is a human-intensive activity, which makes the success of software projects highly influenced by the quality and expertise of their development workforce [62, 85, 108]. Besides, software systems are complex engineering artifacts that demand high technical specialization levels in different areas [85, 108]. These conditions make IT companies focus on creating cross-functional teams with experts in several positions, such as databases, security, front-end design, back-end design, mobile apps, etc. These companies increasingly use social coding platforms like GitHub to search for suitable candidates for their open positions [85]. However, finding developers with the right skills for specific projects is not easy [85]. It is necessary to conduct a long process to select developers, for example, including tests and interviews. Besides, in general, some people avoid self-assess. On the other hand, some people are good to self-assess, but they do not have good technical skills. Therefore, we need ways to support this task of computing hard programming skills of developers. This chapter introduces this thesis. We start by stating our problem and motivation in Section 1.1. Section 1.2 details our goals and intended contributions. Section 1.3 presents the steps of this work. Section 1.4 presents a summary of the results obtained. Finally, Section 1.6 presents the outline of this thesis.

1.1 Problem and Motivation

Software development is a highly technical activity that requires professionals performing diverse roles in software projects, with skills and experience in many different technologies, tools, and techniques [82]. Due to this reason, companies try to build teams composed of developers with a wide diversity of skills. In the Software Engineering industry, with various open positions and not as many qualified candidates, extracting, ranking, and selecting candidates with the right combinations of skills are essential tasks for the success of software projects [33, 58]. Most strategies commonly used for selecting

candidates have limitations. One of these limitations is the source of information used to identify developers hard skills. Many companies rely only on traditional curriculum analysis, which may have inaccurate or outdated information. Besides, non-technical recruiters may identify incorrect developer skills or other skills that are not the focus of the organization. Hiring the wrong developer can lead to additional efforts and resources to train the new employee or expending more time and resources to hire another one. Moreover, wrong hiring decisions are a well-known risk factor for the success of a software project [92]. Software developers have long been using social networking platforms, such as LinkedIn, to showcase their technical skills. They also use other platforms that promote programming contests and serve as a form of recruiting for companies. Employers, on the other hand, use information from social networking platforms to recruit developers [15, 53, 56]. However, the reliability and accuracy of the information provided in such media are not guaranteed [17]. For instance, some individuals can overvalue their skills and omit some skills in a self-authored curriculum.

Software repositories, such as GitHub¹, have been mined for several reasons, such as the extraction and analysis of software metrics [98] and the identification of reuse opportunities [70]. GitHub, in particular, has contributed to the diffusion and publication of open-source code projects that otherwise would not have existed [60]. Moreover, the payoffs are substantial [19, 60]. Although software repositories can reflect the work of software developers, few works [35, 45, 66, 102] have relied on software repository data to identify the skills of developers. In this context, this thesis proposes and evaluates a framework to compute programming skills based on the work that developers perform on GitHub. The proposed framework relies on GitHub data to compute programming skills of developers based on the actual contributions they make. Developers can produce a variety of contributions to the platform, ranging from coding and testing to discussions in forums and documentation of source code. From each type of developer contribution, we can identify essential developer skills [35].

1.2 Goals

As previously mentioned, we lack a complete understanding on how to describe hard skills from source code. Therefore, we propose a framework to help developers and recruiters to assess hard skills from source code analysis. In this context, the general goal of this thesis is to define, investigate, and evaluate this framework. The specific goals can be divided into four:

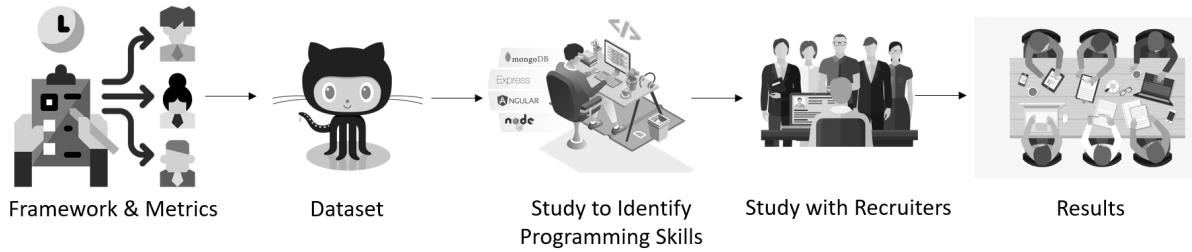
¹<https://github.com>

1. We aim to address the knowledge gap regarding programming skills derived from source code. To achieve this goal, we developed a curriculum for developers based on data that our framework provides. The generated curriculum can help developers self-evaluate their programming skills and guide decision-making to improve hiring processes for software projects, particularly from the viewpoint of recruiters.
2. We analyze hard skills from source code in specific libraries. The goal is to evaluate the knowledge of developers from popular Java libraries. We intend to evaluate the feasibility of finding library experts from the source code.
3. We plan to characterize hard skills from source code analysis. Particularly, this thesis focuses on the hard skills of developers based on data from popular and real projects hosted on GitHub. Besides Java projects, we also studied JavaScript, Python, Go, C++, Ruby, PHP, TypeScript, C#, and C. These analyses were created to generate the profile of developers in four categories: programming languages more used, back-end or front-end profiles, unit tests, and extra data profile.
4. We evaluate the effectiveness of our framework in the context of software recruiting. For this, we conducted interviews with 15 recruiters from three different countries.

1.3 Study Steps

Figure 1.1 presents an overview of the research conducted in this thesis. The first step is present in Chapter 2. This step investigates literature (ad-hoc) concerning techniques, methods, and tools for programming skills. After this research, we develop the framework and tool to compute programming skills (Chapter 3). Next, we select developers from GitHub to evaluate the framework proposed. In sequence, we evaluate the framework with real developers from GitHub through a survey (Chapters 4 and 5). Finally, in the last step, we conducted interviews with recruiters from three different countries to evaluate the framework (Chapter 6).

Figure 1.1: Step by Step



Source: Prepared by the author, 2023

1.4 Results

This section presents a summary of the results obtained. We develop a framework to compute programming skills from source code analysis. We evaluate the framework from three perspectives: 1) programming skills, 2) library experts and 3) recruiters.

From the first perspective, we promote an empirical study to identify developer profiles. As a result, we obtained 54% precision to metric changed files versus 36% to changed lines of code in relation to programming languages skills. To profile back-end & front-end, we obtained 53% precision for changed files versus 45% for changed lines of code. Finally, for the unit test, we observed precision of 45% for changed files versus 30% for changed lines of code. Besides, we noted the necessity of detecting the source code of tests to improve the detection of testing skills.

About the second perspective, we apply an empirical study in libraries. We select the most popular Java libraries from GitHub. We used three metrics. 1) Number of Commits, 2) Number of Imports, and 3) Lines of Code. As overall results, we observed that about 88% of the library experts who answered the survey have high knowledge about the evaluated libraries. In general, when applied solo, we concluded that the metrics could not get good results. Therefore, these metrics need to apply combined to obtain the best results and show the programming skills of developers.

Concerning the third perspective, we conducted a study with 15 software recruiters from three different countries. The recruiters were asked to evaluate two resumes generated by the framework based on GitHub data. To create a realistic scenario, we presented recruiters with a job opening adapted from a real job post on LinkedIn. The recruiters were asked to evaluate whether the framework helped them understand the candidates' profiles, or if it did not help at all. They were also invited to provide feedback on the GitHub data and the resume generated by the framework. The results of the study showed that recruiters found it easier to follow the data presented by the framework because they did not have to manually search GitHub for information. Furthermore, they stated that

the resume generated by the framework could serve as the first round of candidate selection. This could potentially simplify the software developer selection process. Overall, the study demonstrated the effectiveness of the framework and its potential to improve the recruitment process for software development.

1.5 Publications

The results of this thesis generated the following publications:

- **QUATIC'22** Oliveira, J., Souza, M., Flauzino, M., Durelli, R., and Figueiredo, E. Can source code analysis indicate programming skills? survey with developers. In *Proceedings of the International Conference Quality of Information and Communications Technology*, pages 156–171. Springer International Publishing, 2022.
- **JSERD'21** Oliveira, J. A., Vigiato, M., Pinheiro, D., and Figueiredo, E. Mining experts from source code analysis: An empirical evaluation. *Journal of Software Engineering Research and Development*, 9(1):1:1 – 1:16, Feb. 2021.
- **SBES'20** Oliveira, J., Pinheiro, D., and Figueiredo, E. Jexpert: A tool for library expert identification. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, page 386–392. Association for Computing Machinery, 2020.
- **SBQS'19** Oliveira, J., Vigiato, M., and Figueiredo, E. How well do you know this library? Mining experts from source code analysis. In *Proceedings of the XVIII Brazilian Symposium on Software Quality*, SBQS '19, page 49–58. Association for Computing Machinery, 2019.
- **SBCARS'18** Santos, A., Souza, M., Oliveira, J., and Figueiredo, E. Mining software repositories to identify library experts. In *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*, SBCARS '18, page 83–91. Association for Computing Machinery, 2018.

We have submitted a paper to the Journal. However, from the writing date of this thesis, we have not yet received approval from the paper.

1.6 Outline of the Thesis

We organized the remainder of this thesis as follows.

- Chapter 2 – this chapter provides basic concepts on hard skills, metrics, and models to compute programming skills. Besides, we present the directly related works to this thesis.
- Chapter 3 – this chapter provides presents the proposed framework, named FYI - “*For Your Information*” to compute programming skills from source code.
- Chapter 4 – this chapter presents a study to identify developer profiles from GitHub.
- Chapter 5 – this chapter shows a study in Java libraries and evaluates the framework developed to compute libraries experts.
- Chapter 6 – this chapter shows an empirical study with software recruiters.
- Chapter 7 – this chapter summarizes the conclusions we leveraged throughout this thesis. It also outlines some ideas we find interesting to investigate in the future.

Chapter 2

Background and Related Work

Software development is a human-intensive activity, which makes the success of software projects highly influenced by the quality and expertise of their development workforce. Besides, software systems are complex engineering artifacts that demand high levels of technical specialization in different areas. The source code can reflect the developers' skills in programming languages, frameworks, and libraries, for example. This chapter is distributed in the following way. Section 2.1 presents concepts about difference between hard and soft skills. Section 2.2 explains the recruitment process. Section 2.3 investigates the role of GitHub in the analysis of software developers and team behaviors. Section 2.4 presents the works that are closely related to this thesis. Finally, we report our final remarks in Section 2.5.

2.1 Hard Skills and Soft Skills

Hard skills and soft skills are two crucial aspects of software development that organizations consider while hiring software developers [87]. Software developers need to have hard skills, which include technical know-how and aptitude in areas like programming languages, software tools, and frameworks [3, 27]. For a software engineer to properly carry out their daily job, these abilities are crucial. Technical interviews, coding tests, and reviews of the developer's contributions to open-source projects are frequently used to assess hard skills. Soft skills, on the other hand, are the non-technical competencies and character traits of a software development, including teamwork, communication, problem-solving, and flexibility [3, 27]. Non-technical talents or soft skills include psychological phenomena, including social interaction skills, communication, creativity, and teamwork [40]. These abilities are also crucial to a software developer's success and overall effectiveness.

Although soft skills are relevant to selecting a developer candidate, our focus is on hard skills in this work. Since evaluating soft skills is subjective, a face-to-face meet

is recommended to understand the candidates' soft skills in more details. In contrast, we can obtain extensive data about hard skills. They cover the theoretical fundamentals and practical experience one needs to complete the intended task [40]. Developers are frequently viewed as technical people [40, 65]. As a result, their technical competency is heavily stressed in both practical work and research studies.

2.2 Recruitment Process

Developers recruitment is defined as an employer's actions that are intended to 1) bring a job opening to potential job candidates who do not currently work for the organization, 2) influence whether these people apply for the job, 3) verify whether they maintain interest until a job offer is made, and 4) influence whether a job offer is accepted [54]. The recruitment process for selecting a new software developer is a crucial task for companies, as it determines the success of software development projects [54, 64, 67]. The process involves several strategies aimed at evaluating the hard and soft skills of software developers. Some common strategies used in the recruitment process of software developers include interviews and profile mining [54, 69]. Technical interviews are one of the most common strategies used to evaluate the technical abilities of software developers [22, 54]. The interviews usually consist of coding challenges, algorithms, and data structures, aimed at evaluating the candidate's problem-solving and coding skills [21, 69]. Another strategy is GitHub Profile Mining [41].

GitHub is a Web-based platform for version control and collaboration that has become the facto standard for software development professionals [49, 67]. Recruiters can review a candidate's GitHub profile to evaluate their coding skills, contributions to open-source projects, and overall development activity [49]. In this study, we apply this strategy based on a framework [74] that compute the hard skills. It is possible to conduct or combine with other strategies, such as Behavioral Interviews. Behavioral interviews are aimed at evaluating a candidate's soft skills, such as communication, collaboration, problem-solving, and adaptability [13, 67]. The interviews usually consist of questions that require the candidate to provide examples of how they have handled difficult situations in the past [13, 67].

Even when applying these strategies, the recruitment process of software developers is challenge. One of the biggest challenges is evaluating the technical abilities of software developers, as it requires a deep understanding of programming languages, tools, and frameworks [5, 13]. Another challenge is the lack of qualified software developers, which makes it difficult for organizations to find the right candidate [5]. The recruitment process

of software developers is a crucial task for organizations, and recruiters should use a combination of strategies to evaluate the technical and interpersonal skills of software developers. The use of profile mining, technical interviews, and behavioral interviews can provide valuable insights into a candidate's abilities and help recruiters make informed decisions [4]. Despite the challenges, the recruitment process is essential for the success of software development projects, and organizations should invest the necessary time and resources to ensure they find the right candidate [4].

The hiring process might be straightforward, but the initial step finding potential candidates can be extremely difficult due to technological advancements [4]. A company's bottom line may be significantly impacted by practical recruitment activities that make it stand out and more desirable to prospective employees [54]. Software developers can post and share their work in online areas thanks to social networking platforms, for instance GitHub. These experts develop a reputation within a field of expertise, frequently intending to find a job [54].

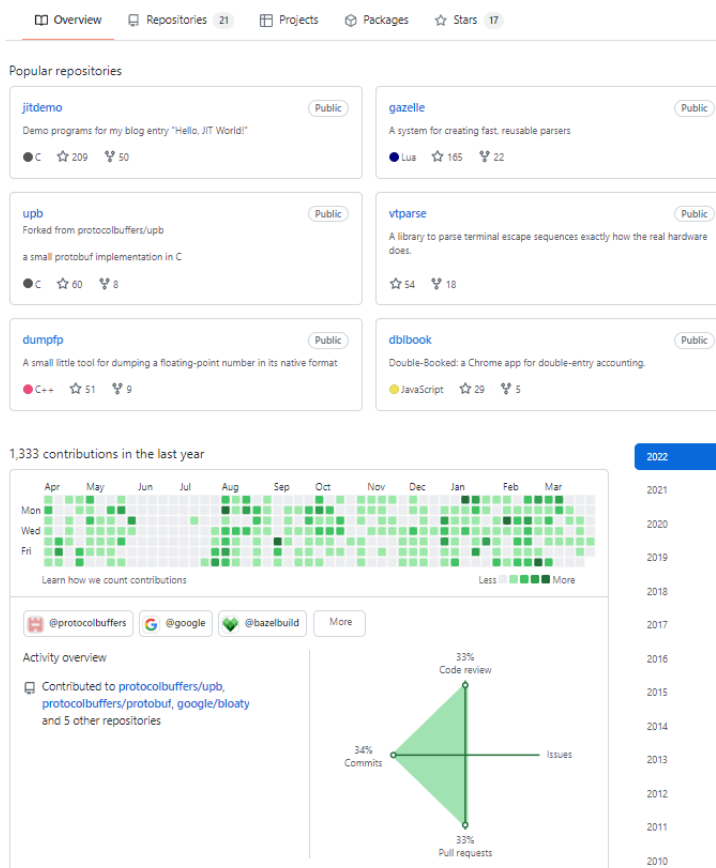
2.3 Using GitHub Data to Understand Software Developers

The use of data from GitHub to understand how software developers work and collaborate has become recurrent in software engineering studies [29, 30, 66]. Some studies seek to understand the behavior of developers concerning their interaction with peers [78]. For example, studies try to find who are the developers with peaceful and those with aggressive behavior and if these developers coexist productively in software development projects [79]. Besides, some studies investigate developers forms of work [24, 29] and seek to understand the emotional behavior of software developers [79]. Another topic that raises the interest of researchers is how knowledge transference, transparency, and collaboration happens in GitHub teams [25, 28]. Researchers [41] also study collaborative development practices in commercial projects using GitHub. Finally, another subject of study is how teams distributed in different geographic locations can be productive and perform tasks with quality using social programming platforms [23, 50].

Figure 2.1 shows an illustration of the information interface from GitHub. This platform provides an activity chart showing how many commits a user has made over the previous year. This information is prominent on a user's profile, but information about the content of changes that a developer has made is significantly more challenging to obtain [35]. Similarly, GitHub profiles link all projects that the user has contributed to or

forked. Nevertheless, they do not provide additional information about the contributions, programming languages, back-end & front-end profile, unit test, and core libraries used. The GitHub profiles can only be considered a signal of the developer's interests. Inexperienced GitHub users might not be aware that the information on the user's profile needs to be verified by examining the user's commits to identify their contributions [35, 53]. Many issues make it challenging to use GitHub profiles directly to assess programming skills [35].

Figure 2.1: GitHub Profile Example



Source: Prepared by the author, 2023

To accurately represent a developer's skills via the GitHub interface, a user needs to select all projects to which the developer has contributed and manually verify the contributions [35]. Programming languages for all projects need to be aggregated to get an overview of the developer's skills set. However, even a developer contributing to a Python project may not have changed any Python code in their contributions. Therefore, the developer's skills need to be verified to individual commit level [35]. It is also non-trivial to find developers using a particular programming language or library on GitHub. GitHub provides an advanced search feature that allows users to find developers that use a specific programming language. However, this does not indicate how much experience a developer has in a language and what other languages they are experienced.

2.4 Related Work

In this section, we provide a comparison of the related work by presenting a detailed table that highlights the key differences among them. Table 2.1 serves as a resource for understanding the distinctive characteristics of each approach. The ‘Author’ line indicates the author of each study. The ‘Metrics’ line shows the metrics used in each study. The ‘Programming Language’ line presents the programming language investigated in each study. The ‘Libraries’ line indicates the libraries evaluated in each study. The ‘Test’ line shows the studies focused on detecting unit tests. The ‘Back-end’ and ‘Front-end’ lines identify the types of technologies analyzed in these studies. Finally, the ‘Evaluation’ line provides the data used to evaluate each study. Notably, a common metric employed across the related work mentioned in Table 2.1 is the Number of Commits. However, our framework distinguishes itself as the sole solution capable of effectively detecting unit tests among all the existing approaches. This unique capability positions our framework as a tool for assessing the quality and robustness of software projects. By presenting these comparative insights, we contribute to the existing literature by shedding light on the distinctive features and benefits offered by our innovative framework.

Studies to identify and classify the knowledge of developers based on their contributions to online platforms have gained relevance in the field of software engineering [8, 30, 66, 68, 88, 95]. The evaluation of such approaches is essential to guarantee the quality of the analysis and to ensure the effectiveness of the specialist identification methods. For instance, tags can be mined from the project’s READMEs and commit messages. Just like we do in Chapters 4 and 5, some studies use questionnaires through forms [88, 99] or choose to carry out such an assessment manually [96]. Others compare current results to previous studies [57]. Researchers also performed automatic evaluations comparing data extracted from GitHub with the content of responses made in Stack Overflow (Q&A) [26, 30, 66, 68, 88, 101]. Approaches to identify and extract technical skills, known as hard skills, have gained prominence among research, often supported by tools to perform the extractions [39, 44, 66]. For instance, Greene and Fischer [35] proposed a tool named CVExplorer, which can be used to assist non-technical users in extracting, filtering, and identifying developers according to technical skills (programming languages, libraries, and frameworks) demonstrated across all of their open-source contributions, in order to support more accurate candidate identification.

Constantinou and Kapitsaki [26] proposed an approach that extracts developers’ expertise in different programming languages, measuring commit activity on GitHub. They consider both the quantity and the continuity of their contributions in isolate projects over time. The authors [26] evaluated the generated developers’ expertise profiles against recognized answering activity in Stack Overflow via a dataset of users that are

Table 2.1: Metrics and Hard Skills by Authors

Author		This work	Montandon et al.[66].	Constantinou and Kapitsaki[26]	Greene and Fischer [35]																
Metrics	No. of Commits No. of Imports Lines of Code Collaboration Knowledge Amplitude Knowledge Intensity Code Proficiency Experience	All programming Languages	No. of Commits No. of Imports Code churn No. of add library import No. of days since the 1 ^o commit No. of days since the last commit No. of days between commits Average interval of the commits No. of client projects	No. of Commits Lines of Code	No. of Commits Files Change																
						JavaScript	All programming Languages	All programming Languages													
									Detect all libraries since uses the import or require statements	REACT MONGODB SOCKET.IO	Not found	Not found									
													Detect. However, need of the folder test in the project	Not found	Not found	Not found					
																	Yes	Yes	Not found	Not found	
																					Yes
Evaluation	Type	Survey Interview	Survey	Experiment	Experiment																
	Participants	Developers Recruiters	Developers	Developers	Developers																
	Dataset	1,137 Developers ~12,000 Repositories 9 Lib 15 Recruiters	575 developers	234 Developers	33 Developers																

Source: Prepared by the author, 2023

active both in GitHub and Stack Overflow. As an outcome, each developer is a ranked list of programming languages according to the expertise over time.

Approaches using machine learning techniques are also gaining space [32, 47, 48, 100]. Although the evaluated model is totally analytical, it is worth highlighting such contributions. Montandon et al. [66] proposed an approach for identifying experts in software libraries and frameworks among GitHub users by applying Machine Learning. More specifically, the authors extended existing expertise identification approaches to the context of third-party software components. However, their key hypothesis is: “*when maintaining a piece of code, developers also gain expertise on the frameworks and libraries used by its implementation*”. To validate this hypothesis, they [66] carried out unsuper-

vised and supervised machine learning classifiers to identify experts in these libraries. Both techniques were applied using features (e.g., number of commits on files, number of client projects, etc.) extracted from selected GitHub users. Hauff and Gousios [36] created a pipeline that use natural language processing to automatically match job advertisements to GitHub users. Their pipeline “translate” both the developer profile and the advertisement into the Linked Open Data [14] space, where they can exploit the background information available in the Linked Open Data cloud to bridge the semantic gap. Their pipeline is threefold [36]: (i) extraction of concepts from job advertisements and social coding user data; (ii) weighting of concepts in such a way that more important concepts receive higher weights; (iii) matching of the 2 (job and coding profile-based) concept vectors.

Marlow and Dabbish [55] investigated how employers evaluate developers based on their GitHub accounts, which give hints about their activities, talents, motivations, and values. These indicators were thought to be more trustworthy than resumes. Open source contributions, according to employers surveyed by Marlow and Dabbish [55], are a sign that a developer has the correct values and is not just in the industry for the money. It was also stated that contributions to high status projects showed some level of proficiency.

Similar research was done by Singer et al. [91]. However, they focused on the social media accounts of developers in general. These include Stack Overflow, Twitter, Codewall, and Masterbranch, as well as GitHub and other profile aggregators. They discovered that some recruiters may find profiles challenging to comprehend or assess. A developer’s public activities must always be judged against their actual artifacts, such as their code, tests, documentation, or debates, in order to be truly evaluated. Furthermore, developers assessed employers and organizations to ascertain their credibility and suitability as potential partners. The majority of developers and recruiters are aware that a developer’s lack of activity does not necessarily indicate something significant; they may simply be a private developer or focus only on closed source projects.

Adnin et al. [1] presents the results of a study that examined employers’ experiences in hiring computer science graduates for software startup companies. The researchers conducted interviews with 23 employers from software businesses. The findings indicate that employers face challenges in finding qualified individuals who possess the necessary real-world experience and specialized skill sets required for startup positions. The companies also highlighted the significance of soft skills, such as communication, teamwork, and a strong willingness to learn.

Our study focuses on investigating the process of detecting experts in software development, improving the recruitment process, and exploring the challenges and problems of current methods. Unlike the related work mentioned, this study interviewed recruiters of software developers in three countries: Brazil, the United States, and Canada. The study aimed to gain insights into improving the detection of experts, creating a good cur-

riculum, and addressing the challenges faced in the recruitment process. The proposed methodology in this study can help to compute programming skills and can be used to improve the recruitment process of the software developer.

Our work distinguishes itself from its counterparts in many aspects. We show in Chapter 3 a framework that aims to compute programming skills, in general, independent of programming language, classify the profiles in the back-end and front-end, programming languages, libraries, and detect if a developer made a unit test. In related work [26, 35, 52, 66, 91], measures of lines of code written and file-specific experience pertain to expertise within a specific project for a developer. In general, the related work does not provide an enough overview profile of developer expertise that can be transferred among software projects. Different of other work [26, 35, 52, 66, 91], we are able to compute skills in many projects. While all of these approaches have a similar step in the same direction as us, they provide a weaker link between developers and their technologies. By using README, for example, CVEplorer [35] files rely on the primary source of developer expertise, while we extract language-specific APIs from files a developer has modified.

2.5 Concluding Remarks

Technology has become the backbone of our everyday lives, and programmers are needed to keep moving that technology forward. However, finding developers with specific skills is challenging. It is also hard self-evaluation hard skills in programming. This way, we need ways to compute programming skills from source code. This chapter described the fundamental concepts used in this thesis and discusses some related works. Besides, this chapter discussed the opportunities to use data from GitHub, its applicability, and concepts about programming skills. Furthermore, we discussed the applications of work related to the proposal of this chapter. In the next chapter, we present a framework to compute programming skills.

Chapter 3

A Framework to Identify Programming Skills

The number of projects and developers involved in open-source software has reached an impressive scale [106]. For instance, GitHub reported that in 2019 alone, over 10 million new developers joined and more than 44 million new projects were created [30]. GitHub serves as a social platform that developers can use to showcase their work [70]. However, identifying developers with the necessary skills for a software project can be a challenging task. As a result, there is a need for methods, frameworks, metrics, and tools that can support this process.

This chapter introduces FYI (For Your Information), a novel framework that employs static analysis to identify programming skills in the Git architecture. The framework is designed to handle large systems that consist of hundreds of source elements, such as files, lines of code, imports, packages, and classes, without compromising response time. FYI is flexible and lightweight, providing developers with an efficient way to compute programming skills from source code. We demonstrate the framework capabilities by means of three instances: library experts, profiles, and tech skills. Section 3.1 provides an overview of the framework, while Sections 3.2 to 3.4 show the three instances of the framework. Finally, Section 3.5 presents a tool to support the framework.

3.1 Framework Overview

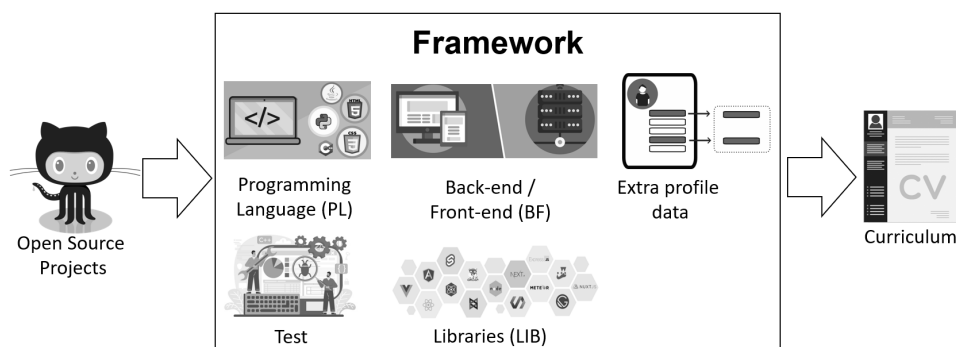
In this section, we present the components of the framework developed in this work. Figure 3.1 provides detailed information about the framework architecture, illustrating how the different components interact. Our comprehensive framework comprises five components: Programming Language, Back-end & Front-end, Extra Profile Data, Test, and Libraries. These components have been carefully designed and integrated to enable the efficient computation of programming hard skills through source code analysis. The

Programming Language component focuses on analyzing the language used in the source code. It encompasses a wide range of programming languages, ensuring the framework's versatility and applicability across various development environments. The Back-end & Front-end component analyzes the code pertaining to server-side and client-side development. It considers the implementation of web services, APIs, database interactions, and user interfaces to provide a holistic evaluation of a developer's proficiency in both back-end and front-end technologies.

The Extra Profile Data component leverages additional information about developers, such as their educational background, work experience, and contributions to open-source projects. This supplementary data enriches the evaluation process, offering a more comprehensive understanding of a developer's capabilities beyond mere source code analysis. The Test component plays a crucial role in assessing a developer's ability to design and implement effective test cases. It evaluates the presence of unit tests. The Libraries component examines the utilization of external libraries, within the source code. It considers how effectively developers leverage these resources to enhance productivity, implement complex functionalities, and adhere to industry best practices. In addition to the core components, our framework includes a module for repository cloning, which enables the automated retrieval of source code repositories for analysis. This functionality streamlines the process, saving time and effort in acquiring the necessary codebase for evaluation.

Furthermore, we have developed another module capable of generating a Developer's Curriculum based on the comprehensive results obtained from the framework analysis. This module synthesizes the developer's strengths, areas for improvement, and notable achievements into a professionally crafted curriculum, providing valuable insights for potential employers or collaborators. The integration and coordination of these components within our framework offer a versatile platform to identify programming skills through source code analysis. By leveraging the capabilities of this framework, stakeholders in the software development industry can make informed decisions regarding developer selection, skill enhancement programs, and project team formation.

Figure 3.1: An Overview of the Framework

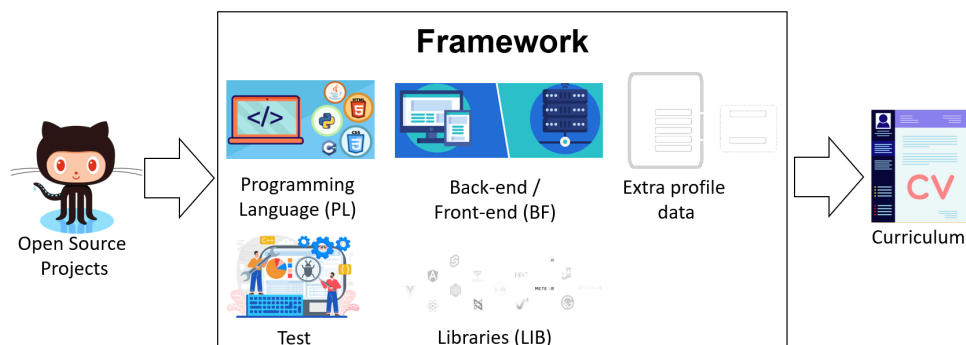


Source: Prepared by the author, 2023

3.2 Profile Instance

The Profile Instance serves as a valuable tool for identifying the programming languages predominantly used by developers, enabling the assessment of their expertise in commonly employed programming languages. In Figure 3.2, the components adopted for this instance are highlighted. The Profile Instance extends its analysis to determine developers' inclination towards either back-end or front-end development, based on the types of technologies they use. This capability provides valuable insights into a developer's skills and preferences, facilitating more targeted evaluations and effective team formations. One of the primary functionalities of this instance is the Programming Language component, which identifies the programming languages most frequently used by a developer. This information is crucial for evaluating a developer's breadth of knowledge and ability to work with diverse programming paradigms. The Back-end & Front-end component examines the types of technologies utilized in the source code, such as server-side, client-side libraries, and database interactions, providing valuable insights into the developer's focus and expertise in these specific areas. This information is important for team formation and role assignment, ensuring a balanced skill set and compatibility among team members. Additionally, the instance incorporates the Test component to identify whether a developer has implemented unit test cases in their source code. Analyzing the presence of unit tests provides an indication of a developer's commitment to software quality and adherence to best practices [45]. This evaluation aspect is essential for assessing a developer's attention to detail, code maintainability, and overall software reliability.

Figure 3.2: Programming Skills Profiles Instance



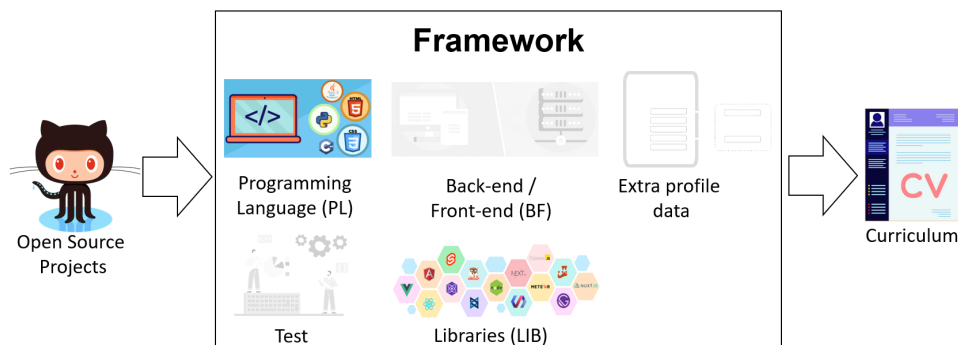
Source: Prepared by the author, 2023

3.3 Library Instance

In this section, we introduce the Library Instance, in Figure 3.3. The components of this instance highlighted are discussed below. The Library Instance incorporates the Programming Language component as mentioned in Section 3.2. Additionally, it introduces the new component Libraries. This instance plays a crucial role in analyzing developers' projects by identifying the primary programming languages they use and the libraries they are familiar with. By examining the codebase, the Library Instance provides valuable insights into developers' skill sets and proficiencies across various development environments. Understanding the libraries that developers have mastered is essential for making informed decisions regarding hiring new talent or forming effective development teams. The identification process relies on the usage of the “import” keyword, commonly employed in programming languages to incorporate functionalities from third-party libraries. Moreover, apart from identifying libraries, the Library Instance also recognizes the programming languages that developers have worked with.

The identification and comprehension of developers' preferred libraries and programming languages bring about several significant advantages. Firstly, it enables organizations to assess the extent of developers' knowledge and experience in utilizing essential tools and resources within their respective ecosystems. Proficiency in widely adopted libraries indicates developers' ability to leverage existing solutions, saving time and effort in the development process while fostering efficiency and productivity. Moreover, familiarity with popular libraries also implies a level of exposure to industry best practices, as these libraries often embody established coding conventions, design patterns, and architectural paradigms. This alignment with industry standards ensures that developers can contribute effectively to projects, follow coding guidelines, and collaborate smoothly within development teams.

Figure 3.3: Library Instance



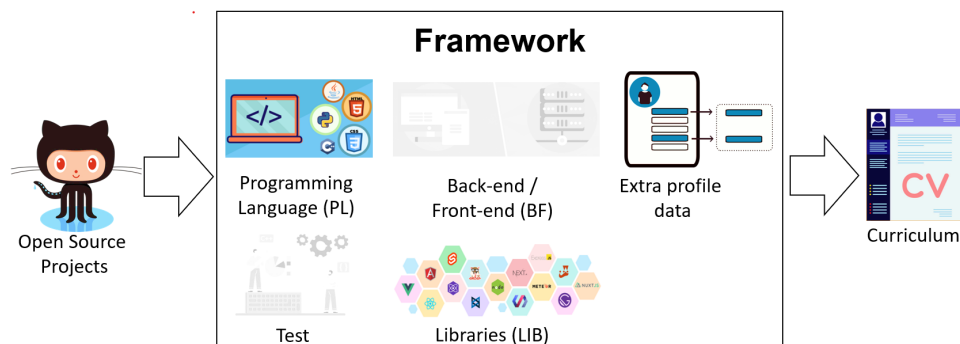
Source: Prepared by the author, 2023

3.4 Tech Skills Instance

In this section, we present the Tech Skills Instance, as illustrated in Figure 3.4. The components of this instance are discussed below. The Tech Skills Instance integrates the components introduced in Sections 3.2 and 3.3. This instance also includes a new component named Extra Profile Data. For this instance, we analyze developers' projects, extracting valuable information about their preferred programming languages and their mastery of third-party libraries. However, what really sets this instance apart is its ability to leverage additional data from external platforms, such as GitHub, to enrich the analysis. The Tech Skills Instance goes beyond the surface-level analysis of code repositories and incorporates metrics from platforms like GitHub. It considers factors, such as the number of repositories they have, a more holistic view of their expertise and reputation within the developer community. These additional data provide a context that can help evaluate developers' credibility, industry recognition, and level of engagement in open-source projects.

Furthermore, the Tech Skills Instance offers a concise yet comprehensive summary of developers' hard skills. It goes beyond simply identifying the programming languages they frequently use and explores further into their commitment and productivity. By analyzing the frequency of their commits and the strict lines of code written per programming language, a detailed understanding of their dedication, efficiency, and contribution to projects is obtained [45]. This information proves valuable for team leaders and project managers when making decisions related to resource allocation, project planning, and identifying potential skill gaps within the team [36]. The insights provided by the Tech Skills Instance have significant implications for various aspects of the software development process. Firstly, they aid in the selection and evaluation of new developers, allowing hiring teams to make informed decisions based on a comprehensive understanding of candidates' technical skills and industry involvement. Additionally, the instance assists in identifying areas where additional training or upskilling may be necessary, enabling companies to invest in their existing talent and foster continuous professional growth. By identifying developers' strengths and weaknesses, the Tech Skills Instance facilitates effective team composition, ensuring the right balance of expertise and complementary skill sets within development teams [45].

Figure 3.4: Tech Skills Instance



Source: Prepared by the author, 2023

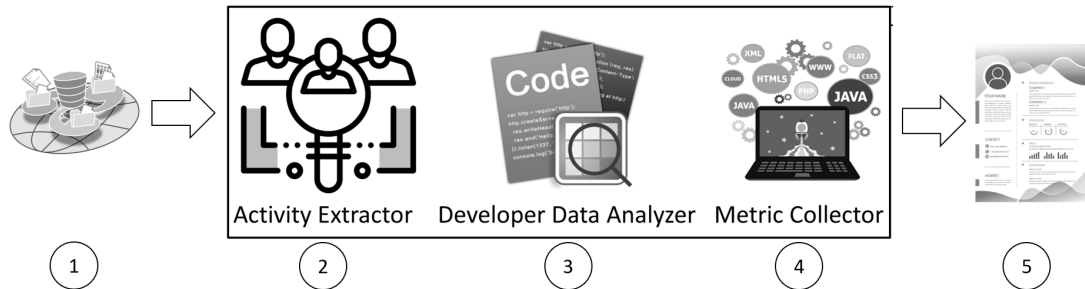
3.5 A Tool to Support Programming Skill Identification

Accurate representation of developer expertise has always been an important research problem [90]. Recruiters have hard work to select good developers with the right skills. Now, we present a tool to support our framework presented in Section 3.1 to compute programming skills from source code. This tool uses source code activities to compute programming skills in four perspectives: i) programming languages, ii) back-end & front-end profiles, iii) unit tests, and iv) libraries experts. In addition, this tool shows details of the developer profile, for example, short bio. Our tool works with Git architecture projects to compute metrics. The tool uses static analysis, but it avoids building the abstract syntax tree. Therefore, it reduces the response time when analyzing large systems with hundreds of source elements, for instance, lines of code, imports, and classes.

Our framework needs to process the massive data. Therefore, we developed a tool able to support the framework. Figure 3.5 presents an overview of the tool. The tool is composed of five steps. Steps 2, 3 and 4 are three main modules: *Activity Extractor*, *Developer Data Analyzer*, and *Metric Collector*, respectively. Step 1 – we select the target repositories to compute the developers' programming skills. Step 2 – the module Activity Extractor is the first module used in the process, which receives as input the projects mined from GitHub and stored locally. Step 3 – from the previous step, the module Developer Data Analyzer computes all data about each developer. This module is responsible for separating the number of commits to libraries from changes made for source code, changed files, changed LOC, and all changes made by a specific developer. In addition, this module is responsible to obtain the extra data profile, for example, short bio and number of repositories available at GitHub. Step 4 – Metric Collector is responsible for calculating some metrics. In this module, the tool computes the parameters according

to the heuristic developed for each metric. Finally, the programming skills are generated with developers' names and skills. This way, the tool shows in Step 5 a CV for each developer.

Figure 3.5: Tool Architecture Overview

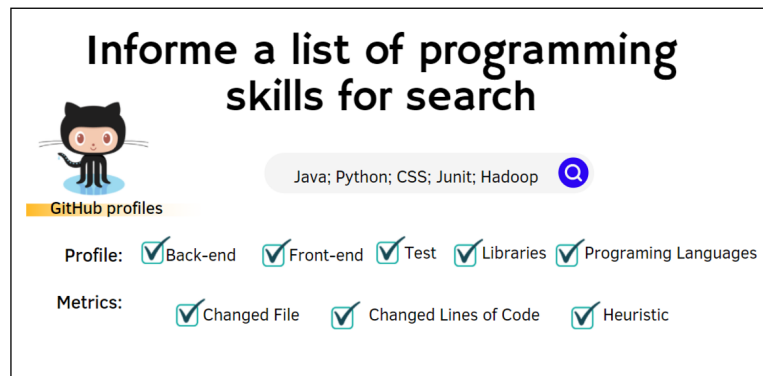


Source: Prepared by the author, 2023

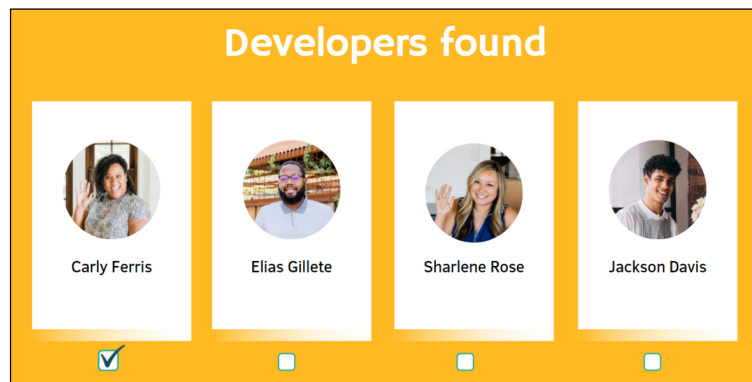
The tool was developed in Python, JavaScript, and React as a Web application. Our tool relies on Git projects from GitHub to extract implementation references and source code metrics to compute programming skills. Figure 3.6 (a) shows the main screen to select the setup profile and metrics to compute programming skills. Note that our tool computes programming skills, but the tool needs repositories to analyze. This tool requires entering programming skills wanted by a user. For example, we wish to search developers by technical skills: Java, Python, library Hadoop, a developer with a profile turned to the back-end and able to write unit tests. Therefore, the tool computes the programming skill from the dataset available. The tool does pick developers with these skills if people check the checkbox back-end, front-end, and test (Figure 3.6(a)). If people check libraries and programming languages, the tool shows in curricula the top-5 libraries and programming languages most used by developers. Figure 3.6 (b) shows the developers found¹ and the option to select a specific developer or all developers to show CV. Finally, Figure 3.6 (c) shows the curricula of a developer in detail. It presents the data about top-5 programming languages, top-5 libraries, profile to back-end & front-end, and test.

¹The developers presented are fictitious

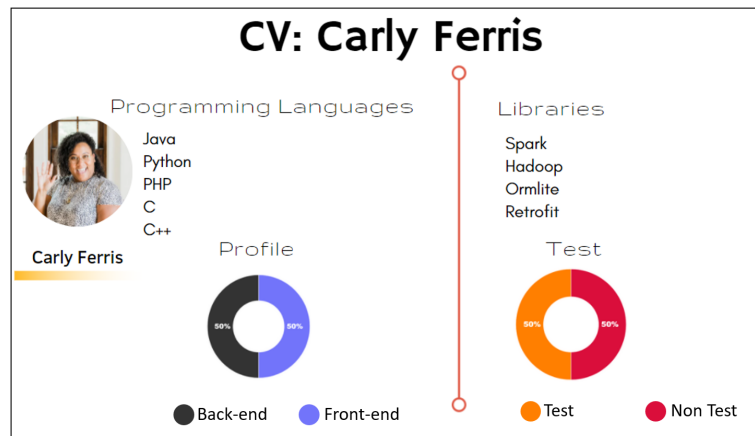
Figure 3.6: Tool to Compute Programming Skills



(a) Home Screen to Select Skills



(b) Developers Found



(c) Curriculum Details

Source: Prepared by the author, 2023

3.6 Concluding Remarks

This chapter presents the FYI framework, which is designed to evaluate the programming skills of software developers using data extracted from their GitHub repositories. The framework uses static analysis to identify hard skills, including programming language, back-end and front-end profiles, tests, and libraries. The tool implements the three instances of our framework: Profile instance, Library Instance and Tech Skills Instance. Chapter 4, we present an evaluation and proof of concept of the our framework from the “Profiles” instance to compute the programming skills of developers.

Chapter 4

Identifying Developer Profiles

It is easier to identify experts, measure, and rank their skills in some fields when we have objective measures to calculate the expertise. For example, time to run a marathon can be used to rank experts. However, it is much more challenging to find reliable and objective metrics to measure expertise in software engineering. This way, we need advanced metrics, frameworks, and tools as discussed in previous chapters. In addition, we need to evaluate them. The activities of software development, traditionally, are considered a collection of many tasks, such as design, coding, testing, and maintenance [99]. However, software development has evolved rapidly since the early 21st century [99]. It has created various pathways for mastering and expertise, such as expertise in programming languages (e.g., JavaScript, C/C++, Python, JavaScript, SQL), frameworks, and libraries (e.g., Node.js, Angular.js, and Spark), technologies (e.g., internet of things, deep learning, machine learning, computer vision, block-chain, quantum computing, cloud computing) and databases (e.g., MySQL and PostgreSQL) [99].

As a result, we need to evaluate developers' programming skills to provide reliable methods and tools. In our case, to evaluate the framework presented in Chapter 3, we select developers and their repositories from GitHub. We chose developers that made at least 300 commits in the last three years, i.e., at least 100 commits per year, to get active developers. Next, we clone their repositories, compute their programming skills, and apply a survey to the target developers. We implemented two metrics from our framework, (i) one based on Changed Files and (ii) the other based on Changed Lines of Code, to compute programming skills. For each metric, we compute three programming skills: Programming Language, Back-end & Front-end profiles, and Tests.

This chapter presents a proof of concept of the framework concerning profile identification, such as programming language, back-end & front-end, and unit test from source code analysis. Note that, at this moment, we evaluate the Profile Instance of the framework developed. This chapter is distributed the following way. Section 4.1 presents the goal and research questions of this study. Section 4.2 presents the steps performed to evaluate the framework. Section 4.3 shows the metrics adopted to compute the hard skills for the Profile Instance. Section 4.4 shows the dataset used. Section 4.5 presents the survey design with details about the questions and target developers. Section 4.6 formulates the

hypothesis from our research questions. Sections 4.7 to 4.11 presents the results for four research questions.

4.1 Goal and Research Questions

It is important to collect the developers' perceptions of their programming skills to get insights into the main strengths and weaknesses of the evaluated framework. Therefore, we decided to perform an exploratory study as an online survey [42]. This type of study is appropriate for approaching a large number of participants and allows the collection of a wide range of data [42]. We performed the study following a predefined protocol and documenting the results of each step. This allows the traceability between our study goal, the research questions, the questionnaire design, and the collected data from the participants. It also supports future studies that may aim at replicating. We define the scope of our study following the Goal/Question/Metric (GQM) method [10]. Therefore, the scope of this study can be summed up as follows.

Analyze a framework that compute programming skills
for the purpose of evaluating its applicability
with respect to developers perception of their programming skills
from the point of view of the researchers and practitioners
in the context of source code analysis of open-source software projects.

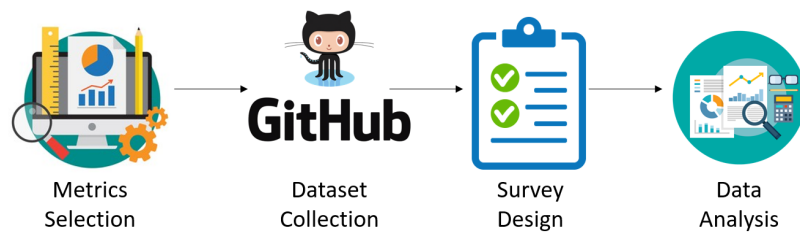
In order to achieve this goal, we framed our research around the following research questions (RQs).

- RQ1 – *What is the precision of the proposed framework to compute the programming language skills of a developer?*
- RQ2 – *What is the precision of the proposed framework to compute the back-end & front-end profiles of a developer?*
- RQ3 – *What is the precision of the proposed framework to compute the test development skills of a developer?*
- RQ4 – *What feedback do developers provide about source code analysis to compute programming skills?*

4.2 Evaluation Steps

The main goal of our analysis is to gather data on various variables to investigate the feasibility of assessing programming skills based on source code. Therefore, our study involves evaluating the skills of real developers using the framework described in Chapter 3 and the metrics developed to quantify these skills as described in Section 4.3. In this study, we look for the agreement of the developers regarding the results obtained from this framework. To achieve this, we designed the study in four steps: (1) Metrics Selection, (2) Dataset Collection, (3) Survey Design, and (4) Data Analysis. Figure 4.1 illustrates each step, and in the following sections, we provide detailed explanations of the different stages involved.

Figure 4.1: Study Steps to Identifying Developer Profiles



Source: Prepared by the author, 2023

Step 1 – Metrics Selection: In this step, we determine the metrics to be used in the Profile Instance of the framework. The selection of these metrics can vary depending on the specific context in which the framework is being employed. Section 4.3 presents these metrics.

Step 2 – Dataset Collection: We randomly selected 1,137 developers from GitHub. The inclusion criterion was that the developer had to have at least ten public repositories because we needed vast source code data to be analyzed. As a result, we obtained a list of 2,758 repositories to be analyzed. We present this step in Section 4.4.


Step 3 – Survey Design: We then conducted an empirical study through a survey with the selected developers. We sent emails asking them their agreement with the estimated programming skills computed by the framework evaluated in this study. This step is presented in Section 4.5.

Step 4 – Data Analysis: We conducted a qualitative and quantitative data analysis of the results in this last step. In addition, we translate the RQ into hypotheses. Section 4.6 shows this step.

4.3 Metrics used to Profile Instance

In this section, we present the metrics used to compute the Profile Instance in this study (see Section 3.2). These metrics can be changed according to the context of use. The metrics used in this thesis are based on practical observations, empirical studies, gray literature, and related work (see Chapter 2). We developed two metrics to compute programming skills from source code: (i) one based on Changed Files and (ii) the other based on Changed Lines of Code to compute programming skills. To illustrate how these two metrics work, Figure 4.2 presents four fictitious developers (Mary, Ashley, John, and Chris) and eight source code files they changed in a given period. The numbers describe the number of changed lines of code. Mary changed all eight files and five lines of code in the file “A.py.” John changed only five files and changed three lines of code (LOC) in file “A.py.”

Figure 4.2: Changed Files and the Respective Number of Changed Lines of Code for Fictitious Developers



	A.py	B.py	C.py	D.java	E.java	F.css	G.css	H.html
Mary	5	2	3	1	1	2	1	2
Ashley	3	1	4	5	2	1	3	4
John	3	4	-	-	-	1	1	1
Chris	-	-	2	2	3	1	2	-

Source: Prepared by the author, 2023

Concerning the metric based on Changed Files, we compute Programming Languages by the number of files changed that share the same file extensions. Considering the example in Figure 4.2, Mary modified 3 Python, 2 Java, 2 CSS, and 1 HTML files. Therefore, according to the Changed Files metric, her skill in programming languages is distributed as follows: 37.5% Python, 25% Java, 25% CSS, and 12.5% HTML. Table 4.1 presents the output of this metric for the given example.

Table 4.1: Distribution of Programming Languages Based on Changed Files

Developer	Python	Java	CSS	HTML
Mary	37.5%	25%	25%	12.5%
Ashley	37.5%	25%	25%	12.5%
John	40%	-	40%	20%
Chris	20%	40%	40%	-

Source: Prepared by the author, 2023

Similarly, the metric Changed Lines of Code considers the number of LOC added or removed by a specific developer. Considering the same example of Figure 4.2, to calculate Johns' skill distribution in programming languages, we need to count the total number of changed LOC for all files that share the same file extension. John, for instance, has a skill distribution in programming languages as follows: 70% Python, 0% Java, 20% CSS, and 10% HTML. Table 4.2 presents the output of this metric for the given example.

Table 4.2: Distribution of Programming Languages Based on Changed Lines of Code

Developer	Python	Java	CSS	HTML
Mary	58.82%	11.76%	17.65%	11.76%
Ashley	34.78%	30.43%	17.39%	17.39%
John	70%	-	20%	10%
Chris	20%	50%	30%	-

Source: Prepared by the author, 2023

We identify the profile of a developer, whether they are a back-end, front-end, or full stack developer. One key aspect of determining the profile is analyzing the programming languages utilized by the developer in their projects. For example, if a developer predominantly uses Python, it indicates an inclination towards back-end development. Conversely, if they frequently employ HTML and CSS, it suggests a focus on front-end development. However, it is important to note that developers can also possess skills in both areas, making them full stack developers. Additionally, we assess whether the developer has implemented unit tests in their projects. This assessment is performed by checking the presence of a dedicated test folder within the developer's project structure. By adopting this approach, we can evaluate the developer's commitment to software quality and their adherence to best practices. The inclusion of unit tests in the project signifies a developer's attention to detail, code maintainability, and overall software reliability. To measure these profile skills, we utilize two metrics: (i) Changed Files and (ii) Changed Lines of Code.

4.4 Dataset

To create our dataset, we conduct a filtering process to select the developers. Figure 4.3 shows the steps performed. First, we select 2,000 developers randomly according to trending GitHub¹. Second, we made a filter by developers with the top-10 programming languages²: JavaScript, Python, Java, Go, C++, Ruby, PHP, TypeScript, C#, and

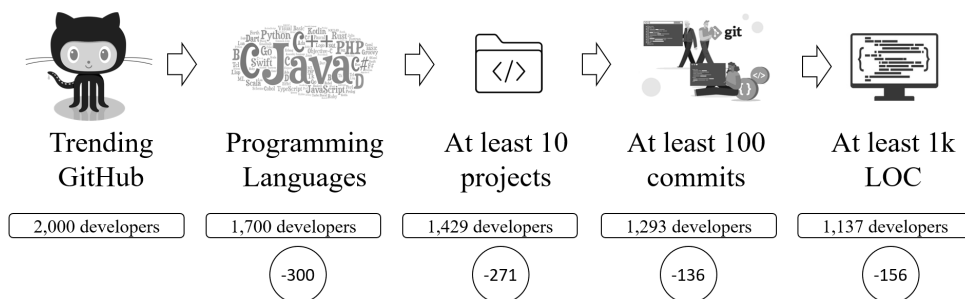
¹<https://github.com/trending/developers>

²https://madnight.github.io/github/#/pull_requests/2020/2

C. Besides, we selected the style sheet language CSS and HTML. Note that the programming languages selected are among the top-10 most used from GitHub³. As a result of this filtering step, we remove 300 developers and select 1,700 developers for the next criteria. Third, we delete developers whose repositories have less than ten projects from the last filter. Therefore, we removed 271 developers and selected 1,429 developers. Fourth, we select from the previous step developers with at least 100 commits to projects. Consequently, we discard 136 developers and select 1,293. In the fifth and last step, we select developers with at least a thousand lines of code committed. That way, we eliminated 156 developers and obtained the 1,137 developers able to participate in our analysis. After, we automatically cloned 2,758 repositories of these developers through Ghcloneall⁴. We extracted from each commit: lines of code and files modified by the file extensions, such as, *.py*, *.java*, and *.css*.

Finally, we computed the developers' skills as described in Chapter 3. We obtained 110 responses from the survey (about 10% response rate). These repositories provide the dataset with high variability in the size and complexity of projects, domains, and technologies. Together, these repositories have a history of over 2.5 million commits, measured at the moment of data collection, i.e., in August of 2020.

Figure 4.3: Steps to Select Developers from GitHub



Source: Prepared by the author, 2023

4.5 Survey Design

According to Easterbrook et al. [31], survey studies are used to identify the characteristics of a population and are usually associated with the application of questionnaires. Besides, surveys are meant to collect data to describe, compare or explain knowledge

³<https://github.info/>

⁴<https://pypi.org/project/ghcloneall/>

[42, 80]. In this study, the survey is the primary empirical method used to collect data to evaluate the framework for computing developers’ programming skills.

Target group – The survey is composed of developers from GitHub. *Questionnaire Structure* – Based on our goal and research questions, we designed a questionnaire supported by a *README* file that explains the details of the study. *Type of questions*– The questionnaire consists of a single answer. The goal of the questionnaire was to collect the participant opinion of two mini curricula generated from our scripts based on the Changed Files (Option A) and Changed Lines of Code (Option B) of our framework. Each questionnaire item asked the participants to rate their agreement with the curriculum items in their skills. The items used five Likert-scale options for answers: “Strongly agree”, “Agree”, “Neither agree nor disagree”, “Disagree”, and “Strongly disagree”. The questionnaire had no mandatory item.

Pilot study – We performed a pilot study with eight developers to assess the questions’ understandability and estimate the time required to answer them. We encouraged the eight developers to take notes on any problems or doubts regarding the meaning of the questions or their answers and track the time they spent filling out the questionnaire. As a result, we changed one question classified as confused by two developers. *Questionnaire length* – The designed questionnaire includes up to five questions written in English (Google Forms)⁵ as presented in Table 4.3. The time needed for answering the questions of the questionnaire was between three and five minutes.

Table 4.3: Survey Questions

ID	Questions		
SQ1	Your GitHub Username:		
		Option A	Option B
SQ2	With respect to programmings languages: we mined some languages you are likely to know. How much do you agree or disagree with the distributions presented in Option A and Option B?	Likert-scale*	Likert-scale*
SQ3	With respect to back-end and front-end: we mined your possible profile. How much do you agree or disagree with the distributions presented in Option A and Option B?	Likert-scale*	Likert-scale*
SQ4	With respect to software test: we mined your likely knowledge. How much do you agree or disagree with the distributions presented in Option A and Option B?	Likert-scale*	Likert-scale*
SQ5	Please leave any comments that you consider relevant to our research.		

*Likert-scale options for answers: “Strongly agree”, “Agree”, “Neither agree nor disagree”, “Disagree”, and “Strongly disagree”

Source: Prepared by the author, 2023

⁵<https://www.google.com/forms/>

4.6 Data Analysis

From the survey results, we performed quantitative and qualitative data analyses to address the research questions. For research questions RQ1, RQ2, and RQ3, we conducted a descriptive study of the data and statistical tests to identify the precision similarity of both metrics. For the statistical tests, each research question was translated into hypotheses.

Regarding *RQ1*, we defined the following hypotheses:

Null Hypothesis, H_{0RQ1} : there is no difference in the precision of both metrics regarding the programming language skills of a developer.

Alternative Hypothesis, H_{1RQ1} : there is a difference between both metrics regarding programming language skills of a developer.

Then, our hypotheses can be formally stated as:

$$H_{0RQ1}: \bar{x}_{changeFile} = \bar{x}_{changeLOC}$$

and

$$H_{1RQ1}: \bar{x}_{changeFile} <> \bar{x}_{changeLOC}$$

Regarding *RQ2*, we defined the following hypotheses:

Null Hypothesis, H_{0RQ2} : there is no difference in the precision of both metrics regarding the back-end front-end skills of a developer.

Alternative Hypothesis, H_{1RQ2} : there is a difference between both metrics regarding back-end & front-end skills of a developer.

Then, our hypotheses can be formally stated as:

$$H_{0RQ2}: \bar{x}_{changeFile} = \bar{x}_{changeLOC}$$

and

$$H_{1RQ2}: \bar{x}_{changeFile} <> \bar{x}_{changeLOC}$$

Regarding *RQ3*, we defined the following hypotheses:

Null Hypothesis, H_{0RQ3} : there is no difference in the precision of both metrics regarding the test development skills of a developer.

Alternative Hypothesis, H_{1RQ3} : there is a difference between both metrics regarding software testing skills of a developer.

Then, our hypotheses can be formally stated as:

$$H_{0RQ3}: \bar{x}_{changeFile} = \bar{x}_{changeLOC}$$

and

$$H_{1RQ3}: \bar{x}_{changeFile} <> \bar{x}_{changeLOC}$$

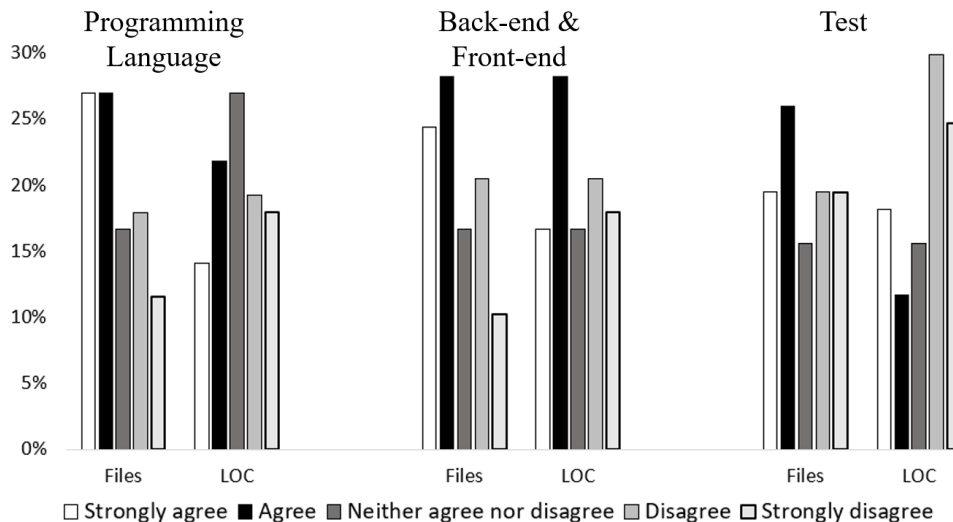
We perform statistical tests over the data in *RQ1*, *RQ2*, and *RQ3*. The independent variables for the first three research questions are the Change Files and Change Lines of Code. Precision is our dependent variable. We verify if answers from every case deviate from normality using the Shapiro-Wilk test (p-value ranges from 0.7 to 0.9). Since normality was not always met, we used a Wilcoxon Signed-Rank Test with a 95% confidence interval. We conducted a Wilcoxon signed-rank test [83] to check the significance of the difference between the two paired groups. Because we are interested in the differences in the values, we report the effect size with the median and mean differences. We use R to conduct these analyses.

For *RQ4*, we used an approach inspired by the open and axial coding phases of Ground Theory [93]. Two researchers analyzed the responses to open questions individually and marked relevant segments with “code” (tagging with keywords). Later, the researchers compared their codes to reach a consensus and grouped them into relevant categories. We examined the data line-by-line using the following questions as a lens to identify codes (open coding) [93]: (1) What is this saying? What does it represent? (2) What is happening here? (3) What is at issue here? (4) What is the participant trying to convey? (5) What process is being described? Consequently, it is possible to count the number of occurrences of codes and the number of items in each category to understand what recurring aspects are pointed out by the participants and then discuss possible lessons learned. The developers are our oracle to evaluate the framework. Then, we compare the results provided by the framework with the answers from developers. For this, we calculated the precision. The precision is the ratio $\frac{tp}{tp+fp}$ where *tp* is the number of true positives and *fp* the number of false positives [89]. The precision is intuitively the ability of the classifier not to label as positive an answer that is negative [89].

4.7 Overview Results

This section presents the summary of our quantitative results. At first, we present an overview of results for the Changed Files and Changed Lines of Code metrics. Figure 4.4 shows the opinion of the developers of each metric used in our framework. In this figure, we have three blocks, one for each evaluated programming skill. The first block of charts represents programming language skills. The second block depicts the back-end & front-end profiles. Finally, the last block shows the test skills. Note that, in this phase, we did not evaluate the library skill. We conducted an independent evaluation of programming skills with respect to a library in Chapter 5. Concerning the Changed Files metric agreement, we observe in the programming language perspective that 27% of the participants answered “strongly agree” and “agree”. On the other hand, in the Changed Lines of Code, 14%, and 22% of the participants answered respectively “strongly agree, and “agree”. Concerning disagreements, we have 12% and 18% to “strongly disagree” and “disagree” with the Changed Files. While for Changed Lines of Code, we observe 18% and 19%, respectively, to “strongly disagree” and “disagree”. This way, we note that the Changed Files, in general, are better evaluated by developers to capture their programming language skills.

Figure 4.4: Overview



Source: Prepared by the author, 2023

Overall, we conclude that both metrics used to compute test skills in this study were not the right way to compute test skills. Developers mostly disagree with both metrics for the Test development. Amongst the other negative comments, we highlight one that the developer reported: “I am a primarily front-end and JS expert, and I maintain 2 of the most popular testing frameworks in the world. Your mining needs a little work”.

This feedback confirms that both metrics fail to compute test skills for some developers. Therefore, it is necessary to apply more efforts to develop new metrics, combine them, and detect the source code of the test. One positive aspect of the presented results is that there is a considerable percentage of developers who agree or strongly agree with the framework's ability to capture programming language skills using the Changed Files metric. This indicates that the framework has the potential to be a useful tool for assessing developers' skills in this particular area.

4.8 Results for Programming Language Skills

This section analyses the results of our empirical evaluation to research question one (RQ1).

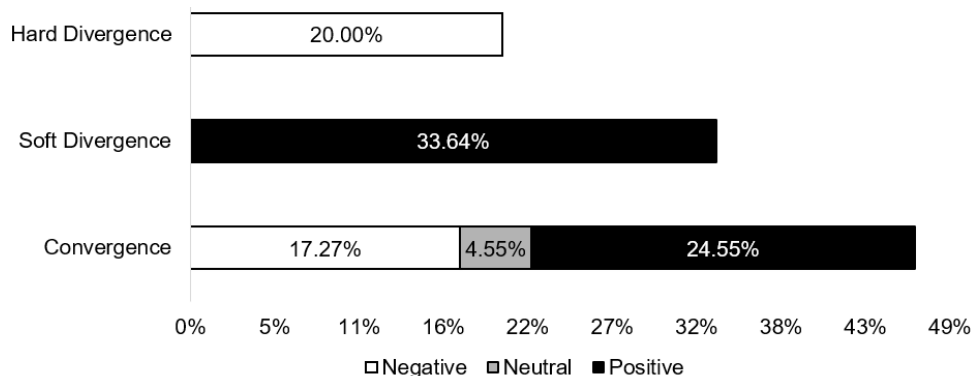
RQ1 – What is the precision of source code analysis to compute the programming language skills of a developer?

For RQ1, we found a statistically significant difference between Changed File and Changed Lines of Code (p-value= 0.02423) with the Wilcoxon-Signed Rank test. This result allows us to refute the null hypothesis. Therefore, the metrics Changed Files and Changed Lines of Code are statistically different. To analyze the precision, we sum “agree” with “strongly agree”. We obtained the 54% precision to Changed Files and 36% to Changed Lines of Code. To understand the factors leading a developer not to agree, we create three categories: Hard Divergence, Soft Divergence, and Convergence. The category Hard Divergence indicates the number of developers against one metric and favorable to the other. The category Soft Divergence means that a developer agrees or disagrees with a given metric and is neutral about the other. Finally, Convergence indicates they share the same opinion (agree, disagree, or neutral) for both metrics. Our focus is on Hard Divergence because our goal is to understand how the metrics differ.

Figure 4.5 presents the opinion of the developers into these three categories. In this figure, we use three colors for Convergence: light grey (negative), dark gray (neutral), and black (positive). “Hard Divergence” and “Soft Divergence” correspond to, respectively, 20% (22) and 33.64% (37) of the answers. Therefore, we can observe that 20% of the developers agree with a metric and disagree with another, 33.64% agree with one and are neutral with the other. The remainder of the answers (46.37%) shows Convergence, where: 17.27% agree on neither of the two metrics, 24.55% agree with both metrics, and 4.55% are neutral towards both. Our qualitative analysis (SQ5) investigates patterns of answers. We note that the metrics were discrepant in some causes. That is, they presented a result different from the other. In some cases, the metrics present the same

languages. However, a metric presented a different sorting (e.g., Java is 1st according to one metric and 3rd language according to the other metric). In general, the metric that prioritizes the most mainstream languages from the viewpoint of a developer is that they agree—for example, C++, JavaScript, and Java were considered mainstream languages. On the other hand, when the metric ranks CSS and HTML above the other languages, overall, the developers disagree with this metric.

Figure 4.5: Programming Language Skills



Source: Prepared by the author, 2023

RQ1 summary – We obtained 54% precision to Changed Files versus 36% to Changed Lines of Code. However, it is possible to see that less than 20% strongly disagree with both evaluated metrics. Moreover, concerning the programming languages, developers tend to select the metric that prioritizes the mainstream languages—for example, Python, JavaScript, and Java.

4.9 Results for Back-end & Front-end Profiles

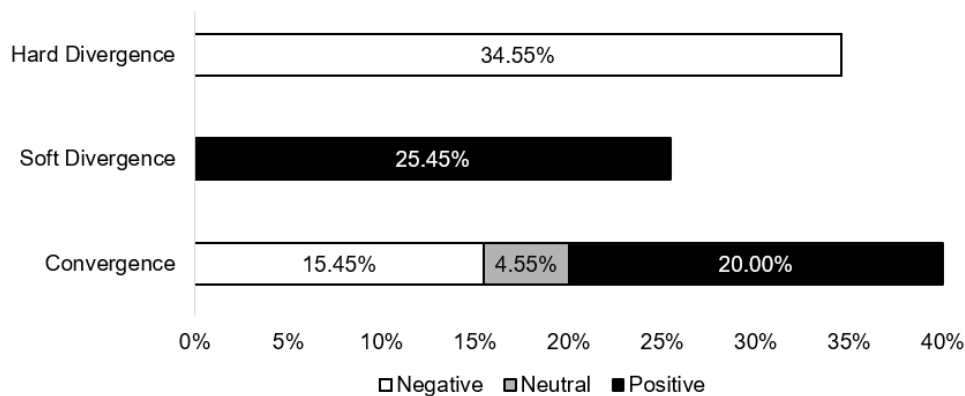
This section analyses the results of our empirical evaluation to research question two (RQ2).

RQ2 – What is the precision of source code analysis to compute the back-end & front-end profiles of a developer?

To investigate *RQ2*, we conducted the same configuration presented in *RQ1*. The Wilcoxon signed-rank test indicated $p\text{-value} = 0.06896$. Since the $p\text{-value}$ is 0.06896 , which is greater than our 0.05 significance level, we could not reject the null hypothesis in this case. Figure 4.6 shows the results of this research question. In this figure, we have the same configurations presented previously. “Hard Divergence” and “Soft Divergence” correspond

to 34.55% (38) and 25.45% (28) of the answers. On the other hand, “Convergence” corresponds to 40% of the answers, where 15.45% (17) of the developers disagreed with the metric. This way, to these developers, both metrics cannot represent their Back-end and Front-end profiles. For this RQ, we obtained a precision of 53% to Changed Files versus 45% to Changed Lines of Code. Therefore, we observe that Changed Files from survey results obtained better precision than Changed Lines of Code.

Figure 4.6: Back-end & Front-end Skill



Source: Prepared by the author, 2023

RQ2 summary – Overall, the precision of the metric evaluated is medium for back-end & front-end profiles. We obtained 53% precision to Changed Files versus 45% to Changed Lines of Code.

4.10 Results Test Development

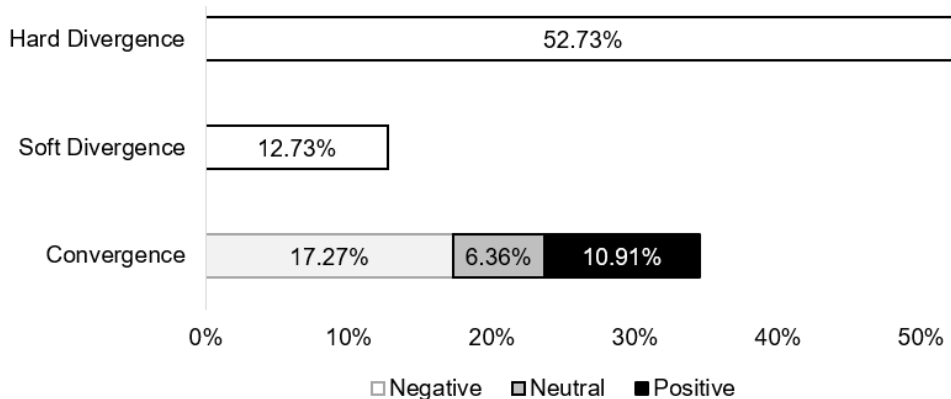
This section analyses the results of our empirical evaluation to research question three (RQ3).

RQ3 – What is the precision of source code analysis to compute the Test development skills of a developer?

To answer this research question, we investigate the test skills of developers computed by the two metrics, Changed Files, and Changed Lines of Code. We used the same configurations presented in the last research questions. The Wilcoxon signed-rank test indicated $p\text{-value} < 0.01$. Since the p -value is smallest than the 0.05 significance level, we reject the null hypothesis. Therefore, there is a difference between the two metrics evaluated concerning test development skills. Figure 4.7 presents the results in the same way as presented previously. From all skills evaluated, tests obtained the worst results.

We compute 52.73% (58) of answers for Hard Divergence against 12.73% (14) for Soft Divergence. From these results, we can observe that 17.27% (19) converge negatively to both metrics. Therefore, to 19 developers, both metric cannot compute their skills in tests. We also note that only 10.91% (12) of developers agree with both metrics evaluated, and 6.36% (7) are neutral.

Figure 4.7: Test Skill



Source:

Prepared by the author, 2023

RQ3 summary – We obtained a precision of 45% to Changed Files versus 30% to Changed Line of Code. We argue that the metrics alone may not precisely compute programming skills for test development.

4.11 Feedback from Developers

This section analyses the results of our empirical evaluation to research question 4 (RQ4).

RQ4 – What feedback do developers provide about source code analysis to compute programming skills?

To gather and synthesize such feedback, we employed an approach inspired by the coding phase of ground theory (as discussed in Section 4.6). We grouped the identified codes into eight categories (shown in Table 4.4): *Imprecision of detected skills*, *Expertise not captured*, *Lack of data from private repositories*, *Problems in presentation*, *Positive feedback*, *Negative Feedback*, and *others*. The most frequent feedback is related to *Imprecision of detected skills* and to *Expertise not captured*, mentioned by 17 and 16 participants, respectively. The first represents the developers' feedback about misalignment identified by the metric. That means imprecise detection of programming language, back-end &

front-end, and test development. Seven participants expressed concerns about the developer's lack of treatment regarding what was created in contrast to what was committed, i.e., third-party code (e.g., automated code, code provided by frameworks, boilerplate code) computed in the developer skills.

Six participants did not agree with the metric for calculating their front-end and back-end profiles. Four participants pointed out problems in the metric for calculating language skills, such as wrong recognition of languages. Two participants pointed criticism towards the results of Test Development. The codes categorized as *Expertise not captured* are mostly related to the absence of specific programming languages or testing skills in the resulting curriculum. Eight participants indicated that the results did not compute any testing skills. These comments helped us understand that the metric of simply considering directory path referencing to test is not enough to capture fragments of codes that implement tests. Seven participants mentioned they missed specific languages in their respective curricula. From these cases, 6 participants mentioned languages that our scripts do not capture (e.g., R, Latex, and Swift). However, developers were also missing languages the script was supposed to capture (HTML, TypeScript, and CSS). Finally, one participant mentioned that the metric could not assess their front-end & back-end orientation since they do not post it on GitHub.

Concerning *Positive feedback*, we have a favorable opinion of the developers about the evaluated metric. Between these comments, we highlight one in particular, in which the developer said that the metric has the potential to use in the industry if improved. Another category we created was *Problems in presentation*. This category represents the problems concerning the presentation of the curriculum. For instance, the type of charts, layout, percentage, and text in the curriculum. This problem typically occurred when the curriculum showed similar data for both metrics. The codes categorized as *Lack of data from private repositories* indicate the observations of the developers about data not captured at private repositories. Some developers related that their skills were not computed correctly because most of their source code is available in private repositories. *Negative Feedback* reports comments from developers that did not like the results presented by the metric. For example, a developer-related: "None of these results captures the useful nuance".

Table 4.4: Feedback Categories

Category	#
Imprecision of detected skills	17
Expertise not captured	16
Other	6
Positive feedback	6
Problems in presentation	5
Lack data from private repositories	4
Negative Feedback	4
Time factor	3

Source: Prepared by the author, 2023

RQ4 summary – In general, we observe that the most frequent feedback is related to the imprecision of detected skills and Expertise not captured.

4.12 Threats to Validity

As any empirical study, ours is also subject to several threats of validity. Therefore, in this section, we detail the threats that may affect the validity of the study and how they are handled.

Construct Validity refers to the extent the operational measures in this thesis really represent what we intended to measure. Therefore, it is concerned with issues that may arise due to improper design of the survey instrument, which may not correctly measure what it is supposed to measure. We believe this issue has been mitigated since the questionnaire was iteratively designed and updated by the authors based on well known [104]. Further, we conduct a survey pilot to improve our survey before applying to all developers. Self-selection is another threat. However, we try to select the biggest group of developers from GitHub without evaluating the number of commits, amount of stars, programming languages, and organizations. The use of generated distribution of skills based on GitHub contributions as a proxy for general individual skill level evaluation is a limitation. Professional experiences can be much broader than individual contributions to open-source.

Internal Validity is related to uncontrolled aspects that may affect the results, such as the subjects' experience. We developed many steps to mitigate this threat: (i) Respondents were assured of their anonymity to avoid evaluation apprehension; (ii) We sent an email only to developers mined by GitHub; and (iii) All statements were not mandatory.

External Validity is related to the possibility to generalize our results. The sur-

vey was answered by 110 subjects. This number might not be a representative sample. However, our sample is diversified; the subjects have different programming experience and work with freelancers or other companies. Therefore, we believe that these steps contributed to obtaining a sample that is quite heterogeneous in terms of knowledge, job role, profile, and company. The selected projects are also open source. It is possible that closed-source projects developed within an industrial setting could have different properties, but several of the systems we include in our sample are developed by industry. The participants were sampled by convenience. Therefore, generalizing the results to a different population (e.g., software developers proprietary) poses a threat.

Conclusion Validity is related to the possibility of reaching incorrect conclusions about observations due to errors, such as the use of inadequate statistical tests or measures. The participants did not answer the questionnaire honestly. Participants may feel that they needed to provide positive feedback for better scores. This threat is minimized by announcing that the questionnaire is not mandatory, so participants provide feedback willingly and without pressure. In this study, we only used the absolute number and percentages to compute programming skills. In addition, we only considered complete responses in our analysis. Another threat is subjectivity. We manually interpret participants' feedback, and subjective or bias might be included during the experiment. This threat is addressed by cross discussion among co-authors. Although we did not inform the participants about our research goals, they were aware of being part of an investigation on programming skills. Thus, there could be the risk of hypothesis guessing. It is noteworthy that we did not use advanced techniques, such as GitHub Linguist, to detect the type of programming language, for example. The reason is because our study aim to evaluate in this first moment the feasibility of computing programming skills from the source code analysis. In the future, we can use this technology to detect the primary programming languages, source code generated automatically, and binary data.

4.13 Concluding Remarks

Contemporary software development demands breadth and in-depth knowledge for a tremendously large set of technologies, tools, and practices. Retaining a competitive development team becomes even more pronounced considering the rapid evolution of software technologies and the continuous emergence of novel programming languages, frameworks, and libraries. Considering that developers' skills also evolve, project managers face the challenge of tracking the kind and extent of knowledge within the development team to map people to tasks efficiently. Therefore, it is essential to select right develop-

ers to compose the time development. Identifying these developers is not easy, then it is necessary methods able to compute programming skills from source code in order to provide results must be reliable. This way, besides purpose a framework and metrics, we conducted an evaluating of this framework with real developers from GitHub. In Chapter 5, we depth our evaluation, focusing on compute libraries most used by developers, in order to identify library experts.

Chapter 5

Identifying Library Experts

Identifying the third-party libraries in which a developer excels is crucial in the field of software development. These libraries play a pivotal role in boosting productivity, facilitating rapid development, and delivering robust and reliable solutions. By identifying the specific libraries that a developer has mastered, we gain valuable insights into their skill set, expertise, and ability to effectively utilize external resources. This knowledge not only helps in making informed decisions when hiring or forming development teams but also enables us to allocate resources strategically and assign projects that align with the developer's strengths. Moreover, understanding a developer's proficiency in specific libraries fosters more effective collaboration and knowledge-sharing within the team, fostering a culture of continuous learning and innovation. Overall, identifying a developer's expertise in third-party libraries empowers organizations to build high-performing teams and deliver exceptional software solutions that meet the demands of today's dynamic technological landscape [12, 63]. Expertise encompasses the characteristics, skills, and knowledge that distinguish experts from novices and less-experienced individuals [63, 107]. Software development is a complex activity as it relies on the knowledge of individuals, and each software or new feature introduces new challenges. Therefore, it is crucial to identify developers with multidisciplinary skills, although this task itself can be complex. This chapter presents a proof of concept of the framework concerning the Library Instance. This specific instance of the framework calculates the programming skills of developers in relation to the most commonly used programming languages and libraries. Therefore, in this study, we focus on computing the expertise of developers in Java-specific libraries.

This chapter is organized as follows. Section 5.1 presents the research questions that guide this study. Section 5.2 describes the steps taken for the empirical evaluation. Section 5.3 introduces the metrics adopted to compute hard skills in libraries within our framework. Section 5.4 outlines the construction of the dataset used to evaluate the framework. Section 5.5 presents the setup of the survey conducted, and Section 5.6 provides an overview of the results obtained from the developers who participated in the survey. Sections 5.7 to 5.9 present the results of the empirical evaluation in response to the research questions. Section 5.10 discusses the threats to validity and the corresponding treatments. Finally, Section 5.11 concludes this chapter with our final remarks.

5.1 Goal and Research Questions

The primary goal of this study is to evaluate the applicability of the framework to identify library experts from source code analysis. We are interested in whether the framework can precisely identify experts in a specific library. We are also concerned with assessing the relevance of the results provided by the framework. For this purpose, we select the 10 most popular and standard Java libraries among GitHub developers. One library was later excluded (Section 5.4) and, therefore, we evaluate the framework with the 9 most popular libraries. To achieve this goal, we use the Goal-Question-Metric framework to select measurements of source code. The GQM method proposes a top-down approach to define measurement; goals lead to questions, which are then answered with metrics [10].

Table 5.1 shows the GQM with the research questions and metrics investigated in this study. Through RQ1, we are interested in investigating the efficiency of the number of commits (metric) to indicate the level of activity of a developer in a specific library. In other words, we aim to analyze the number of commits involving a specific library performed by a developer to compute their level of activity in the library. With RQ2, we aim at assessing the knowledge extension based on the number of imports to a specific library. That is, from all imports made by a developer at the source code, we investigate the number related to the specific library. Finally, the last research question (RQ3) analyzes the knowledge intensity of the developers from the number of LOC related to the library (metric). In this last question, we aim to evaluate the amount of LOC implemented by a developer using a specific library. For this purpose, we evaluate the relation of total LOC and LOC related to a specific library.

Table 5.1: The Metrics Analysis as GQM method

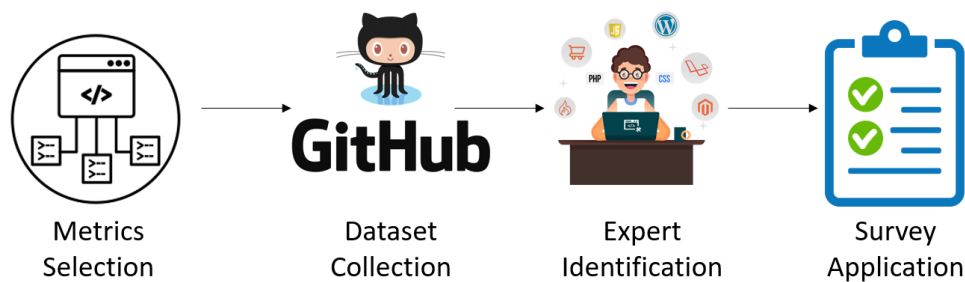
Questions	Metrics
RQ1– How to evaluate the level of activity of a developer in a library?	Number of Commits
RQ2– How to evaluate the knowledge extension of a developer in a library?	Number of Imports
RQ3– How to evaluate the knowledge intensity of a developer in a library?	Lines of Code

Source: Prepared by the author, 2023

5.2 Evaluation Steps

This section describes the steps to evaluate the identification of library experts from source code. To answer the research questions presented in Section 5.1, we designed a mixed-method study composed of four steps: 1) *Metrics Selection*, 2) *Dataset Collection*, 3) *Expert Identification*, and 4) *Survey Application*. Figure 5.1 presents the steps of our research, which are discussed next. For *Metrics Selection* (Section 5.3) we create metrics to identify library experts. In the *Dataset Collection* (Section 5.4), we clone the projects from GitHub. For *Expert Identification*, we compute the skills of developers based on three metrics: *Number of Commits*, *Number of Imports*, and *Lines of Code*. Finally, we performed a *Survey*. This survey was conducted to evaluate the accuracy of the framework according to the responses of developers. Section 5.5 presents details about the survey.

Figure 5.1: Study Steps to Identifying Library Experts



Source: Prepared by the author, 2023

5.3 Metrics used to Library Instance

We adopted metrics to compute libraries experts. As cited in Chapter 3, these metrics were developed empirically and can be changed by users of the framework. For this instance of the framework, we implement three metrics: Number of Commits, Lines of Code, and Number of Imports. Number of Commits calculates the activity of each developer through the number of commits using a particular library. Through this metric, it is possible to measure the amount of use of the library in a project that a specific developer works. Number of Imports presents the intensity of use of a particular library. For this metric, we count all imports to the library. Lines of Code has a heuristic to count the amount of lines of code related to a specific library, as follows. First, we obtain the ratio of changed lines of code by the number of all imports in the file. Then, we multiply the ratio by the num-

ber of imports related to the library ($LOC = \frac{\# \text{ of LOC Altered by Commit}}{\# \text{ of All Imports}} \times \# \text{ of Library Imports}$). The concept of skills is essential in the proposed framework because we aim to search for professionals with specific skills and with different capabilities for each type of task [6, 7, 18, 20, 38]. That is, a recruiter needs to have several dimensions of skills from each candidate so that they can have a benchmark to select the most appropriate candidate for the job [2, 51].

5.4 Dataset

To create our dataset, we select the 10 most popular and common Java libraries among GitHub developers: Hibernate, Selenium, Hadoop, Spark, Struts, GWT, Vaadin, Primefaces, Apache Wicket, and JavaServer Faces. The selection was made based on a survey provided by Stack Overflow¹ in 2018 with answers of over 100,000 developers around the world. Table 5.2 summarizes the definitions of each library. All definitions of the libraries were retrieved from Stack Overflow and their Web pages. We selected Java because it is one of the most popular programming languages² and there are many Java projects available on GitHub.

Figure 5.2 illustrates the criteria for defining our dataset. We apply the following exclusion criteria. 1) We excluded systems with less than 1 KLOC because we considered them toy examples or early-stage software projects. 2) We removed projects with no commit in the last 3 years because the developers may forget their code [46]. Finally, in the last exclusion criteria, 3) we removed projects which did not contain imports related to the selected libraries. Besides, we excluded all official projects of these libraries, because we assume all developers of a library project are experts in the corresponding library. We also removed libraries with less than 100 projects (the case of JavaServer Faces) because we need a representative number of projects to evaluate our framework. We analyzed only files with extension .java. Therefore, we end up analyzing 9 libraries in this study. Table 5.3 shows the number of remained projects after each step in our filtering process.

From the dataset projects, we computed all commits with the libraries evaluated in this study and identified 1.6 million different developers who made commits. Figure 5.3 shows the number of developers per library. The library, with more developers that made commits, was Selenium. This library has 811,844 developers. In contrast, Apache Wicket was the library with fewer developers, 5,440. It is important to say that these developers made at least one commit for the respective library. However, we cannot consider them

¹<https://insights.stackoverflow.com/survey/2018#most-popular-technologies>

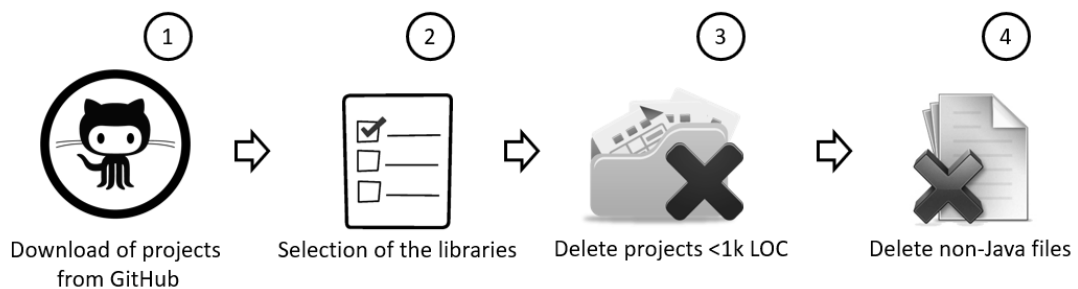
²<https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018>

Table 5.2: Library Descriptions

Library	Description
Hibernate	Hibernate is a library of object-relational mapping to object-oriented.
Selenium	A test suite specifically for automating Web.
Hadoop	A library that facilitates the use of the network from many computers to solve problems involving massive amounts of data [97, 105].
Spark	A general-purpose distributed computing engine used for processing and analyzing a large amount of data
Struts	It helps in developing Web-based applications.
GWT	It allows Web developers to develop and maintain complex JavaScript front-end applications in Java.
Vaadin	It includes a set of Web components, a Java Web library, and a set of tools and application starters. It also allows the implementation of HTML5 web user interfaces using the Java.
PrimeFaces	A library for JavaServer Faces featuring over 100 components.
Apache Wicket	A library for creating reusable components and offers an object-oriented methodology to Web development while requiring only Java and HTML.
JavaServer Faces	A Java view library running on the server machine which allows you to write template text in client-side languages (like HTML, CSS, JavaScript, etc.).

Source: Prepared by the author, 2023

Figure 5.2: Steps for Collecting Software Projects from GitHub



Source: Prepared by the author, 2023

all experts, since a single library use may not indicate high expertise.

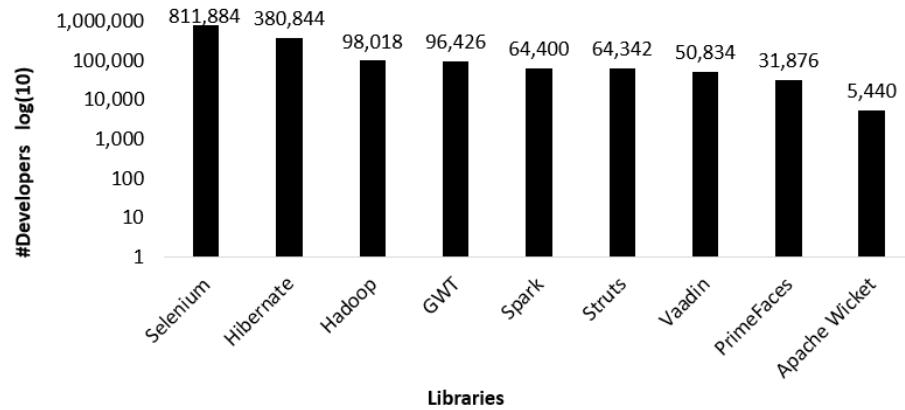
Figures 5.4, 5.5, and 5.6 show an overview of the metrics computed to our dataset. Figure 5.4 presents the results to Number of Commits per library. Figure 5.5 presents an overview of metric Number of Imports per library. Finally, Figure 5.6 shows the results of the metric Lines of Code per library. In general, Lines of Code (Figure 5.6) was the metric that presented more variation in our dataset. For instance, GWT has developers that wrote more than 130 KLOC. Similarly, for Hibernate, it is possible to see an outlier

Table 5.3: Projects Selected for Analysis

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

Source: Prepared by the author, 2023

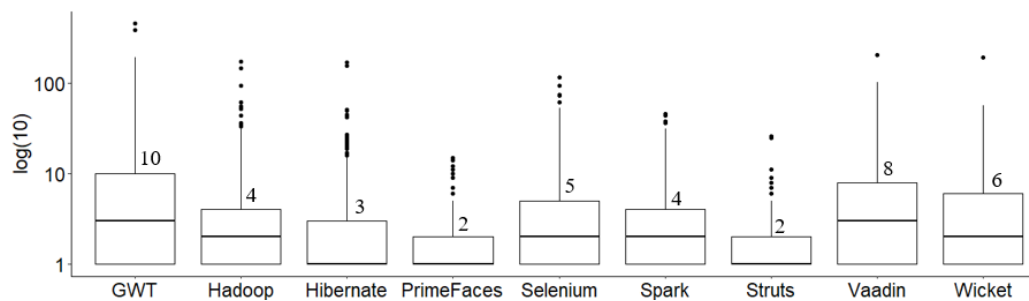
Figure 5.3: Number of Developers by Library



Source: Prepared by the author, 2023

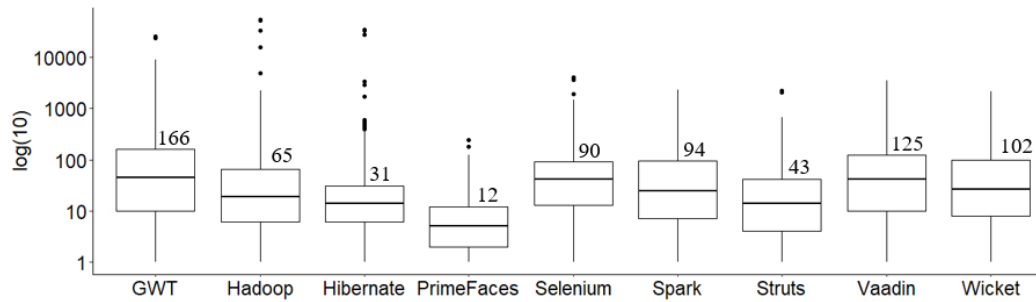
developer who wrote more than 500 KLOC. In contrast, some developers wrote less than 10 lines of code, for example, to the library PrimeFaces.

Figure 5.4: Number of Commits per Library



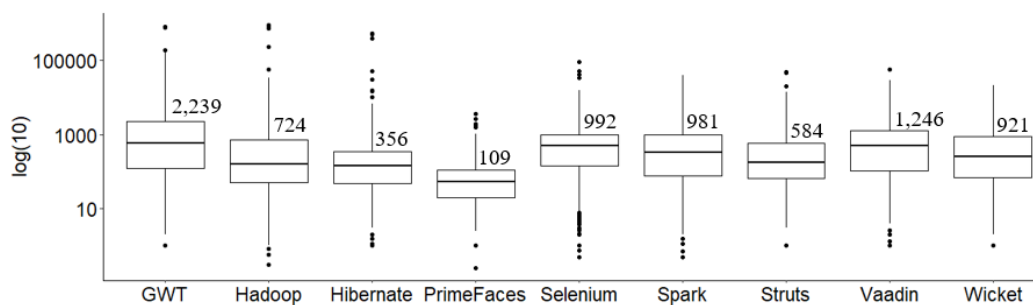
Source: Prepared by the author, 2023

Figure 5.5: Number of Imports per Library



Source: Prepared by the author, 2023

Figure 5.6: Number of LOC per Library



Source: Prepared by the author, 2023

5.5 Survey Design

Survey studies are used to identify characteristics of a population and are usually associated with the application of questionnaires [31]. Besides, surveys are meant to collect data to describe and compare or explain knowledge [80]. We selected the library experts with the best values in the evaluated metrics to validate them through a survey. We designed and applied a survey with the top developers identified by our framework. We selected developers with the top-20% highest values in at least two (out of three) metrics. We created a questionnaire on Google Forms³ with two parts: the first one was composed by 5 questions about the background of the expert candidates; the second part also had 5 questions about the knowledge of the expert candidates regarding the evaluated libraries. Table 5.4 contains the tag `<libray name>` meaning a specific library, for instance, Hadoop. In addition, this table shows the possible answers to the survey.

To obtain the email used by the developer to perform the commits in the source code, we used the Git-Blame⁴ tool. The emails were collected in order to send the survey. We sent an email to developers asking them to assess their knowledge on each library. For instance, the developers were invited to rank their knowledge (Table 5.4, SQ1) using

³<https://www.google.com/forms/>

⁴<https://git-scm.com/docs/git-blame>

a scale from 1 (one) to 5 (five), where (1) means no knowledge about the library; and (5) means extensive knowledge about the library. Questions are not mandatory because they may require knowledge on exceptional features of the library. Therefore, participants are not forced to provide an answer when they did not remember a specific element of the library, such as, time of development using the library and the approximate frequency of commits that contains the library. The survey remained open for 15 days in January 2019. In this section, we present the results of the accuracy evaluation based on a survey with expert candidates in each library. The goal of this evaluation is to verify the precision of the library expert identification. We empirically selected 1,045 developers among the top-20% values in at least 2 metrics. The questionnaire was sent January 2019. After a period of 15 days, we obtained 137 responses resulting in a response rate of about 15%. We asked the 137 developers about their software development experience in general (background), and the use of the specific libraries investigated in this thesis.

Table 5.4: Survey Questions on the Use of the Libraries

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

Source: Prepared by the author, 2023

5.6 Overview

In this section, we present an overview of some relevant findings. Table 5.5 presents an overview of the experts' candidates contacted to answer our survey. This table has the following structure. The Library column indicates the name of the analyzed library. The second Emails sent column shows the number of emails collected and sent to expert candidates. The Invalid email column presents the number of emails invalid which returned by the server. The Remaining emails column indicates the number of valid emails. The #Answers column shows the number of answers we obtained for each library. Finally, in the last (%) column, we show the response rate of each library.

Table 5.5: Top 20% from Library Experts Selected to Answer the Survey

Library	Emails sent	Invalid email	Remaining email	# Answers	%
GWT	160	18	142	31	22%
Hadoop	181	33	148	11	7%
Hibernate	155	10	145	16	11%
Spark	138	19	119	11	9%
Struts	42	2	40	9	23%
Vaadin	107	18	89	15	17%
PrimeFaces	30	1	29	9	31%
Wicket	23	2	21	8	38%
Selenium	209	31	178	27	15%
TOTAL	1,045	134	911	137	15%

Source: Prepared by the author, 2023

It is worth highlighting that half respondents are graduated in Computer Science and 7% Ph.D. degree. Concerning time dedicated to software development, 47% have more than 10 years of experience, and only 2% have less than 1 year of experience. Therefore, we can conclude that, in general, the participants are not novices. Our study also shows that a significant amount of expert candidates make commits when writing code related to a specific library, perform many imports of particular libraries and write lines of code in relation to the library. We support this affirmation through metrics, mainly a heuristic that evaluates the amount of LOC written by a developer when it is performed a commit.

Table 5.6 shows the results about knowledge that surveyed developers claim to have in each library. The developers were invited to rank their knowledge using a scale from 1 to 5, where (1) means no knowledge about the library; and (5) means extensive knowledge about the library. If we analyze the data about the precision of the framework from the sum of levels 3, 4 and 5 of Likert-type scale, we obtain on average 88.49% of accuracy in relation the knowledge of the developers, i.e., identification is correct in more than 88% of the cases. On the other hand, although a score three may represent an acceptable knowledge, if we followed a more conservative criteria, only classifying as library experts the developers that informed a higher (≥ 4) knowledge on the libraries, we obtain, on average, 63.31% of precision. This way, we conclude that most of the identified expert candidates identified by the framework contain high knowledge about the evaluated libraries. In contrast, to a level of knowledge < 3 , we achieved only 11.51% of the developers, i.e., possibly the framework fails by selecting these developers.

Summary – About 88% of the library experts who answered the survey have a high knowledge about the evaluated libraries.

Table 5.6: Level of Knowledge in Each Library

Library	Likert scale					Total	3-4-5	4-5
	1	2	3	4	5			
GWT	1	1	4	9	16	31	94%	81%
Hadoop	0	1	3	4	3	11	91%	64%
Hibernate	1	3	6	3	3	16	75%	38%
Spark	0	1	4	2	4	11	91%	55%
Struts	2	2	1	4	0	9	56%	44%
Vaadin	0	2	5	3	5	15	87%	53%
PrimeFaces	0	0	4	4	1	9	100%	56%
Wicket	1	0	2	4	1	8	88%	63%
Selenium	0	1	4	13	9	27	96%	81%

Source: Prepared by the author, 2023

5.7 Level of Activity

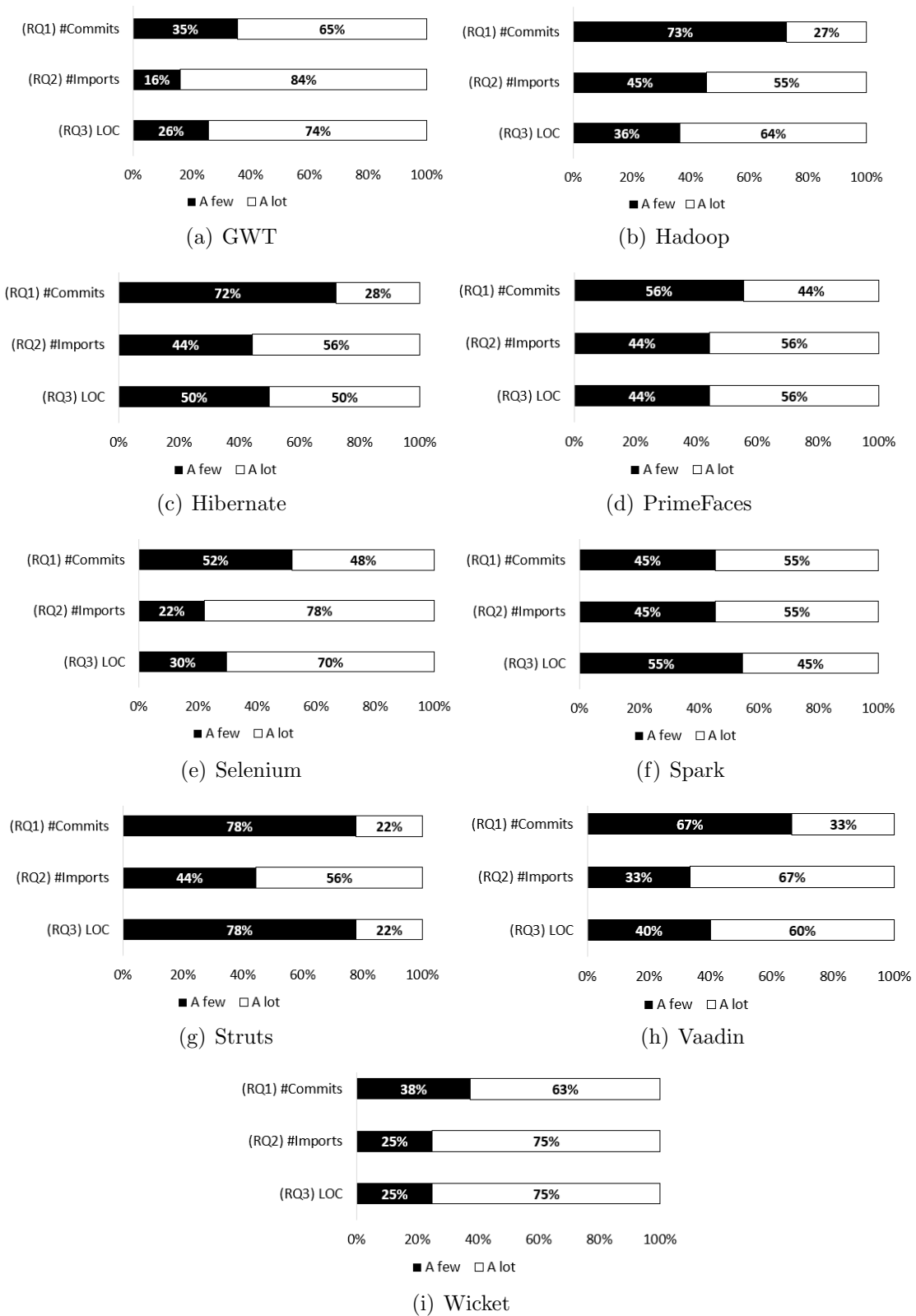
This section analyses the results of our empirical evaluation to research question one (RQ1).

RQ1– How to evaluate the level of activity of a developer in a library?

To answer this research question, we asked the library experts the following question. “How often are your commits related to the *<library name>* library”? Figure 5.7 shows the results to this question in the first line in each chart to each library. For most libraries, the majority of the participants answered they made “few” commits using the evaluated libraries. This way, if we evaluated the results obtained for this label, it is possible to see that from 137 experts, 54% made “few” commits. For instance, in the library Hibernate, 87% of developers said they made few commits related to this library. Another library that deserves special attention is Struts. In this library, 88% of the developers responded that they made few commits. Regarding the label “a lot”, only 39% of experts polled said they performed many commits. GWT was the library with a higher rate of answers to this label (62%). Therefore, the numbers indicate that the metric *Number of Commits* needs to be combined with other metrics to achieved conclusive results about the skill from developers and even develop other metrics to identify the level of activity ability.

RQ1 summary – A large proportion of library experts make “few” commits using the library. Therefore, we concluded that the solo use of number of commits cannot identify library experts.

Figure 5.7: Results of the survey questions for each library



Source: Prepared by the author, 2023

5.8 Knowledge Intensity

This section analyses the results of our empirical evaluation to research question two (RQ2).

RQ2– *How to evaluate the knowledge intensity of a developer in a library?*

Regarding the number of imports to indicate a library expert, we ask the developers the following question: “How often do you include an import of *<library name>* library in your commits?”. Figure 5.7 shows the results to this question from second line in each chart to each library. We analyze the number of imports performed by developers. The main reason for this analysis is to evaluate the feasibility of inferring the skills of the developers from the types of imports performed. In general, the label “few” and “a lot” are tied or with little difference between them. For example, Hibernate, Spark, and PrimeFaces have practically tied. These libraries did not show significant differences; in some cases, the difference was only of 1 absolute point. In only three cases, the label “a lot” remained significantly higher: GWT (83%), Vaadin (67%) and Selenium (78%). From 137 experts, 68% said that they made “a lot of imports”. However, the number informed by the experts indicates that this metric requires a combination with other metrics to achieve better results, because 32% of experts said they made few imports to libraries evaluated. Therefore, from the survey results, the metric *Number of Imports*, as well as the metric *Number of Commits*, are not able to identify library experts, when we apply one at a time.

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

5.9 Knowledge Extension

This section analyses the results of our empirical evaluation to research question three (RQ3).

RQ3– *How to evaluate the knowledge extension of a developer in a library?*

In this research question, we analyze the developers skill from the number of LOC related to library. We evaluate the number LOC implemented by a developer to specific library. For this purpose, we asked the library experts from the survey the following question. “How much of your code is related to the *<library name>* library when you

perform a commit?”. Figure 5.7 shows the results to this question in the third line in each chart to each library. The libraries GWT, Wicket, Selenium, and Hadoop, for instance, obtained 74%, 71%, 70%, and 64% respectively to label “a lot”. We noted, however, the label “a few” also remained at a high level in some cases, for instance, the libraries Struts (88%) and Spark (55%). In fact, the library Hibernate remained tied to labels “a few” and “a lot”. In general, from 137 experts, 39% said they write “a few” LOC and 61% write “a lot” LOC with respect to libraries. Therefore, it is possible to infer that the metric *Lines of Code* alone also does not provide indications about developer skills, although this metric achieved better precision than the metric *Number of Commits*.

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

5.10 Threats to Validity

We based our study on related work to support the evaluation of a strategy to identify library experts. Regarding the assessment, we conducted a careful empirical study to assess the efficiency of the strategy from software systems hosted from GitHub. The strategy evaluated is able to analyze source code from platforms that follow the Git architecture. However, some threats to validity may affect our research findings. The main threats and respective treatments are discussed below [104].

Construct Validity is related to whether measurements in the study reflect real-world situations [104]. Before running the strategy, we conducted a careful filtering of software systems from GitHub repositories. However, some threats may affect the correct filtering of systems, such as human factors that wrongly lead to the discard of a valid system to be evaluated. Considering that the exclusion criteria to system selection were applied in a manual process, we may have discarded interesting systems that we identified as non-Java, for instance.

Internal Validity is related to uncontrolled aspects that may affect the strategy results [104]. The strategy may be affected by some threats. To treat this possible problem, we selected a sample of 5 software systems that contain the library Hadoop from our dataset, with a diversified number of LOC. Then, we manually identified the number of commits from the GitHub repository, the number of imports, and the number of LOC codified to the specific library. We compared our manual results with the results provided by the tool and observed a loss of 5% in metrics terms computed through the automated

process. We believe that this error rate does not invalidate our main conclusions.

External Validity this validity is related to the possibility of generalizing our results [104]. We evaluated the strategy with a set of 16,703 software projects from GitHub. Considering that these systems may not include all existing libraries, our findings may not be generalized. Furthermore, we evaluated the strategy with an online survey with only 137 developers that implemented projects with the investigated libraries. We analyzed the data with only 9 Java libraries. However, we chose the top libraries from the survey reported by StackOverflow in 2018, with over 100,000 responses developers around the world. This way, we believe these libraries can represent a reasonable option to evaluate the framework.

Conclusion Validity this validity is a factor that can lead research to reach an incorrect conclusion about a relationship between observations [104]. Having a fair comparison between programming skills from source code and real hard skills is challenging. We show that source code presents data about the programming skills of developers. However, the data can suffer interference. For example, other people made a commit by the developer. Our study reveals a high dependency on hash commits. Although it is possible to find an optimal threshold configuration to detect commits atypical.

5.11 Concluding Remarks

In this chapter, we presented an evaluation of our framework based on the Library Instance. The framework developed aims to identify library experts from source code in technologies that use the import keyword. At this stage, we evaluate the framework composed of three metrics: i) Number of Commits, ii) Number of Imports and, iii) Lines of Code. Besides, we evaluated our framework with nine Java Libraries: Hibernate, Selenium, Hadoop, Spark, Struts, GWT, Vaadin, PrimeFaces, and Apache Wicket. We conducted the empirical evaluation with real developers from GitHub. We concluded that the metrics can be used to identify library experts but they need to be combine with another for better results. In the next chapter, we present a study involving software developer recruiters from three different countries: Brazil, Canada, and the United States.

Chapter 6

A Study with Software Recruiters

The selection of competent software developers is crucial for ensuring the quality of software products [4]. In the rapidly evolving field of software development, where technology advancements and customer expectations continuously shape the landscape, identifying skilled developers becomes imperative. The capability of developers directly impacts the software development process, including requirements analysis, design, implementation, and testing [67]. This chapter describes the study conducted to evaluate the Tech Skills Instance by means of interviews with actual recruiters. The study utilizes the data generated by the Tech Skills Instance of our framework, which takes as input the developers' repositories and computes their hard skills based on programming languages and libraries. The Tech Skills Instance provides insights into the developers' profiles using data mined from GitHub. These data are processed to showcase the developers' curricula.

This chapter is distributed the following way. Section 6.1 outlines the aims of our study and the research questions we address. Section 6.2 presents the evaluation steps. Section 6.3 presents the metrics used in this instance, which can be adapted according to the goal of computing hard skills. Section 6.4 explains the instrumentation applied during the interviews, including the adoption of concepts such as Focus Interview [16] and Think Aloud [11]. Section 6.5 details the steps taken to select the participants, and Section 6.6 provides a summary of the interviews. Section 6.7 presents the details about data analysis. Section 6.8 shows the demographic information of recruiters. Sections 6.9 to 6.12 present the results. Section 6.13 discusses some limitations and threats to the study. Finally, Section 6.14 concludes with our final remarks.

6.1 Goal and Research Questions

The main goal of this study is to understand the problems in hiring a new software developer to fill a specific position from the point of view of recruiters. More specifically, we would like to know if the selection of software developers is appropriate and how profile

mining can help in this step. Since we are interested in comprehending the recruiters' opinions about this process, we conducted exploratory interviews with them. This means prospective recruiters can see a developer's profile of projects listed on GitHub and a history of their code-related actions over time on both these and other people's projects. We show below the four Research Questions (RQ) of this study.

RQ1- How do companies recruit software developers?

RQ2- What are the possible applications of the framework to mine a developer profile?

RQ3- How do the framework results complement GitHub information?

RQ4- What are the opportunities for improving the hiring process?

From the RQ1, our goal is to understand the strategies and processes used by companies to recruit software developers. The answer to this question provides insights into the frameworks used by companies to evaluate the hard skills of software developers, such as technical interviews, behavioral interviews, and reviewing GitHub profiles. The goal of RQ2 is to understand the feasibility of applying a new framework to complement the process of selecting software developers. In other words, whether, for example, it is possible to use other strategies to support the step of selecting new software developers. RQ3 investigates ways to complement GitHub information, for instance, with summarized profiles. For this, we investigate extras information from GitHub repositories. After, we check whether these data can help recruiters to select a developer. The goal of RQ4 is to identify areas for improvement in the framework used in this study and the hiring process. The answer to this question provides insights into the limitations of the framework, areas for improvement, and how the framework can be adapted to changing requirements and technologies. The information obtained from this question can be used to make the recruitment process more efficient and effective and to ensure that companies are hiring the best software developers.

6.2 Evaluation Steps

This section describes the steps to answer the research questions presented in Section 6.1. We designed a study composed of five steps: 1) *Metrics Selection*, 2) *Instrumentation*, 3) *Selection Subjects*, 4) *Interview with Recruiters*, and 5) *Data Analysis*. Figure 6.1 shows the steps. In Step 1) Metrics Selection (Section 6.3), we adopt the metrics to identify hard skills from our framework. Step 2) Instrumentation (Section 6.4). describes the

instrumentation used to conduct the study. In Step 3) Selection Subjects (Section 6.5), we investigate software developer selection with 2 randomly selected developers and recruiters with at least 2 years of experience invited from LinkedIn and networking contacts, speaking either Portuguese or English. Step 4) Interview with Recruiters (Section 6.6) is responsible to describe the online semi-structured interviews with recruiters, focusing on their use of GitHub during the hiring process. Step 5) Data Analysis (Section 6.7) is responsible to compute the data about the interviews.

Figure 6.1: Study steps with Software Recruiters



Source: Prepared by the author, 2023

6.3 Metrics used to Tech Skills Instance

In this section, we provide an explanation of the metrics used to compute hard skills for the Tech Skills Instance. While these metrics share similarities with those adopted in previous studies (Chapters 4 and 5), there are some differences that we will discuss here. The primary goal of this instance is to gather more detailed information about a developer’s profile from their GitHub repositories. To achieve this, we rely on two key metrics: Lines of Code and Number of Commits. As mentioned in Section 5.3, we have previously used the metric Lines of Code, but for this instance, we have made certain modifications. In this study, we employ a different approach to identify the programming language used by a developer, based on the file extension and confirmation from Linguist¹. Subsequently, we use Git Blame and Git Log to determine the number of Lines of Code changed (added, removed, or updated) by a specific developer for a particular programming language. This metric provides insights into the developer’s level of engagement and activity in coding with a specific language. Additionally, we leverage the metric Number of Commits to quantify the quantity of commits made by a developer in relation to specific programming languages and the most commonly used libraries. To further identify the main libraries used by a developer, we compute the frequency of “imports” or “require” statements found in their code files. These keywords are typically used to

¹<https://github.com/github-linguist/linguist>

call third-party libraries, and by analyzing their presence, we obtain valuable insights into the developer's familiarity with different libraries and their level of expertise in utilizing external resources.

6.4 Instrumentation

In this section, we describe the instrumentation used to conduct the study. We create one fictitious job opportunities from open jobs on LinkedIn. Next, we randomly select two developers from GitHub. However, each developer must have similar profile concerning the programming languages (in this case, JavaScript). It is not mandatory that the developers have the same programming skill level. We then compute the programming skills of these two developers from the framework selected in this study [74] and generate each developer's CV. Before conducting the interview with recruiters, we select them. We select recruiters with at least 2 years of experience invited from LinkedIn and our networking contacts. Recruiters with more experience are often better equipped to identify and evaluate the skills, qualifications, and cultural fit of potential candidates, resulting in more successful hires. After that, we conducted a semi-structured interview with recruiters. In this type of interview, open questions can be applied off-order. This way, the participants can express themselves more freely and elaborate on their answers. To obtain more details about the opinion of the recruiters, we adopt the think-aloud protocol [11, 34, 77].

Participants in think-aloud protocols speak their thoughts out loud as they carry out a series of predetermined tasks [11, 34, 77]. In our case, we observe the use of GitHub from the recruiters actions. Participants are invited to say whatever comes to mind as they finish the activity. This could involve what they are seeing, contemplating, acting upon, and feeling. For instance, when a recruiter looks at the developer's repositories, we need to understand the recruiter's viewpoint. Making thought processes as explicit as feasible throughout task execution allows observers to gain insight into participants' cognitive processes rather than just their end product. Every verbalization is recorded, transcribed, and examined. We also request that the interviewees share their screens with us. This step is essential to view points evaluated by recruiters. Before starting the recorder screen and voice, we solicit the consent of the interviewees to record. Finally, we analyze the data generated by interviews using the open and axial coding inspired to Grounded the Theory [37, 59, 94]. A replication package for this study is made available at Zenodo [75].

6.5 Selection Subjects

This study relies on two profiles of developers and selected 17 recruiters to investigate the software developers selection process. We selected two developers by which the GitHub profiles had information about the developer which matches the job description. We create one fictitious job opportunities from open jobs on LinkedIn. We also researched LinkedIn by software recruiters to invite them for our interview. As a result, we invite 20 recruiters from our personal network and mined from LinkedIn of known.

To invite a recruiter, they need to have experience in at least two years as a recruiter. This period is an important factor in being interviewed to be able to contribute to the study. In addition, our participants to interview should speak one of the two idioms, Portuguese or English. This is important for us to avoid language barriers and to understand each other. To conduct an interview, we select two curricula generated from GitHub data [74]. These curricula have the following data of the developer: 1) a short bio, 2) the number of repositories, 3) a bar chart with lines of code change by programming language, 4) a bar chart with commits made by the developer to a specific programming language, and 5) the number of import to most used APIs by programming language.

6.6 Interview with Recruiters

We used interviews for data collection conducted online in November and December 2022. We adopt the concept of Focus Interview [16]. This type of interview emphasizes the interviewees subjective and personal responses, where the interviewer engages in eliciting more information [9]. According to Merriam [61], interviews effectively elicit information about things that cannot be observed. We used semi-structured interviews with open-ended questions because this approach gathers richer responses compared to structured interviews [84]. Interviewees: we contacted each participant by e-mail. Each interview occurred in a private Google Meeting room. We asked the recruiters to describe a recent past hire, focusing on how they used GitHub during the hiring process. We request what information on the platform was relevant and what that information is saying about the candidate.

Interviewees had to agree with the Informed Consent Form, which guarantees the confidentiality of the data provided, the anonymity of the participants, and the right to withdraw from the research at any moment. We collected data and enhanced the

interview in two rounds. During the first round, we piloted the interview design with two recruiters. We then discarded the pilots' data. Through the pilot tests, we understood that it was necessary to change some interview questions, enhance CVs of the developers, and emphasize the focus on hard skills to conduct the interview. Therefore, our final pool of interviews was composed of 15 recruiters because two were used in the study pilot. Table 6.1 presents the Interview Questions (IQ) in detail. The first column indicates the research questions used to create the interview question. The second column indicates the number of IQs, and the last column presents the interview questions. Note that from IQ1 to IQ4 are questions about data exclusive of GitHub. On the other hand, from IQ5 to IQ8 are questions about the framework data.

Table 6.1: Interview Questions

RQ	IQ	Interview Questions
RQ1	IQ1	How does the company evaluate a profile of a new software developer to compose the team? What are the steps?
RQ1, RQ2	IQ2	Based on GitHub, how does the company evaluate the candidate's skills? Can you compare Candidate 1 with Candidate 2 using GitHub data?
RQ1	IQ3	What the characteristics do you observe on these profiles of the developers ?
RQ3,RQ4	IQ4	Which developers would the company choose between these two developers? Why?
RQ3,RQ4	IQ5	From the PDF file of the mined profiles, what data do you consider as important?
RQ3,RQ4	IQ6	From PDF file of the mined profiles, how do you compare the two developers ?
RQ3,RQ4	IQ7	From PDF file of the mined profiles, what developer do you choose for this job position? Why?
RQ3,RQ4	IQ8	From the PDF file of the mined profiles, what are the recruiter's observations? How can the PDF file help the recruiter to select a software developer from GitHub?

Source: Prepared by the author, 2023

6.7 Data Analysis

We conducted a series of semi-structured interviews with 15 recruiters to identify how activity traces are used and assessed to infer a developer's abilities and personal qualities by recruiters. When we finished all interviews, we compile the results. To compile the results, we create a transcript of the interview and made highlights in crucial

points about opinion and actions of the recruiters. In our analysis, we coded the interview transcripts to identify the various ways in which profiles were employed during the hiring process, as well as the diverse types of inferences drawn about individuals under evaluation based on the GitHub environment.

Using HyperResearch [54], a qualitative analysis software tool, we identified relevant sentences or broader segments in interview transcripts related to each research question. We used open coding to analyze the interview transcripts. We start by reading the transcripts, identifying key points, and assigned them a code (i.e., a 2–3 words statement that summarizes the key point). In the context of this work, the 15 attributes identified in the previous step were used as seeds for this analysis. By constantly comparing the codes [94], we grouped them into four categories that gave a high-level representation of the codes. This coding process was conducted by one researcher and constantly discussed with other two researchers.

6.8 Results for Participants' Demographic Information

This section shows the demographic information of the interview participants. Table 6.2 shows the 17 participants of this study, including two in the pilot. In the first column, we present the participant ID. We remove the real name of participants to preserve their identity. The second column shows the gender of the participants. The third column shows the company size. We separate the data by company size into four groups: a) Micro Enterprises: fewer than 10 people, b) Small Enterprises: between 11 and 50 people, c) Medium-Sized Enterprises: between 51 and 250, and d) Large Enterprises: from 251 people or more. In the fourth column, we show the country of the recruiter company. Finally, in the last column, we present the approximate number of employees in each company. The last two rows of Table 6.2 present the demographic data about the pilot study. In general, the companies of participants in the study are Large enterprises. For instance, participant P1 from Brazil works in a company with $\sim 16,500$ employees. In contrast, we also interviewed recruiters from micro-enterprises, for instance, participant PS2 (pilot study). Out of 15 participants, four are female.

Table 6.2: Participant demographics

Participant	Gender	Company Size	Country	Employees
P1	M	Large Enterprises	Brazil	16,500
P2	M	Large Enterprises	Canada	9,700
P3	M	Large Enterprises	Canada	9,700
P4	M	Large Enterprises	Brazil	8,452
P5	M	Large Enterprises	Brazil	5,571
P6	M	Large Enterprises	Brazil	4,000
P7	F	Large Enterprises	Brazil	3,600
P8	F	Large Enterprises	Brazil	2,990
P9	M	Large Enterprises	Brazil	2,990
P10	F	Large Enterprises	Brazil	2,990
P11	M	Large Enterprises	Brazil	2,788
P12	M	Large Enterprises	Brazil	1,873
P13	M	Large Enterprises	United States	500
P14	F	Large Enterprises	Brazil	174
P15	M	Medium-Sized Enterprises	United States	52
Pilot Study				
PS1	M	Large Enterprises	Brazil	4,419
PS2	M	Micro Enterprises	Brazil	3

Source: Prepared by the author, 2023

6.9 Results for Recruitment Channels

In this section, we answer the RQ1: How do companies recruit software developers?

Our first research question focused on how recruiters use recruitment channels to evaluate new candidates. We investigate the channels and strategies used to select software developers from the interviews. In general, all interviewees believed that a GitHub profile provided insight into an individual's technical abilities and personal qualities more reliably than resumes or code samples taken out of context. The GitHub profiles provide recruiters with a history of individual contributions over time. In some situations, searching for complementary information about the candidate from other channels, for instance, LinkedIn, is necessary; for instance, when the candidate is junior and the GitHub profile is clean or very simple. On the other hand, we observe that some companies like to receive curricula in PDF of a candidate and conduct a face-to-face interview. In the same case, they provide a candidate with a fictitious programming problem. In this way, the candidate needs to implement a source-code as a solution from a time and programming language stipulated by the recruiters. In this case, the recruiter evaluates the source-code quality. In general, according to interviewees, companies look at soft skills in the first moment and then select a developer based on hard skills. Therefore, the soft and hard skills are complementary.

Nevertheless, this situation depends on also the requirements of the open job. For example, if the company searches for a junior developer, it is optional that this developer knows many technologies, such as Java, Python, JavaScript, and frameworks. On the other hand, if the company is searching for a senior developer, this candidate needs excellent soft and hard skills because, generally, this developer is more expensive and is expected to have more experience. Besides, they will guide the team to solve a problem. As an example to support this claim, recruiter P2 said:

[...] The best software developer is not someone mastering JavaScript or React, for instance. The best software developer is one who can contribute to the team. The best software developer needs a balance between soft and hard skills. The best software developer needs to listen to the team for giving and receiving help. To develop a software is hard then, we require people with the same propose. As said the popular saying “one bad apple can spoil the whole barrel”.

Finding a new software developer can be difficult, especially if the recruiter only looks at the curricula and promote test with toy problems. During the recruitment process, whether it’s done using paper or online platforms like LinkedIn, important factors like cultural fit, communication skills, and cooperation skills can often be overlooked. However, these factors are crucial for success in a collaborative work environment. Furthermore, although source-code exams can give a candidate a look of their technical proficiency, they might not fully capture their problem-solving abilities or their capacity to design clear, maintainable code. To get a more thorough assessment of a candidate’s prospective fit the business, it is crucial to combine these methodologies with other evaluations, like behavioral interviews and real-world projects.

RQ1 summary. *We have observed that software developers need to balance both soft and hard skills. Moreover, we have found that recruiters consider GitHub to be an important platform for obtaining reliable information about job candidates.*

6.10 Results for Applicability

In this section, we answer RQ2: What are the possible applications of the framework to mine a developer profile?

Table 6.3 shows the computed coding to answer this question. It includes four

columns: Category, Codes, Participants, and Frequency. Category indicates a group created to aggregate the similar codes. Codes are abstract models that emerge during grounded theory analysis’s sorting and demonstration stages. They conceptualize the integration of substantive codes as hypotheses of a theory. This study uses coding inspired by grounded theory but does not necessarily apply all ground theory steps. Participants indicates the number of participants that cited something related to the code. Finally, Frequency shows the number of times a code was cited.

Table 6.3 shows four codes: 1) “Effort Reduction”, 2) “First Step to Filter”, 3) “Rework” and 4) “Profile Type”. Concerning “Effort Reduction”, we identify from the interviews with recruiters that it is necessary to optimize the process of recruiting a developer. Sometimes, there are many candidates for a job position. We identify this code from the view point of the 13 (87%) participants with frequency of 17 occurrences. The number of candidates can harder the selection process because it is expensive in time and resources to interview them all. That is, the team needs to stop some tasks to help the recruiter to select a new developer. Therefore, the hiring process may require less effort to find a new candidate by using the framework to generate the CV of the candidates from hard skills. The recruiter interviewed P1 said:

[...] GitHub is a rich source of data about candidates for job vacancies. However, we need to spend time identifying the candidate’s hard skills, for example, the primary programming language and whether the candidates know a specific API / framework. From the generated CV, we reduced the work to identify the developer’s abilities. In addition, we are able know more details about the candidate.

Table 6.3: Applicability of the framework Evaluated

Category	Codes	Participants (%)	Frequency
Application	Effort Reduction	13 (87%)	17
	First Step to Filter	12 (80%)	14
	Rework	8 (53%)	8
	Profile Type	7 (47%)	7
	Total		46

Source: Prepared by the author, 2023

The code “First Step in Filtering” is about recruiters’ viewpoint of using the CV as an instrument to pre-select the candidates. We observed that the recruiters usually have a considerable effort to conduct the hiring process because, generally, there are many candidates. Sometimes, to find a suitable candidate in a universe of twenty candidates, for instance, is exhausting. This way, from the viewpoint of recruiters, it is possible to reduce the effort by using the CV generated by the framework that computes hard skills. The idea is to use the CV as a pre-selection of the candidates. This code was manifested

by 12 (80%) participants in 14 times. In that case, it is possible to adopt the framework, but if the job opening is for a junior position, the recruiter needs to evaluate in more detail because the GitHub profile of the candidate may be less important. In the following, we quote recruiter P4.

[...] This CV can help us to pre-selecting the candidates. It is possible to evaluate the developer from the specific hard skill as a target of our interest, for example, knowing React. Therefore, the sector of the humans resource can be streamlined in search specific abilities and conduct the filter to the next steps. As such, the recruiter needs less work to conduct the process of selecting a new candidate.

The other code identified was “Rework”. This code summarizes the situation required to restart the recruitment process. We note that 8 (53%) of the 15 recruiters point out the problems and costs associated with the flawed selection process. Therefore, it is necessary to be the most assertive as possible in selecting a software developer because conducting the recruitment process again is expensive and exhausting. In addition, the team generally needs to support the newly hired developer or offer training. This way, the costs to the company can be high, because it may be necessary to fire the developer and hire another. Through the framework, it is possible to help in the selection process with more details about the candidate available from source-code analysis. In this way, the recruiter has more data to decide about the software developer, for instance, the candidate’s main programming language or knowing APIs. In the following, we show the quote of recruiter P7.

[...] The selection process is challenging for us recruiters because we need to select a developer to help our team. If we select wrong, we will have problems in the future because it will be necessary to hire other developers and restart the process. Therefore, this problem can delay the project. The framework cannot substitute the face-to-face meeting or the recruiter, but the generated CV can help to understand the developer’s profile in more details.

The source-code available in GitHub can provide idea of the profile of the developer. The code “Profile Type” indicates the suitability to back or front-end, for example. This code was manifested by 7 (47%) participants in 7 times. We observe that recruiters analyzing the CV generated by the framework had more details of the candidate. For example, if a candidate like more Python or made more commits and wrote many lines of code to SQL, then this developer has characteristics of the back-end and database manager. These data could be obtained from GitHub without the mining framework.

However, to investigate all repositories of a specific developer manually is very exhausting to recruiter. Recruiter P6 said:

[...] Sometimes we need to look at many CVs by day. This way is inevitable to lose some information. In addition, if we need deep information into each developer's repository manually, selecting a developer will be more complicated. Therefore, sometimes we look at GitHub profiles in general, only the tip of the iceberg after selecting a candidate from the best. We apply, for example, source-code tests to obtain more information about the hard skill of the developer. This summarized PDF file can help us in this step to avoid looking only at the tip of the iceberg. From the CV, it is possible to understand the developer's profile and know if they can work with us. In addition, if the job requires a junior, we immediately understand the deficiency of some technologies, and we can provide training, for example. Therefore, the presented framework can help decision to select a candidate.

RQ2 summary. *Usually, recruiters need to evaluate many profiles of the candidates for a job. This task is complex and extremely relevant to the selection of an excellent developer for the team. Therefore, through the framework, it is possible to obtain more details about each profile from GitHub. This way, the recruiter decision for a candidate is better supported. In addition, the framework can be used as a first step to select a developer and provide more details about them.*

6.11 Alternative Visualization

In this section, we answer the RQ3: How do the framework results complement GitHub information?

To respond to this research question, we asked the recruiters about the data in the PDF file and GitHub. Table 6.4 shows the category Alternative Visualization. This table presents the exact configuration of Table 6.3. The code Programming Language represents the recruiters' viewpoint about the details they did not observe on GitHub but noted from the PDF file of mined profile. During the interview, recruiters analyzed the developer's programming language from repositories in GitHub. GitHub presents the primary programming language of a repository. However, it does not present the programming language that the developer used in their repositories. For instance, if a developer creates a script only in Python, but the primary programming language of the

repository is JavaScript, only observing the primary programming language informed by GitHub is insufficient. That way, the recruiters praised the alternative visualization of the programming language provided in the framework presented in the PDF file. This code was manifested by 13 (87%) participants, 15 times. As an example, recruiter P15 said.

[...] In fact, it is more convenient to observe the programming language that a specific developer uses with the PDF file. Because from the GitHub repository, we observe other programming languages that are not the developer's skills.

Another interesting code found was “Frequency of Commits”. This code was manifested by 12 (80%) participants, 15 times. “Frequency of Commits” is presented in a PDF file generated by the framework about the commits made by a specific developer concerning a programming language. The “Frequency of Commits” made by a developer can indicate the amount of work they are doing on a project for a specific a programming language. However, it is essential to remember that quantity is not necessarily a direct measure of the quality of work. Recruiter P11 said.

[..] The number of commits made by a developer can be visualized from GitHub too. However, the PDF file complements this view because it puts a bar chart with commits by programming language side by side. This way is more convenient to analyze.

“Time Experience” is a code concerning a developer’s period using GitHub. This code was manifested by 11 (73%) participants, 13 times. “Time Experience” is data essential to understand the knowledge of developers about the version control, for example, Git. Note that these data are available from GitHub, but sometimes needs to be highlighted to recruiters. From the PDF file, the recruiter notes these data and can be associated with other data, for example, programming language, API, and frequency of commits. The recruiter P9 said.

[...] These data is available on GitHub. However, from the PDF file, it is possible to complement the data presented by GitHub because it is possible to compare the period that the developer used GitHub and the technologies adopted by them.

“API/Framework” is a code about the technologies developers use together with a programming language, for instance, Hadoop. This code was manifested by 8 (80%) participants, 12 times. GitHub shows a pool of data that sometimes can confuse the recruiter and can be hard to be analyzed, for example, the API used by them. The PDF file shows the API/Framework most used by developers from a specific programming language. The

Table 6.4: Category Visualization

Category	Codes	Participants (%)	Frequency
Alternative visualization	Programming Language	13 (87%)	15
	Frequency of Commits	12 (80%)	15
	Time Experience	11 (73%)	13
	API / Framework	8 (53%)	12
	Repositories	7 (47%)	11
	Pollution Profile	3 (20%)	3
	Total		69

Source: Prepared by the author, 2023

use of API by a developer can indicate familiarity with the technologies. Developers often use APIs and frameworks that they are familiar with, as this can help them work more efficiently and produce higher-quality code. Familiarity with a particular API or framework makes it easier for developers to find solutions to problems they encounter during development. In addition, the recruiter can take these data to understand more about the developer because the API can be associated with the application’s performance, for example. The performance of an API or framework can significantly impact the overall performance of the software application. Therefore, it is possible to check if a developer knows API most adopted by the company. It is crucial to consider the speed and efficiency of the API or framework used and ensure it meets the project’s performance requirements. Recruiter P7 said.

[...] Recruiters face a non-trivial task when searching for specific APIs in source-code via import. As the framework shows the mostly used API, it is more practical for a recruiter to understand what is being used and, if necessary, ask questions to the developer. Since this import can be presented in many files, and they must be opened individually, the process can be time-consuming when made manually.

The code “Repositories” indicates the number of repositories of a developer. The number of repositories a developer has on GitHub can be one factor to consider when assessing their skills and experience. However, it should be evaluated alongside other factors, including code quality and collaborative skills. This code was manifested by 7 (80%) participants 12 times. Recruiter P6 said.

[...] It is easy to look at these data from GitHub. However, combining these data with the level of commits by programming language helps the recruiter analyze the developer's profile. Moreover, this mechanism in GitHub is not trivial. Therefore, the data from the PDF file complements the data presented in GitHub. These data is vital because the number of open-source projects a developer has contributed to can positively indicate their commitment to the community and willingness to share knowledge and expertise. The developer is more comfortable working in a distributed environment and is skilled at navigating open-source processes and tools.

“Pollution Profile” is a code about the presentation problems in GitHub. The pollution of GitHub profiles refers to developers cluttering their profiles with unnecessary pictures and emojis. While GitHub is primarily used for hosting and sharing code repositories, many developers use their profiles to showcase their personalities and interests. However, some developers take this to the extreme and fill their profiles with many images and emojis, making it challenging to navigate and find relevant information. Additionally, these unnecessary elements can slow down the page's loading time and make it less accessible. This practice harms the recruiter in finding the data about the developer. This code was manifested by 3 (20%) participants 3 times. Recruiter P4 said.

[...] I think that the pollution of GitHub profiles to be a significant problem in the candidate selection process. While I appreciate that developers want to showcase their personality and interests, excessive pictures and emojis on their GitHub profiles can be distracting and make it difficult for me to evaluate their technical skills and experience. It can also slow down the process of reviewing profiles, mainly if I am working with many candidates. Additionally, cluttered profiles can make it challenging to find essential information, such as links to code repositories, projects, and contributions to open-source software. Therefore, I urge developers to keep their profiles clean and straightforward, highlighting their technical accomplishments and experience clearly and concisely. This will help them stand out as a serious candidate and increase their chances of being selected for the next stage of the hiring process.

RQ3 summary. *In general, the framework to summarize profiles is able to show an alternative visualization of the data provided by GitHub. For example, pollution profile was observed by the recruiters. The pollution profile refers to cluttered profiles with unnecessary pictures and emojis, making it difficult to navigate and find relevant information. The framework is able to filter out this problem.*

6.12 Miss Information

In this section, we answer the RQ4: What are the opportunities for improving the hiring process?

To respond to this research question, we asked the recruiters about the need for more data from PDF generated by the framework. Table 6.5 shows the category “Lacking Information”. This table presents the exact configuration of Table 6.4. We create a code “Starts”. Stars on GitHub are a way for users to indicate that they liked a particular project and follow its development. For software developers, having many stars in their projects can indicate that their work is popular and well-received by the community and can be a positive factor when looking for work or receiving contributions from other developers. In addition, stars also help make a project stand out on GitHub, making it more visible to other developers looking for solutions to a particular problem or inspiration for their own projects. This code was manifested by 11 (73%) participants, 12 times. In general, the recruiters missed these data from the PDF file, as recruiter P6 said:

[...] I have the responsibility of looking for the best professionals for open positions. And, to evaluate a developer's experience, in addition to analyzing the profile of the developer, it is important to check other relevant data, such as the number of stars that their projects have on GitHub. This information can indicate the popularity and quality of the work done by the candidate.

The code “Dev History” means the historical from GitHub of a developer, for example, actual job, country of origin, and complete read-me. The PDF file limits the developer's text and does not present the data about the contact, for example, address or country, when informed by the developer. This limitation from the viewpoint of the recruiters could be better. This code was manifested by 5 (33%) participants 8 times. Recruiter P8 said:

[...] For me it is interesting to understand the developer's culture, and it is possible to see this from, for example, from the country of origin. Another fascinating data is about the actual job position of the candidate. From this, it is possible to know more details about the candidate. These data need to be presented.

The code “Seeking Information” represents the necessity of the recruiter to search for more details about the candidate from other social networking, by lack of data from GitHub and PDF presented, for instance, research from LinkedIn. This code was manifested by 5 (33%) participants in 6 times. The recruiter P6 said.

[...] Sometimes, GitHub does not have reliable data about the candidate, for example, when they are a junior. Supplementing a candidate's profile with additional research, such as reviewing their online presence, including their LinkedIn profile, is essential. The framework could support this research to help the recruiter.

The code “Followers” indicates a developer's popularity and interest in their projects. Generally, a higher number of followers means that a developer has a larger audience and their work is well-regarded within the GitHub community. This code was manifested by 4 (27%) participants 4 times. The PDF file generated by the framework does not show these data. From the viewpoint of the recruiters, these data are essential, mainly if the job opening is for senior. Recruiter P3 said.

[...] The number of followers is a data relative but very important to analyze, mainly if the job opening is for a senior. Overall, senior has the characteristics of a guide to the team, and the number of followers could be a point of attention for us.

From the interviews, we note the recruiters are interested in analyzing the source-code about the add, delete, and changes made by a developer. This way, we create the “Code Churn”. On GitHub, code churn is a metric that measures the number of changes made to a source-code repository. This metric can be used to understand how the code in a project changes over time, helping developers identify issues and opportunities for improvement. This code was manifested by 2 (13%) participants 2 times. The PDF file generated by the framework does not show this metric. Recruiter P14 said.

[...] I believe that Code Churn is a valuable metric for evaluating the quality of a developer's work on GitHub. Code churn can help understand how code evolves over time, identify problems and opportunities for improvement, and even predict quality risks in software projects.

Table 6.5: Category Lacking information

Category	Codes	Participants (%)	Frequency
Lacking Information	Star	11 (73%)	12
	Dev History	5 (33%)	8
	Seeking Information	5 (33%)	6
	Followers	4 (27%)	4
	Code Churn	2 (13%)	2
	Total		32

Source: Prepared by the author, 2023

RQ4 summary. *The recruiters reported the lacking of data from the PDF file, including information on a developer's popularity, historical data from GitHub, and data about the developer's contact information. The recruiters also expressed the necessity of seeking information from other social networking sites when there is a lack of data from GitHub and PDF. Overall, recruiters believe that improvements in the framework could better support their recruitment processes by providing more comprehensive data about developers on GitHub.*

6.13 Threats to Validity

Due to some limitations and threats to the study validity, we need a cautious interpretation of the results of the present study. The results could be different in other geographical and cultural areas. Thus, further work is needed to replicate the results in other geographical areas and software development. In the following, we present the main threats to validity, organized in four typical groups [103].

Construct Validity. To answer our research questions, we asked recruiters open questions. These questions may not cover all data about the recruitment process. However, we scheduled the interview sessions to be relatively long (forty minutes), making sure that we gave the participants enough time to express their ideas and share their thoughts. At the beginning of each interview section, we asked the participants to answer the questions in their own words and provide as much detail as they feel is relevant to

address each question. We also placed an open question at the end of the interview to allow the participants to share any additional information about the topic.

Internal Validity. The framework we used in our study, as the interviews, can be affected by bias and inaccurate responses. This effect could be intentional or unintentional. We repeatedly and constantly used phrases to encourage the participants to provide their honest opinions, using the phrase “based on your experience” in most of the questions. We also indicated that the participants should “feel free to change/add/delete information or not.” Sometimes, we also indicated that “there is no right or wrong answer; we are interested in what you think and your perspective.” Using a study pilot, we also took multiple steps to reduce potential confirmation bias [81]. We asked participants to describe their examples of hiring a new software developer. Another threat to the validity of our study is drawing conclusions based on recovered memories [43]. We are interested in capturing recruiters’ opinions about what components constitute rationale, independently of how accurate their memories are. We encouraged participants to take their time to recall situations and to report the hire that mattered in their experience.

External Validity. Our studied recruiters may only partially represent part of the employers’ population. To mitigate this threat, we recruited a diverse sample of the population with diverse types and amounts of experience. In addition, our interviewees are from three different countries: Brazil, Canada, and United States.

Conclusion Validity. The participation of the author who followed the Grounded Theory procedures poses another threat. His beliefs might have caused some distortions when interpreting the data. To mitigate this threat, the Grounded Theory coding activities were shared with other researchers. Moreover, the identification of the constructs and the depicting of propositions were performed separately by the first author and other researchers. In fact, three authors participated in the Grounded Theory procedures independently; then we merged their results to shape the theory. Thus, the contents were compared and discussed by the researchers until reaching a consensus.

6.14 Concluding Remarks

In this study, we aimed to gain insight into the perceptions of software recruiters from Brazil, United States, and Canada on detecting software experts, crafting an effective resume, and the challenges and problems associated with current recruitment frameworks. Through interviews with 17 recruiters from various organizations, we identified valuable strategies to assess potential candidates and their skills. We highlighted the importance of a well-crafted resume, which should include clear and concise information about the

candidate's experience, projects, and contributions. Our research also identified challenges recruiters face in evaluating candidates, such as assessing hard skills, needing more standardization in the recruitment process, and time constraints. Our study provides practical guidance for recruiters and developers to improve their recruitment processes and increase their chances of success in the highly competitive software industry. In the next chapter, we present the conclusion of this thesis.

Chapter 7

Conclusion

Software development competes to a critical role in today's technological landscape, and the success of projects heavily relies on the quality and experience of software developers. However, evaluating the software engineering experience and skills of developers can present significant challenges. Accurately measuring their hard skills, such as programming proficiency and knowledge of specific technologies, is a complex task. Selecting highly skilled software developers can be a challenging task for recruiters. Traditional resources, such as LinkedIn profiles and coding tests, have their limitations when it comes to accurately assessing the developer's abilities. One of the main challenges lies in effectively evaluating a developer's core skills solely based on these resources. Core skills encompass the technical competencies and expertise that developers possess, which are essential for successfully carrying out software development tasks. However, relying solely on LinkedIn profiles or coding tests often fails to provide a comprehensive understanding of a developer's true capabilities. LinkedIn profiles can be subjective and limited in showcasing a developer's actual coding proficiency, while coding tests typically assess only a specific portion of a developer's skill set. Consequently, recruiters face difficulties in accurately assessing a developer's core skills and making informed hiring decisions based on these traditional resources. This highlights the need for alternative approaches that can delve deeper into a developer's technical abilities and provide a more comprehensive evaluation of their core skills.

This chapter concludes the thesis by discussing the main contributions and prospecting future works. Specifically, in Section 7.1, we revisit an overview of this thesis content and its structure. We summarize the results and highlight the main contributions in Section 7.2 while discussing future research in Section 7.3.

7.1 Thesis Recapitulation

This thesis presents a framework to identify core skills of software developers through source code analysis. For this purpose, three studies were conducted to evaluate the framework, extensively analyzing scenarios involving core skills and the recruitment process. In Chapter 2, the main concepts adopted in this thesis were presented. This chapter served as a foundation for understanding the key principles and methodologies used throughout the research. Additionally, a comprehensive review of the relevant literature in the field of identifying hard skills was provided. The chapter highlighted the existing studies and approaches related to the identification of hard skills in software development, offering insights into the current state of research and establishing the context for the subsequent chapters.

In Chapter 3, we introduced the framework and three instances. Each of these three distinct instances fulfills a specific purpose in evaluating software developers. The first instance focuses on identifying the developer's profile, providing insights into their skills, programming languages, and areas of expertise. The second instance delves into the developer's proficiency in third-party libraries, analyzing their familiarity and utilization of these resources. Lastly, the third instance leverages additional data from the GitHub platform to enrich the developer's profile with supplementary information. By combining these three instances, the framework offers a comprehensive and multi-faceted approach to assess and understand the skills of software developers to help in the recruitment process.

In Chapters 4 and 5, we conducted two proof-of-concept studies to validate the effectiveness of the framework, focusing specifically on the Profiles and Library Experts instances. These studies aimed to demonstrate the framework's capability to identify developers' profiles and assess their expertise in various programming languages, back-end, front-end, unit test, and libraries. Additionally, in Chapter 6, we presented an empirical study involving software recruiters to evaluate the developed framework. This study provided valuable insights into the practical utility of the framework in real-world recruitment scenarios and validated its effectiveness in assisting recruiters in making informed hiring decisions. The combination of these three chapters demonstrates the comprehensive evaluation and validation of the framework, establishing its potential as a valuable tool in the software development industry.

7.2 Contributions

We summarize our contributions as follows:

- We have made a contribution to the field by developing a comprehensive **framework for assessing and computing hard skills** based on source code analysis. This framework offers support to the recruitment process, as it enables the identification and evaluation of developers' skills through the analysis of their Git repository structures. By leveraging the information embedded in developers' source code, the framework provides a more accurate and objective assessment of their hard skills.
- We conducted a study involving library experts to evaluate the effectiveness of the **Libraries Experts instance** in assessing the hard skills related to third-party libraries. This study is important as it provides valuable insights into the framework's capability to accurately evaluate the hard skills of software developers when working with these libraries. By assessing the developers' proficiency and expertise in utilizing libraries, the study contributes significantly to the validation and reliability of the framework, ultimately promoting more informed hiring decisions and enhancing the overall effectiveness of development teams.
- We conducted a study to evaluate the **Profile instance** of the developed framework, focusing on assessing the hard skills of developers. This study provides a thorough evaluation of developers' profiles, including their proficiency in programming languages, their use of unit tests, and their preferences in terms of back-end, front-end, or full-stack development. By evaluating these aspects, the study contributes to a deeper understanding of developers' hard skills and provides valuable insights for effective team composition and skill matching. The findings from this study further validate the Profile Instance and its relevance in assessing the diverse skill sets of software developers.
- We conducted a study to evaluate the **Tech Skill instance** of the developed framework, focusing on analyzing developers' projects and extracting information about their preferred programming languages and proficiency in third-party libraries. This instance goes beyond code analysis by incorporating external data from platforms like GitHub to enhance the analysis. Additionally, we conducted interviews with recruiters to obtain insights into the limitations of the framework and understand how it can support the recruitment process. This study sheds light on the effectiveness of the Tech Skill Instance in assessing developers' hard skills and provides feedback for further improvement and refinement of the framework.

- We designed and implemented a **supporting tool** that facilitates the configuration and use of the framework instances. This tool serves as an interface for importing the necessary data, such as source code repositories, and to run the framework instance to generate the curricula.
- We have also contributed a **set of key metrics** that played a significant role in evaluating the studies conducted with the framework. These metrics, such as Lines of Code, Number of Imports, and Number of Commits, offer valuable insights into developers' coding practices and engagement within their projects. The Lines of Code metric provides an understanding of the volume and complexity of developers' code contributions, highlighting their productivity and involvement in software development tasks. The Number of Imports metric helps identify the libraries and external dependencies utilized by developers, revealing their familiarity and proficiency in leveraging third-party resources. Lastly, the Number of Commits metric offers insights into developers' commitment and frequency of contributions, highlighting their level of engagement and activity within the development process. These metrics serve as fundamental building blocks for evaluating developers' hard skills and are instrumental in the empirical studies conducted within the scope of this research.

7.3 Future Work

Throughout this thesis, we have delved into the complexities and challenges associated with selecting software developers. We have explored the perspectives of both recruiters and developers themselves, focusing on the evaluation of hard skills. Through our investigations, we have identified new avenues of research that aim to enhance the recruitment process. In the subsequent paragraphs, we will discuss these potential areas of study, offering insights and prospects for further exploration.

Soft skill. Soft skills are essential for effective collaboration, communication with stakeholders, and understanding user needs. They contribute to a positive work environment and promote teamwork, creativity, and innovation. On the other hand, hard skills ensure that developers have the technical knowledge and expertise required to carry out their tasks effectively. While this thesis primarily focuses on the evaluation of hard skills, it is equally important to investigate and understand the soft skills of software developers in more detail. Future research could explore methodologies and frameworks to assess and measure soft skills objectively. One way for assessing the soft skills of software develop-

ers could be to use Artificial Intelligence (AI) to develop a chatbot-based system. This chatbot could simulate typical teamwork and stakeholder communication scenarios. By analyzing the developers' responses and behaviors using natural language processing and AI algorithms, the chatbot could evaluate their effective communication, problem-solving abilities, teamwork, and adaptability. The system could present various scenarios, such as conflict situations, team decision-making, requirement negotiations, or project presentations. Through sentiment analysis, natural language processing, and machine learning techniques, the system could assess the developers' communication skills, problem-solving abilities, and decision-making skills. Incorporating a scoring system that assigns weights to different competencies would provide an overall score for the developers' soft skills. This approach would enable a more objective and standardized evaluation of developers' soft skills, allowing for large-scale data collection for statistical analysis and comparison across different profiles. However, it is important to note that assessing soft skills through AI should be supplemented with other evaluation methods, such as behavioral interviews and references, to obtain a comprehensive and accurate understanding of developers' skills.

Instances of the framework. In addition to the existing instances of the framework, there are several other potential instances that could be created to further enhance the assessment of software developers' skills. Some ideas for additional instances include: Project Complexity Instance, Testing and Quality Instance, and Continuous Learning Instance. Project Complexity Instance: could focus on evaluating developers' ability to handle complex software projects. It could analyze factors such as the size of the codebase, the number of dependencies, the level of abstraction, and the use of design patterns. By assessing how developers tackle complex projects, this instance could provide insights into their problem-solving skills, architectural understanding, and ability to manage large-scale software systems. Testing and Quality Instance: could assess developers' proficiency in writing unit tests, conducting code reviews, and ensuring code quality. It could analyze metrics such as code coverage, adherence to coding standards, and the presence of code smells or anti-patterns. By evaluating developers' testing practices and commitment to quality, this instance could provide valuable insights into their attention to detail, software reliability, and ability to deliver robust and maintainable code. Continuous Learning Instance: could assess developers' commitment to continuous learning and professional development. It could analyze factors, such as their participation in online courses, attendance at conferences, contributions to open-source projects, and engagement in community forums. By evaluating developers' efforts to stay updated with the latest technologies and industry trends, this instance could provide insights into their adaptability, eagerness to learn, and potential for growth.

Longitudinal Analysis. Another venue for future work could be to conduct longitudinal

studies to track and analyze the growth and development of developers' hard skills over time. This could involve collecting data at different intervals and comparing the changes in their skill profiles, providing insights into the effectiveness of learning and training programs.

DevOps Integration. We can explore ways to integrate DevOps practices and principles into the framework. DevOps focuses on the collaboration and integration between software development and IT operations teams, emphasizing automation, continuous delivery, and monitoring. Investigate how the framework can incorporate metrics and indicators that assess developers' proficiency in DevOps-related skills, such as infrastructure automation, version control, continuous integration, and deployment processes. This integration can provide a comprehensive assessment of developers' abilities to work effectively in a DevOps environment and contribute to the continuous improvement of software development practices.

Bibliography

- [1] Adnin, R., Afroz, S., Ulfat, M., and Iqbal, A. A hiring story: Experiences of employers in hiring cs graduates in software startups. In *Companion Publication of the 2022 Conference on Computer Supported Cooperative Work and Social Computing, CSCW'22 Companion*, page 126–129, New York, NY, USA, 2022. Association for Computing Machinery.
- [2] Ahmed, F., Capretz, L. F., Bouktif, S., and Campbell, P. Soft skills requirements in software development jobs: a cross-cultural empirical study. *Journal of systems and information technology (JSIT)*, 2012.
- [3] Al-Ani, B., Bietz, M. J., Wang, Y., Trainer, E., Koehne, B., Marczak, S., Redmiles, D., and Prikladnicki, R. Globally distributed system developers: Their trust expectations and processes. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13*, page 563–574, New York, NY, USA, 2013. Association for Computing Machinery.
- [4] Amreen, S., Karnauch, A., and Mockus, A. Developer reputation estimator (dre). In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering, ASE '19*, page 1082–1085. IEEE Press, 2020.
- [5] Assyne, N. Hard competencies satisfaction levels for software engineers: a unified framework. In *Software Business: 10th International Conference, ICSOB 2019, Jyväskylä, Finland, November 18–20, 2019, Proceedings 10*, pages 345–350. Springer, 2019.
- [6] Bailey, J. and Mitchell, R. B. Industry perceptions of the competencies needed by computer programmers: technical, business, and soft skills. *Journal of Computer Information Systems*, 2006.
- [7] Bailey, J. L. and Stefaniak, G. Industry perceptions of the knowledge, skills, and abilities needed by computer programmers. In *Conference on Computer personnel research (SIGCPR)*, 2001.
- [8] Baltes, S. and Diehl, S. Towards a theory of software development expertise. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018.

-
- [9] Barring, M., Nåfors, D., Henriksen, D., Olsson, D., Johansson, B., and Larsson, U. A vsm approach to support data collection for a simulation model. In *Proceedings of the 2017 Winter Simulation Conference, WSC '17*. IEEE Press, 2017.
- [10] Basili, V., Caldiera, G., and Rombach, H. D. *The Goal Question Metric Approach*. Online Technical Report, 1994.
- [11] Behroozi, M., Parnin, C., and Brown, C. Asynchronous technical interviews: Reducing the effect of supervised think-aloud on communication ability. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022*, page 294–305, New York, NY, USA, 2022. Association for Computing Machinery.
- [12] Bergersen, G. R., Sjøberg, D. I., and Dybå, T. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering*, 40(12):1163–1184, 2014.
- [13] Bhasin, T., Murray, A., and Storey, M.-A. Student experiences with GitHub and stack overflow: An exploratory study. In *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 81–90. IEEE, 2021.
- [14] Bizer, C., Heath, T., and Berners-Lee, T. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, 2011.
- [15] Black, S. L., Washington, M. L., and Schmidt, G. B. How to stay current in social media to be competitive in recruitment and selection. In *Social Media in Employee Selection and Recruitment*. Springer, 2016.
- [16] Bloor, M. *Focus groups in social research*. Sage, 2001.
- [17] Brown, V. R. and Vaughn, E. D. The writing on the (facebook) wall: The use of social networking sites in hiring decisions. *Journal of Business and psychology (JBP)*, 2011.
- [18] Cappel, J. J. Entry-level is job skills: A survey of employers. *Journal of Computer Information Systems (JCIS)*, 2002.
- [19] Capretz, L. F. and Ahmed, F. Making sense of software development and personality types. *IT professional*, 2010.
- [20] Cavaiani, T. P. Cognitive style and diagnostic skills of student programmers. *Journal of Research on Computing in Education (JRCE)*, 2014.

-
- [21] Cherubini, M., Venolia, G., DeLine, R., and Ko, A. J. Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07*, page 557–566, New York, NY, USA, 2007. Association for Computing Machinery.
- [22] Chinn, D. and VanDeGrift, T. Gender and diversity in hiring software professionals: What do students say? In *Proceedings of the Fourth International Workshop on Computing Education Research, ICER '08*, page 39–50, New York, NY, USA, 2008. Association for Computing Machinery.
- [23] Constantino, K., Zhou, S., Souza, M., Figueiredo, E., and Kästner, C. Understanding collaborative software development: An interview study. In *International Conference on Global Software Engineering, ICGSE '20*, page 55–65, New York, NY, USA, 2020. Association for Computing Machinery.
- [24] Constantino, K., Belém, F., and Figueiredo, E. Dual analysis for helping developers to find collaborators based on co-changed files: An empirical study. *Software: Practice and Experience*, 53(6):1438–1464, 2023.
- [25] Constantino, K., Souza, M., Zhou, S., Figueiredo, E., and Kästner, C. Perceptions of open-source software developers on collaborations: An interview and survey study. *Journal of Software: Evolution and Process*, 35(5):e2393, 2023.
- [26] Constantinou, E. and Kapitsaki, G. M. Identifying developers' expertise in social coding platforms. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016.
- [27] da Silva, F. Q., Costa, C., França, A. C. C., and Prikladinicki, R. Challenges and solutions in distributed software development project management: A systematic literature review. In *2010 5th IEEE International Conference on Global Software Engineering*, pages 87–96, 2010.
- [28] Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. Social coding in github: transparency and collaboration in an open software repository. In *Conference on computer supported cooperative work (CSCW)*, 2012.
- [29] Destefanis, G., Ortu, M., Counsell, S., Swift, S., Marchesi, M., and Tonelli, R. Software development: do good manners matter? *PeerJ Computer Science*, 2016.
- [30] Dey, T., Karnauch, A., and Mockus, A. Representation of developer expertise in open source software. In *International Conference on Software Engineering (ICSE)*, 2021.

-
- [31] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. *Selecting empirical methods for software engineering research*, pages 285–311. Springer London, 2008.
- [32] Gaaloul, K., Menghi, C., Nejati, S., Briand, L. C., and Wolfe, D. Mining assumptions for software components using machine learning. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 159–171, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370431. doi: 10.1145/3368089.3409737. URL <https://doi.org/10.1145/3368089.3409737>.
- [33] Garcia, V. C., Lucrédio, D., Alvaro, A., de Almeida, E. S., de Mattos Fortes, R. P., and de Lemos Meira, S. R. Towards a maturity model for a reuse incremental adoption. In *Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)*, 2007.
- [34] Gill, A. M. and Nonnecke, B. Think aloud: Effects and validity. In *Proceedings of the 30th ACM International Conference on Design of Communication*, SIGDOC '12, page 31–36, New York, NY, USA, 2012. Association for Computing Machinery.
- [35] Greene, G. J. and Fischer, B. Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In *International Conference on Automated Software Engineering (ASE)*, 2016.
- [36] Hauff, C. and Gousios, G. Matching GitHub developer profiles to job advertisements. In *Working Conference on Mining Software Repositories (MSR)*, 2015.
- [37] Hoda, R. Decoding grounded theory for software engineering. In *Proceedings of the 43rd International Conference on Software Engineering: Companion Proceedings*, ICSE '21, page 326–327. IEEE Press, 2021.
- [38] Huang, H., Kvasny, L., Joshi, K. D., Trauth, E. M., and Mahar, J. Synthesizing it job skills identified in academic studies, practitioner publications and job ads. In *Special interest group on management information system's conference on Computer personnel research (SIGMIS CPR)*, 2009.
- [39] Huang, W., Mo, W., Shen, B., Yang, Y., and Li, N. CPDScorer: Modeling and evaluating developer programming ability across software communities. In *International Conference on Software Engineering and Knowledge Engineering (ICISDM)*, 2016.
- [40] Jia, J., Chen, Z., and Du, X. Understanding soft skills requirements for mobile applications developers. In *2017 IEEE International Conference on Computational*

- Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, volume 1, pages 108–115, 2017.
- [41] Kalliamvakou, E., Damian, D., Blincoe, K., Singer, L., and German, D. M. Open source-style collaborative development practices in commercial projects using GitHub. In *International Conference on Software Engineering (ICSE)*, 2015.
- [42] Kitchenham, B. A. and Pfleeger, S. L. *Personal Opinion Surveys*, pages 63–92. Springer London, 2008.
- [43] Koriat, A., Goldsmith, M., and Pansky, A. Toward a psychology of memory accuracy. *Annual Review of Psychology*, 51(1):481–537, 2000. PMID: 10751979.
- [44] Kourtzanidis, S., Chatzigeorgiou, A., and Ampatzoglou, A. Reposkillminer: Identifying software expertise from GitHub repositories using natural language processing. In *International Conference on Automated Software Engineering (ASE)*, 2020.
- [45] Kourtzanidis, S., Chatzigeorgiou, A., and Ampatzoglou, A. Reposkillminer: Identifying software expertise from GitHub repositories using natural language processing. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, ASE '20*, page 1353–1357, New York, NY, USA, 2021. Association for Computing Machinery.
- [46] Krüger, J., Wiemann, J., Fenske, W., Saake, G., and Leich, T. Do you remember this source code? In *International Conference on Software Engineering (ICSE)*, 2018.
- [47] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521:436–444, 2015.
- [48] Lee, J. H., Shin, J., and Realff, M. J. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. *Computers Chemical Engineering*, 114:111–121, 2018. ISSN 0098-1354. FOCAPO/CPC 2017.
- [49] Liang, J. T., Zimmermann, T., and Ford, D. Towards mining oss skills from GitHub activity. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER '22*, page 106–110, New York, NY, USA, 2022. Association for Computing Machinery.
- [50] Lin, B., Zagalsky, A., Storey, M.-A., and Serebrenik, A. Why developers are slacking off: Understanding how software teams use slack. In *Conference on Computer Supported Cooperative Work and Social Computing Companion (CSCW)*, 2016.
- [51] Litecky, C. R., Arnett, K. P., and Prabhakar, B. The paradox of soft skills versus technical skills in is hiring. *Journal of Computer Information Systems (JCIS)*, 2004.

- [52] Manes, S. S. and Baysal, O. How often and what stackoverflow posts do developers reference in their GitHub projects? In *Proceedings of the 16th International Conference on Mining Software Repositories, MSR '19*, page 235–239. IEEE Press, 2019.
- [53] Marlow, J. and Dabbish, L. Activity traces and signals in software developer recruitment and hiring. In *Conference on Computer Supported Cooperative Work (CSCW)*, 2013.
- [54] Marlow, J. and Dabbish, L. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13*, page 145–156, New York, NY, USA, 2013. Association for Computing Machinery.
- [55] Marlow, J. and Dabbish, L. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13*, page 145–156, New York, NY, USA, 2013. Association for Computing Machinery.
- [56] Marlow, J., Dabbish, L., and Herbsleb, J. Impression formation in online peer production: activity traces and personal profiles in GitHub. In *Conference on Computer supported cooperative work (CSCW)*, 2013.
- [57] Matturro, G. Soft skills in software engineering: A study of its demand by software companies in uruguay. In *International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2013.
- [58] McCuller, P. *How to Recruit and Hire Great Software Engineers: Building a Crack Development Team*. Apress, USA, 1st edition, 2012. ISBN 143024917X.
- [59] McDonald, D. W. and Ackerman, M. S. Expertise recommender: A flexible recommendation system and architecture. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, CSCW '00*, page 231–240, New York, NY, USA, 2000. Association for Computing Machinery.
- [60] McDonald, N. and Goggins, S. Performance and participation in open source software on GitHub. In *Extended Abstracts on Human Factors in Computing Systems (CHI EA)*, 2013.
- [61] Merriam, S. B. and Tisdell, E. J. *Qualitative research: A guide to design and implementation*. John Wiley & Sons, 2015.
- [62] Meyer, A. N., Fritz, T., Murphy, G. C., and Zimmermann, T. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT*

- International Symposium on Foundations of Software Engineering*, FSE 2014, page 19–29, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330565.
- [63] Meyer, A. N., Barton, L. E., Murphy, G. C., Zimmermann, T., and Fritz, T. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 43(12):1178–1193, 2017.
- [64] Meyer, A. N., Zimmermann, T., and Fritz, T. Characterizing software developers by perceptions of productivity. In *Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '17, page 105–110. IEEE Press, 2017.
- [65] Mockus, A. and Herbsleb, J. Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 503–512, 2002.
- [66] Montandon, J. E., Silva, L. L., and Valente, M. T. Identifying experts in software libraries and frameworks among GitHub users. In *International Conference on Mining Software Repositories (MSR)*, 2019.
- [67] Montandon, J. E., Silva, L. L., and Valente, M. T. Identifying experts in software libraries and frameworks among GitHub users. In *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR '19, page 276–287. IEEE Press, 2019.
- [68] Montandon, J. E., Politowski, C., Silva, L. L., Valente, M. T., Petrillo, F., and Guéhéneuc, Y.-G. What skills do it companies look for in new developers? a study with stack overflow jobs. *Information and Software Technology (IST)*, 2021.
- [69] Muratbekova-Touron, M. and Galindo, G. Leveraging psychological contracts as an hr strategy: The case of software developers. *European Management Journal*, 36(6):717–726, 2018.
- [70] Oliveira, J., Fernandes, E., Souza, M., and Figueiredo, E. A method based on naming similarity to identify reuse opportunities. *Revista Brasileira de Sistemas de Informação (iSys)*, 2017.
- [71] Oliveira, J., Vigiato, M., and Figueiredo, E. How well do you know this library? Mining experts from source code analysis. In *Proceedings of the XVIII Brazilian Symposium on Software Quality*, SBQS '19, page 49–58. Association for Computing Machinery, 2019.

- [72] Oliveira, J., Pinheiro, D., and Figueiredo, E. Jexpert: A tool for library expert identification. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, page 386–392. Association for Computing Machinery, 2020.
- [73] Oliveira, J., Souza, M., Flauzino, M., Durelli, R., and Figueiredo, E. Can source code analysis indicate programming skills? survey with developers. In *Proceedings of the International Conference Quality of Information and Communications Technology*, pages 156–171. Springer International Publishing, 2022.
- [74] Oliveira, J., Souza, M., Flauzino, M., Durelli, R., and Figueiredo, E. Can source code analysis indicate programming skills? a survey with developers. In Vallecillo, A., Visser, J., and Pérez-Castillo, R., editors, *Quality of Information and Communications Technology*, pages 156–171, Cham, 2022. Springer International Publishing.
- [75] Oliveira, J., Souza, M., and Figueiredo, E. Evaluating a method to select software developers from source code analysis, 2023. URL <https://doi.org/10.5281/zenodo.7739022>.
- [76] Oliveira, J. A., Viggiano, M., Pinheiro, D., and Figueiredo, E. Mining experts from source code analysis: An empirical evaluation. *Journal of Software Engineering Research and Development*, 9(1):1:1 – 1:16, Feb. 2021.
- [77] Olmsted-Hawala, E. L., Murphy, E. D., Hawala, S., and Ashenfelter, K. T. Think-aloud protocols: A comparison of three think-aloud protocols for use in testing data-dissemination web sites for usability. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 2381–2390, New York, NY, USA, 2010. Association for Computing Machinery.
- [78] Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M., and Tonelli, R. Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. In *Working Conference on Mining Software Repositories (MSR)*, 2015.
- [79] Ortu, M., Destefanis, G., Counsell, S., Swift, S., Tonelli, R., and Marchesi, M. Arsonists or firefighters? affectiveness in agile software development. In *International Conference on Agile Software Development*, 2016.
- [80] Pfleeger, S. L. and Kitchenham, B. A. Principles of survey research: Part 1: Turning lemons into lemonade. *Software Engineering Notes (SIGSOFT)*, 2001.
- [81] Pohl, R. and Pohl, R. *Cognitive Illusions: A Handbook on Fallacies and Biases in Thinking, Judgement and Memory*. Psychology Press, 2004.
- [82] Radermacher, A., Walia, G., and Knudson, D. Investigating the skill gap between graduating students and industry expectations. In *International Conference on Software Engineering (ICSE)*, 2014.

- [83] Rosner, B., Glynn, R. J., and Lee, M. L. The wilcoxon signed rank test for paired comparisons of clustered data. *Biometrics*, 2006.
- [84] Rubin, H. J. and Rubin, I. S. *Qualitative interviewing: The art of hearing data*. sage, 2011.
- [85] Ruvimova, A., Lill, A., Gugler, J., Howe, L., Huang, E., Murphy, G., and Fritz, T. An exploratory study of productivity perceptions in software teams. In *International Conference on Software Engineering (ICSE)*, pages 234–235. ACM, 2022.
- [86] Santos, A., Souza, M., Oliveira, J., and Figueiredo, E. Mining software repositories to identify library experts. In *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS '18*, page 83–91. Association for Computing Machinery, 2018.
- [87] Sarma, A., Chen, X., Kuttal, S., Dabbish, L., and Wang, Z. Hiring in the global stage: Profiles of online contributions. In *2016 IEEE 11th International Conference on Global Software Engineering (ICGSE)*, pages 1–10, 2016.
- [88] Saxena, R. and Pedanekar, N. I know what you coded last summer. In *Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*, 2017.
- [89] Schütze, H., Manning, C. D., and Raghavan, P. *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008.
- [90] Silveira, K. K., Musse, S., Manssour, I. H., Vieira, R., and Prikladnicki, R. Confidence in programming skills: Gender insights from stackoverflow developers survey. In *International Conference on Software Engineering (ICSE)*, pages 234–235. IEEE, 2019.
- [91] Singer, L., Filho, F. F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Conference on Computer supported cooperative work - (CSCW)*, 2013.
- [92] Sommerville, I. *Software Engineering*. Pearson, 2015.
- [93] Stol, K.-J., Ralph, P., and Fitzgerald, B. Grounded theory in software engineering research: A critical review and guidelines. In *International Conference on Software Engineering (ICSE)*, 2016.
- [94] Strauss, A. L. and Corbin, J. M. *Basics of qualitative research: techniques and procedures for developing grounded theory*. Sage Publications, Thousand Oaks, Calif, 1998.

- [95] Tantisuwankul, J., Nugroho, Y. S., Kula, R. G., Hata, H., Rungsawang, A., Leelaprute, P., and Matsumoto, K. A topological analysis of communication channels for knowledge sharing in contemporary GitHub projects. *Journal of Systems and Software (JSS)*, 2019.
- [96] Teyton, C., Palyart, M., Falleri, J.-R., Morandat, F., and Blanc, X. Automatic extraction of developer expertise. In *International Conference on Evaluation and Assessment in Software Engineering - (EASE)*, 2014.
- [97] Tong, J., Ying, L., Hongyan, T., and Zhonghai, W. Can we use programmer's knowledge? fixing parameter configuration errors in hadoop through analyzing q amp;a sites. In *International Congress on Big Data (BigData Congress)*, 2016.
- [98] Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Penta, M. D., Lucia, A. D., and Shybyvanyk, D. When and why your code starts to smell bad. In *International Conference on Software Engineering (ICSE)*, 2015.
- [99] Vadlamani, S. L. and Baysal, O. Studying software developer expertise and contributions in stack overflow and GitHub. In *International Conference on Software Maintenance and Evolution (ICSME)*, 2020.
- [100] Wan, C., Liu, S., Xie, S., Liu, Y., Hoffmann, H., Maire, M., and Lu, S. Automated testing of software that uses machine learning apis. In *Proceedings of the 44th International Conference on Software Engineering, ICSE '22*, page 212–224, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392211. doi: 10.1145/3510003.3510068. URL <https://doi.org/10.1145/3510003.3510068>.
- [101] Wan, Y., Chen, L., Xu, G., Zhao, Z., Tang, J., and Wu, J. SCSMiner: mining social coding sites for software developer recommendation with relevance propagation. *World Wide Web (WWW)*, 2018.
- [102] Wang, Z., Sun, H., Fu, Y., and Ye, L. Recommending crowdsourced software developers in consideration of skill improvement. In *International Conference on Automated Software Engineering (ASE)*, 2017.
- [103] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
- [104] Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.
- [105] Ye, C. Research on the key technology of big data service in university library. In *International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, 2017.

-
- [106] Zacchiroli, S. A large-scale dataset of (open source) license text variants. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, page 757–761, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393034. doi: 10.1145/3524842.3528491. URL <https://doi.org/10.1145/3524842.3528491>.
- [107] Zhou, M. and Mockus, A. Developer fluency: Achieving true mastery in software projects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 137–146, 2010.
- [108] Zieris, F. and Prechelt, L. Explaining pair programming session dynamics from knowledge gaps. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 421–432. IEEE, 2020.