A REUSE-BASED APPROACH FOR EVALUATING DEVELOPER CONTRIBUTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Yahya Elnouby
July 2025

A Reuse-Based Approach for Evaluating Developer Contribution
By Yahya Elnouby
July 2025

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Eray Tüzün(Advisor)

Can Alkan

Engin Demir

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

A REUSE-BASED APPROACH FOR EVALUATING DEVELOPER CONTRIBUTION

Yahya Elnouby
M.S. in Computer Engineering
Advisor: Eray Tüzün
July 2025

Evaluating developer contributions is essential for effective resource allocation and recognizing expertise. Traditional metrics—such as lines of code (LOC) or commit counts—lack sufficient context, as code varies in complexity and importance. Reusable code, however, is a key indicator of quality and can serve as a better metric for contribution evaluation. Our goal is to develop a methodology and practical tool to assess developers' reusable code contributions in software projects. Inspired by Hirsch's H-Index used in academia, we introduce the Developer H-Index (DH-Index), which tracks method usage through project-wide call graphs, analogous to academic citations. These usage references are linked to individual developer contributions. We further propose the Lines-of-Code-Weighted Developer H-Index (LWDH-Index), which incorporates method length to account for the significance of reused code. We implemented this approach in DevScholar, an open-source prototype that analyzes Java project repositories to extract method-based developer metrics. We evaluated DevScholar against GitHub Insights using two Open-Source Software (OSS) projects: Spring Boot and Apollo. The LWDH-Index proved more robust, reducing the distortion caused by overly simple but frequently reused methods or complex methods used sparingly. Additionally, we conducted a user study with an international airline company, analyzing three internal projects. Our findings suggest that, compared to traditional LOC- or commit-based metrics, DH-Index and LWDH-Index offer richer insight into the impact and quality of developer contributions, particularly in terms of code reusability. This contributes to a more nuanced understanding of developer performance in team-based software development.

Keywords: Developer Contribution, Mining Software Repositories, Key Developers, Core developers, H-Index, Call Graphs, Lines of Code, Reusable Code.

ÖZET

GELİŞTİRİCİ KATKISINI DEĞERLENDİRMEK ÜZERE YENİDEN KULLANIM ODAKLI BİR YAKLAŞIM

Yahya Elnouby
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Eray Tüzün
Temmuz 2025

Geliştirici katkılarının değerlendirilmesi, kaynakların etkin tahsisi ve uzmanlığın görünür kılınması açısından kritik öneme sahiptir. Ancak satır sayısı (LOC) veya commit adedi gibi geleneksel metrikler, kodun karmaşıklık ve önem farklılıklarını yansıtmadığından yeterli bağlam sağlayamamaktadır. Bunun yerine yeniden kullanılabilir kod, kaliteyi gösteren güçlü bir belirteç olup katkı ölçümü için daha sağlıklı bir ölçüt sunmaktadır. Bu çalışmada, yazılım projelerinde geliştiricilerin yeniden kullanılabilir kod katkılarını ölçmeye yönelik metodolojik ve pratik bir yaklaşım ortaya konmaktadır. Akademide kullanılan Hirsch H-Endeksi'nden esinlenerek, proje genelindeki çağrı grafiklerinden yöntem kullanımını izleyen Geliştirici H-Endeksi (DH-Index) tanımlanmaktadır. Bu kullanım referansları, akademik atıflara benzer biçimde geliştirici katkılarıyla ilişkilendirilmektedir. Ayrıca yöntem uzunluğunu da hesaba katarak yeniden kullanılan kodun önemini yansıtan Satır-Sayısı Ağırlıklı Geliştirici H-Endeksi (LWDH-Index) önerilmektedir. Önerilen yaklaşım, Java proje depolarını analiz ederek yöntem tabanlı geliştirici metrikleri çıkaran açık kaynaklı DevScholar prototipine uygulanmıştır. DevScholar'ın etkinliği, SpringBoot ve Apollo adlı açık kaynak yazılım projeleri üzerinde GitHubInsights ile karşılaştırmalı olarak değerlendirilmiştir. LWDH-Index'in, basit ancak sıkça kullanılan yöntemler ile nadiren kullanılan karmaşık yöntemlerin oluşturduğu sapmayı azalttığı gözlemlenmiştir. Ayrıca uluslararası bir havayolu şirketinde yapılan kullanıcı çalışmasında, üç iç proje analiz edilmiştir. Bulgular, DH-Index ve LWDH-Index'in geliştirici katkılarına dair daha zengin içgörüler sunduğunu göstermektedir.

Anahtar sözcükler: Yazılım Geliştirici Katkısı, Yazılım Havuzu Madenciliği, Kilit Geliştiriciler, Çekirdek Geliştiriciler, H-Endeksi, Çağrı Grafikleri, Kod Satırları, Yeniden Kullanılabilir Kod.

Acknowledgement

I would like to express my heartfelt gratitude to my supervisor, Associate Professor Eray Tüzün. His unwavering support, insightful guidance, and continued trust in my abilities have been instrumental throughout my academic journey. His expertise and mentorship have played a vital role in shaping both my academic and personal growth.

Furthermore, I sincerely thank the esteemed members of my examination committee, Associate Professor Can Alkan and Associate Professor Engin Demir. I am truly grateful for the time they devoted to reviewing my work and for their thoughtful insights and evaluations.

My heartfelt gratitude also goes to my beloved family—my mother, Gihan; my father, Mahmoud; my brother, Abdulrahman; and my sister, Jana. Their unwavering love and support have been a constant source of strength and inspiration throughout this journey.

Last but not least, I am deeply grateful to İlayda, Nursenem, and their family (the Altıntaş family) who have been more than just friends and have stood by me whenever I needed them. I also wish to thank my fellow Master's colleagues, Kübra, Can, and Alp, for their support and friendship. Additionally, I extend my sincere appreciation to my Layermark family for their encouragement and support throughout my professional career.

Yahya

July 2025, Ankara

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software projects typically involve the collaboration of multiple developers. Determining each developer's contribution to software projects is intricate due to the software's intangible and changing nature. Moreover, a development team's effectiveness is gauged by its collective output, including both software as well as other assets such as documentation. These factors make accurately evaluating the quality of each individual's input within the team a complicated task.

Platforms like GitHub [1] rank repository contributors based on the number of commits and lines of code added or deleted. Although commits serve as contribution indicators, they are of different lengths and importance, rendering them incomparable and potentially misleading. This approach can encourage a higher quantity of commits, but not necessarily of better quality or with more impact. Ranking developer contributions based on the number of added or deleted lines can lead to the same issue.

Developing reliable metrics to evaluate developers' contributions can yield valuable insights into their skills, proficiency, and experience level. Such insights enable managers to optimize work allocation, identify training needs, pair experienced developers with newcomers, and enhance overall productivity. These metrics can also potentially identify key contributors, offering a more objective

assessment than peer or managerial reviews.

Software companies rely on various metrics to inform decisions and improve their operations. The selection and accuracy of these metrics are crucial, as they may affect company efficiency. Although inaccurate metrics can be detrimental, well-chosen, precise metrics can drive positive and constructive decisions. This principle extends to metrics for evaluating developer contributions. Effective metrics can ensure equitable work distribution, whereas flawed metrics might inadvertently encourage gaming the system.

1.1 Research Problem

The thesis was motivated by the absence of a metric capable of evaluating the impact and reusability of the developer's contribution in a balanced way.

Similar efforts have been made in academia to quantify and measure the author's contributions. Academic indexes such as the H-Index are used to evaluate individuals' cumulative academic work. Hirsch's Index (H-Index) is based on the idea that a researcher's work cannot be measured solely by the number of publications or citations they have. The H-Index is defined as the largest h number of publications with at least h citations, which estimates the cumulative impact of a researcher's work [2].

Our motivation is to develop a metric aligned with the notion of the H-Index. To that end, we investigated whether the principles of the H-Index can help us evaluate the quality of the developers' contributions to a software project. A similar study was conducted in 2012 by Capiluppi et al. [3]. The authors developed three distinct indices to evaluate the contributions of the developers. The first index measures a developer's involvement by counting how many projects they rank as top contributors. The second index assesses the developer's engagement by considering the community sizes of the projects they have been part of. Lastly, the third index quantifies a developer's impact by using the number of times a

project they contributed to was downloaded. In contrast to this study, we wanted to investigate and develop a metric and a tool to quantify a developer's contributions within a single project. The following research question guides the thesis:
RQ: How can we assess individual developers' contributions within a software project from the perspective of contributions' reusability?

In software development, method references can be a valuable synonym for citations as they reveal the interdependence and reusability of the code authored by individual developers. When a method is referenced many times, the developer's work is utilized throughout the project.

Simply counting a developer's total method references can be problematic, similar to citation counts in academic work, where the frequency of citations varies greatly. Some methods are referenced extensively, while others are seldom used. To address this variability, we applied the H-Index concept, commonly used in academia, to identify and rank critical methods, thus providing a more meaningful metric.

1.2 Contribution of the Thesis

To implement our metrics, we focused on method references (also referred to as method usage in this thesis). We initially gathered detailed method data, capturing each method's contributors and their specific contributions. This data was collected by parsing all relevant commits to account for contributions that might not be reflected in the final version of the code. Additionally, we tracked references to these methods throughout the project.

It should be noted that definition of contribution within this thesis is limited to contribution to the codebase. Other contributions,such as documentation, system design, requirement definition, etc., are not involved in our metrics.

As a concrete implementation of our approach, we developed DevScholar, a tool

designed to extract and analyze developer data to present developer contributions using multiple metrics, including new H-Index-inspired ones. To validate our results, we ran our tool on four different Open Source Software (OSS) projects, presenting the results of two, and performed a user study. The tool offers a diverse set of rankings for developers based on various contribution metrics, which are Lines-of-Code-Weighted Developer H-Index (LWDH-Index), Developer H-Index (DH-Index), and total contributed LOC. It also displays information about co-developers and their contributions to individual methods.

Thesis Organization. Chapter 2 provides the background. Chapter 3 explains the adaptation of citation metrics to software development and the generation of the new H-Index-inspired indexes. Chapter 4 presents the results of our validation with DevScholar. Chapter 5 discusses the implications of the work. Threats to validity are discussed in Chapter 6. Chapter 7 discusses the related work. Finally, conclusions are stated in Chapter 8.

Chapter 2

Background

In this chapter, we introduce the H-Index metric, which was the inspiration for our work.

In academia, several indexes have been proposed to evaluate the research performance, including the total number of papers, the total number of citations, or citations for each paper. However, the measurement of the research performance of each researcher was problematic due to not having of a statistically stable and reliable metric. Often, a researcher has few publications with many citations and many publications with few citations.

To strike a balance between these two dimensions, the h-index was introduced by Jorge Hirsch, which is a single-number metric that reflects both the productivity and citation impact of a researcher's publications, defined as the highest number h such that h papers have at least h citations each [2]. It has been widely adopted due to its simplicity, robustness, and ability to represent consistent scholarly performance while minimizing the influence of extremely high or low citation counts. The h-index does not decline over time and favors researchers with a steady record of impactful publications, making it especially useful for evaluating long-term scientific contribution.

Several studies have investigated the validity of the h-index, comparing it with

traditional bibliometric indicators and peer evaluations. Findings indicate a generally strong correlation between the h-index and both peer-reviewed judgments and other metrics, such as citation counts, suggesting that it is a reasonable proxy for scholarly impact. However, some research questions the index's precision, especially for fine-grained assessments, indicating that while useful, it may not fully capture the nuances of scientific quality [4].

Bornmann et al. [4] also mentioned that despite its strengths, the h-index faces several criticisms. It oversimplifies multidimensional scholarly performance into a single metric, lacks field normalization, and disadvantages early-career researchers due to its cumulative nature. It can also be skewed by name ambiguities in databases and incomplete citation records. Moreover, because citation practices differ across disciplines and evolve over time, comparisons using the h-index are only meaningful within similar fields and career stages, necessitating caution in its interpretation and use for evaluative decisions.

In Table 2.1, we can see that Author 2 has the H-Index value of two, because they have two publications with at least two citations. In contrast, Author 1 has the H-Index value of three since they have three publications with at least three citations. Although Author 2 has more citations in total, the H-Index gives importance to equal distribution of citations to avoid, for example, giving more importance to an author with only one popular article than an author who has a steady stream of citations across their portfolio of publications.

Table 2.1: An Example of H-Index

	Author 1	**Author 2**
No of Citations of Publication 1	5	20
No of Citations of Publication 2	4	2
No of Citations of Publication 3	3	1
Sum of All Citations	12	23
H-Index	3	2

Chapter 3

Methodology

In this thesis, we applied the concept of an H-Index to software development to evaluate developer contributions in terms of code reusability. Table 3.1 depicts an analogy between academia and software development. Later, we used this table to create developer contribution metrics in line with the H-Index concept.

We started by applying the H-Index concept to software development, generating the Developer H-Index (DH-Index) metric by treating method usage as analogous to publication citations. While the DH-Index quantifies developer contributions based on method usage, it fell short in capturing contributions for methods not designed for frequent calls by other methods. Investigating this issue from other perspectives, we incorporated LOC into our measurement, generating Lines-of-Code-Weighted Developer H-Index (LWDH-Index). To apply these metrics, we implemented a tool named DevScholar[1].

[1]https://devscholar.netlify.app/

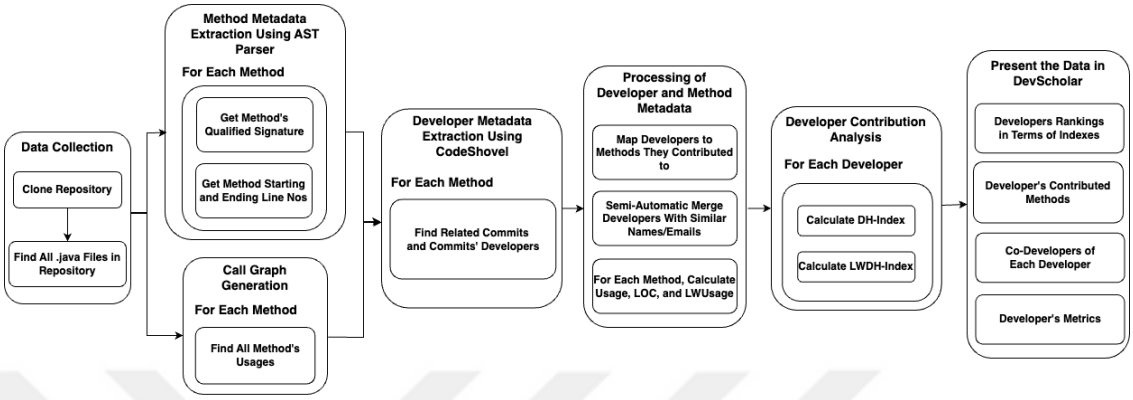

Figure 3.1: Workflow of DevScholar

3.1 Generation of DH-Index and LOC Weighted DH-Index

Applying the H-Index concept to software development, we generate the DH-Index to evaluate the quality of the developers' contributions in terms of reusability. DH-Index is represented as the largest number h such that h methods have at least h usages, which is calculated as follows,

$$\textit{DH-Index (Usage)} = \max\{h \in \mathbb{N} : \textit{Usage}(h) \geq h\} \tag{3.1}$$

To further elaborate, we draw an analogy and comparison of the H-Index in academia and the DH-Index in software development, as shown in Table 3.1. For the H-Index, the primary artifact under consideration is the academic publication, while in software development, the principal artifact becomes the method implemented by the developers. While the publication is written by the author(s), a method is written by the developer(s), the key difference is that the authors of a publication do not change over time. In contrast, developers of a method are dynamic; as new developers contribute to an implemented method, we consider them to be one of the developers of that method. The publication's authors are ordered according to who contributed more to the publication. In a software project, the developers' ranking is dynamic as the developers are ranked according to their methods' number of usage, which evolves. H-Index concept

Table 3.1: Mapping H-Index Analogy to Software Development

Comparison Criteria	**H-Index Concept**	**DH-Index Concept**
Artifact	Publication	Method
Artifact Contributor	Publication's Author(s)	Method's Developer(s)
Ranking of Artifact Authors	Author Order	Developer Ranking
Level of Contribution to the Artifact	Order of Authors Listed On the Publication	Comparative LOC contribution
Stability of Contributor Rankings over Time	Stable	Changeable
Importance Determinant of Artifact	Number of Citations to the Publication	Number of Usages of the Method
Artifact Scope	All Recognized Publication Venues	Project
Self Citation	Not Preferred	Accepted (Recursive Methods or Same Author's Methods)
Lifetime of an Artifact	Infinite	Finite
Acceptance of an Artifact	Publication in Conference/Journal	Approved through a Merge/Pull Request
Artifact Contents	Stable	Changeable
Editability of the Artifact	None	Rewriting/ Refactoring the Method

covers all recognized publication venues. However, the DH-Index is computed only for a specific project. A conceptual difference between citations and code usage is that, in academia, self-citation is believed to inflate an author's H-Index value, but in software projects, a developer reusing code, whether their own or another developer's, is natural, so differentiating self-use is not necessary. The lifetime of a publication is infinite since a publication cannot be deleted, while in a software project, code fragments, including methods, can be deleted. Another difference concerns how contributions are legitimized: publications are legitimized by their acceptance to a recolonized publication venue, after which citations to that publication are counted. In software projects, there is no standard vetting of contributed code, even when projects are required to use pull requests (PR) with quality checks. If that is the case, we could think of a PR as analogous to a contributed code fragment. Other differences relate to the artifact content and editability of the artifact. A publication's content is stable once published, but code is ever-changing: in a software project, a method written by a contributor can be deprecated, rewritten, or refactored at any point.

When experimenting with the DH-Index, some deficiencies came to light, which prompted us to create a variation. For example, we noticed that methods such as getters and setters could disproportionately inflate a developer's ranking. Since these methods tend to be called more frequently than other methods, their inclusion without any adjustment can create a bias. In the academic version, H-Index, citations are the single contribution factor, but this does not need to be the case in DH-Index: more nuanced and fair variations are possible by incorporating code length into the index through LOC.

We use LOCs to adjust the DH-Index. We first multiply method usage with LOC to generate a new count, the LOC-Weighted Usage (LWUsage). We add one to the usage before weighting to prevent a possible discontinuity when usage equals zero so that the volume of the contributed code in a method is counted even when the method is not called from somewhere else. This may happen, for example, in API-level methods that are not meant to be called anywhere else from the codebase but can still be important: the LOC weighting scheme thus allows such methods to be included. Finally, we adjust the result by the percentage

of the total contributed LOC attributed to the particular developer for whom the index is calculated. The resulting adjusted counts are given by the following equations:

$$LWUsage = \lceil \text{Method's LOC} \cdot (Usage + 1) \cdot \%ofContribution \rceil \tag{3.2}$$

where

$$\%ofContribution = \left(\frac{\text{Developer's LOC Contribution}}{\text{Total Method LOC Contribution By Contributors}} \right) \tag{3.3}$$

Then, the variation of the DH-Index, the LWDH-Index, can be defined based on LWUsage. LWDH-Index is calculated as the largest number h such that h methods have at least h LWUsage as shown below:

$$LWDH\text{-}Index\ (LWUsage) = \max\{h \in \mathbb{N} : LWUsage(h) \geq h\} \tag{3.4}$$

3.2 Implementation of DevScholar

We implemented a web application named DevScholar that analyzes software projects by extracting developers' contributions and calculating metrics for the developers and methods. The workflow of how DevScholar works is shown in Figure 3.1 and will be explained in detail in this chapter.

3.2.1 Data Collection

As the first step in data collection and as shown in Figure 3.2, the user is prompted to provide the repository link, the path to clone the repository to, and the git provider that has the project to be cloned, in which the tool supports cloning projects from Github, Bitbucket, and GitLab. Using the inputs provided, the project is cloned, and all Java file paths are extracted.

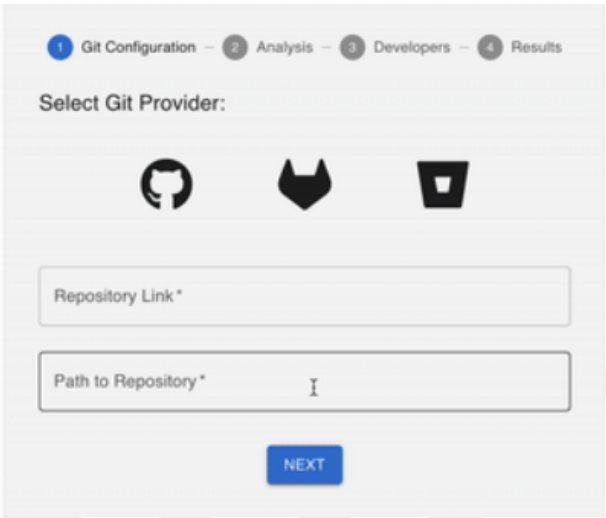

Figure 3.2: User Input for Project to be Analyzed

3.2.2 Method Metadata Extraction using AST Parser

In the next step, using Java's Abstract Syntax Tree (AST) parser [2], the metadata of the methods, including the starting and ending lines and the method's signature, are extracted.

3.2.3 Call-Graph Generation

In parallel, using a call-graph generation tool for each method, we obtain all its references throughout the project. The tool generates a graph where the nodes correspond to the methods and the edges indicate the caller-callee relationships between the methods. This graph is extracted from the call information provided by tracing the AST of each file [5].

[2]https://github.com/javaparser/javaparser

3.2.4 Developer Metadata Extraction using CodeShovel

Next, for each method, we use CodeShovel [6] to extract the related method's commit history. These commits include changes to the method, such as to its signature, body, name, location, or the name of the file containing the method. The commit data is composed of the developer's name and email, the date of the commit, and the changes made.

3.2.5 Processing of Developer and Method Metadata

Using the data extracted in the previous steps, we first process the method's data. This includes calculating the method's LOC (Current LOC), number of references (Usage), and LWUsage. We then remove the bots from the developer's list if the developer's email or username contains '[bot]'. Then, we map the developers to the methods to which they have contributed. In addition, when mapping, we merge authors using different commit identities into a single developer if a similarity is detected using the fuzzy string matching algorithm. The merging step is a semiautomatic process, where we take the input from the user to confirm whether or not the similar detected authors are the same author, as shown in Figure 3.3. A sample result of this step is shown in Tables 3.2 and 3.3. Dev / Total (Contributed LOC) represents all the additions and deletions made by the method contributor, including overwritten lines divided by all the contributions to the method. The additions or deletions of empty lines or comments are excluded from the contribution calculation.

Table 3.2: Methods to which the Developer X has Contributed

Method Name	**Usage**	**LWUsage**	**Current LOC**	**% Contribution**	**Dev/Total (Contributed LOC)**
assignAndNotify()	5	36	10	60%	12 / 20
sendMessage()	2	9	15	20%	6 / 30
getUser()	5	12	5	40%	4 / 10

Figure 3.3: User Step for Merging Developers

Table 3.3: Methods to which the Developer Y has Contributed

Method Name	**Usage**	**LWUsage**	**Current LOC**	**% Contributed**	**Dev/Total (Contributed LOC)**
assignAndNotify()	5	24	10	40%	8 / 20
sendMessage()	2	36	15	80%	24 / 30
deleteMessage()	1	20	10	100%	20 / 20

Table 3.4: Developers Indexes Calculation Result

Developer	**LWDH-Index**	**DH-Index**	**Total Contributed LOC**
X	3	2	22
Y	3	2	52

3.2.6 Developer Contribution Analysis

After the data is processed, we calculate the metrics for each developer. The calculated metrics include the LWDH-Index, DH-Index, and Total Contributed LOC, as shown in Table 3.4. The total Contributed LOC is calculated as the total number of LOCs that the developer has contributed to the methods.

3.2.7 Present the Data in DevScholar

Finally, these processed data are displayed to the user. In the application, the user can select a developer to see their processed data. Developer data, such as their associated names and emails, the number of methods attributed to them, co-developers, the calculated indexes, and the methods attributed to them, are shown in Figure 3.4. On the same page, shown in Figure 3.5, the top developers of the project, with their calculated indexes, are shown in an ordered manner based on the LWDH-Index by default. The developers can be sorted according to any of the other metrics. After clicking on a method, the method's developers and the number of LOCs they contributed can be viewed as shown in Figure 3.6. With these features of our application, we present a developer contribution analysis and provide a user-friendly interface to navigate through the work of developers and the developers of methods in the history of the project.

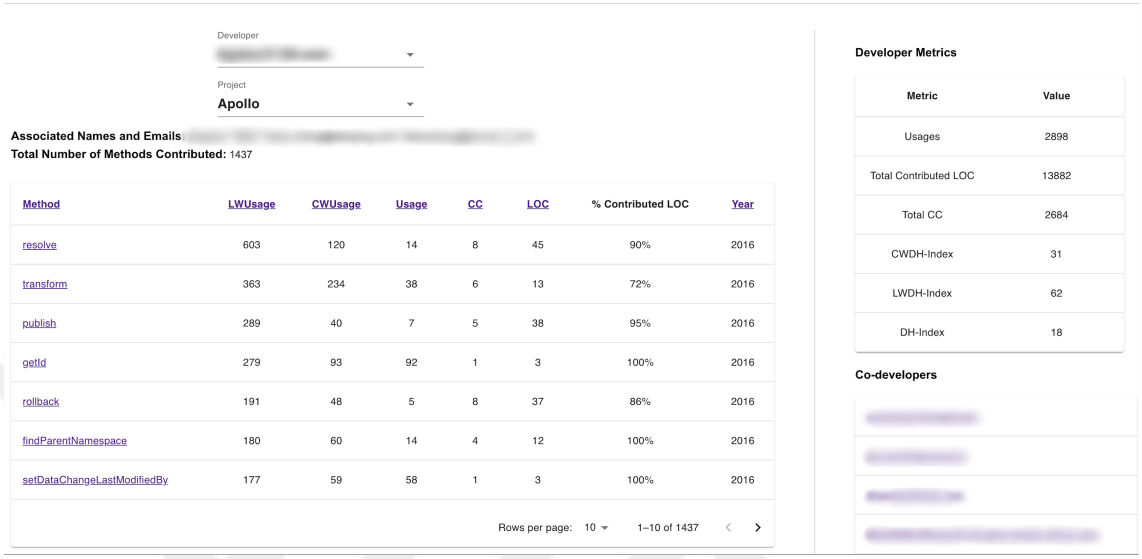

Figure 3.4: Developer's Calculated Metrics

Top Developers

#	Developer	LWDH-Index	CWDH-Index	DH-Index	Total CC	Total Contributed LOC
1		62	31	18	2684	13882
2		48	33	13	2405	11624
3		32	15	7	656	2541
4		28	19	8	865	1992
5		24	17	10	427	2674
6		18	9	4	162	691
7		17	9	4	223	751

Figure 3.5: Top Project's Developers

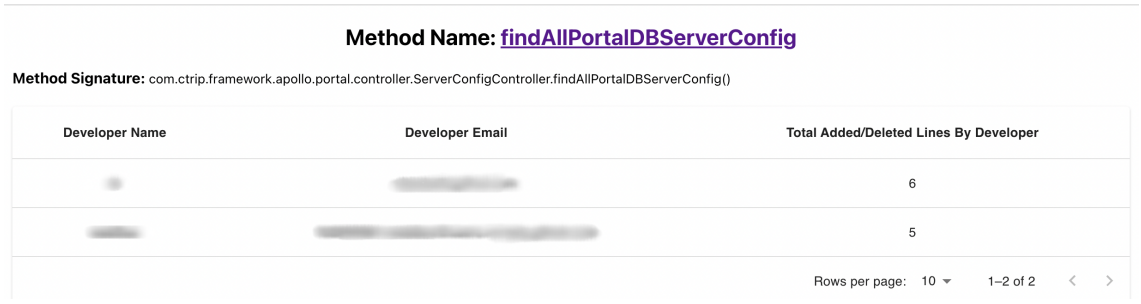

Figure 3.6: Developer's Contribution Data to Method

Chapter 4

Validation

We started our validation approach in the absence of a definitive ground truth, which prevented us from using conventional validation techniques. Validating the developer contribution metrics that embrace a method that deviates from conventional norms has been a point of contention [7].

Therefore, we conducted a comparative case analysis by contrasting the capabilities and top developers identified by GitHub Insights with those identified by our tool for four OSS projects but due to the similarity of the results we present two of them in this thesis. Our goal was to generate illustrative examples to highlight the differences and discuss the advantages and disadvantages. We extended our validation with a user study conducted with an international airline company, Company A, which has used DevScholar in its own settings and ran it on three of its internal projects.

In the following sections of this chapter, we discuss the selection of sample projects, provide a comparison with GitHub Insights, provide comparison of features between DevScholar and Github Insights, and present the process and results of our user study.

4.1 Selection of OSS Projects

We first determined relevant criteria to select the projects for our analysis. The first criterion was the programming language, where we focused on Java, as DevScholar relies on CodeShovel, which is Java-based. We aimed to include projects with a substantial history of changes to ensure rich data on method modifications. Thus, our second and third criteria required projects to have (>2000) commits and (>1000) pull requests. The fourth criterion was a contributor count (>100), as a larger, more diverse contributor base increases the likelihood of encountering edge cases for further analysis. Lastly, we assessed project popularity, selecting those with (>25000) stars on GitHub. Using these criteria, we employed the GitHub search tool by Dabic et al. [8] to filter and select suitable projects.

There were 64 projects that fit the criteria we mentioned above, and from those projects, for space purposes, we randomly chose four projects as shown in Table 4.1, Apollo, Spring Boot, Retrofit, and Dubbo, and ran DevScholar on them.

Table 4.1: Selected Projects

Project	**Stars**	**Contributors**	**Commits**	**Pull Requests**
Apollo	29.3k	162	2925	1734
Spring Boot	76k	1137	52691	6656
Retrofit	43.3k	160	2453	1459
Dubbo	40.7k	592	8291	7673

Due to the similarity of the results of the analyzed projects, only the results of two OSS projects, Apollo and Spring Boot, are discussed. Data extraction was completed on 8 October 2023. The results of those two projects are discussed in the following sections of the chapter.

4.2 Analysis Time for the OSS Projects

Table 4.2 presents the analysis time required for four open-source software (OSS) projects, broken down into three sequential steps: method extraction, call graph generation, and CodeShovel analysis. The **AST Parser** column represents the time taken to extract methods from source code using abstract syntax tree (AST) parsing. The **Call Graph** column shows the duration required to generate the call graph for each project, which captures method invocation relationships. Finally, the **CodeShovel** column indicates the time spent analyzing method histories using the CodeShovel tool. As seen in the table, larger projects like Spring Boot and Dubbo require significantly more time, especially during the CodeShovel phase, where Spring Boot's analysis took approximately 26.4 hours. In contrast, smaller projects like Retrofit and Apollo complete all steps considerably faster due to their smaller codebases, as can be seen by the **Number of Methods** column. After all the steps are done and the data is ready, the indexes are calculated, which takes a few seconds, and are nearly the same for all the projects.

Although the analysis process can take up to one or two days for large projects, this is not a significant limitation in practice. DevScholar is not intended to run with every single code change but rather periodically, such as every few days or at the end of each sprint. This periodic execution is sufficient to provide meaningful insights without burdening the development workflow. As a result, the processing time remains manageable and well-aligned with common agile practices.

Table 4.2: Analysis Time for OSS Projects

Project	**Number of Methods**	**AST Parser**	**Call Graph**	**CodeShovel**
Apollo	3933	15.4 Seconds	50.9 Seconds	3.2 Hours
Spring Boot	34421	86.4 Seconds	1205.1 Seconds	26.4 Hours
Retrofit	2523	6.4 Seconds	19.9 Seconds	1.9 Hours
Dubbo	28037	68.1 Seconds	1075.3 Seconds	21.5 Hours

4.3 Comparison of DevScholar Results with GitHub's Insights (Number of Commits)

In Table 4.3, the GitHub Insights ranking of developers, their total commits, the DH-Index ranking and value, the LWDH-Index ranking, the value, and the total contributed LOC and methods of developers are given. Looking at the GitHub Insights and the DevScholar results of the Apollo project in Table 4.3, we find that the rankings are similar but display several notable differences. For example, although dev1 has the highest number of commits, it comes second in both the DH-Index and LWDH-Index rankings. This is an interesting case, as the commits dev1 has made are four times dev2's commits, so dev1 could be expected to be ranked first in all metrics. We can also see from DevScholar that dev2 has contributed to 1437 methods with a total LOC of 13882, while dev1 has contributed to 1146 methods with a total LOC of 11624. This shows that the number of commits of a developer is not necessarily correlated with a developer's contribution in terms of the number of code abstractions, which we consider to be more important than the number of commits. Commit numbers can be artificially inflated by adding empty lines or comments, and the committed lines may not necessarily represent valuable code abstractions (they could be trivial or part of dead code). The usage of the contributed lines is not measured in GitHub Insights, which makes it impossible to assess the centrality of the contributed work. DevScholar metrics include usage as a proxy of contributions' impact or importance, which puts dev2 above dev1 in Table 4.3. Also, DevScholar disregards empty lines and comments, avoiding their potentially disproportionate inflationary effect.

Also, we notice that dev7, dev9, and dev10 are not in the top 10 developers in the DevScholar ranking system. dev7 is ranked 34^{th} in the DH-Index and 32^{nd} in the LWDH-Index ranking. The differences between these rankings are much higher. When we go through dev7's contributed methods, we see that dev7 has contributed to one method with a usage of four, while the usage of other methods is zero. We can argue that dev7's method contribution is minimal

Table 4.3: Comparison of GitHub Insights, DH-Index and LWDH-Index Ranking of Developers for Apollo Project

Github Insights		**DH-Index**		**LWDH-Index**			
Ranking	**No of Commits**	**Ranking**	**Value**	**Ranking**	**Value**	**Total LOC**	**Methods**
dev1	1140	dev2	18	dev2	62	13882	1437
dev2	252	dev1	13	dev1	48	11624	1146
dev3	135	dev3	10	dev5	32	2541	341
dev4	47	dev4	8	dev4	28	1992	316
dev5	47	dev5	7	dev3	24	2674	228
dev6	37	dev8	7	dev11	18	691	94
dev7	33	dev6	5	dev12	17	751	99
dev8	30	dev11	4	dev8	16	1015	252
dev9	24	dev12	4	dev14	13	235	27
dev10	12	dev13	4	dev6	12	425	75

concerning their usage in the code, so dev7 does not appear in DevScholar's top 10 listing. In addition, dev9 is 19^{th} in the DH-Index and 12^{th} in the LWDH-Index ranking, while dev10 is 29^{th} in the DH-Index and also 22^{nd} in the LWDH-Index ranking. Neither developer is in the top 10 in DevScholar, although they are in GitHub Insights. This difference again stems from the same phenomenon: both dev9 and dev10's contributed methods have few or zero usages.

Our last observation is about dev1: the most used method of dev1 is a getter method with a usage of 71 and LOC of three. The seventh most used method is not a getter/setter method; its usage is 15, and LOC is 13. The LWUsages of these methods are 124 and 208, respectively. While getter/setter methods are easier to write and tend to have higher usage, their significant impact on rankings is questionable. LWDH-Index compensates for the effect trivial methods can have on the rankings by equalizing long methods' contributions relative to those of short methods: this can, in turn, prevent many short methods with plenty of usage from assigning higher scores to one developer and fewer methods with significant content but less usage from assigning lower scores to another developer.

Another difference that is noticed is that our tool provides the data of 69 developers, while the number of contributors to that project, according to GitHub, is 148 developers. Upon investigation, we found that developers ranked from 55 onwards had made only one commit, with contributions that either did not involve methods, were limited to Java files, or were merely whitespace and comment

changes.

When we look at the GitHub Insights and DevScholar rankings of the Spring Boot project in Table 4.4, we see that dev1 is ranked first in GitHub Insights, but dev3, who committed less than half of dev1, is ranked first in both our DH-Index and LWDH-Index rankings. We can see that dev3 contributed to 16157 methods with a total LOC of 119091. On the other hand, dev1 contributed to 14429 methods with a total LOC of 86936. We can state that in terms of methods and contributed LOC, dev3 contributed more to the project, and the method usage was higher compared to dev1, but dev3's recognition of the project is less in GitHub Insights.

Table 4.4: Comparison of GitHub Insights, DH-Index, and LWDH-Index Ranking of Developers for Spring Boot

Github Insights		**DH-Index**		**LWDH-Index**			
Ranking	**No of Commits**	**Ranking**	**Value**	**Ranking**	**Value**	**Total LOC**	**Methods**
dev1	9502	dev3	15	dev3	45	119091	16157
dev2	8502	dev1	13	dev1	45	86936	14429
dev3	3918	dev2	11	dev2	32	50397	8832
dev4	919	dev4	10	dev4	27	21822	3670
dev5	713	dev10	8	dev9	24	17845	2265
dev6	573	dev8	6	dev8	20	10505	1783
dev7	544	dev7	6	dev10	18	14736	1879
dev8	452	dev6	5	dev6	18	9657	1574
dev9	385	dev5	5	dev11	15	2286	391
dev10	239	dev9	4	dev12	13	703	122

In addition, there is a substantial difference in the ranking of dev5. In GitHub Insights, dev5 is ranked fifth, but in the DH-Index ranking, they are ranked ninth. The LWDH-Index ranks dev5 as 13^{th}, which is not even in the top 10. When we sort dev5's methods according to their usages, we see that the only top 66 methods have usages, and the remaining 1301 contributed methods have zero usage, as the majority of them are test methods. Compared to other developers, this contribution to the core of the project is less, and thus this developer is not ranked in the top 10 in both of our metrics.

Furthermore, when we compare the results of the DH-Index and LWDH-Index, we can see that dev9 has more total contributed LOC and a higher LWDH-Index value but a lower DH-Index value than dev8. Although the methods to which

dev8 contributed have more usages and, thus, a bigger DH-Index value, dev9 has contributed with more LOC and is ranked higher based on the LWDH-Index value. This illustrates how the DH-Index focuses solely on method usage, while the LWDH-Index also considers the LOC of these methods.

Similar reasoning to the Apollo project, it is noticed that our tool provides the data of 562 developers, while the number of contributors to that project, according to GitHub, is 1061 developers. Moreover, GitHub Insights's ranking shows only the top 100 developers.

An analysis comparing our indexes with GitHub Insights reveals a strong correlation between the two. In all the open-source projects we examined, the top three developers identified by our indexes match those highlighted by GitHub Insights. These developers have contributed the highest number of lines of code (LOC) to the codebase, according to GitHub Insights, and the largest number of methods, as determined by DevScholar. Additionally, they are also the oldest contributors to their projects. For instance, in the case of the Spring Boot project, the top three contributors—referred to here as Dev 1, Dev 2, and Dev 3—have been active since 2013, whereas the remaining contributors began participating approximately three years later.

4.4 Feature Comparison of GitHub Insights with DevScholar

Since GitHub is one of the most used software development platforms, we provide a general comparison of our tool to GitHub's Insight page in terms of functionality (how developer-method relations are presented) and ranking approach. This comparison is given in Table 4.5.

On the GitHub Insights page, users must click on a developer's commits and manually browse through the changed files to identify the contributed methods.

Table 4.5: Feature Comparison of GitHub Insights with DevScholar

Criteria	**GitHub Insights**	**DevScholar**
Contributed Methods	Traverse the developer's commits to see the contributed methods.	Click on the developer's name to see his/her contributed methods.
Method's Developers	Have to traverse the file's commits.	Displayed.
Ranking Developers	According to the number of commits, added or deleted lines.	According to the DH-Index and LWDH-Index.
Developer Collaboration	Not displayed.	Displayed.

In DevScholar, the user can display to which methods the developer has contributed by clicking on their username. Additionally, clicking on the method name directs the user to the beginning line of that method on the project's GitHub page.

Moreover, identifying a method's developers on GitHub can be problematic. For each source code line, GitHub only shows the last developer's name that changed that line. The user is forced to traverse the file's previous commits to determine the other developers who have worked on the method. However, DevScholar readily displays all developers' names who touched that method in the method's entire commit history.

DevScholar thus requires much less effort and digital forensics on the part of the user to identify a developer's contributions and a method's contributors than is readily accessible through GitHub Insights.

DevScholar provides a richer insight into developers from multiple perspectives. It is also important to mention that DevScholar calculates the number of contributed lines as the sum of added and deleted lines for a method. DevScholar excludes empty lines and comments, even when they occur in a method, as it values only the functional part of the code. In contrast, GitHub Insights counts all added or deleted lines, including those inside methods, comments, or empty

lines. In the case of developer contributions, GitHub does not show which developers contributed to the same methods, whereas in DevScholar, this information is displayed.

4.5 User Study

Table 4.6: Key Metrics of Company A's Projects Analyzed by DevScholar

	Project 1	**Project 2**	**Project 3**
No of Developers	25	33	36
No of Methods	2183	29158	58463
Project Age (In Months)	24	6	36
Programming Language	Java	Java	Java

To evaluate the tool and the proposed DH-Index and LWDH-Index metrics, we conducted a user study. This study aimed to assess the tool's usefulness and practical utility in a real-world setting and gather feedback for further refinement. We used convenience sampling and recruited volunteers via LinkedIn, presented and demonstrated the tool's features, and provided a trial period during which volunteers independently tested the tool on their internal projects. We concluded the study with a qualitative survey to capture detailed insights. Our user study process included the following steps:

4.5.1 Volunteer Recruitment

To solicit volunteers, we shared a post on LinkedIn[1] outlining the tool's features and inviting volunteers to test it and help validate its effectiveness and results. From the interested respondents, we selected one company to participate in our study. The study was conducted in partnership with an international airline company, Company A, involving three of their internal projects developed by a large-scale software development team. Those three projects are used for their airline check-in system.

[1]https://www.linkedin.com/

4.5.2 Introducing DevScholar

Our first meeting with Company A was to clarify our objectives and present the underlying metrics and an explanation of our tool, DevScholar. The meeting took about one hour and the participants included two technical directors, two developer leads from Company A, and two of our researchers. After our presentation, one of the directors stated that "*Quantifying and evaluating the quality of the developer's contributions is very essential for our company*". They have also mentioned that they previously did not use any tools to measure the contributions of the developers based on code reusability.

4.5.3 Deploying DevScholar and Demonstrating the Features

One week later, we conducted another meeting with two of their developer leads that took two hours. The purpose of the meeting was to demonstrate the tool's features in a more detailed way and set up and deploy the tool on their private servers. With the help of two of our researchers, the tool was deployed to the company server. After that, the tool was run on Project 1, shown in Table 4.6. The aim was to pilot the tool with a relatively small project for testing, demo, and training purposes.

4.5.4 Trial Period

After the previous step, Company A experimented with our tool for two weeks, by running the tool on two larger projects, Project 2 and Project 3 shown in Table 4.6. During this time, the DH-Index and LWDH-Index metrics were computed to evaluate their meaningfulness in measuring developer contributions in the company's context.

4.5.5 Feedback Session

Following the trial period, a one-hour follow-up feedback session was conducted with two of our researchers and two developer leads from the company. This session was for receiving feedback to improve the tool, where we also noticed that there was some misunderstanding regarding some of the concepts such as the co-developer concept, which needed clarification. One of the developer leads stated, "*As a lead, I would like to filter the performance of the developers based on time, i.e. last three months, as there are some developers who have left the company but as the tool analyzes the whole history they are still shown in the list*". Therefore, their feedback included requests for new features, such as time-based filtering (e.g., monthly), graphs to track changes over time, and minor UI improvements. The managers emphasized the clarity and granularity of the insights provided by the DH-Index and LWDH-Index. In particular, they appreciated the ability to see not just the volume of contributions but also the reusability and significance of these contributions.

4.5.6 DevScholar Improvement Period

Next, we reviewed Company A's feedback, implemented the requested features such as time filtering, improved the UI by implementing new interfaces for user inputs and for showing the candidate developers that are similar and might need to be merged, and enhanced performance, reducing analysis time from seven to five hours on average for the two large projects.

4.5.7 Final Session

We concluded the process with a final one-hour session with two of the developer leads and two of our researchers. The session was divided into two parts: (a) a demo showcasing the improvements made based on the feedback provided earlier and (b) a qualitative survey [9]. The qualitative survey had the semi-structured

interview format: the researchers asked both pre-prepared, structured survey-type questions and open-ended questions that allowed the session to proceed in an organic manner.

4.5.7.1 Improvements Demonstration

The session began with a demonstration of the updated tool, which featured a redesigned user interface for inputting repository data, improved functionality for merging developers with similar names and emails, time-based filtering, and other minor UI enhancements. Following the demo, the company participants gave additional feedback aimed at further improving the usability. Some of their feature requests included integrating the tool with version control systems for multi-repository analysis, displaying the contribution percentages for co-developed methods, and providing more detailed tool tips to clarify each metric. They also noted that for managers to use the tool regularly, it would be advantageous to integrate it with Version Control System (VCS) to automatically fetch all repositories, allowing users to select which ones to analyze, instead of analyzing each repository one by one.

4.5.7.2 Qualitative Survey

The interviewees were the two developer leads, Developer Lead 1 and Developer Lead 2, who had used our tool during the previous steps and analyzed its features and results. The interviewers were two of our researchers who had been involved in the user study process. The discussion and answers were transcribed into interview notes during the session. To collect additional data and in addition to our questions, we also integrated the System Usability Scale (SUS) questions into the session. Developed by Brooke [10], SUS consists of 10 questions using a 5-point Likert scale ranging from Strongly Disagree to Strongly Agree. Each response is assigned a numerical value, and the total score is calculated by summing these values. This total is then multiplied by 2.5 to produce a score between 0 and 100. According to research, a SUS score above 68 is considered above average, while

a score below 68 is considered below average [10].

We first took 20 minutes to solicit the answers, and later discussed the answers in an open format, and explored further insights. The first question we asked was about how useful and meaningful the metrics, LWDH-Index and DH-Index, were in assessing the developer's contributions. Developer Lead 1 stated that "*The relevance of traditional metrics' values like LOC can vary, therefore the new metrics provided by DevScholar as LWDH-Index and DH-Index that take into consideration of the usage of the methods provide a valuable and better perspective on developers' contributions*". In addition, Developer Lead 2 mentioned that "*Metrics like usage of the method shows the reusability of the implemented methods that indicates a higher quality of the written code*". This indicates that the DH-Index and LWDH-Index metrics offer a more nuanced view aligned with the project's goal of fostering reusable, high-quality code. One of our questions was to evaluate the alignment of the DevScholar information and rankings with the technical leads' subjective opinions of the developers. Both of the developer leads have stated that although it is hard to provide a static and definitive ranking of the developers, they agreed with the DevScholar results by providing a score of four out of five, with a score of five corresponding to "completely aligned". Evaluating the answers to the SUS questions, the tool received an average SUS score of 79, which suggests the usability of the tool is above average.

In addition, the technical director was interviewed separately for an executive perspective. We asked him the following question "*What are the most important criteria you consider when you evaluate the developers?*". The technical director stated "*As our developers work remotely, they can sometimes be distracted by other tasks, which impacts both their productivity and code quality. That's why having a tool that evaluates them from multiple perspectives, beyond just lines of code, is essential.*" He also added, with respect to DevScholar in particular, "*The metrics provided by DevScholar allow us not only to assess code quality, especially in terms of reusability, but also to rank developers. Although it's challenging to perfectly rank developers due to various perspectives, by identifying outliers and observing the best and worst-ranking developers, we can gain valuable insights*

into each developer's performance." The feedback suggests the value and importance of our metrics in giving an idea and evaluating developer contributions according to reusability and according to reusability combined with the volume of contributions.

Chapter 5

Discussion

Understanding and evaluating developer contributions accurately is a critical aspect of software project management. Traditional metrics, such as the number of commits or lines of code (LOC), offer limited insight due to their simplistic nature and susceptibility to manipulation. In the following sections, we assess the quality and robustness of our proposed LWDH-Index in comparison to conventional metrics. First, we utilize the Kaner-Bond framework to systematically compare the LWDH-Index with the number of commits across multiple dimensions of metric evaluation in Section 5.1. Then, we discuss the broader implications of using contribution metrics, highlighting both the potential benefits and the risks of misinterpretation or misuse in section 5.2.

5.1 Comparison Based on Kaner-Bond Framework

While suggesting new metrics, it is important to assess their quality of measuring the intended attribute. Kaner and Bond [11] proposed an evaluation framework for software engineering metrics. We used this framework to compare the LWDH index alongside the number of commits.

The evaluation framework consists of 10 questions, each targeting a different aspect of the metric [11]:

1. What is the purpose of this measure?
2. What is the scope of this measure?
3. What attribute are we trying to measure?
4. What is the natural scale of the attribute we are trying to measure?
5. What is the natural variability of the attribute?
6. What is the metric (the function that assigns a value to the attribute)? What measuring instrument do we use to perform the measurement?
7. What is the natural scale for this metric?
8. What is the natural variability of readings from this instrument?
9. What is the relationship of the attribute to the metric value?
10. What are the natural and foreseeable side effects of using this instrument?

As both metrics (LWDH-Index and the number of commits) share the same measured attribute (developer contribution), seven framework questions result in similar answers (questions 1, 2, 3, 4, 5, 7 & 8). Therefore, we will focus on the remaining three questions to compare the LWDH-Index with the number of commits.

The sixth question in the framework delves into the function that assigns a value to the measured attribute [11]. On the one hand, for the number of commits, the metric function is simply counting. On the other hand, the LWDH-Index first calculates a score based on LOC and method usage and then uses these scores to calculate an H-Index notion metric. The score calculation step in the LWDH-Index is an effort to account for productivity and contribution within the project. Though the scores can deviate from method to method, the

calculated index points to a cumulative contribution that evens out the effect of single methods.

The ninth question in the framework digs into the attribute-metric relationship, examining how the metric is affected by the changes in the attribute [11]. The relationship for the number of commits is linear; it increases with each commit. This linearity can cause concern as commits rarely contribute equally to the project. For the LWDH-Index, the relationship is not linear. By definition, the LWDH-Index only increases if a new method reaches an LWUsage score larger than the index value. This means that it becomes harder to increase the index at each increment.

The tenth question in the framework is concerned with the natural and foreseeable side effects of the metric [11]. When a metric is used, it can lead to an inevitable behavior change. Such a change could cause more harm than the intended benefit. A foreseeable side effect of adopting any metric to quantify developer contribution is shifting the focus of developers to the metric, eventually harming the development. This phenomenon is known as Campbell's Law, which states that when quantifying metrics are used for social decision-making, they are likely to corrupt the process they intend to monitor [12]. Adopting the number of commits as a contribution metric might potentially encourage developers to split their work into several commits. This simple act allows developers to increase their metrics without changing their contribution. LWDH-Index can introduce corruptions similar to the process, while it is harder to manipulate due to the calculation of the index.

5.2 Implications of Contribution Metrics

Associating developer contributions solely with the number of commits or LOC can be deceptive, as they lack standard formats. A commit might contain a single minor change or hundreds, while LOC could represent a simple refactor or a complex lambda function. As the developer leads noted in our user study, these

traditional metrics vary, highlighting the need for alternative perspectives. To address these limitations, the LWDH-Index applies the H-Index formula to a score called LWUsage, which quantifies each method's reusability based on its usage throughout the software and the LOC contributed. Method usage reflects its importance across the codebase, while LOC indicates the contribution's volume. LWUsage scores are then distributed to developers according to their contribution percentage for each method. Despite this, both LOC and method usage have their own biases—simple methods like getters and setters tend to be heavily used, and high-LOC methods may have limited project-wide use.

We noticed that once the ranking is performed according to LWUsage, the rank of straightforward methods goes significantly lower, as they often have a low number of LOCs. The same is true when we compare it to the LOC-based ranking. The rank of methods with many LOCs decreases if they have limited usage. However, the LWUsage metric may not always accurately reflect the contribution of developers in certain scenarios, such as when a method is called infrequently or when multiple developers have contributed to the same codebase. For that, we rely on the calculation logic of the H-Index, which can even out the impact of outliers.

Overall, the LWDH-Index is less susceptible to manipulation through LOC inflation or excessive self-usage of a developer's own methods. This is due to the methodology of the H-Index calculation. The index measures a distributed contribution rather than the cumulative contribution of single methods. Similar concerns were expressed for H-Index. Although excessive self-citations can increase the H-Index by one or two, the metric is robust when the H-Index difference is large [13].

Academic indexes might incentivize authors to do self-citations and publish many papers. The H-Index calculation process mostly evens out the effect of such metric-manipulating behaviors. Yet still, researchers try to publish frequently as they do not know which publication will be cited more. Similar metric-manipulating efforts could affect the software production process as well.

Ranking developers can also cause problems. When people realize that their performance is judged based on a metric, they might stress about the metric and neglect the value they bring to their work. Software development is not immune to such perverse incentive effects, also known as the Cobra effect [14]. Employment termination decisions based on metrics are also problematic [7]. As the technical director from Company A has said, it is difficult to have an accurate ranking as this ranking would change according to the perspective; however, looking at the outliers of the rank provides an idea about the developer's contributions.

We explored different usage opportunities for our metric. One potential use case for our metrics is at both the level of developers and the level of methods. At the developer level, outlier values in the LWDH-Index scores may indicate that those developers lower the truck factor. The truck factor is the smallest number of critical team members whose absence would significantly impact the project's continuity. As other developers are less aware of the conducted work, it is fairly hard to compensate for the loss of information when a key developer leaves. As a precaution, developers with higher LWDH-Index scores could be diverted to pair programming or documentation to mitigate such information losses.

At the method level, it is natural to assume that a high-usage method is critical for the project. Method calls are associated with the ripple effect phenomenon, where changes in a particular part of the software can cause unintended changes in others [15]. Additionally, a method with high LOC is likely to have a long-method code smell, one of the most frequently encountered code smells [16]. Therefore, LWUsage scores could be used to prioritize methods in reviews, refactoring, and documentation.

Overall, it is essential to interpret developer contributions cautiously, given the inherent complexity of software development. In our user study, the technical director highlighted that with the rise of remote work, developers may face distractions, such as freelance commitments outside their primary roles, which can impact two key criteria: code quality and productivity. Engaging with DevScholar interactively can enhance the project by providing a broader perspective on the contributions made by various developers.

Chapter 6

Threats to Validity

6.1 Construct Validity

Our research aims to evaluate the developer's contribution in terms of reusability, which is an intangible attribute. Our metrics are based on the premise that a highly contributing developer will be responsible for methods of significant importance. Such a direct relation between contribution and method reusability can only be valid for developers who write code. Other contributions, such as documentation, system design, requirement definition, etc., are not involved in our metrics.

As a threat to construct validity, it could be argued that a developer can have more dimensions of contributions, and our metric does not consider them. A developer can make a verbal contribution by leading the software team toward a goal. Additionally, a developer can contribute to software with different artifacts such as PDF, JSON, and README. In random experimentation, Jergensen et al. [17] found that two developers had fewer contributions, and another did not contribute to the source code of six developers. Furthermore, our metrics focus on method definitions. While methods are essential code components, other coding elements, such as classes, global variable definitions, comments, etc., also

contribute to the software. It can be argued that other coding elements, such as comments, possess descriptive information rather than impactful work.

6.2 Internal Validity

A threat to validity that concerns our methodology for measuring the impact of a method is that if a method is referenced many times by other methods that are referenced many times, that method is important. However, this approach could lead to false negatives where a method of high importance has only a limited number of method calls. Additionally, it could lead to false positives where simple methods used frequently will be credited as essential, such as getters and setters. However, we mitigate this effect by considering both LOC and usage and calculating the LWDH-Index metric.

In addition, the call graph tool we have employed regards the nodes as the method signatures, and directed unweighted edges represent the existence of a method call. Therefore, if there are multiple method calls between the same nodes, they are only counted as one. Although the call graph tool can accurately calculate the number of calls within and across classes, not generating a weighted edge between nodes might result in calculating the number of method calls that a method has less than it should be. High coupling in software projects, where components are excessively interconnected, and low cohesion where the components are not closely related, might have a negative impact on maintainability and scalability.

Another threat is that prioritizing a high number of method calls could unintentionally encourage high coupling by increasing inter-module dependencies. Moreover, rewarding methods with more LOC can result in low cohesion, as the methods may perform multiple unrelated tasks to increase the number of LOC rather than focusing on a single cohesive purpose.

6.3 External Validity

Another area of validity concern can be the generalizability of the results due to the limited dataset, where only four open-source projects were used and three projects were used in the user study.

Decisions based on our metrics within a software project should be taken with caution, as developers with low rankings may significantly impact the project through means other than coding.

Unlike OSS projects, in software industrial environments, tasks are assigned to the developer by developer leads for instance, which can affect the results and contributions.

6.4 Conclusion Validity

In addition, our data collection methodology exposes another threat. Using CodeShovel [6], we get the commit history of the methods. While IntelliJ and git log -L are tools used for change history, it is known that they cannot track inter-file method moves, and git log -L also cannot track file renames. Since these refactorings are common in software development, we chose to use CodeShovel because it gives a more accurate history. Grund et al. [6] state that CodeShovel can return complete and accurate histories for 90% of the methods and 97% of the method changes. As we use the CodeShovel results, the inaccuracies of the data collection affect our results as well. In addition, data collection accuracy can also be impacted by factors like commit squashing and developers using different names or emails for commits. In commit squashing, multiple commits are combined into a single one, which may include contributions from different developers, resulting in data loss.

Additionally, while we attempt to merge contributions from developers with

similar names or emails, some developers commit under randomly generated identifiers, making it nearly impossible to link them back to the correct individual, thus losing track of those contributions. Although we exclude commits involving only empty lines or comments, minor changes like indentation formatting are harder to filter out and may affect the developer's contribution count.

Chapter 7

Related Work

In this chapter, we examine studies that apply the H-index concept to software development, followed by research centered on identifying key developers and evaluating their contributions. Assessing developer contributions or identifying key developers and critical code components in a software project has been a long-standing area of research. We have categorized the related papers into sections: H-Index In Software Development, Identifying Key Developers, and Assessing Developer Contributions. The main differences between the work done in this thesis and the related work is presented in the Comparison Against Existing Methodologies section.

7.1 H-Index in Software Development

Capiluppi et al. [3], to our knowledge, was the first to apply the concept of H-Index in software development. They formulated three different H-Index-based indices to evaluate the developer contributions over multiple OSS project, as discussed in the introduction. However, our aim throughout this thesis is to quantify and evaluate the contributions within a single project.

Ding et al. [18] proposed four H-Index-based metrics to identify key classes in

a weighted class network of software such that the nodes represent the classes, the edges imply the dependency between classes, and the weights are given based on the number of dependencies between classes. While this study focuses on identifying key classes, the thesis focuses on the developers themselves and their contributions.

7.2 Identifying Key Developers

Orrei et al. [7] identified key developers in software projects using the *Pony Factor* metric, which defines core contributors as those responsible for more than 50% of total commits, calculated across parameters such as total changed lines, recent contributions, and file types. Similarly, Yamashita et al. [19] used commit-based, LOC-based, and access-based heuristics, identifying core developers as those responsible for 80% of total commits or LOC changes, or as developers with write access and contributions. Joblin et al. [20] took a relational approach, analyzing both frequency-based metrics (commit and LOC counts) and network-based metrics, like centrality and connectivity, to distinguish core and peripheral developers. Bock et al. [21] generated a method to identify core developers using events triggered in GitHub issues and pull requests. While these studies focus on identifying core developers, we aim to quantify and evaluate developer contributions in terms of code reusability. Çetin et al. [22], [23] identify key developers by constructing an artifact graph with nodes representing developers, change sets, files, and issues, and edges based on commits, reviews, issue links, and file inclusions. While these studies focus on identifying core developers, we aim to quantify and evaluate developer contributions in terms of code reusability.

7.3 Assessing Developer Contributions

Previous work on evaluating developer contributions has employed diverse metrics and methodologies. Ren et al. [24] developed DevRank, utilizing PageRank

based on method usage and applying NLP and ML techniques to commit messages to detect non-code contributions. De Bassi et al. [25] used software quality metrics, while Curry et al. [26] correlated development history metrics, highlighting First Authorship and Recency of Modification as key indicators. Other approaches, such as those by Nagwani and Verma [27] and Gousios et al. [28], assessed contributions using combined metrics like bug-fix frequency, comment volume, and repository activities. In contrast, our approach focuses solely on direct codebase contributions.

Lima et al. [29] assessed developer contributions using metrics such as lines of code (LOC), method complexity, and bug-fixing activities, finding that code contribution and complexity metrics were positively received, while bug-related metrics were viewed negatively by evaluators. Sun et al. [30] combined Abstract Syntax Tree (AST) differences, complexity metrics, and call graph changes to evaluate developer contributions based on commits, outperforming the LOC metric according to skilled programmer assessments. Another notable area of research is the *truck (bus) factor*. It is defined as the minimum number of people who have to leave the project to create serious problems. It is highly related to our research since one has to define a way of measuring developer contribution and knowledge to estimate the truck factor. Avelino et al. [31] presented an approach, using source files and commit history for developer contribution evaluation, to calculate the truck factor. During their validation with developers of 67 systems, 84% of the time, the developers agreed or partially agreed that the truck factor developers were the main authors of their systems. In contrast, the work in this thesis evaluates contributions by focusing on implemented methods, their usage, and the number of LOC each developer contributed to the method. Jabrayilzade et al. [32] broadened the metric by incorporating code-review activity and meeting participation, and an industry survey confirmed that engineers regard bus-factor tracking as critical for project health. Complementing these algorithmic advances, Klimov et al. [33] released Bus Factor Explorer, a web tool that computes the metric and offers interactive treemap visualisations and "what-if" simulations to pinpoint at-risk components. Haratian et al. [34] introduced BFSig, which weights files by their structural importance in the dependency

graph and cuts estimation error by up to 18%, while reducing false-negative risk cases. In contrast, this thesis evaluates contributions at the granularity of individual methods, tracking who implemented each method, how those methods are reused, and the exact lines of code each developer added, yielding a finer-grained perspective on knowledge concentration and bus-factor risk.

7.4 Comparison Against Existing Methodologies

The primary distinction of work done in this thesis from previous work is that we do not focus on bug reports, network relationships, documentation, recency, or file/project size-related factors. Rather than identifying key or core developers, we aim to quantify and rank developer contributions across multiple metrics, providing diverse perspectives in a multi-dimensional context. We prioritize information on all developers rather than limiting analysis to key developers, as this comprehensive approach offers insights into each developer's work. For code fragments (methods in our analysis), we calculate contribution percentages for each developer, allowing us to identify not only key developers for the project but also each method. Contributions are measured based on methods rather than entire files, and our LOC calculation excludes changed lines unless they are part of a method. While prior studies focus on core developer identification, none provide a tool similar to what DevScholar does.

However, GitHub Insights displays contribution metrics based on commits and added or deleted LOC. These metrics lack consideration of contribution quality, treating all commits and lines (including empty lines) equally. DevScholar, on the other hand, offers a more nuanced set of metrics that differentiate low- and high-quality contributions based on reusability.

Orrei et al. [7] introduced Pony Factor metrics to identify key developers. Since our LWDH-Index also considers line changes, we compared our results with their "Ponies onLines" and "Ponies onLines NoBinary" factors. In both the Apollo and Spring Boot projects, the key developers identified by these factors align with the

top developers in our LWDH-Index. However, differences in motivation, output, and consistency distinguish our approach. Orrei et al. [7] focus solely on identifying key developers without ranking others, while we evaluate all participating developers. Their output includes only key developer names, while DevScholar provides a detailed breakdown of each developer's LWDH-Index score, including method names and contribution details.

Additionally, we address name consistency issues that Orrei et al. [7] encountered, where the same developer appeared under different names or aliases. DevScholar merges contributions from the same individual, even if they use different names or emails, ensuring accurate rankings. Beyond identifying developer impact, DevScholar offers a complete overview of contributions, with options to visualize metrics linked to GitHub projects and specific methods, enabling in-depth exploration of developer contributions throughout the project.

Chapter 8

Conclusion

In this thesis, we introduced two indexes to evaluate the quality of the contribution of the developers from a reusability perspective. The first metric, called the DH-Index, applies the H-Index concept by treating method usage similarly to publication citations. Although beneficial, it has some limitations as a result of only considering usage to quantify method-level contributions. The second metric developed within this thesis is the LWDH-Index, which considers the LOC of a method, the usage of the method, and the percentage of the developer's contribution to the method. This metric can help identify key developers and code ownership, as well as it can indicate the need for pair programming opportunities and documentation to mitigate information loss and increase the truck factor.

We also implemented "DevScholar"[1], designed to calculate metrics and present results in a user-friendly interface. The replication package[2] contains the datasets, scripts, and documentation needed to reproduce our results and adapt the tool to new contexts. The tool integrates a variety of metrics at both the method and the developer level, facilitating project tracking for users.

In summary, our evaluation of DevScholar across both open-source and industry projects demonstrates its potential to offer a more nuanced and meaningful

[1] `https://devscholar.netlify.app/`
[2] `https://zenodo.org/records/15072359`

understanding of developer contributions. Unlike conventional metrics such as commit counts or lines of code—which often fail to capture the true impact or reusability of contributions—the LWDH-Index provides a balanced approach by accounting for both method complexity and usage across the codebase. Feedback from an international airline company further highlights the value of this perspective in real-world settings, where promoting sustainable and maintainable development practices is essential. Overall, our findings suggest that DevScholar, with its contribution-focused indexes, can complement existing tools by enabling more informed assessments of developer impact and fostering better engineering practices.

Bibliography

[1] "Github." https://github.com/. Accessed: 2023-07-01.

[2] J. Hirsch, "An index to quantify an individual's scientific research output," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, pp. 16569–72, 11 2005.

[3] A. Capiluppi, A. Serebrenik, and A. Youssef, "Developing an h-index for oss developers," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 251–254, 2012.

[4] L. Bornmann and H.-D. Daniel, "What do we know about the h index?," *Journal of the American Society for Information Science and technology*, vol. 58, no. 9, pp. 1381–1385, 2007.

[5] I. S. Göçmen, A. S. Cezayir, and E. Tüzün, "Enhanced code reviews using pull request based change impact analysis," *Empirical Software Engineering*, vol. 30, no. 3, p. 64, 2025.

[6] F. Grund, S. A. Chowdhury, N. Bradley, B. Hall, and R. Holmes, "CodeShovel: Constructing method-level source code histories," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2021.

[7] V. Orrei, M. Raglianti, C. Nagy, and M. Lanza, "Contribution-based firing of developers?," in *ESEC/FSE 2023 Ideas, Visions and Reflections*, 2023.

[8] O. Dabic, E. Aghajani, and G. Bavota, "Sampling projects in github for MSR studies," in *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*, pp. 560–564, IEEE, 2021.

[9] "Qualitative surveys." https://www2.sigsoft.org/EmpiricalStandards/docs/standards?standard=QualitativeSurveys. Accessed: 2024-11-02.

[10] J. Brooke, *SUS – a quick and dirty usability scale*, pp. 189–194. 01 1996.

[11] C. Kaner and W. Bond, "Software engineering metrics: What do they measure and how do we know?," in *IEEE International Software Metrics Symposium*, 2004.

[12] D. T. Campbell, "Assessing the impact of planned social change," *Evaluation and Program Planning*, vol. 2, no. 1, pp. 67–90, 1979.

[13] L. Engqvist and J. Frommen, "The h-index and self-citations," *Trends in ecology evolution*, vol. 23, pp. 250–2, 06 2008.

[14] H. Siebert, *Der Kobra-Effekt: wie man Irrwege der Wirtschaftspolitik vermeidet*. Dt. Verlag-Anst., 2001.

[15] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Introducing a ripple effect measure: A theoretical and empirical validation," in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–10, 2015.

[16] M. Shahidi, M. Ashtiani, and M. Zakeri-Nasrabadi, "An automated extract method refactoring approach to correct the long method code smell," *Journal of Systems and Software*, vol. 187, p. 111221, 2022.

[17] C. Jergensen, A. Sarma, and P. Wagstrom, "The onion patch: Migration in open source ecosystems," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, (New York, NY, USA), p. 70–80, Association for Computing Machinery, 2011.

[18] Y. Ding, B. Li, and P. He, "An improved approach to identifying key classes in weighted software network," *Mathematical Problems in Engineering*, vol. 2016, pp. 1–9, 2016.

[19] K. Yamashita, S. McIntosh, Y. Kamei, A. E. Hassan, and N. Ubayashi, "Revisiting the applicability of the pareto principle to core development teams in open source software projects," in *Proceedings of the 14th International Workshop on Principles of Software Evolution*, IWPSE 2015, (New York, NY, USA), p. 46–55, Association for Computing Machinery, 2015.

[20] M. Joblin, S. Apel, C. Hunsen, and W. Mauerer, "Classifying developers into core and peripheral: An empirical study on count and network metrics," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pp. 164–174, IEEE, 2017.

[21] T. Bock, N. Alznauer, M. Joblin, and S. Apel, "Automatic core-developer identification on github: A validation study," *ACM Trans. Softw. Eng. Methodol.*, 2023.

[22] H. A. Çetin and E. Tüzün, "Identifying key developers using artifact traceability graphs," in *Proceedings of the 16th ACM international conference on predictive models and data analytics in software engineering*, pp. 51–60, 2020.

[23] H. A. Çetin and E. Tüzün, "Analyzing developer contributions using artifact traceability graphs," *Empirical Softw. Engg.*, vol. 27, May 2022.

[24] J. Ren, H. Yin, Q. Hu, A. Fox, and W. Koszek, "Towards quantifying the development value of code contributions," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, (New York, NY, USA), p. 775–779, Association for Computing Machinery, 2018.

[25] P. R. de Bassi, G. M. P. Wanderley, P. H. Banali, and E. C. Paraiso, "Measuring developers' contribution in source code using quality metrics," in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, pp. 39–44, 2018.

[26] O. Cury, G. Avelino, P. Santos Neto, R. Britto, and M. Túlio Valente, "Identifying source code file experts," in *Proceedings of the 16th ACM / IEEE*

International Symposium on Empirical Software Engineering and Measurement, ESEM '22, (New York, NY, USA), p. 125–136, Association for Computing Machinery, 2022.

[27] N. Nagwani and S. Verma, "Rank-me: A java tool for ranking team members in software bug repositories," *Journal of Software Engineering and Applications*, vol. 05, pp. 255–261, 01 2012.

[28] G. Gousios, E. Kalliamvakou, and D. Spinellis, "Measuring developer contribution from software repository data," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, (New York, NY, USA), p. 129–132, Association for Computing Machinery, 2008.

[29] J. Lima, C. Treude, F. Figueira Filho, and U. Kulesza, "Assessing developer contribution with repository mining-based metrics," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 09 2015.

[30] Y. Sun, Z. Xu, C. Liu, Y. Zhang, and Y. Liu, "Who is the real hero? measuring developer contribution via multi-dimensional data integration," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, (Los Alamitos, CA, USA), pp. 825–836, IEEE Computer Society, sep 2023.

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

[32] E. Jabrayilzade, M. Evtikhiev, E. Tüzün, and V. Kovalenko, "Bus factor in practice," in *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '22, (New York, NY, USA), p. 97–106, Association for Computing Machinery, 2022.

[33] E. Klimov, M. U. Ahmed, N. Sviridov, P. Derakhshanfar, E. Tüzün, and V. Kovalenko, "Bus factor explorer," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 2018–2021, 2023.

[34] V. Haratian, M. Evtikhiev, P. Derakhshanfar, E. Tüzün, and V. Kovalenko, "Bfsig: Leveraging file significance in bus factor estimation," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, (New York, NY, USA), p. 1926–1936, Association for Computing Machinery, 2023.