# Aged to Perfection? Analyzing the Impact of Years of Experience on Code Quality

Jefferson G. M. Lopes
jeffersonmoreira@dcc.ufmg.br
Federal University of Minas Gerais (UFMG)

Johnatan Oliveira
johnatan.oliveira@ufla.br
Federal University of Lavras (UFLA)

Eduardo Figueiredo
figueiredo@dcc.ufmg.br
Federal University of Minas Gerais (UFMG)

## ABSTRACT

The association between developer experience and code quality is a significant debate within the software engineering community. To explore this relationship, we quantitatively evaluated 401 GitHub software repositories in JavaScript, PHP, and Python, maintained by 98 developers with Workana profiles – a freelancing platform. For quality assessment, we rely on SonarQube, a widely adopted tool in both industry and academia. We analyzed several dimensions, such as the programming languages and severity (high, medium, and low) of maintainability and reliability issues. We observed that developers with the highest level of experience presented code with fewer issues. However, developers with intermediate level of experience presented more code quality issues than their novice and experienced counterparts, revealing a more complex and nuanced relationship between level of experience and code quality. Despite that, our analysis did not indicate significant statistical differences between the density of the issue at varying levels of experience, suggesting that other factors may also contribute to code quality outcomes. This study contributes to an open question with implications for software engineering recruitment and code quality assurance. We provide a new dataset of developers and their projects based on recent data. All extracted data and accompanying scripts are available, aiming to enable further replications of our study.

## CCS CONCEPTS

• **Information systems** → *Information retrieval*; **Web mining**; • **Social and professional topics** → **Professional topics**.

## KEYWORDS

Code quality, developer experience, repository mining

## 1 INTRODUCTION

The level of experience developers possess may affect the quality of the software they produce. It is often assumed that increased individual experience correlates with enhanced competency [14]. However, the association between developer experience and code quality is a significant debate within the software engineering community [7, 33, 34, 36]. The definition of software quality may be related with different code aspects, such as readability [6], technical debts [3], bug frequency [16], and adherence to coding standards [46]. Similarly, experience of developers can be defined in several ways, such as the number of years working in the field [31] and the contributions made to software projects [20]. Regardless of their definitions, a common belief is that increasing individual experience is a way to achieve better software products; e.g., a program which is easier to comprehend and to maintain. For this reason, companies focus on evaluating work experience and educational backgrounds, as these factors enhance hiring recommendations through recruiters perceptions [44]. As a result, software developers nowadays have reported their previous experiences in freelance platforms, such as Workana [49], and in professional social networks, such as LinkedIn [1].

In fact, several studies have explored the relationship between the experience of developers and the quality of code they produce. However, a recent literature review has found conflicting results [24] and other prior investigations yield inconsistent conclusions [3, 7, 12, 24, 34, 36, 50]. For instance, Dieste et al. [12] found experience failed to predict code quality among 126 programmers, while Alfayez et al. [3] observed experienced developers introduced less technical debt in 38 Apache systems. Furthermore, several of these studies were conducted more than ten years ago, according to a comprehensive literature survey [24], and thus may not accurately represent present-day software development methodologies and practices.

Divergent findings prompted us to conduct a more detailed investigation with the objective of investigating the influence of developer experience on code quality in contemporary development environments. To explore this relationship, we examined 401 GitHub repositories from 98 Workana developers with varying self-reported JavaScript, PHP and Python years of experience. Using SonarQube, we assessed quality across languages, issue types maintainability and reliability, alongside their severities. With a formed dataset, we applied statistical methods to understand the impact of different levels of years of experience in the issue density of the developers. Additionally, we independently analyze the severity of the issues, their type and the language used by the developers to further improve isolation between the variables. Interestingly, the statistical analysis did not reveal a substantial correlation between experience levels and issue density across all dimensions, indicating that other elements impact quality. Our findings point towards a complex, non-linear association between various aspects of code quality and developer experience. Consistently, we found that developers with the highest analyzed experience presented fewer code quality issues compared to their peers. Nevertheless, mid-level developers, those with three to five years of experience, showed a higher occurrence of code quality issues than both more and less experienced developers, underscoring a complex non-linear dynamic.

In conclusion, this research emphasizes the necessity for additional investigations to elucidate and extend the understanding of the correlation between years of experience and code quality, potentially by incorporating new explanatory variables. Our contributions are threefold. Firstly, we address an unresolved issue that has significant implications for recruitment in software engineering and for practices concerning code quality. Secondly, we demonstrate developer portfolios as a feasible source of code examples and other pertinent data, such as years of experience, given that some portfolios are connected to social professional networks. Lastly, we provide a new dataset, which links Workana profiles to GitHub projects, complete with comprehensive replication materials. [1].

## 2 BACKGROUND

This section defines the main variables and concepts used and their relevance to the study. Section 2.1 reviews the concept of experience and how it is measured in this work. Section 2.2 discusses code quality. Finally, Section 2.3 explores social professional networks and portfolios and their applicability in this study.

### 2.1 Quantifying Experience

In professional contexts, employers and organizations often equate experience with competency and expertise [28], treating it as a critical factor in hiring and career advancement. This emphasis holds particular weight in software engineering, where technical proficiency and practical problem-solving skills directly influence outcomes [8, 37]. Yet, research on how experience affects code quality reveals inconsistencies. A recent literature review [24] investigating this relationship identifies two findings central to our work. First, studies operationalize experience in different ways. Researchers typically define it as either career experience (years of formal education or industry work) or project experience (quantitative contributions to a project, such as commit frequency or lines of code written). Second, only 3 of the 18 analyzed studies [27, 45, 50] used career experience as a predictor of code quality. Among these, two [45, 50] link greater career experience to improved code quality, while the third [27] argues that it cannot serve as a standalone measure.

This discrepancy and the lack of relevant studies point to a gap in understanding experience. To further investigate this, we use the definition of experience as self-reported years of active programming in a particular programming language as an independent variable. By focusing on this narrow definition of experience, our study addresses the limited attention Career Experience receives in existing research. Clarifying its role could contribute to software engineering recruitment and management, where experience metrics can influence hiring decisions and team structure.

### 2.2 Code Quality

Code quality plays a crucial role in a product's success for several reasons. First, high-quality code is typically more reliable and less prone to bugs, leading to a better user experience and reduced maintenance costs [19, 30]. Second, well-structured and maintainable code allows easier updates and feature additions, faster iteration, and adaptation to market demands [38]. The ability of a developer to write high-quality code can affect their career success, as it enhances their reputation among colleagues and potential employers. For instance, Li et al. [23] suggests that code quality is a key factor in the evaluation of software engineers.

[1]Replication package: https://doi.org/10.6084/m9.figshare.28306316.v1

In evaluating software quality, industry standards provide structured frameworks to guide assessment processes. Among these, ISO/IEC 25010 [2] serves as a globally recognized benchmark with eight core attributes to comprehensively evaluate software systems. These include functional suitability (alignment with requirements), performance efficiency (resource optimization), compatibility (seamless integration with other systems), usability (user effectiveness), reliability (consistent operation under defined conditions), security (data protection), maintainability (ease of modification), and portability (adaptability across environments). These criteria collectively ensure that the software meets the technical, operational, and user-centric benchmarks.

In this study, we focus on the maintainability and reliability dimensions of ISO/EIC 25010 [2] and rely on SonarQube to measure them. Although SonarQube can detect security issues, incomplete rule coverage prevented us from using security as a metric. By scanning code for vulnerabilities, redundancies, and complexity, SonarQube aligns with ISO's maintainability and reliability metrics: for instance, unused code or excessive complexity signals maintainability concerns, while error-prone logic flags relate to reliability [32]. In this context, we use metrics from SonarQube as dependent variables of years of experience. The platform classifies issues into three severity levels, ranging from critical flaws to minor inefficiencies with a lower risk. As defined by SonarQube, the code quality aspects evaluated in this study are the following.

- **Maintainability:** Maintainability issues affect the ease with which code can be modified or extended, often relating to code smells or complex structures. Addressing these issues can reduce technical debt and improve code comprehension.
- **Reliability:** Reliability issues refer to elements of code that could lead to unexpected behavior or system failures. Ensuring high reliability is essential for maintaining the stability and correct operation of the software system.

Additionally, each issue that SonarQube encounters, being a maintainability or a reliability issue, is classified in the following levels of severity.

- **High Severity:** This category includes critical issues that can lead to functional errors or security vulnerabilities, requiring immediate attention to maintain the stability of the application.
- **Medium Severity:** This level includes issues that, while less urgent than high severity issues, may still affect functionality or performance and warrant prompt resolution to prevent escalation.
- **Low Severity:** These issues are generally cosmetic or related to code style, affecting readability or maintainability without posing immediate functional risks.

### 2.3 Social Professional Networks and Portfolios

Social professional networks (SPNs) are online platforms designed to foster connections among professionals for career-oriented purposes. These networks enable users to create detailed profiles, establish connections, share industry-relevant information, and participate in discussions within their respective fields. Brandão and Moro [5] offer a comprehensive survey and taxonomy of SPNs, exploring their various types, definitions, and applications. Platforms such as Workana and GitHub exemplify SPNs under this definition. On Workana [2] developers can list their expertise across various skills, such as React, JavaScript, Java, and Web Design, along with their corresponding years of experience, to connect with potential employers. GitHub [3], on the other hand, serves as a widely used platform to host code repositories, contribute to open source projects, and engage in professional interactions. Notably, many Workana profiles feature links to corresponding GitHub accounts, which serve as portfolios. These portfolios consist of curated projects designed to demonstrate the technical skills of a developer, thus improving their prospects of employment [13]. This research relies on Workana and Github SPNs to assemble a dataset designed to evaluate the relationship between developer experience and code quality. It achieves this by mining GitHub portfolios and extracting developers' self-reported experience from Workana.

## 3 RESEARCH METHOD

Figure 1 describes the overall data collection process. We rely on Workana to collect developer profiles, on GitHub to collect code samples, and on SonarQube to measure code quality. Section 3.1 explains the study goal and research questions. Section 3.2 presents the Workana freelance platform and developer profiles. Section 3.3 explains how the collection of developer's GitHub repositories is collected. Section 3.4 reveals the collection and filtering of SonarQube quality metrics. Section 3.5 describes how we related Workana profile data with GitHub and SonarQube metrics. Section 3.6 describes the method applied to analyze the data.

### 3.1 Study Goal and Research Questions

The primary goal of this study is to examine the relationship between developer experience and code quality. By investigating profiles from the Workana freelancing platform and analyzing code repositories on GitHub, this study aims to uncover to what extent years of coding experience on a given set of programming languages, as self-reported by developers, might correlate with various code quality metrics obtained through SonarQube. The following research questions guide the study:
***RQ1:*** What is the impact of developer experience on code quality across programming languages?
***RQ2:*** How does the type of code quality problems vary across developer experience levels?
***RQ3:*** How does the severity of code quality problems (High, Medium, Low) vary across developer experience levels?

RQ1 inspects how developer experience and code quality vary across different programming languages. RQ2 and RQ3 go deeper by specifically examining the impact of the type and severity of code quality issues. This set of research questions is designed to minimize overgeneralization, ensuring a more accurate and meaningful analysis.

### 3.2 Workana Profile Collection

Our analysis begins by extracting profiles of developers from Workana, focusing on reported skills and years of experience. Each selected profile also includes a link to the developer's GitHub profile, enabling further investigation. A customized variant of EXTRACTPRO [15] was utilized for the data gathering process. This tool is intended for collecting Workana profiles and GitHub repositories. EXTRACTPRO locates Workana profiles using a search term, navigates through all resulting pages, and retrieves essential data including years of language experience, profile description, demographics, and a link to a GitHub profile, if available. The search term 'github.com' was employed to locate profiles with a GitHub link associated. Once the GitHub profiles were identified, they were downloaded into a local directory. Although EXTRACTPRO offers a web interface, we only rely on its extraction capabilities.

The Workana profile data are stored in a CSV file and subsequently imported into a Pandas DataFrame [29] for processing. Using regular expressions, we parse the skills data to identify programming languages and corresponding years of experience. Workana displays experience levels per programming language in five categories, 1<, 1–3, 3–5, 5–10, >10 years of experience. To ensure sufficient sample sizes and clearer distinctions among skill levels, we consolidated Workana's original experience categories into fewer bins 3<, 3-5, >5 years of experience, following guidelines on avoiding small cell sizes and improving statistical power according to our dataset [9, 21, 47]. In total, we collected all 266 Workana profiles, returned with the GitHub profile link, and stored their description and a list of skills. The next step filters the skills into a defined programming language set.

### 3.3 GitHub Repository Collection

For each developer, we scan their downloaded GitHub profiles and select repositories created within the last five years (from 2019 onward) that match the target programming languages (JavaScript, PHP, or Python). To ensure we analyze only the code authored by the developer being inspected, we exclusively download repositories with a single author, which must be the owner of the GitHub profile, avoiding complications related to team dynamics, different contributions of developers outside the sample, and other issues. We downloaded up to five repositories per developer to have a manageable dataset. A total of 552 repositories were collected matching the specified criteria.

### 3.4 SonarQube Scan and Quality Evaluation

SonarQube is applied to evaluate the downloaded GitHub repositories. These repositories are organized in a specified directory, with analysis restricted to files that have the selected programming language extensions: .php for PHP, .py for Python, and .js for JavaScript. In this study, we focus on public profiles featuring three specific skills—JavaScript, PHP, and Python—linked to an associated GitHub repository. We selected those languages following a pilot study, which identified them as the most commonly used in the collected profiles and compatible with SonarQube analysis.

We derive metrics from SonarQube reports and export them as spreadsheets summarizing code issues and lines of code (NKLOC) per file type. The code issues are of the type Maintanability and Reliability, and are further classified into the severity levels High,

[2] https://www.workana.com
[3] https://github.com

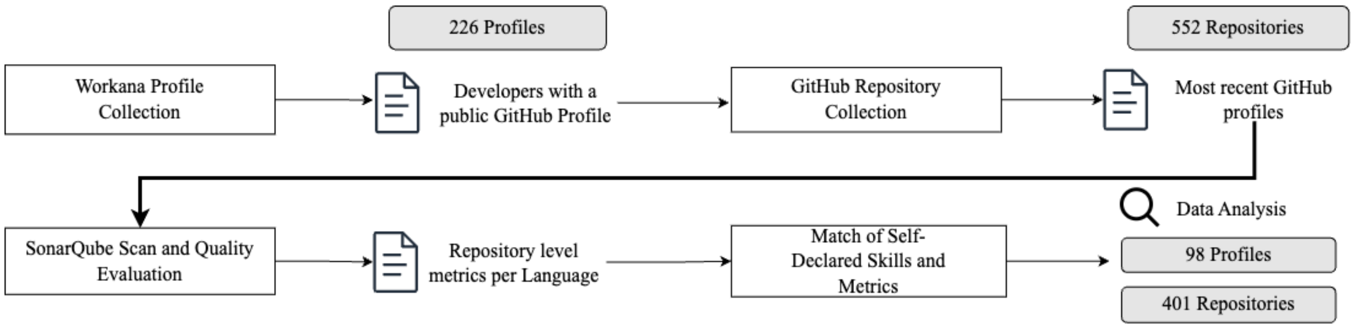
226 Profiles
552 Repositories
Workana Profile Collection
Developers with a public GitHub Profile
GitHub Repository Collection
Most recent GitHub profiles
Data Analysis
SonarQube Scan and Quality Evaluation
Repository level metrics per Language
Match of Self-Declared Skills and Metrics
98 Profiles
401 Repositories

**Figure 1: Process to collect profiles and repositories**

Medium and Low [40, 42]. The NKLOC metric returned by SonarQube only counts code lines that contain at least one character that is not a blank space, tabulation, or comment [41].

## 3.5 Matching of Self-Declared Skills and Metrics

Upon completion of the SonarQube analysis, the data from Workana and SonarQube are merged using GitHub usernames as identifiers. This matching process generates a dataset that links developer experience levels (from Workana) with corresponding code quality metrics (from SonarQube). A portion of the initial collection of Workana profiles did not declare experience in any of the studied languages or had GitHub profiles that did not contain code in the analyzed languages. After excluding them, the final collected dataset contained 98 Workana profiles and 401 repositories.

Table 1 displays a summary of the data collected. To assess the representativeness of the collected profiles, we extracted and categorized demographic data from their Workana profiles, considering country of origin and their self-reported role in software development. The self-reported role is an open-ended section where individuals can input any response. Consequently, we consolidated these roles, taking into consideration translations and various other differences. Among the 98 profiles, 28 identified themselves as Full Stack Developers, 17 as Software Developers, 14 as Web Developers, 9 as Frontend Developers, and 6 as Backend Developers. The other 24 profiles were categorized as "Other" due to the diversity of roles, which included titles such as CTO, Analyst, and Data Scientist. Despite the range of roles, all individuals provide software development services and have expertise in programming languages, along with GitHub portfolios[4] showcasing their work. Positions reported fewer than 5 times were included in the Other category. Regarding their locations, 52 individuals are from Brazil, 9 from Argentina, 6 from Venezuela, 4 from India, 4 from Malaysia, 3 from Turkey, 3 from Chile, and 17 are from various other countries, such as Israel, China, and Colombia, among others. The "Other" category combines countries with fewer than 3 entries.

[4]A portfolio, in this context, is not a normal set of GitHub repositories as they are linked by developers on their Workana profiles specifically to showcase skills and attract clients, suggesting the developer intends them to be viewed as a curated representation of their professional abilities, unlike potentially less curated profiles not linked for commercial purposes

## 3.6 Data Analysis

The data analysis phase began with a detailed refinement and descriptive statistical assessment of the dataset. We established a threshold of 1,000 lines of code in each programming language per developer to ensure that each sample included in the analysis sufficiently represented the developer's coding practices. Based on this threshold, we calculated an issue density metric, representing the number of code issues per a 1,000 non-commented lines of code for each programming language. Issue density metrics are useful for normalizing project size variations when analyzing issues [11, 25]. This metric allowed us to evaluate each developer's contribution to code quality by quantifying the rate of issues in their code. Each developer-language pair was treated as an individual data point, enabling us to include developers with varying experience levels across different programming languages. For example, if the Developer 1 has the following characteristics: an average issue density of 10.5 with more than 5 years of experience in Python, an average issue density of 70.1 with less than 3 years of experience in PHP, and an average issue density of 2.2 with less than 3 years of experience in JavaScript, then the Developer 1 will be treated as three separate data points, one for each language and experience pair. This technique was possible due to high independence of the issue density of one language over the others for the same developer in our dataset. In that sense, a developer that have high issue density in JavaScript, for example, may not necessarily have high issue density in Python, i.e. one language density has little influence over the other for the same developer.

A normality test was performed on the issue density data points for each language and on the overall issue density per developer. The results indicated that all the analyzed distributions do not follow a normal distribution. As a result, all subsequent statistical analyses rely on non-parametric tests.

To enhance the robustness of our analysis, we applied the Interquartile Range (IQR) method to detect and exclude outliers. For each programming language, experience level and issue type we calculated the first quartile (Q1) and third quartile (Q3) of the issue density values. The IQR is defined as the difference between Q3 and Q1. We established lower and upper bounds for acceptable data as Q1 - 1.5*IQR and Q3 + 1.5*IQR, respectively. We classified data points falling outside these bounds as outliers and excluded from further analysis. This approach minimized the influence of

**Table 1: Raw data summary**

| Programming Language | Years of Experience | Severity | | | Type | | Total Issues | NKLOC | Developers |
|---|---|---|---|---|---|---|---|---|---|
| | | High | Medium | Low | Maintainability | Reliability | | | |
| JavaScript | <3 | 4831 | 2515 | 1376 | 8122 | 1837 | 8722 | 225079 | 48 |
| | 3-5 | 2855 | 1252 | 833 | 4718 | 1035 | 4940 | 99091 | 15 |
| | >5 | 179 | 284 | 128 | 550 | 211 | 591 | 24758 | 13 |
| Python | <3 | 324 | 214 | 184 | 701 | 27 | 722 | 20324 | 16 |
| | 3-5 | 48 | 177 | 194 | 418 | 6 | 419 | 6662 | 5 |
| | >5 | 113 | 186 | 227 | 525 | 3 | 526 | 21830 | 1 |
| PHP | <3 | 504 | 473 | 1674 | 2471 | 221 | 2651 | 46267 | 26 |
| | 3-5 | 375 | 586 | 1290 | 1817 | 449 | 2251 | 70923 | 7 |
| | >5 | 186 | 277 | 740 | 1042 | 182 | 1203 | 19266 | 9 |

extreme values, allowing for a more accurate representation of central tendency and variability within the dataset.

We analyzed the correlations between overall issue density, total issue density by programming language, issue density by type (Reliability and Maintainability) and issue density by severity (High, Medium, Low) to answer the research questions. For this analysis, we employed the Kruskal-Wallis test [22], a non-parametric test that assesses whether the median ranks of different groups come from the same population. Formally, the Kruskall-Wallis test has following hypotheses:

- **Null Hypothesis ($H_0$):** The median ranks of issue density are equal across all groups (e.g., less than 3 years of experience, 3 to 5 years of experience, more than 5 years of experience), indicating no statistically significant differences.
- **Alternate Hypothesis ($H_a$):** At least one group has a different median rank of issue density, indicating a statistically significant difference among the groups.

We adopted a significance level of 0.05, as conventionally used in statistical hypothesis testing [10, 26], to determine whether to reject the null hypothesis. A p-value below this threshold indicates sufficient evidence to reject the null hypothesis in favor of the alternative, suggesting that the population medians are not equal.

## 4 RESULTS AND DISCUSSION

This section addresses the research questions by reporting and discussing the results of this study.

### 4.1 Overall Analysis

**Table 2: Average of Issue Densities by Years of Experience**

| Issues Counted | Years of Experience | | |
|---|---|---|---|
| | <3 | 3-5 | >5 |
| All Issues | 45.0 | 56.6 | **28.4** |
| Maintainability | 42.2 | 49.8 | **23.5** |
| Reliability | **4.9** | 6.7 | 7.6 |
| High | 10.5 | 11.4 | **7.0** |
| Medium | 12.5 | 20.9 | **11.9** |
| Low | 14.1 | 23.4 | **10.8** |

Table 2 presents the average issue density categorized by years of experience. When considering all issues and programming languages, the average issue density is 45.0 for developers with less than three years of experience, 56.6 for those with three to five years, and 28.4 for those with more than five years. These results challenge the assumption that increased years of experience in programming directly correlate with improved code quality. Instead, the data suggests a non-linear relationship between years of experience and code quality. A similar pattern is observed when issue types and severities are considered in isolation. Developers with less than three years of experience have a lower average issue density compared to those with three to five years of experience. The exception to this trend is observed in reliability-related issues, where issue density increases with greater developer experience. Finally, we observed that more experienced developers consistently posses a lower issue density when compared to the other experience levels.

The Kruskal-Wallis test yielded a p-value of 0.15. This result indicates that the null hypothesis cannot be rejected at standard significance levels. Although a non-linear relationship is apparent in the data, the statistical test suggests that there is insufficient evidence to conclude a significant difference in issue density across the years of experience analyzed.

### 4.2 Analysis of Programming Languages

*RQ1:* What is the impact of developer experience on code quality across programming languages?

Figure 2 shows the total issue density by years of experience and classified per programming language. The Y axis displays the issue density. The X axis shows the three categories of years of experience in a given language. Each bar represents a programming language, either Python, PHP or JavaScript. Figures 3 to 7 follow the same configuration[5].

Analyzing each programming language individually reveals distinct and intriguing patterns when it comes to issue density and the relationship with developer experience. For Python, the trend aligns closely with the broader observation made throughout the overall analysis: issue density tends to increase as developers gain

[5]Note that some categories, such as 'Python > 5 years of experience', contain small samples (n=1), which may compromise the interpretation of the visual distribution in this group.

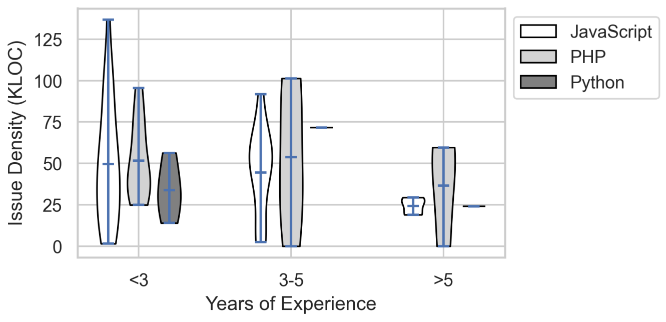

**Figure 2: Total issue density by years of experience**

experience up to the five-year mark and after it begins to decline again. This suggests that developers may be encountering a more manageable set of issues after reaching their highest level of experience.

In contrast, PHP shows a different pattern; the issue density remains relatively stable for developers with less than three years of experience through to five years. This stability is noteworthy and might imply that PHP developers face a consistent level of challenges regardless of their growing experience, although a decline is observed among those with more than five years of experience. This could indicate an effect where seasoned developers are better equipped to tackle complex issues, thus reducing the frequency of problems reported.

JavaScript presents a different trend, exhibiting a gradual decrease in issue density as experience increases, ranging from those with less than three years of experience to developers who have worked with the language for over five years. The decline might reflect a learning curve that allows developers to streamline their coding practices and avoid common pitfalls as their experience deepens. This results for JavaScript better align with a common belief that the quality positively correlates with experience.

Moreover, the Kruskal-Wallis p-values for Python, PHP, and JavaScript stand at 0.28, 0.90, and 0.29, respectively. These values displays the lack of statistically significant differences between the groups. Despite the visible conflicting trends observed in issue density across the languages, the statistical results indicate that we cannot reject the null hypothesis at conventional significance levels. In essence, these data suggest that we did not find any strong evidence in issue density patterns when analyzing each programming language individually, demonstrating the complexity of the relationships between experience and reported issues in software development.

***RQ1 summary****: The distribution of code quality problems varies across programming languages. Python shows a non-linear trend, with issue density increasing and then decreasing, whereas PHP remains relatively stable for low and moderate experience. The common belief that quality continuously increases with experience was only verified for JavaScript. The more experienced developers consistently display less code quality issues. The series are not statistically different ($p > 0.05$).*

## 4.3 Analysis of Issue Types

*RQ2:* How does the type of code quality problems vary across developer experience levels?

To answer RQ2, this section splits the analysis for type of issue: Maintainability and Reliability.

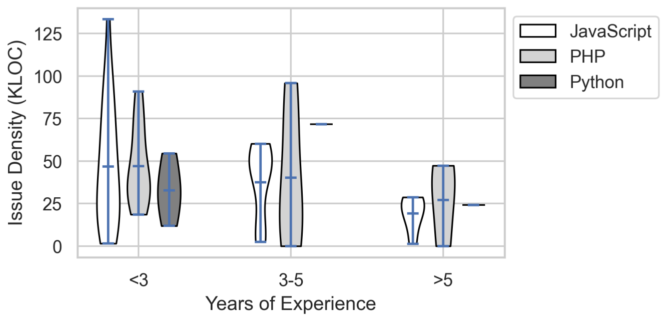

**Figure 3: Maintanability issue density by years of experience**

### 4.3.1 *Maintainability Issue Density.*

Figure 3 presents an analysis of Maintainability issue density correlated with varying levels of developer experience. Developers with less than three years of experience reported an average maintainability issue density of 42.2 issues per NKLOC, which rose to 49.8 for those with three to five years of experience. In contrast, developers with over five years of experience experienced a notable reduction, with an average of just 23.5 issues per NKLOC. This finding, again, illustrates a non-linear relationship between developer experience and the density of Maintainability issues.

An examination of specific programming languages revealed nuanced differences. Python adhered to the overall trends previously identified, showing a lower count in issues for developers with less than three years of experience, an increase for those with three to five years, and a significant drop for those with more than five years. On the other hand, PHP and JavaScript data agreed with the common belief that more experience results in better code quality. Both PHP and JavaScript showed a marked and steady decline in maintainability issue density as developer experience increased.

Statistical analysis conducted with the Kruskal-Wallis test yielded p-values of 0.17 for JavaScript, 0.74 for PHP, 0.28 for Python, and 0.08 for all languages collectively. These findings suggest that while certain trends are observable, the differences in maintainability issue density across various experience levels are not statistically significant at conventional thresholds. This indicates that while fluctuations and improvements can be noted, they do not provide conclusive or uniform evidence regarding maintenance issues across the programming languages analyzed.

### 4.3.2 *Reliability Issue Density.*

Figure 4 illustrates the Reliability issue density based on years of experience. This analysis reveals

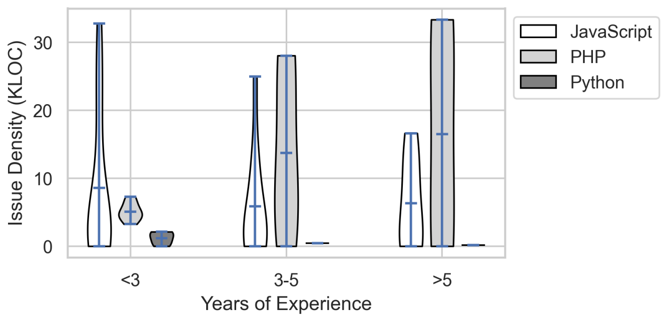

**Figure 4: Reliability issue density by years of experience**

the most drastical variations among the series. For Reliability issues, the average issue density shows a gradual increase with experience: developers with less than three years of experience encounter 4.9 issues per NKLOC, while those with three to five years report 6.7 issues, and those with more than five years experience an increase to 7.6 issues. This trend, encompassing all programming languages, contrasts with the non-linear patterns observed in other categories. A significant factor in these findings is PHP, which has a higher absolute issue density compared to other languages, influencing the overall average in a notable way.

When examining the languages individually, distinct trends emerge. Python exhibits a markedly lower absolute issue density that declines with increased experience. In contrast, PHP shows a significant increase in issue density as developers gain experience. JavaScript displays a non-linear relationship: higher issue counts during the initial three years, followed by stability for those with three to five years, and a slight increase for developers with over five years of experience. The variability in issue densities across different languages suggests the presence of underlying factors, such as a programming language's propensity to generate issues in SonarQube. For instance, Python may favor simpler constructs that lead to fewer mistakes by developers.

Nevertheless, the results from the Kruskal-Wallis test indicate no statistically significant differences, with p-values of 0.84 for JavaScript, 0.58 for PHP, 0.60 for Python, and 0.99 when all languages are considered together. While the individual trends are evident, the marginal differences in issue counts among programming languages do not provide sufficient evidence for meaningful distinctions.

***RQ2 summary**: Code quality problems differ by issue type and language. Maintainability issues exhibit a non-linear pattern, with density increasing before declining significantly with more than five years of experience. Reliability issues can increase or decrease over experience depending on the programming language analyzed. However, the correlation tests did not yield a statistical significant relationship ($p > 0.05$) for both types of issues.*

## 4.4 Analysis of Issue Severity

*RQ3:* How does the severity of code quality problems (High, Medium, Low) vary across developer experience levels?

In this section, we separated our analysis in three categories: High, Medium and Low Severity Issues.

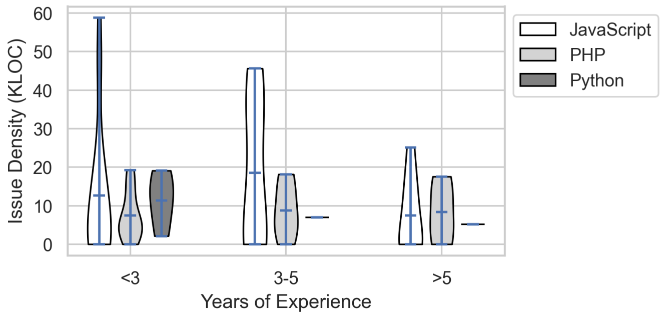

**Figure 5: High severity issue density by years of experience**

*4.4.1* ***High Severity Issue Density***. Figure 5 displays the average issue density by years of experience for High severity issues. The average issue density for developers with less than three years of experience was 10.5 issues per NKLOC, increasing to 11.4 for those with three to five years of experience and then decreasing to 7.0 for those with more than five years of experience across all programming languages combined. This pattern again indicates a non-linear relationship between experience and high-severity issue density.

Examining each language individually revealed mixed results. Python exhibited a consistent decline in issue density with increasing experience, dropping from an average of 11.36 issues per NKLOC for developers with less than three years of experience to 5.18 for those with more than five years. PHP showed relatively stable issue densities across experience levels, with averages of 7.50 for less than three years, 8.73 for three to five years, and 8.34 for more than five years. JavaScript presented a peak in issue density for the three to five years category, with averages of 12.67, 18.54, and 7.41 for less than three, three to five, and more than five years of experience, respectively.

The observed non-linear relationship between developer experience and High severity issue density indicates that increased experience does not uniformly lead to fewer critical issues. The rise in issue density among developers with three to five years of experience may reflect a phase where they are engaged in more complex or risk-prone tasks, potentially increasing the likelihood of introducing High severity issues. Language-specific trends further suggest that the nature of the work and the programming language influence this relationship. In Python, the consistent decline in issue density with experience implies that proficiency gains effectively reduce critical issues. In contrast, PHP's stable issue density across experience levels suggests that other factors, such as the types of projects, may mitigate the impact of experience on High severity issues. The peak in JavaScript issue density for mid-level developers could be due to the complexity of JavaScript applications or the specific challenges associated with intermediate-level tasks in that language. The Kruskal-Wallis test results yielded p-values of 0.40 for JavaScript, 0.80 for PHP, 0.60 for Python, and 0.48 for all languages combined, indicating no statistically significant differences across experience levels within each language.

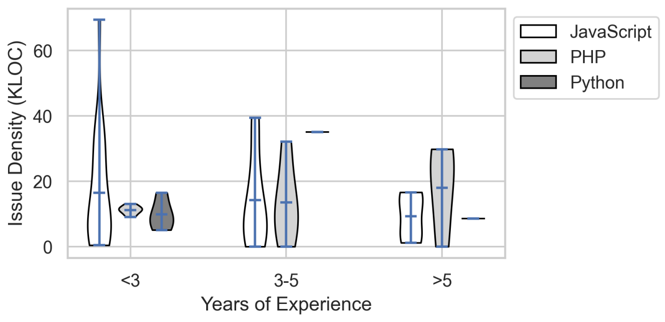

**Figure 6: Medium issue density by years of experience**

*4.4.2* ***Medium Severity Issue Density.*** Figure 6 presents the average issue density categorized by years of experience for Medium severity issues. Developers with less than three years of experience exhibited an average issue density of 12.5 issues per NKLOC. This figure escalated to 20.9 issues per NKLOC for those possessing three to five years of experience, only to decline to 11.9 issues per NKLOC for developers with over five years of experience. This non-linear trend parallels the pattern observed with High severity issues, where issue density initially increases before subsequently decreasing with additional experience.

A detailed analysis of individual programming languages again reveals distinct trends. In Python, issue density rises through the three to five years experience category and then declines, reflecting the overall observed trend. PHP, conversely, demonstrates relatively stable issue density across all experience levels, indicating that experience has a minimal impact on Medium severity issues within this language. In contrast, JavaScript displays a consistent decrease in issue density as developer experience increases, supporting the hypothesis that higher experience may enhance code quality within this context.

However, the results of the Kruskal-Wallis test indicate the absence of statistically significant differences in issue density across the various experience levels. The p-values obtained were 0.69 for JavaScript, 0.61 for PHP, 0.34 for Python, and 0.94 for the aggregate of all languages. These elevated p-values and marginal absolute differences between the issue densities suggest that the observed differences in Medium severity issue density are likely attributable to other factors rather than a systematic influence of developer experience.

*4.4.3* ***Low Severity Issue Density.*** Figure 7 shows that Low severity issue density begins at 14.1 issues per NKLOC for developers with less than three years of experience, peaks at 23.4 for those with three to five years, then drops to 10.8 for over five years. This non-linear pattern mirrors trends found in higher severity issues.

A deeper investigation by language reveals distinct behaviors. In Python, issue density is relatively low for beginners, rises for

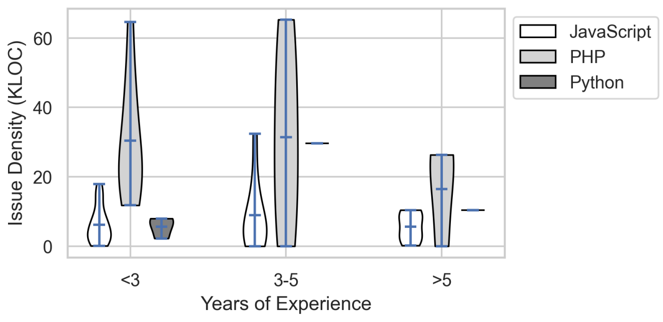

**Figure 7: Low severity issue density by years of experience**

intermediate-level developers, then declines for those with more than five years. PHP remains high for those below five years of experience but drops significantly thereafter, suggesting seasoned PHP developers gain deeper insights to reduce minor issues. JavaScript shows minimal fluctuations across all experience levels, with only a slight increase in the three-to-five-year bracket.

Kruskal-Wallis tests indicate no statistically significant differences in Low severity issues across experience levels (p-values of 0.88 for JavaScript, 0.62 for PHP, 0.20 for Python, and 0.67 overall). Because these issues often concern style or minor inefficiencies rather than functionality, their impact on overall quality may be limited. However, trends suggest that less experienced developers adhere closely to standards, intermediates may explore advanced techniques (potentially incurring more minor issues), and veteran developers produce cleaner code, potentially after internalizing best practices. Differences in language features also likely shape how minor issues arise and are addressed.

***RQ3 summary****: The severity of code quality problems varies with developer experience. High and Medium severity issues show non-linear trends, peaking at moderate experience levels and decreasing with more extensive experience. Low severity issues follow a similar trajectory. The statistical correlation test points no significant difference ($p > 0.05$) between the analyzed series.*

## 4.5 Discussion

Our findings reveal a complex, non-linear relationship between developer experience and code quality across different programming languages, issue types, and severity levels. In our observations, code quality was consistently improved by developers with the highest level of experience. This pattern was observed across High, Medium, and Low severity issues and Maintainability issues, indicating that the relationship between experience and code quality is more intricate than a simple linear correlation. Contrary to the common belief that more experience uniformly and directly leads to higher code quality, the frequency of code quality issues, or issue density, often increases for developers with three to five years of experience before declining for those with more than five years.

One possible explanation for this non-linear trend is grounded in models of skill acquisition, such as the Dreyfus Model of Skill Acquisition [18]. According to this model, individuals progress through stages from novice to expert, and during the intermediate 'competent' stage, they have enough experience to tackle more complex tasks but may lack the intuitive grasp that experts possess. In our context, this stage may lead to increased introduction of code quality issues as developers experiment with advanced features and take on challenging problems without fully developed expertise. Additionally, the Dunning-Kruger effect highlights that individuals with low to moderate skill levels may overestimate their abilities [39], potentially leading to overconfidence and a higher likelihood of making mistakes or overlooking best practices.

Contextual factors, such as project complexity, task assignments, and organizational practices likely contribute to these variations. The choice of programming language being analyzed is also a crucial factor that can significantly influence code quality. Different programming languages have inherent characteristics, paradigms, and syntax that can lead to variations in how code is structured and maintained. While the impact of the programming language on code quality is generally recognized to be marginal, it is essential to acknowledge that these differences can affect aspects, such as reliability and maintainability. For instance, languages that enforce stricter typing or offer robust error-handling mechanisms may contribute to fewer bugs and improved overall code reliability. On the other hand, languages that promote rapid prototyping may facilitate faster development cycles but can potentially lead to lower code quality in terms of long-term maintainability and scalability [35]. Experienced developers may produce cleaner code due to their internalized best practices and deeper understanding of language nuances. Less experienced developers might adhere strictly to coding guidelines, resulting in fewer Low severity issues but potentially not addressing complex problems effectively. Intermediate developers may face new challenges without the full expertise required, which can temporarily impact code quality until they gain further experience.

Research opportunities exist to systematically investigate the distinct challenges faced by mid-level developers compared to those at other levels of seniority. Such inquiry can contribute to a deeper understanding of the unique experiences and obstacles in the software development profession, thereby informing strategies for effective professional development. One potential solution to the noted phenomenon is to provide ongoing training on more complex tasks for mid-level developers or to engage them with case studies that prepare them to tackle more sophisticated challenges. The lack of statistically significant differences in issue densities across experience levels, as indicated by the Kruskal-Wallis tests (p-values > 0.05), suggests that experience alone may not be a strong predictor of code quality. It also implies that other factors, such as team dynamics, code review processes, or individual learning curves, might impact code quality more than experience measured in years. Our findings align partially with existing literature showing the multifaceted nature of software development expertise. Practically, these findings call for further research on other factors that may play a more substantial role in influencing developer quality.

## 5 THREATS TO VALIDITY

In this section, we discuss the potential threats to the validity of our study, categorized according to Wohlin et al. [48].

**Internal Validity** – this validity concerns the use of self-reported experience levels from the Workana platform. Self-reported data may introduce biases if developers misrepresent their experience. To mitigate this, we carefully categorized experience levels and filtered out profiles lacking relevant information. However, the reliance on self-reported data could still impact findings, as actual proficiency may not be exactly aligned with years of reported experience. Furthermore, a discrepancy may exist between the experience developers assert on Workana and the recent activity in their GitHub repositories. While some repositories may not reflect the latest coding abilities of a developer, our analysis shows that for 75% of the developers, the oldest repository in their portfolio was updated at most two years before their last Workana login. This indicates that a large portion of the developer's GitHub portfolio is quite current, alleviating worries about the recency of outdated code compared to their Workana profile. However, for a subset of developers, repositories may not accurately reflect their current proficiency, introducing a potential source of bias in the correlation analysis.

**External Validity** – this study is potentially limited by the selection of only three programming languages: JavaScript, PHP, and Python. While these languages are widely used and represent diverse paradigms, the results might not generalize to other programming languages with distinct ecosystems and development practices. Furthermore, as we only analyzed developers active on GitHub and Workana, our findings may not apply to developers in other contexts, such as enterprise environments or other freelancing platforms.

**Construct Validity** – this validity concerns the alignment between our measurements and the theoretical constructs we intend to examine. In this study, we operationalized code quality using SonarQube, a static analysis tool that detects maintainability and reliability issues in source code. We selected SonarQube due to its widespread use in industry and academia for analyzing software quality [3, 35], and its classification of code issues aligns with key attributes of the ISO/IEC 25010 [2] software quality framework. Additionally, we restricted our dataset to single-author repositories, ensuring that the detected issues reflect the practices of individual developers rather than team dynamics. However, we acknowledge that our study focuses primarily on maintainability and reliability, and does not assess other dimensions of code quality, such as performance and security. Despite this, maintainability and reliability are critical aspects of software quality, making SonarQube an appropriate tool for our analysis.

**Conclusion Validity** – this validity is related to the soundness of our statistical analyses. We employed non-parametric tests due to the non-normal distribution of our data; however, these tests may have reduced sensitivity in detecting subtle relationships between experience and code quality. Further, while outliers were identified and removed using the Interquartile Range method, these values might have contained useful information regarding the variability in developer performance. Consequently, future studies with larger datasets and more nuanced statistical techniques could help strengthen the conclusions drawn from this research.

## 6 RELATED WORK

This section introduces the related work relevant to our study, covering key findings on the relationship between developer experience, human factors, technical debt, and code quality metrics.

The impact of developer experience on code quality has been widely debated in the field of software engineering [7, 33, 34, 36]. Studies show that experience is not always a valid indicator of quality. For example, Dieste et al. [12] explore how industry experience does not necessarily lead to improved code quality, finding that factors like academic background and familiarity with productivity tools can have more influence. This result suggests that although experience is often valued, it should be considered alongside other variables, such as specific tool expertise.

Salamea and Farre [36] examined human factors in software quality, focusing on aspects like the level of involvement and project experience of developers. They found a positive correlation between experience and the amount of technical debt (TD) introduced, though communication skills appeared to have minimal impact on TD. Their results indicate that developers who are more familiar with a project may tend to introduce TD, possibly due to the complexity of their contributions. This study adds to the discussion on experience by showing that developer characteristics can impact both code quality and maintenance practices.

Research on the effects of code quality in production settings, like the study by Tornhill and Borg [43], demonstrates that low-quality code significantly increases defects and the time required to resolve them, which in turn affects developer productivity. Their findings indicate that TD leads to higher unpredictability, organizational stress, and additional costs, highlighting the importance of maintaining code quality as a business goal. This is particularly relevant in our study, as SonarQube was used as a tool to identify and mitigate maintainability and reliability issues.

Alfayez et al. [3] conducted an exploratory study on the influence of developers on TD by analyzing 19,088 commits across 38 Apache Java systems. Their findings indicate that TD is unevenly distributed among developers, with seniority and frequency of commits negatively correlated to TD introduced, while longer intervals between commits are positively correlated. This suggests that more active and experienced developers tend to reduce TD accumulation, while less frequent contributors are more likely to increase TD. Their use of SonarQube for TD measurement aligns with our methodological approach and highlights the importance of individual developer characteristics in maintaining code quality.

While previous studies have explored topics like developer experience, technical debt, and design practices, our study differs by conducting a large-scale analysis across 401 GitHub repositories in multiple languages (JavaScript, PHP, and Python) to investigate the complex relationship between developer experience and code quality. By integrating SonarQube metrics with developer profiles from Workana, we provide a detailed analysis of how experience influences maintainability and reliability issues in varied contexts. Unlike other studies that often suggest linear trends, our findings indicate a non-linear relationship between experience and code quality, suggesting that as developers gain experience, the complexity and reliability challenges in their projects may increase, rather than showing a simple improvement in code quality.

## 7 CONCLUSION

It is a common belief is that experience of developers has a positive linear correlation with code quality. Although several studies [7, 33, 34, 36] have explored their relationships, literature reports conflicting results. To deeper investigate this open question with high implications for software engineering and related areas, such as program comprehension, this paper performed an empirical study with publicly available data mined from GitHub and Workana. Based on profiles of 98 developers and 401 open-source software repositories, we analyzed different perspectives of the experience-quality relationship, such as programming language, types of code issues, and severity of issues.

Our observations consistently indicated that experienced developers exhibit a lower density of issues compared to their less experienced counterparts, particularly in terms of maintainability and issues classified with High, Medium, and Low severity across all issue types. Furthermore, our analysis revealed that developers with three to five years of experience exhibited higher issue densities compared to both less experienced and more experienced developers. This non-linear relationship was relatively consistent across different programming languages and issue types. These findings suggest that factors, such as the complexity of tasks at different career stages, perception of competency and even the specific characteristics of programming languages may influence code quality more significantly than experience alone.

For future work, we plan to expand our study to cover other programming languages and more developers. We may achieve this goal by replicating our study with different freelance platforms and professional social networks, such as LinkedIn [1]. Another possible venue for further work is using varying proxies for software quality, such as the number of reported bugs or other evaluation tools. Human subjects could also be used to evaluate other dimensions of code quality, such as portability and usability.

## ARTIFACT AVAILABILITY

This research examines data from public repositories on GitHub and Workana, ensuring the adherence to ethical standards in software repository mining as outlined in the software engineering literature [17]. The data set utilized in this study and all the scripts are accessible through a public repository [4]. We have taken steps to anonymize developer identifiers from both Workana and GitHub profiles, where feasible, to address privacy concerns. We have also refrained from disclosing confidential information, such as names, LinkedIn profiles, Facebook pages, and other personal information that could identify or affect the study subjects. Although our objective is to investigate the link between experience and code quality, we recognize that software quality may be shaped by numerous factors beyond one's self-reported experience.

**Acknowledgements.** This research was partially supported by Brazilian funding agencies: CNPq (Grant 312920/2021-0), CAPES, and FAPEMIG (Grant APQ-01488-24).

## REFERENCES

[1] [n. d.]. LinkedIn. https://linkedin.com. Accessed: 2024-11-01.

[2] 2023. ISO/IEC 25010:2023: Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models. https://www.iso.org/standard/78176.html ISO/IEC Standard.

[3] Reem Alfayez, Pooyan Behnamghader, Kamonphop Srisopha, and Barry Boehm. 2018. An exploratory study on the influence of developers in technical debt. In *Proceedings of the 2018 international conference on technical debt*. 1–10.

[4] Authors. 2025. replication-package-Analyzing-the-Impact-of-Years-of-Experience-on-Code-Quality-main. (1 2025). https://doi.org/10.6084/m9.figshare.28306316.v1

[5] Michele A Brandão and Mirella M Moro. 2017. Social professional networks: A survey and taxonomy. *Computer Communications* 100 (2017), 20–31.

[6] Raymond PL Buse and Westley R Weimer. 2009. Learning a metric for code readability. *IEEE Transactions on software engineering* 36, 4 (2009), 546–558.

[7] Denivan Campos, Luana Martins, and Ivan Machado. 2022. An empirical study on the influence of developers' experience on software test code quality. In *Proceedings of the XXI Brazilian Symposium on Software Quality*. 1–10.

[8] Luiz Fernando Capretz and F. Ahmed. 2018. A Call to Promote Soft Skills in Software Engineering. *ArXiv* abs/1901.01819 (2018). https://doi.org/10.17140/PCSOJ-4-e011

[9] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Routledge, New York, NY, USA.

[10] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, Carlo A Furia, and Ziwei Huang. 2019. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. *Journal of Systems and Software* 156 (2019), 246–267.

[11] Massimiliano Di Penta, Luigi Cerulo, and Lerina Aversano. 2009. The life and death of statically detected vulnerabilities: An empirical study. *Information and Software Technology* 51, 10 (2009), 1469–1484.

[12] O. Dieste et al. 2017. Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study. *Empirical Software Engineering* 22, 5, 2457–2542.

[13] Nikolina Dragicevic, Matantsev Maxim, and Artem Kruglov. 2023. A Study of Effective Strategies for Personal Development and Success for Software Engineers. In *2023 IEEE 14th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 101–104.

[14] K Anders Ericsson, Robert R Hoffman, Aaron Kozbelt, and A Mark Williams. 2018. *The Cambridge handbook of expertise and expert performance*. Cambridge University Press.

[15] Jefferson G. Lopes, Johnatan Alves, and Eduardo Figueiredo. 2022. EXTRACT-PRO: A Data Mining Tool for Developer Profile Generation based on Source Code Analysis. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering*. 112–117.

[16] E. Giger, M. Pinzger, and H. C. Gall. 2011. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceedings of Data Archiving and Networked Services (DANS)*.

[17] Nicolas E Gold and Jens Krinke. 2022. Ethics in the mining of software repositories. *Empirical Software Engineering* 27, 1 (2022), 17.

[18] Nora Honken. 2013. Dreyfus five-stage model of adult skills acquisition applied to engineering lifelong learning. In *2013 ASEE Annual Conference & Exposition*. 23–443.

[19] .. Khyber, Sikandar Ali, Fazli Wahid, S. Baseer, A. Alkhayyat, and Akram Al-Radaei. 2024. Smell-Aware Bug Classification. *IEEE Access* 12 (2024), 14061–14082. https://doi.org/10.1109/ACCESS.2023.3335175

[20] S. Ozcan Kini and A. Tosun. 2018. Periodic Developer Metrics in Software Defect Prediction. In *IEEE Xplore*.

[21] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-001. EBSE Technical Report. https://www.cs.auckland.ac.nz/~norsaremah/2007-Kitchenham-GuideSLR.pdf

[22] William H. Kruskal and W. Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *J. Amer. Statist. Assoc.* 47, 260 (Dec. 1952), 583–621.

[23] Paul Luo Li, Amy J Ko, and Jiamin Zhu. 2015. What makes a great software engineer?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 700–710.

[24] Jefferson GM Lopes, Johnatan Oliveira, and Eduardo Figueiredo. 2024. Evaluating the Impact of Developer Experience on Code Quality: A Systematic Literature Review. In *Congresso Ibero-Americano em Engenharia de Software (CIbSE)*. SBC, 166–180.

[25] Ivano Malavolta, Roberto Verdecchia, Bojan Filipovic, Magiel Bruntink, and Patricia Lago. 2018. How maintainability issues of android apps evolve. In *2018 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 334–344.

[26] James Miller. 2004. Statistical significance testing—-a panacea for software technology experiments? *Journal of Systems and Software* 73, 2 (2004), 183–192.

[27] O. Dieste et al. 2018. Empirical Evaluation of the Effects of Experience on Code Quality and Programmer Productivity: An Exploratory Study. In *Proceedings of the 2018 International Conference on Software and System Process*. 111–112.

[28] Susanna Paloniemi. 2006. Experience, competence and workplace learning. *Journal of Workplace Learning* 18 (10 2006), 439–450. https://doi.org/10.1108/13665620610693006

[29] Pandas. 2019. Python Data Analysis Library — pandas: Python Data Analysis Library. https://pandas.pydata.org. Accessed: Nov. 1, 2024.

[30] Shravan Pargaonkar. 2023. Cultivating Software Excellence: The Intersection of Code Quality and Dynamic Analysis in Contemporary Software Development within the Field of Software Quality Engineering. *International Journal of Science and Research (IJSR)* (2023). https://doi.org/10.21275/sr23829092346

[31] V. Piantadosi, S. Scalabrino, A. Serebrenik, N. Novielli, and R. Oliveto. 2023. Do attention and memory explain the performance of software developers? *Empirical Software Engineering* 28, 5 (Aug. 2023).

[32] Alifia Puspaningrum, Muhammad Anis Al Hilmi, . Darsih, Muhamad Mustamiin, and Maulana Ilham Ginanjar. 2022. Vulnerable Source Code Detection Using Sonarcloud Code Analysis. In *Proceedings of the 5th International Conference on Applied Science and Technology on Engineering Science*. SCITEPRESS - Science and Technology Publications, 683–687. https://doi.org/10.5220/0011862600003575

[33] Yilin Qiu, Weiqiang Zhang, Weiqin Zou, Jia Liu, and Qin Liu. 2015. An empirical study of developer quality. In *2015 IEEE International Conference on Software Quality, Reliability and Security-Companion*. IEEE, 202–209.

[34] Foyzur Rahman and Premkumar Devanbu. 2011. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*. 491–500.

[35] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. 2014. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*. 155–165.

[36] María José Salamea and Carles Farré. 2019. Influence of developer factors on code quality: A data study. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 120–125.

[37] Sigrid Schefer-Wenzl and I. Miladinovic. 2019. Developing Complex Problem-Solving Skills: An Engineering Perspective. *Int. J. Adv. Corp. Learn.* 12 (2019), 82–88. https://doi.org/10.3991/ijac.v12i3.11067

[38] D. Shyamal, P. Asanka, and D. Wickramaarachchi. 2023. A Comprehensive Approach to Evaluating Software Code Quality Through a Flexible Quality Model. *2023 International Research Conference on Smart Computing and Systems Engineering (SCSE)* 6 (2023), 1–8. https://doi.org/10.1109/SCSE59836.2023.10215004

[39] Daniel J Simons. 2013. Unskilled and optimistic: Overconfident predictions despite calibrated knowledge of relative skill. *Psychonomic bulletin & review* 20 (2013), 601–607.

[40] SonarSource. 2024. Clean-Code-based analysis | SonarQube Docs. https://docs.sonarsource.com/sonarqube/10.7/core-concepts/clean-code/code-analysis/. Accessed: Nov. 1, 2024.

[41] SonarSource. 2024. Metrics | SonarQube Docs. https://docs.sonarsource.com/sonarqube-server/10.7/user-guide/code-metrics/metrics-definition/. Accessed: Nov. 1, 2024.

[42] SonarSource. 2024. Software qualities | SonarQube Docs. https://docs.sonarsource.com/sonarqube/10.7/core-concepts/clean-code/software-qualities/. Accessed: Nov. 20, 2024.

[43] Adam Tornhill and Markus Borg. 2022. Code red: the business impact of code quality-a quantitative study of 39 proprietary production codebases. In *Proceedings of the International Conference on Technical Debt*. 11–20.

[44] W.-C. Tsai, N.-W. Chi, T.-C. Huang, and A.-J. Hsu. 2010. The Effects of Applicant Résumé Contents on Recruiters' Hiring Recommendations: The Mediating Roles of Recruiter Fit Perceptions. *Applied Psychology* 60, 2 (Oct. 2010), 231–254.

[45] V. Piantadosi et al. 2023. Do Attention and Memory Explain the Performance of Software Developers? *Empirical Software Engineering* (2023).

[46] Y. Wang, B. Zheng, and H. Huang. 2008. Complying with Coding Standards or Retaining Programming Style: A Quality Outlook at Source Code Level. *Journal of Software Engineering and Applications* 1, 1 (2008), 88–91.

[47] Claes Wohlin, Per Runeson, Martin Host, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

[48] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[49] WORKANA. 2024. Hire Freelancers and IT Developers | Workana. https://www.workana.com. Accessed: Nov. 1, 2024.

[50] Z. Karimi et al. 2016. Links Between the Personalities, Styles and Performance in Computer Programming. *Journal of Systems and Software* 111 (2016), 228–241.