Evaluating the Impact of Developer Experience on Code Quality: A Systematic Literature Review

Jefferson G. M. Lopes[1] **, Johnatan Oliveira**[2] **, Eduardo Figueiredo**[1]

[1]Department of Computer Science (DCC)
Federal University of Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

[2]Department of Computer Science (DCC)
Federal University of Lavras (UFLA)
Lavras – MG – Brazil

{jeffersonmoreira, figueiredo}@dcc.ufmg.br, johnatan.oliveira@ufla.br

Abstract. *The relationship between developer experience and code quality continues to provoke extensive debate and diverging interpretations in software engineering. To investigate this subject, we conducted a systematic literature review and identified 18 relevant papers from which we aim to answer an overarching research question: to what extent does developer experience impact on code quality? Our analysis reveals different definitions and dimensions for both developer experience and code quality, highlighting the complexity and multifaceted nature of their relationship. We also observed contradictory results on the impact of developer experience on code quality. This literature review contributes in two key ways. First, it synthesizes various perspectives on developer experience and code quality, offering a consolidated viewpoint of the current academic work. Second, it uncovers significant gaps in our understanding of the relationship between these two concepts, pinpointing areas for further research and emphasizing the needs for more focused studies to bridge these knowledge gaps.*

1. Introduction

In software engineering, the experience of developers may play a pivotal role in shaping the quality of the software they produce. A common belief is that increased individual experience correlates with enhanced competency [1]. This notion, seemingly straightforward, has driven numerous studies [2, 3] to explore the relationship between a developer's experience and the quality of code they create. However, empirical research in this area often yields conflicting results, underscoring the complexity in defining and measuring both experience and code quality in software development [2].

The concept of code quality itself is diverse, with aspects such as technical debt [4], bug frequency [5], and adherence to coding standards [6]. Additionally, there are a similar diversity while defining a developer's experience, that can relate to years worked in the field [7] or contributions made to a project [8]. Previous work [2, 3] in this field have yielded inconsistent findings. For example, Dieste et al. [2] conducted a study of the effects of different definitions of experience (academic background, programming experience, unit testing experience) on the performance of 126 programmers

and they found that experience is not able to predict code quality and productivity. On the other hand, Alfayez et al. [3], after analyzing 38 Apache Java systems and applying multiple statistical analysis, concluded that more experienced developers introduce less technical debt. With many definitions of developer experience and code quality, different results and the lack of review studies, it is not clear how those two concepts relate.

Our paper aims to delve into this intricate relationship, providing a comprehensive analysis of the existing literature to shed light on the nuances that underlie the connection between developer experience and code quality. This paper provides two main contributions. First, a comprehensive review of the definitions of code quality and developer's experience used by previous studies. Second, an overview of the explored areas and gaps in our understanding of the relationship between the two concepts. To accomplish that, we defined a systematic literature review protocol and reviewed 1026 candidate studies to answer the research questions. With the results, we aim to offer valuable insights for both practitioners and researchers in software engineering. For practitioners, understanding this relationship can guide better team formation, project management, and training strategies. For researchers, our study provides a foundation for future investigations into the nuances of developer experience and its implications for software quality.

By conducting our systematic review protocol, we found and classified 7 distinct ways of measuring code quality and 5 distinct ways of accounting a developer's experience over the reviewed studies. Additionally, we show that exists a clear trend of previous work in defining a developer's experience as project experience and taking code quality as the number of bugs introduced. Other combinations of experience and code quality needs more investigation as there are contradictory results and unexplored areas.

Following this introduction, the paper is organized as follows: Section 2 presents our review protocol. Section 3 discusses the results of our review, focusing on the definitions of experience and code quality used in the studies and the relationship between them. Section 4 outlines potential threats to the validity of our findings. Section 5 reviews the related work, situating our study within the existing body of research. Finally, Section 6 concludes the paper with a summary of our findings and suggestions for future research directions.

2. Systematic Literature Review Protocol

This section explains the research process used in this study and the overall objectives of the initiative.

2.1. Study Goal and Research Questions

A systematic literature review in software engineering is an approach for collecting, analyzing, and interpreting all available research relevant to a specific research question, topic area, or phenomenon of interest [9]. Systematic reviews aim to present an unbiased evaluation of a research topic by employing a reliable, thorough, and auditable method [9]. They are distinct from traditional literature reviews in their commitment to comprehensiveness and in minimizing bias through exhaustive literature searches and explicit selection criteria [9, 10]. This systematic review adheres to the guidelines specified by Kitchenham and Charters [9], ensuring a detailed and replicable method. Furthermore, the search protocol was designed and a pilot study was conducted in a very similar way

as previous work [11]. The review process involves defining research questions, establishing inclusion and exclusion criteria, searching for studies, assessing the quality of the included studies, extracting data, and synthesizing the findings.

The primary goal of this study is to explore how developer experience impacts on code quality in the field of software engineering. This investigation is driven by the ongoing debate and diverse perspectives within the academic and professional communities about this relationship. To achieve this goal, we created three research questions (RQs) to guide this systematic literature review.

RQ1: How is developer experience evaluated in the context of software engineering?

RQ2: What metrics or methods are used to evaluate software quality?

RQ3: How does developer experience relates to code quality?

Since developer experience and code quality have different and complex meanings, *RQ1* and *RQ2* are meant to be used in conjunction with *RQ3*. Our goal in doing this is to more thoroughly and correctly examine their relationship. It is anticipated that the study's conclusions improve comprehension of these ideas and provide guidance for future research in academia and industry.

2.2. Search String and Selection Criteria

We created a search string and particular selection criteria that are in line with our goal and research questions to gather an extensive and relevant set of literature papers. To ensure that as many relevant papers as possible were found, we ran a preliminary pilot search and then gradually improved the search string. We just used the search string on the metadata, which consists of keywords, abstracts, and titles. We define the search string as follows.

(developer AND (experience OR "years of experience" OR "programming experience" OR "career experience")) AND ("code quality" OR defect* OR "technical debt" OR bug* OR "code smell*")*

In order to include as many research as possible that addressed different aspects of code quality and developer experience, we considered the various terms we included in the search string. Three primary components make up the search string: first, it contains terms associated with developers; second, it contains common terms connected to experience, including years of experience and professional experience; and third, it looks for terms associated with quality, like technical debt and defects. We ensure a comprehensive review of pertinent literature by allowing research addressing related topics and variations to be included by utilizing asterisks (*) as wildcard characters.

Table 1 presents the Inclusion and Exclusion Criteria. With this set of criteria, we aim to highlight research that is both accessible and pertinent to the scope of our study. Our emphasis on Computer Science publications guarantees the technical applicability of the content. Furthermore, our language and format requirements are designed to make the review process easier to use and more reliable. To ensure that the publications addressed the range of our study topics, we only included those that directly compare a developer's experience with code quality. We exclude grey literature and purely theoretical papers to maintain the scientific rigor and empirical focus of the review. By limiting the scope to papers published after 2000, we ensure that the review is based on contemporary understandings and technologies relevant to current software engineering practices.

Table 1. Inclusion and Exclusion Criteria

Inclusion Criteria	Exclusion Criteria
Papers published in Computer Science.	Grey literature.
Papers written in English.	Papers published before 2000.
Papers that investigate the relationship between developer's experience and code quality.	Purely theoretical papers.

2.3. Data Sources and Search

Identifying the right data sources and crafting an effective search strategy are crucial steps in conducting a systematic literature review [9]. This section details the selection of data sources and the method employed in searching for relevant studies.

A search was conducted across multiple data sources to ensure a broad and diverse range of relevant studies. The selected databases are recognized for their extensive collections of scientific and technical papers, particularly in the field of Computer Science and Software Engineering. The following databases were used: ACM Digital Library, IEEE Xplore, Scopus, and Web of Science. These databases were chosen for their relevance on the field, peer-reviewed content, and indexing features, which are essential for conducting a thorough and effective literature review in software engineering. It was essential that the data sources allowed searches on metadata only (title, abstract, and keywords) and the use of logic operators.

We used Zotero [1] to manage the reference and metadata. One of the authors executed all the phases and constantly reviewed the results with the other authors in frequent meetings. When we started the search string was first run through all data sources, 1026 papers were found. After that, the reference manager downloaded the publications metadata. After the first extraction, this review was divided into four stages. Figure 1 shows these phases and overviews the paper extraction from the selected databases.

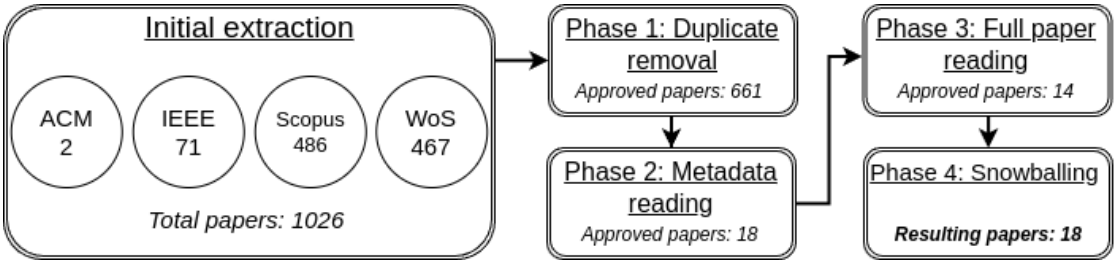

Figure 1. Paper selection and extraction

Phase 1: Duplicate Removal. We removed all duplicated papers from Zotero. After that, the list of studies were sorted by alphabetical order and checked one by one. This first phase removed 365 studies and 661 went to the next phase.

Phase 2: Metadata Reading. In this phase, we read the title and abstract of all 661 papers. We made this phase to verify its adherence to the research questions and to the inclusion/exclusion criteria. The title and abstract were available as part of the initial extraction (we removed 643 papers and 18 went to the next phase).

[1] https://www.zotero.org/

Phase 3: Full Paper Reading. We read the remaining papers. This analysis led to the removal of 4 papers, mainly because they did not adhere to the inclusion/exclusion criteria. The common reasons for exclusion were due to not directly comparing the developer's experience with code quality, or failing to generate measurable conclusions. Consequently, only 14 papers met the criteria and were approved in this phase.

Phase 4: Snowballing. This study analyzed all references from the previous 14 selected papers and triaged them based on their metadata, using the same review process from Phase 1 to Phase 3. After this procedure, the final count of papers found is 18. This step was composed only by a backward snowballing process.

2.4. Data and Metadata Analysis

To respond the research questions and extract information using our search string, we reviewed all pertinent data we had gathered. We looked at the years that the gathered papers were published in order to gain a better understanding of the activity and novelty in this field of study. The majority of the papers were released in 2018 (4), suggesting a lot of activity that year. Figure 2 shows the distribution of the research throughout time. Table 2 presents the 18 selected primary studies that compose this systematic literature review.

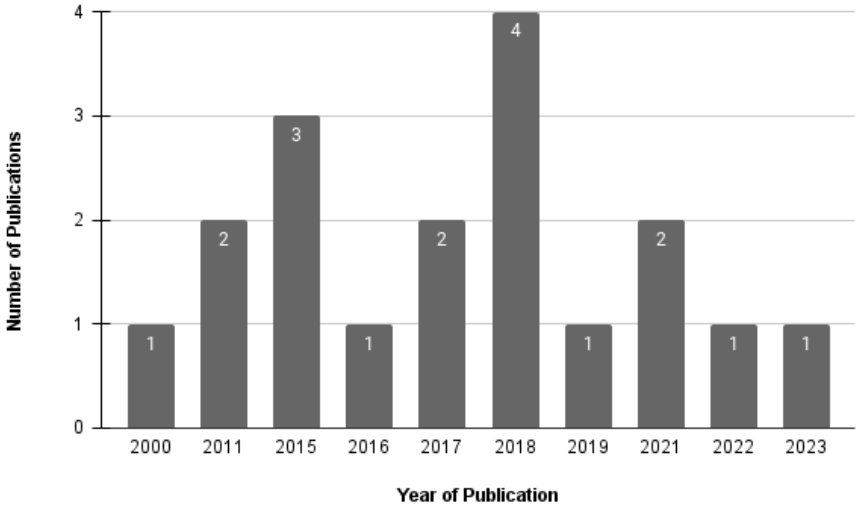

Figure 2. Distribution of the relevant studies by year of publication.

Considering the range of study methods, topics, and formats, we kept a spreadsheet to record the definition of code quality, experience, and corresponding findings of each publication. We classified their definitions of quality, experience, and research outcomes into more general ones based on accepted ideas in the literature so that the results from other studies could be compared. A thematic synthesis, based on the methods described by D. Cruzes et al. [26], was employed to construct dimensions and subdimensions that can represent the studies data.

Table 2. List of papers included.

Authors	Paper Title
González et al.	*A preliminary investigation of developer profiles based on their activities and code quality: Who does what?* [12]
Qiu et al.	*An Empirical Study of Developer Quality* [13]
Tufano et al.	*An empirical study on developer-related factors characterizing fix-inducing commits* [14]
Campos et al.	*An empirical study on the influence of developers' experience on software test code quality* [15]
Alfayez et al.	*An exploratory study on the influence of developers in technical debt* [3]
Piantadosi et al.	*Do attention and memory explain the performance of software developers?* [7]
Eyolfson et al.	*Do Time of Day and Developer Experience Affect Commit Bugginess?* [16]
Dieste et al.	*Empirical evaluation of the effects of experience on code quality and programmer productivity: an exploratory study* [2]
Tsunoda et al.	*Evaluating the work of experienced and inexperienced developers considering work difficulty in software development* [17]
Ando et al.	*How Does Defect Removal Activity of Developer Vary with Development Experience?* [18]
Salamea et al.	*Influence of Developer Factors on Code Quality: A Data Study* [19]
Hokka et al.	*Linking Developer Experience to Coding Style in Open-Source Repositories* [20]
Karimi et al.	*Links between the personalities, styles and performance in computer programming* [21]
Falcão et al.	*On Relating Technical, Social Factors, and the Introduction of Bugs* [22]
Rahman et al.	*Ownership, Experience and Defects: A Fine-Grained Study of Authorship* [23]
Kini et al.	*Periodic Developer Metrics in Software Defect Prediction* [8]
Mockus et al.	*Predicting Risk of Software Changes* [24]
Amanatidis et al.	*Who is Producing More Technical Debt? A Personalized Assessment of TD Principal* [25]

3. Results and Discussion

This section aims to answer RQ1, RQ2 and RQ3 while conducting a discussion over the results. Section 3.1 presents definitions of experience to answer RQ1. Section 3.2 presents definitions of code quality to answer RQ2. Section 3.3 discusses the relationship between experience and code quality found in the relevant studies to answer RQ3. Finally, a brief discussion of the results is included in Section 3.4.

3.1. Definitions of Experience

This section presents the results of our first research question (RQ1), defined as follows.

RQ1: How is developer experience evaluated in the context of software engineering?

Figure 3 illustrates the dimensions and sub-dimensions we identified, along with their respective counts. It is important to note that a study may employ multiple definitions of experience; as a result, the count in a given row can exceed 18. In this figure, each ellipse represents the number of papers we found in each dimension.

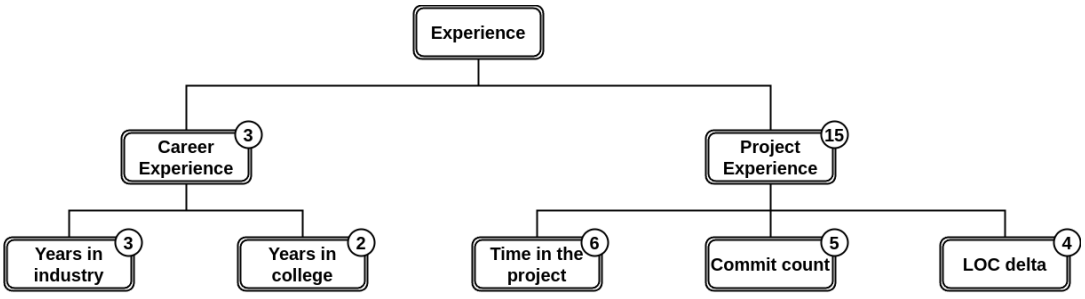

Figure 3. Dimensions and sub-dimensions of a developer's experience and a count of how many papers utilized it. A study can have more than one definition.

The literature frequently operationalizes experience as a theoretical construct in two ways [1, 27]. First, it defines experience as the duration of time spent in the field.

Second, it define it as particular skills and information acquired by active involvement in various tasks. Career Experience and Project Experience are two separate but related concepts that are captured by these operational definitions. We classify the primary studies into one of these two dimensions, notwithstanding the wide range of definitions of experience in the examined papers.

Career Experience is similar to the classic idea of experience as time-based in that it takes into account the chronological amount of time spent in the area [1]. A comprehensive understanding of the field and a depth of knowledge gained over years of working in numerous positions and technologies are correlated with this dimension of experience [28, 29]. This definition was used by three out of the studies in our review [2, 7, 21] . Project Experience, on the other hand, places more emphasis on the volume and variety of work involved in a given project. It reflects the specialized knowledge and problem-solving capabilities developed in real-world situations and includes the practical, hands-on skills gained via direct involvement in software projects [30, 31]. Most of the papers (15) used this concept, albeit they employed various methods for measurement.

Project Experience turns on the breadth and applicability of unique skills to tasks and projects, whereas Career Experience shows the long-term accumulation of knowledge. Understanding how experience appears in a software engineer's career path and how it affects their performance and project success is made possible in large part by this distinction. While some research used numerous sub-dimensions of a single dimension, the majority of studies used one dimension
with one sub-dimension.

Table 3 presents the sub-dimensions we defined in for comparison of results. To maintain comparable results, we restricted our study to only five sub-dimensions and two dimensions, despite the fact that the definitions and calculation techniques for the developer's experience vary greatly and may be further refined in granularity. For instance, the Project Experience sub-dimensions exhibit the most variation among these indicators. For example, some studies solely consider the experience on files that have defects, while others weight metrics based on the timestamp of changes.

Table 3. Comparison of Career and Project Experience Metrics

Career Experience	Project Experience
Years in industry–Refers to the total duration a developer has spent working in the software industry. A total of 3 papers utilized this metric and collected it via a questionnaire.	Time in the project– Tries to quantify the time a developer has been involved with a specific software project. The analyzed papers often calculated this metric using the difference between the date of the first commit and the last commit by a developer. A total of 6 studies utilized this metric.
Years in college– Represents the duration of formal education in software engineering or related fields. This metric was used by two papers.	Commit count– Refers to the number of code contributions (commits) a developer has made to a project's repository. This metric was used by five studies.
	LOC delta– Stands for lines of code delta, which measures the net change in lines of code a developer contributes to a project. The metric was used by 4 of the relevant studies.

RQ1 summary: In the context of software engineering, developer experience is primarily evaluated through 2 dimensions: Career Experience and Project Experience. Career Experience focuses on the chronological time spent in the field, reflecting a broad understanding and depth of knowledge, while Project Experience emphasizes the diversity and volume of work on specific projects, highlighting specialized skills and problem-solving capabilities.

3.2. Definitions of Code Quality

This section presents the results of our second research question (RQ2), defined as follows:

RQ2: What metrics or methods are used to evaluate software quality?

Figure 4 presents the different dimensions and sub-dimensions of code quality classified in the studies. Software quality in software engineering is multifaceted and evaluated using a range of metrics and methods. This systematic review categorizes these into three distinct areas: Technical Debt, Bug, and Code Style. Technical Debt [4] refers to the cost of additional work caused by choosing a faster approach in development rather than a more robust solution. It includes the idea of code smells and the effort required for remediation, reflecting the long-term impact of initial development choices [4].

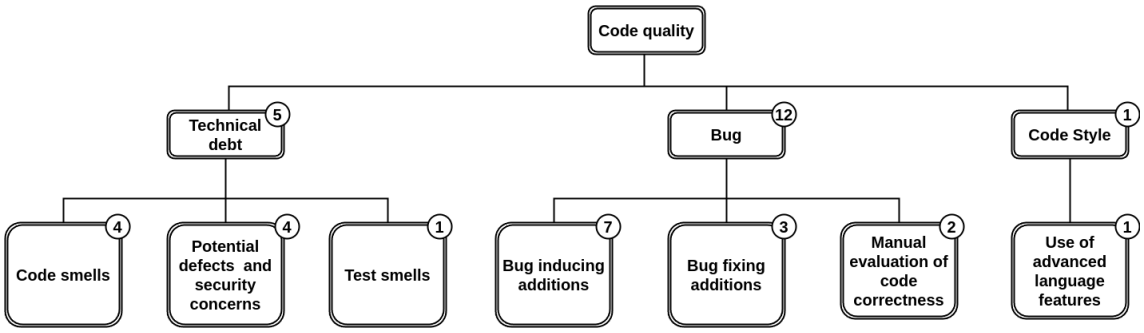

Figure 4. Dimensions and sub-dimensions of a developer's code quality. A study can have more than one definition.

Software quality is assessed through the frequency, severity, and types of software defects in the context of bugs [32]. Bugs indicate direct issues in the software, affecting its functionality and reliability [32]. This study only classified bugs as actual issues in software identified or verified by humans. This definition includes bug tickets in issue tracking systems, code additions considered buggy, or code additions to repair bugs. Possible bugs, such as those reported by static analysis tools or similar probabilistic software, were classified as technical debt as its indication is no certainty of a bug. This definition of bug avoids the well-known miss classification errors from static code analysis tools and restricts its use for actual confirmed problems in software.

Code Style pertains to coding practices that influence the maintainability and readability of the software. This includes adherence to coding standards and using advanced language features, contributing to the software's overall quality [33].

From each dimension of code quality, Table 4 shows the sub-dimensions to better understand and compare how the different studies measure quality. With those results,

most of the studies try to quantify code quality by measuring defects or problems in the code and using that as an inverse measure of quality. With that, the higher the Bug count or Technical Debt count, the less quality a developer's code has. The exception was the study that performed a manual evaluation of code correctness and verified the use of advanced features in the code [20]. However, both alternatives often require expert revision and manual work to obtain.

Table 4. Comparison of Technical Debt, Bug, and Code Style Metrics

Technical Debt	Bug	Code Style
Code smells – Refers to certain structures in the code that suggest a violation of fundamental design principles. Used in four studies.	Bug-inducing additions – Changes in the code base that introduce errors. Used in 7 studies.	Code style– Adherence to coding standards and readability. Used in 1 study.
Potential defects and security concerns – Issues identified by static analysis tools. Considered technical debts in this review. Used in 4 studies.	Bug-fixing additions – Modifications aimed at correcting bugs. Used in 3 studies.	
Test smells – Characteristics in test code indicating potential issues. Used in 1 study.	Manual evaluation of code correctness – Manual code review process. Used in 2 studies.	

RQ2 summary: Software quality in engineering is evaluated through metrics and methods that can be categorized into Technical Debt, Bugs, and Code Style. Technical Debt assesses the long-term impact of initial development choices, Bugs focus on software functionality issues confirmed by human verification, and Code Style examines practices for maintainability and readability.

3.3. Relationship between Experience and Code Quality

After a clear understanding of the definitions of experience and code quality for the reviewed studies, we can answer the RQ3, defined as follows:

RQ3: How does developer experience relates to code quality?

The relevant studies reached different conclusions, and this systematic review classified them into three, as follows: more experience relates to better code quality; more experience relates to worst code quality; it is not possible to establish a relationship. As the classical claim that more experience leads to better performance is only hypothesized, it is imperative that we outline that the studies' empirical results only show evidence of a relationship, or correlation, between the variables experience and code quality. A statistical correlation does not automatically imply causation between the first and second variables [34], meaning that it is not possible to affirm that a developer produces code with high quality because of their experience. However, evidence that they are related is the basis of theoretical work that can potentially explain the observed phenomenon. As there are different dimensions and sub-dimensions for code quality and experience, we aggregated the results based on the type of experience to facilitate the comparison of results.

Career Experience and Code Quality. From the 3 studies that used career experience as a definition of experience, 2 concluded that more experience relates to better

code quality [7, 21] and one did not find any evidence of a relationship between the two factors [2]. Interestingly, no study on this dimension used technical debt or code style as a metric of code quality, only bugs. Given that the experience was manually collected via questionnaire in all these studies, 2 also manually evaluated the code's correctness [2, 7]. Despite having no study pointing out that more years of experience relates to worst code quality, the evidence to support the contrary is sparse, with only 2 studies pointing the opposite and one with inconclusive results.

Project Experience and Code Quality. A majority of the analyzed studies used project experience as a metric for the developer's experience, with 15 studies using it. From those, 8 concluded that more experience relates to better code quality [3, 8, 13, 14, 16, 18, 20, 24], 3 concluded that more experience relates to worst code quality [12, 15, 19] and 4 were inconclusive [17, 22, 23, 25]. From the 8 studies that found more experience is related to better code quality, 6 used bugs [8, 13, 14, 16, 18, 24], and the remaining 2 used technical debt or code style as a metric for quality [3, 20]. The 3 studies that concluded more experience relates to worst code used technical debt [12, 15, 19]. Finally, 3 studies with inconclusive results used bugs [17, 22, 23] and one used technical debit as a metric for quality [25].

The results show a clear predominance in academia that, while using project experience as a measure of a developer's experience and the number of bugs as a metric of quality, the experience of the developer and its code quality are correlated. However, no clear indication of consensus can be drawn by using tech debt or code style as a metric of quality in the scope of project experience.

RQ3 summary: Studies on the relationship between developer experience and code quality show mixed results: some indicate that more experience improves quality, others suggest a negative impact, and a few find no clear link. Evidence highlights a correlation, not causation, between experience and code quality, with varied findings across different metrics of quality.

3.4. Discussion

Figure 5 illustrates the overall findings of the study. The data obtained show conflicting findings - while certain studies identified a positive link between experience and code quality, others reported either a negative correlation or no significant relationship at all. This inconsistency may be attributed to the diverse ways in which experience and code quality are defined and measured across studies, suggesting that the impact of experience on code quality is not straightforward and is influenced by various nuanced factors.

Different types of experience, like career length and project involvement, appear to have distinct impacts on code quality. However, the results are mixed, especially regarding career experience, calling for further investigation into what aspects of experience most significantly affect code quality. Additionally, the varied methodological approaches and quality metrics used in these studies highlight the lack of standardization in this research area. Our study concludes that while a clear evidence exists regarding the positive impact of project experience on code quality, other aspects of the experience-code quality relationship remain ambiguous and asks further research to achieve a more comprehensive understanding about the subject.

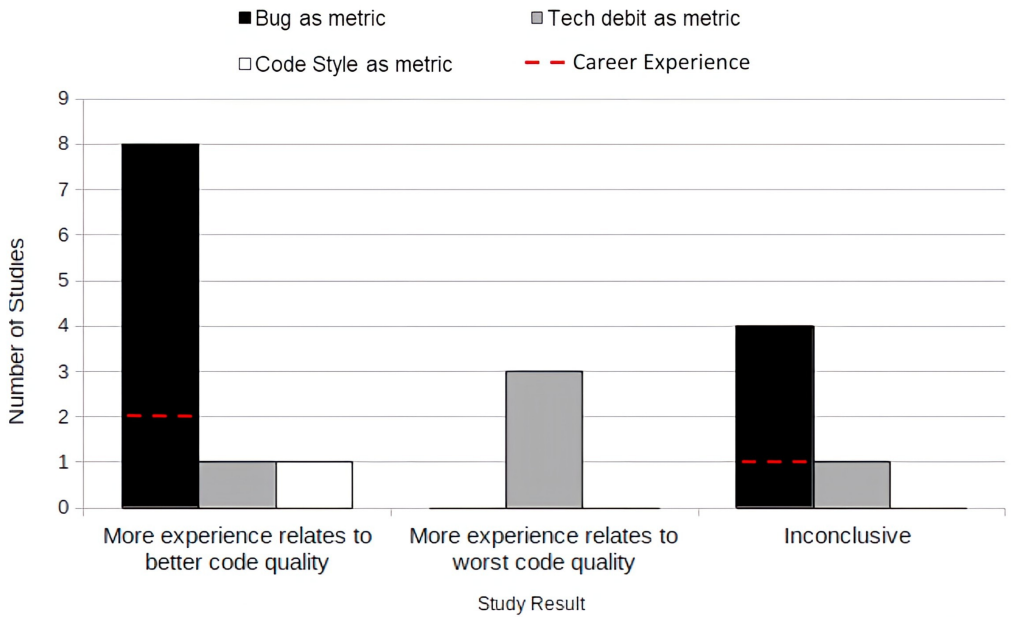

Figure 5. Overall relationship between experience and code quality.

4. Threats to Validity

Even with careful planning, some threats may affect the validity of our findings. This section discusses some threats and our actions to mitigate them, organizing by construct, internal, external, and conclusion validity [35].

Construct Validity: Construct validity concerns the mapping of the results of the study to the concept or theory [35]. For instance, the electronic database selected might not provide all relevant papers. To minimize this threat, we selected four different databases that aggregate papers from multiple publishers. Another threat refers to the search string used which may not return all relevant papers to answer our research questions. However, we tuned our search string by trying several combinations of different combination of terms in the digital libraries.

Internal Validity: Influences that can affect the independent variable to causality [35]. In our study, we favor internal validity by providing a detailed description of the search string, the search engines used, and the inclusion and exclusion criteria. However, one possible threat to internal validity may be related to the judgment and (sometimes subjective) selection of the relevant primary studies. To minimize this threat, the involved researchers have discussed all criteria and selected papers.

Conclusion Validity: Threats to the conclusion validity are concerned with issues that affect the ability to draw the correct conclusion between the treatment and the outcome [35]. In our context, the data summaries extracted from the literature represent the researchers' point of view. To minimize this conclusion threat, we documented all steps to allow replication from external researchers. Moreover, in case of missing or unclear data to answer the research questions, cross-discussions among the paper authors often took place to reach a common agreement.

External Validity: Threats to external validity are conditions that limit our ability to gen-

eralize the results of our paper [35]. For instance, this threat is related to the representativeness of the selected papers to answer our research questions. We used several digital libraries to support a broad representation of the selected papers, but other relevant papers may have been published elsewhere or indexed using unusual terms (not present in our search string). Our results may also not generalize to other perspectives of developer experience or code quality. Therefore, further studies can be performed to complement our findings and investigate possible gaps.

5. Related Work

The experience of the developers may have a direct impact on the software product quality. Besides the code they write, an experienced developer may define architectural solutions essential to the system's maintainability. Some works have studied the impact of developer experience and use metrics to study the relationship with the software maintainability and evolution [36, 37, 38]. R. Brasil-Silva et al. [38], for example, present a catalog of metrics that have been used to quantify developers' experience. They conducted a systematic mapping from 34 relevant papers. These metrics are classified by the type of experience they represent and the purpose of using them. The results showed that most of the selected studies used metrics based on counting developer activities in code. The main difference between this study and our work is that while they provide a comprehensive catalog of metrics for quantifying developers' experience, our study specifically focuses on the relationship between metrics of experience and the direct impact on code quality of developers.

E. Kalliamvakou et al. [36] describe the main challenges to be faced when mining software repositories and propose solutions to alleviate them. Instead, they do not classify nor organize existing papers in terms of such dimensions. Their goal is to help new researchers to conduct well-designed software mining studies rather than to provide a state-of-the-art of the current research in the field. The key distinction between their study and ours is that our research explicitly categorizes and analyzes the impact of developer experience on code quality, which is not the primary focus of Kalliamvakou et al.'s work [36].

T. Carvalho et al. [39] studied machine learning methods for predictive maintenance through a systematic literature review. Their review focused on two scientific databases and provided a valuable foundation for machine learning techniques and their main results, challenges, and opportunities. It also supports new research work in the predictive maintenance field. As a result, the authors identified that each proposed approach addressed specific equipment, making it more challenging to compare it to other techniques. The primary difference between their study and ours is the specific focus on machine learning techniques in predictive maintenance, whereas our study centers on the correlation between developer experience and code quality across various software engineering contexts.

J. Morales et al. [40] conducted a systematic literature review on the Programmer eXperience (PX) over the past ten years (January 2009 to October 2018). Initially, 977 papers were identified, which was refined to 73 relevant papers after applying selection criteria. They appoint key findings. The primary factors influencing PX are the motivation of programmers and the choice of tools they use, such as programming environments. A

majority (59%) of the studies focused on evaluating the influence of Programmer eXperience, User Experience, and usability on programming environments. About 70% of the studies used usability tests and heuristic evaluation methods. Four sets of heuristics were identified for evaluating software development artifacts related to programming environments, languages, and application programming interfaces. The main difference between their study and ours lies in the focus; while Morales et al [40] delve into the broader aspects of Programmer eXperience and its tools, our study specifically targets the impact of this experience on the quality of code produced by individual developers.

6. Conclusion

We selected and reviewed 18 papers, out of a total of 1028, in a systematic manner. To better comprehend the relationship between experience and code quality, we grouped the definitions of each after classifying and evaluating the pertinent publications. Some research indicates that experience leads to better code quality, whereas other studies find no significant correlation or the contrary. This discrepancy most likely results from the different criteria and definitions applied to experience and code quality evaluations. These differences highlight how complex this relationship is, implying that code quality may not always be predicted only by experience.

When considering career and project experience separately, this review suggests they have different impacts on code quality. The inconsistency in findings, particularly with career experience showing limited evidence of a positive correlation, highlight the need for a more detailed understanding of "experience" when measured in years of work or study. This situation raises questions about the specific aspects of experience that influence code quality. These aspects may include the domain of work, the complexity of projects undertaken, or the development methodologies employed.

Finally, the results of this review point to a gap in the existing literature and suggest a direction for future work. Most research relies on bugs as a proxy for code quality, although a very small number of studies use other possible proxy measures. Comparably, few studies use years of experience as a criterion, even though the majority concentrate on project experience. As demonstrated, experience and code quality have a complex relationship that needs more research to effectively guide scholarly discourse and industrial practice. There is general agreement, according to this study, between increased project experience and higher-quality code. Other combinations of code quality and developer experience, however, produce contradictory findings and call for additional data to provide a more comprehensive general academic understanding.

Acknowledgments. This research was partially supported by Brazilian funding agencies: CAPES, CNPq, and FAPEMIG.

References

[1] K. A. Ericsson. *The Cambridge Handbook of Expertise and Expert Performance*. 2006.

[2] O. Dieste et al. 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*, pages 111–112, 2018.

[3] R. Alfayez et al. An exploratory study on the influence of developers in technical debt. In *Proceedings of the 2018 International Conference on Technical Debt*, pages 1–10, 2018.

[4] W. Cunningham. The wycash portfolio management system. In *ACM SIGPLAN OOPS Messenger*, volume 4, pages 29–30, 1992.

[5] E. Giger et al. Comparing fine-grained source code changes and code churn for bug prediction. 2011.

[6] Y. Wang et al. Complying with coding standards or retaining programming style: A quality outlook at source code level. *Journal of Software Engineering and Applications*, 01:88–91, 2008.

[7] V. Piantadosi et al. Do attention and memory explain the performance of software developers? *Empirical Software Engineering*, 2023.

[8] S. Kini et al. Periodic developer metrics in software defect prediction. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 72–81, 2018.

[9] B. Kitchenham et al. Guidelines for performing systematic literature reviews in software engineering. 2, 2007.

[10] P. Brereton et al. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80:571–583, 2007.

[11] L. Olsina et al. Specifying the process model for systematic reviews: An augmented proposal. *Journal of Software Engineering Research and Development*, 7:7:1 – 7:23, Dec. 2019.

[12] C. González et al. A preliminary investigation of developer profiles based on their activities and code quality: Who does what? In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, pages 938–945, 2021.

[13] Y. Qiu et al. An empirical study of developer quality. In *2015 IEEE International Conference on Software Quality, Reliability and Security-Companion*, pages 202–209, 2015.

[14] M. Tufano et al. An empirical study on developer-related factors characterizing fix-inducing commits. *Journal of Software: Evolution and Process*, 29(1):e1797, 2017.

[15] D. Campos et al. An empirical study on the influence of developers' experience on software test code quality. In *Proceedings of the XXI Brazilian Symposium on Software Quality*, pages 1–10, 2022.

[16] J. Eyolfson et al. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 153–162, 2011.

[17] T. Tsunoda et al. Evaluating the work of experienced and inexperienced developers considering work difficulty in software development. In *2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 161–166, 2017.

[18] R. Ando et al. How does defect removal activity of developer vary with development experience? In *SEKE*, pages 540–545, 2015.

[19] M. Salamea et al. 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)*, pages 120–125, 2019.

[20] H. Hokka et al. Linking developer experience to coding style in open-source repositories. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 516–520, 2021.
[21] Z. Karimi et al. Links between the personalities, styles and performance in computer programming. *Journal of Systems and Software*, 111:228–241, 2016.
[22] F. Falcão et al. On relating technical, social factors, and the introduction of bugs. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 378–388, 2020.
[23] F. Rahman et al. Ownership, experience and defects: A fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500, 2011.
[24] A. Mockus et al. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.
[25] T. Amanatidis et al. Who is producing more technical debt? a personalized assessment of td principal. In *Proceedings of the XP2017 Scientific Workshops*, pages 1–8, 2017.
[26] D. Cruzes et al. Recommended steps for thematic synthesis in software engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*, 2011.
[27] B. Curtis et al. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.
[28] W. Chase et al. The mind's eye in chess. 1973.
[29] K. Ericsson et al. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, 100(3):363–406, 1993.
[30] R. Jeffries et al. The processes involved in designing software. 1981.
[31] K. McKeithen et al. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13(3):307–325, 1981.
[32] M. Hamill et al. Common trends in software fault and failure data. *IEEE Transactions on Software Engineering*, 35:484–496, 2009.
[33] A. Mohan et al. Programming style changes in evolving source code. *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.*
[34] N. Altman et al. Association, correlation and causation. *Nature Methods*, 12:799–800, 2015.
[35] C. Wohlin et al. *Experimentation in Software Engineering*. Heidelberg, 2012. ISBN 978-3-642-29043-5.
[36] E. Kalliamvakou et al. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101, 2014.
[37] K. Stol et al. Reporting empirical research in open source software: The state of practice. In *Open Source Ecosystems: Diverse Communities Interacting*, pages 156–169, 2009.
[38] R. Brasil-Silva et al. Metrics to quantify software developer experience: A systematic mapping. *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022.
[39] T. Carvalho et al. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers Industrial Engineering*, 137, 2019.
[40] J. Morales et al. Programmer experience: A systematic literature review. *IEEE Access*, 7:71079–71094, 2019.