



DOI: <https://doi.org>

# ComputeX - Journal of Emerging Technology & Applied Science

Journal homepage: <https://rjsaonline.org/index.php/ComputeX>



## Code Review Quality and Software Reliability: The Mediating Role of Defect Detection and the Moderating Role of Developer Experience

Ayesha Khan<sup>1</sup>

<sup>1</sup> Department of Software Engineering, University of Central Punjab, Lahore, Pakistan

Email: [ayeshak99@hotmail.com](mailto:ayeshak99@hotmail.com)

### ARTICLE INFO

### ABSTRACT

#### Received:

March 06, 2026

#### Revised:

March 28, 2026

#### Accepted:

April 23, 2026

#### Available Online:

May 06, 2026

#### Keywords:

Code Review Quality;  
Defect Detection;  
Developer Experience;  
Software Reliability;  
PLS-SEM; Mediation;  
Moderation

#### Corresponding Author:

[ayeshak99@hotmail.com](mailto:ayeshak99@hotmail.com)

This study aims to explore how the quality of code review relates to software reliability, while focusing on the role of defect detection as a mediator and developer experience as a moderator between code review and software reliability in an information technology (IT) organization with the software engineers and software quality assurance (QA) teams. The researcher used a quantitative approach and explanatory research design to gather the data from the software engineers and QA professionals using validated multi-item scale of code review quality, defect detection, developer experience, and software reliability. To assess data's validity and reliability, first data validation was performed, second confirmatory factor analysis was performed to ensure the consistency of data measurements across constructs. Partial Least Squares Structural Equation Modeling (PLS-SEM) was used to analyze the relationships between the variables in this study because of their hierarchical, second-order and interaction-based forms. The impact of defect detection on software fault was estimated using bootstrapped indirect effects, while the moderating effect of developer experience in the relation between software and defect parameters was analyzed using a hierarchical interaction modeling approach within the PLS framework. In addition to traditional path significance testing, two measures (effect size ( $f^2$ ) and predictive relevance ( $Q^2$ )) were introduced to assess the strength of the model. All constructs showed good levels of reliability, convergent and discriminant validity of the measurement model. There was a significant positive effect of code review quality on both defect detection and software reliability, a significant positive predictiveness of software reliability on defect detection, and there was a partial mediation between code review quality and software reliability. It was determined that the relationship between defect detection and software reliability was significantly moderated by whether the developer was highly or less experienced, and was higher in the cases of high experience. The results indicate that such translation of code review investment into software reliability works significantly, but mainly through defect detection, and that this translation depends both on the level of experience of the development team and on the contraction-level of the code, implying software organizations may be able to optimize code review investment and team by substituting experienced developers with less experienced ones, or by implementing non-experimental code reviews on less criticized layers of the contraction.

## Introduction

Software reliability is one of the most persistent and consequential challenges in information technology organizations that has caused significant financial costs, security vulnerabilities and reputational damage when software defects are not discovered in deployment environments prior to reaching the production environment (Boehm & Basili, 2001). Over the years, software systems have been expanding in scope, complexity and interconnectedness and the cost of finding and fixing a software defect has been proven to rise sharply the deeper it goes into the development process, as documented in software engineering economic research (Boehm & Basili, 2001) and why an organization can invest in running earlier and more intensive quality assurance procedures. Among the most common quality assurance techniques used in modern software development is code review, a systematic examination of source code carried out by developers other than the original developer before it is integrated with other code or committed to other systems (e.g., automated testing or production environments; Bacchelli & Bird, 2013).

In spite of many efforts in establishing code review processes, the link between code review quality and the software reliability elsewhere is seldom direct and unambiguous. The value of code review is not guaranteed; it is only when the code review has the ability to find defects that would otherwise remain untouched in subsequent stages of development or production (Runeson et al., 2006). Therefore, defect detection (the fraction of the total detected defects at a particular review or testing step) is the mechanism that couples the quality of code review with software reliability, and is not directly and unconditionally related to it (Wohlin, Runeson, & Brantestam, 1994). Owners of a process and tool investment might invest significantly in software systems and software assurance processes and tooling, and achieve little to no gains in software systems reliability, for any of a variety of reasons that the process can be performed superficially, reviewers may not be participating in the process at the level needed, reviewers may lack sufficient domain knowledge, etc.

Meanwhile, performance of code review and code review defects isn't going to be the same for everyone involved in the development and testing teams. In software engineering practice, accumulated domain knowledge, knowledge about the codebase, and the director's debugging and review intuition gained over years of experience are better understood to be strong factors that influence the quality of the code and defect rates (Mockus, Fielding, & Herbsleb, 2002). While the effect of developer experience on software reliability is seen in its own right, it may condition (Curtis et al., 1988) or moderate (Krasner & Curtis, 1989) the strength of a relationship between defect detection and software reliability; in other words, the same degree of defect detection may result in different degrees of software reliability depending on the level of developer experience in the team applying the defect detection.

Although it is plausible that this mediated and moderated structure is valid, very few empirical studies have independently validated defect detection as a mediator between code review quality and software reliability, and developer experience as a moderator for the downstream portion of this mediated relationship simultaneously, within a single integrated model. Existing software engineering research has paid attention on code review practices and software reliability as a direct and bivariate relationship (Runeson et al., 2006), as well as on developer experiences and code quality in isolation of code review (Mockus et al., 2002), but important questions remain uninvestigated on how these mechanisms work jointly within the same causal structure. This is an important gap, as organizations interested in increasing the reliability of their software must consider the amount and structure of a code review process in order to ensure that the investment in code reviews optimizes the software reliability increase that it delivers.

Since the proposed model is hierarchical and second order and contains interaction that are especially suitable to be included in a single model, and the mediation and moderation variables pose fewer distributional assumptions compared to covariance-based SEM approaches, which is why PLS-SEM is a highly appropriate method for this study (Hair, Risher, Sarstedt, & Ringle, 2019). In this model of PLS-SEM, the study contributes to understanding the data path of code review investment to software reliability, as it allows defect detection to be an explicit confounder and experience of the developers to be an explicit moderator variable, providing a more complete and methodologically rigorous explanation of how and when code review investment leads to software reliability outcomes.

This study has the following 4 objectives: (1) to explore direct effect of the code review quality on software reliability; (2) to explore the effect of the code review quality on defect detection and defect detection on software reliability; (3) to test the mediating role of the defect detection between the code review quality and software reliability; and (4) to test the moderating role of developer experience between the defect detection and software reliability. This study is significant and contributes to the software engineering literature in the following aspects: Mediation and moderation approaches have been integrated in the same PLS-SEM model and rigorous measurement validation procedures, such as CFA, reliability and validity measurement, have been applied in this study and model strength assessed using effect size and predictive relevance metrics,

applicable to the PLS-SEM approach. In practice, the work suggests specific directions for how to plan a software investment and stakeholder team for code review to deliver the greatest gains in software reliability.

## **Literature Review**

In general, the term Code Review refers to a systematic and structured way for developers to examine source code, often done by another developer before the change is incorporated into the common source code of a software development organization (Bacchelli & Bird, 2013). Bacchelli and Bird (2013) conducted a seminal field study at Microsoft, as current practitioners and managers of code review state that code review serves a variety of “secondary” benefits to them beyond just defect detection: it is used to gain knowledge, share “project awareness,” and discover alternative or better solutions, meaning that the quality of code reviews is a multidimensional construct that goes beyond the rigor of searching out defects, also representing the level of substantive engagement with design and maintainability issues. Software defect economics: Boehm and Basili (2001) offers basic research that suggested that as defects are found later in the software development lifecycle the cost to correct them tends to increase rapidly, a finding that has sparked decades of software defect economics research targeted at identifying defects early to increase the likelihood that they will be corrected. ( Beck, 2003)

In fact, it has been investigated and studied as a construct separate from review or testing process, because the amount of review activity does not necessarily mean that it delivers or results in proportional defect detection success (Wohlin, Runeson, & Brantestam, 1994). Runeson et al. (2006) performed a comparative study of various types of defect detection techniques, such as testing, inspection, and reading techniques, and discovered that defect detection during reviews or test varies significantly among the techniques, in the context of each technique, and with respect to the amount of review effort or intensity. McIntosh et al. (2014) took this body of research into a modern context of code review and found that a significant relationship exists between the extent of how the code is covered via code review—how much of it gets reviewed—and post-release code defect density as well as the number of defects that get found and are changed by reviewers—how many are modified, how many are fixed—in independent bases of proof. These findings led them to conclude that treating the defect-detection as a mechanism by which the quality of code reviews converts into reliability outcomes is warranted, not assuming a direct and unmediated relationship between review actions and reliability. A large-scale empirical study on open-source projects by Bavota and Russo (2015) confirmed that code reviews that took place over a wider number of lines were associated with significantly reduced defect densities after releases.

Properly, in the software engineering literature software reliability has been conceptualized along various dimensions – such as defect density in production, mean time between failures, and the extent to which it meets with functional and non-functional requirements – and empirical software quality measurement research has found that structural and design-level measures of code quality are themselves highly correlated with defect proneness (Basili, Briand, & Melo, 1996; Kitchenham & Pfleeger, 1996). However, empirical studies that have tried to establish a link between code review practices and software reliability outcomes have so far provided a positive association between the two, with some studies observing stronger associations than others, likely because many of these studies have not measured the intervening role of defect detection in its own right, thereby potentially introducing noise into the relationships reported (McIntosh et al., 2014). Implicit or explicit measurements of defect detection have typically shown that there is a positive direct correlation with software reliability (observed) measures such as defect density after release and failure in the field, which supports the hypothesis that defect detection is also an important process link between code review quality and reliability results (Wohlin et al., 1994).

A considerable amount of work has been done as to the developer experience as a sole metric of good code quality and defect rates, and there is strong empirical evidence that more experienced developers are less likely to have defects in their code and to be more efficient code reviewers, partly because of their accumulated domain knowledge, more sophisticated mental models of the codebase, and more efficient debugging heuristics gained over the years of practice (Mockus, Fielding, & Herbsleb, 2002). In an early and influential field study of software design practices, Curtis, Krasner and Iscoe (1988) concluded that one predictor of a successful software development team was depth of knowledge of the application domain, commonly related to programmer experience. However, Bird, Nagappan, Murphy, Gall and Devanbu (2011) showed more empirical detail that software defect density for certain modules was significantly related to the developer who had written the code, as well as their level of familiarity with such modules, which corresponded to their level of developmental experience; some of the developers were able to provide high-quality feedback during code reviews, others provide low-quality feedback.

Focusing more on the ultimate developer user experience, however, has received significantly less research interest as to its moderating effect on the relationship between defect detection and software reliability, distinct from its own predictive effect on software reliability. This differentiation has practical significance; someone says that a common level of defect detection – same fraction of defects found during a specific review method, say – would yield more or less meaningful software reliability

outcomes, depending on the developer expertise involved in understanding and reacting to the detected defects. If this is true, then theoretically it should be possible for developers who have more experience to catch a defect at the time of review and pinpoint the root cause, and then be able to make fixes that are both more effective and less likely to introduce other bugs into the software, and end up seeing related latent defects as a result of the defect they detected, which would strengthen the overall capability benefits of catching them when they first appeared. Similarly, Rigby and Bird (2013) compared code review practices of several large open-source and industrial projects, and observed that both the type of review feedback given and its subsequent impact on code quality depended on the reviewer's experience and familiarity with the code being reviewed, which supports our proposed moderator, developer experience.

From the methodological perspective, using the Partial Least Squares Structural Equation Modeling (PLS-SEM) approach has become one of the commonly used methods in testing models that encompass mediators and moderators because PLS-SEM has a relatively good method for small samples (Hair et al., 2019), supports second order constructs both formative and reflective (Chin, 1998), and has a prediction-oriented estimation process based on a purely covariance fitting (Hair et al., 2019). In the measurement model assessment in the present study Henseler, Ringle, and Sarstedt (2015) acknowledged that correlations using the heterotrait-monotrait approach are a more feasible criterion for discriminant validity in the PLS-SEM framework than those applied earlier based on the Fornell-Larcker criterion.

On the basis of this theory and evidence, the following hypotheses were developed. In line with the software engineering community's understanding of the value of code review for software quality assurance (Bacchelli & Bird, 2013), the first hypothesis (H1) is: There is a significant positive relationship between code review quality and software reliability. Hypothesis 2 (H2): The higher the quality of code review, the higher the likelihood that defects will be discovered in the code. Hypothesis 2 (H2) states that there is a positive relationship between the quality of code review and defect detection, as code review helps identify latent defects before they are integrated into the system (McIntosh et al., 2014). Hypothesis 3 (H3), which is supported by evidence that defect detection and remedial action directly decrease production defect density and field failure rates (Wohlin et al., 1994), is that there is a significant positive correlation between defect detection and software reliability. Hypothesis 4 (H4) suggests that code review quality and software reliability are related in a manner that has defect detection as a mediator link between these two variables, capturing the synthesis of the above hypotheses (i.e., H1, H2 and H3). Finally, Hypothesis 5 (H5) suggests that the experience of the developer in Software Development has an effect on the link between defect detection and software reliability, with the link being stronger for more experienced developers, hence, experience amplifies the reliability benefit obtained based on a given amount of defect detection (Mockus et al., 2002; Rigby & Bird, 2013). All put together, these hypotheses provided the theoretical foundation of PLS-SEM mediation and moderation analyses that are discussed in the next methodology and analysis section.

## **Methodology**

### **Research Design**

A quantitative explanatory research design was used in this investigation to analyze the correlation between code review quality and software reliability, the intervening position of the phenomenon of defect detection and the moderating role of developer experience. The reasoning behind the choice of explanatory design was that the main interest in the study was to test a hypothesized causal pattern between the variables under investigation, rather than just providing a descriptive, or anecdotal, account of development practice as is commonly found in the software engineering research methodology literature (Sjøberg et al., 2005). The sampling unit was the individual software engineer and the quality assurance (QA) team working in information technology (IT) organizations for which data were gathered.

### **Population and Sampling**

The profiles in the target population included software engineers, software developers, software quality assurance professionals engaged in software development and software code review activities within an IT organization. All participants were required to have at least one year professional software developer or QA experience, have to be part of a team that practiced formal or semi-formal code review in its software development process. After screening, 320 valid responses were collected and kept for analysis because they had either completed the questionnaire or they were corrected due to the inclusion of attention-check items in the data answers.

### **Measures**

All constructs were operationalized by using multi-item reflective scales that were developed as a result of the literature search conducted on measurement scales from software engineering and information systems. There were also slight changes in the wording to make it specific to evaluating code review, described below. We evaluated code review quality with a set of

items that quantify the extent of code review conducted by the respondent's team, the consistency, and the substance of code review practices, similar to previous code review process measurement studies (Bacchelli & Bird, 2013). Defect detection was evaluated based on the extent to which the team's review and test efforts were able to detect defects before delivery in terms of the perceived percentage of defects detected over the course of the year and their consistency. Developers' experience was determined through the assessment of years of professional software development experience as well as a domain and codebase familiarity scale previously successfully used in operationalizing developer experience in software engineering research (Mockus et al., 2002). The dependent construct that was measured was software reliability, where items measured perceived post-release defect frequency, system stability, and conformance with software reliability expectation. Items measured on a Likert scale were rated on a 5-point scale of 1 (strongly disagree) to 5 (strongly agree).

### **Data Collection and Validation Procedure**

A structured online questionnaire approach was used to gather data with the target population of software engineers and QA professionals in participating IT organizations and with available self-progressively formed lists. Data was collected by distributing a structured, self-administered online questionnaire to eligible software engineers and professionals available in the participating IT organizations with self-progressively formed lists. The questionnaire was pilot tested with a small group of practicing developers to clarify items and ensure that they were relevant, and some minor changes were suggested as a result of the pilot test that were implemented. After data had been collected, the data were then screened for missing data and subsequently for straight-lining and inconsistent response patterns that would suggest inattentive responding (outliers and data validation). At the initial validation step, reliability assessment (Cronbach's alpha and composite reliability) was employed as a preliminary evaluation of the measurement's quality, before the ensuing confirmatory factor analysis when a more extensive test was conducted.

### **Data Analysis Technique**

Initial data validation and reliability assessment led to confirmatory factor analysis (CFA) measuring the consistency of measurement across the constructs before the estimation of the structural model, making that sure of the standardized factor loadings, composite reliability and average variance extracted of the constructs. The structure of the proposed model on structural relationships was constructed in a hierarchical, second order, and interaction fashion, and considering the less number of distributional assumptions in PLS-SEM approaches for prediction-oriented research, as well as its recognized suitability for modeling structural relationships involving mediation and moderation (Hair et al., 2019), the method of PLS-SEM was employed. To assess the discriminant validity, the heterotrait-monotrait ratio of correlations (HTMT) was computed, while variance inflation factor (VIF) was calculated to detect multicollinearity among all the variables in the structural model. Consistent with established guidelines for testing mediation in regression and structural equation modeling settings (Hayes, 2017), bootstrapping was used to obtain 95% bias-corrected confidence intervals for the bootstrapped indirect effect, which allows for testing the mediating role of defect detection between code review quality and software reliability. To explore the moderation effects of the developer experience on the relationship between defect detection and software reliability, the moderation term was added to the structural model of the PLS model as a product indicator, and the relevant moderating variables were mean-centred first, which helped to reduce the multicollinearity and the interpretation results. Beyond the traditional path coefficient significance testing by bootstrapping, model strength was assessed for each structural path by computing the effect size ( $f^2$ ) and for each endogenous construct by computing their predictive relevance ( $Q^2$ ) with the blindfolding procedure.

## **Analysis and Results**

### **Respondent Profile**

A sample of software professionals who answered the questionnaire—a group of software engineers and/or software developers (61.6%), software quality assurance professionals (28.1%) and respondents with both development and QA duties (10.3%)—was retained for analysis as part of the 320 (300 valid) items that were not discarded in the process. For subsequent moderation analysis purposes, respondents were averaged for 6.4 years of professional software development experience ( $SD = 4.2$ ), with 46.9% being more experienced (5 or more years of experience) and 53.1% as less experienced (fewer than 5 years of experience). The repartition of the sample with respect to their professional profile is summarised in Table 1.

**Table 1. Respondent Professional Profile (N = 320)**

| Category                      | Classification                | Frequency | Percentage (%) |
|-------------------------------|-------------------------------|-----------|----------------|
| Role                          | Software Engineer / Developer | 197       | 61.6           |
|                               | QA Professional               | 90        | 28.1           |
|                               | Combined Dev / QA             | 33        | 10.3           |
| Experience Level              | Less than 5 years             | 170       | 53.1           |
|                               | 5 years or more               | 150       | 46.9           |
| Organization Size             | Small (< 100 employees)       | 88        | 27.5           |
|                               | Medium (100-999)              | 146       | 45.6           |
|                               | Large (1,000+)                | 86        | 26.9           |
| Code Review Practice Maturity | Informal / Ad Hoc             | 79        | 24.7           |
|                               | Formal, Tool-Supported        | 241       | 75.3           |

Source: Survey data (2026).

### Data Validation and Reliability Testing

Data were validated through procedures prior to substantive analysis and were excluded from the analytic pool of 338 responses received to arrive at the analytic sample of 320 responses. As can be seen in Table 2, preliminary reliability data, using Cronbach's alpha, showed satisfactory internal consistency for all of the constructs, with all alpha's greater than 0.70.

**Table 2. Initial Reliability Testing**

| Construct            | No. of Items | Cronbach's Alpha |
|----------------------|--------------|------------------|
| Code Review Quality  | 6            | 0.864            |
| Defect Detection     | 5            | 0.838            |
| Developer Experience | 4            | 0.812            |
| Software Reliability | 5            | 0.847            |

### Confirmatory Factor Analysis

Prior to estimating the structural model, consistency of measurement across constructs was confirmed by conducting confirmatory factor analysis. All the standardized factor loadings are >0.70, the average variance extracted (AVE) is >0.50 for all constructs, and composite reliability (CR) is >0.85 for all constructs as presented in Table 3 and hence it gives strong evidence to convergent validity (Hair et al., 2019).

**Table 3. Confirmatory Factor Analysis: Convergent Validity**

| Construct            | Items | Loading Range | CR    | AVE   |
|----------------------|-------|---------------|-------|-------|
| Code Review Quality  | 6     | 0.738-0.847   | 0.901 | 0.604 |
| Defect Detection     | 5     | 0.721-0.832   | 0.882 | 0.601 |
| Developer Experience | 4     | 0.712-0.818   | 0.864 | 0.614 |
| Software Reliability | 5     | 0.746-0.851   | 0.892 | 0.622 |

Note. CR = Composite Reliability; AVE = Average Variance Extracted.

Discriminant validity was researched via the heterotrait-monotrait ratio of correlations (HTMT) which was a better criterion than the classical Fornell-Larcker for PLS-SEM (Henseler, Ringle, & Sarstedt, 2015). The results indicate that all HTMT values were < 0.85, Thus the constructs were empirically separable from each other (See Table 4). The variance inflation factor (VIF) was used to check for possible problems of multicollinearity in the structural model predictors as shown in Table 5 and no predictor had VIF>3.0 suggesting that multicollinearity was not a threat for the estimates of the structural model.

**Table 4. Discriminant Validity (HTMT Ratios)**

| Construct               | 1     | 2     | 3 |
|-------------------------|-------|-------|---|
| 1. Code Review Quality  | —     |       |   |
| 2. Defect Detection     | 0.624 | —     |   |
| 3. Software Reliability | 0.658 | 0.671 | — |

**Table 5. Multicollinearity Diagnostics (Inner VIF Values)**

| Path                                                                        | VIF   |
|-----------------------------------------------------------------------------|-------|
| Code Review Quality →Defect Detection                                       | 1.000 |
| Code Review Quality →Software Reliability                                   | 1.642 |
| Defect Detection →Software Reliability                                      | 1.598 |
| Developer Experience →Software Reliability                                  | 1.284 |
| Defect Detection × Developer Experience (Interaction) →Software Reliability | 1.317 |

#### Structural Model Results: Direct and Mediation Effects

The structural model showed good level of explanatory power, in particular with regard to defective detection, explaining 35.8% of the variance ( $R^2 = 0.358$ ) and better with regard to software reliability, 46.2% ( $R^2 = 0.462$ ). For both software reliability (0.287) and defect detection (0.224),  $Q^2$  values were greater than zero, respective to software reliability and defect detection, showing a satisfactory predictive relevance (Hair et al., 2019). The results of the structural path for Hypotheses 1–3 can be seen in Table 6, along with the effect sizes ( $f^2$ ).

**Table 6. Structural Model Results: Direct Effects**

| Hypothesis | Path                                      | $\beta$ | t-value | p-value | $f^2$ | Decision  |
|------------|-------------------------------------------|---------|---------|---------|-------|-----------|
| H1         | Code Review Quality →Software Reliability | 0.296   | 5.218   | 0.000   | 0.118 | Supported |
| H2         | Code Review Quality →Defect Detection     | 0.598   | 12.104  | 0.000   | 0.554 | Supported |
| H3         | Defect Detection →Software Reliability    | 0.384   | 6.742   | 0.000   | 0.187 | Supported |

Note. Bootstrapping based on 5,000 resamples; two-tailed test; significance level  $p < 0.05$ .

To test Hypothesis 4, the interaction between code review quality and software reliability in the presence of defect detection as the intervening variable was analysed using bootstrapped indirect effects. To test Hypothesis 4, the interaction between code review quality and software reliability on the presence of defect detection as intervening variable was analysed using bootstrapped indirect effects. Inspection of the indirect effect showed that the quality of code review was statistically significant (with a value of  $\beta = 0.230$ ,  $t = 5.462$ , and a 95% bias-corrected confidence interval (0.148, 0.318)) in explaining the relationship between software reliability and the software developer. The mediated relationship was significant (H2), suggesting that the quality of code review is a full mediator in the relation between code review quality and software reliability with a variance accounted for (VAF) of 43.7%, and the direct effect of code review quality on software reliability (H1) was still represented in the results. The mediation analysis is summarized in Table 7.

**Table 7. Mediation Analysis Results**

| Relationship                                                | Direct Effect ( $\beta$ ) | Indirect Effect ( $\beta$ ) | t (Indirect) | 95% CI         | VAF (%) | Type              |
|-------------------------------------------------------------|---------------------------|-----------------------------|--------------|----------------|---------|-------------------|
| Code Review Quality →Defect Detection →Software Reliability | 0.296**                   | 0.230**                     | 5.462        | [0.148, 0.318] | 43.7    | Partial Mediation |

Note. \*\*  $p < 0.001$ .

#### Moderation Analysis: Developer Experience

To explore intermediating role of developer experience between defect detection and software reliability, the moderating role of developer experience was investigated through the path of introducing mean centered product-indicator interaction term (Defect Detection × Developer Experience) into the structural model via hierarchical interaction modeling in the context of PLS. As revealed in Table 8, the interaction term is positive and significant ( $\beta = 0.184$ ,  $t = 3.246$ ,  $p = 0.001$ ,  $f^2 = 0.042$ ); thus the p value is less than 0.05, suggesting that Defect detection x Developer experience has a significant effect on strengthening the link between defect detection and software reliability thereby supporting Hypothesis 5 (H5). On the simple slope analysis, it shows the defect detection-software reliability path to be significantly stronger if the developer has more experience (simple slope = 0.486,  $p < 0.001$ ) than if the developer has less experience (simple slope = 0.298,  $p < 0.001$ ), as can be seen by the slope difference reported in Table 8.

**Table 8. Moderation Analysis Results (PLS-SEM Interaction Modeling)**

| Path / Effect                                                  | $\beta$ | t-value | p-value | f <sup>2</sup> | Decision     |
|----------------------------------------------------------------|---------|---------|---------|----------------|--------------|
| Developer Experience → Software Reliability (direct)           | 0.162   | 2.918   | 0.004   | 0.038          | Significant  |
| Defect Detection × Developer Experience → Software Reliability | 0.184   | 3.246   | 0.001   | 0.042          | H5 Supported |
| Simple Slope: High Experience                                  | 0.486   | 8.124   | 0.000   | —              | —            |
| Simple Slope: Low Experience                                   | 0.298   | 5.012   | 0.000   | —              | —            |

Note. Interaction term constructed using mean-centered indicators prior to product term formation, consistent with standard PLS-SEM moderation procedures.

Combining these results it can be concluded that all five hypotheses are supported. The variable of developer experience partially mediated the effect of code review quality on software reliability (H4) and significantly moderated the effect of defect detection/warning on software reliability (H5), whereas the latter variables had a significantly positive direct effect on both defect detection (H1) and software reliability (H2). The satisfactory effect sizes and predictive relevance (Q<sup>2</sup>) values for all the endogenous constructs further corroborate the strength and predictive validity of the estimated model.

## Discussion

The results of this study offer strong empirical evidence for a mediated and moderated model between code review quality and software reliability amongst software developers and JQA professionals in IT organizations. The significant direct impact of code review quality on software reliability (H1) and strong impact on defect detection (H2) provides further evidence for the key, long acknowledged value of code review as a quality assurance activity (Bacchelli & Bird, 2013), and extends this evidence with a validated, multi-item measurement approach and beyond single-item or purely archival proxies for review activity. Nearly half of the variance in code review quality explains software reliability, but most of the prediction of software reliability is indirect through defect detection as confirmed by H4, which illustrates the mediating role of code review in detecting defects, with a mediating proportion of about 43.7%. This is a significant explanation of the relationship between code review and software reliability, and confirms that much about code review's impact on software reliability is mediated by its effect on defect detection.

It is an important extension to this mediated model to acknowledge the moderating role of developer experience (H5), as confirmed in this study. The simple slope is significantly higher for the more experienced developers (0.486) than it is for the less experienced ones (0.298), suggesting that the same proportion of defects turned in is associated with a significantly larger reliability benefit when understood and implemented by more experienced team members. This result complements and extends the existing research on how developer experience and familiarity with code influences defect rates and the quality of reviews (Mockus et al., 2002; Bird et al., 2011; Rigby & Bird, 2013) by showing a specific impact: experience enhances the downstream results of the translation of defects into reliability improvements, not just as an autonomous and parallel influence on reliability.

The implications of these findings apply to the practice of IT and engineering management. In conclusion, while defect detection is partially and not fully mediating the relationship between code review and reliability, organizations should not base their code review maturation efforts completely on raising their defect detection rates, but also intentionally focus on the other practices that rely on code review that also have a strong direct impact on reliability, such as structured feedback and knowledge sharing, as shown in this study. The moderation results also indicate that the reliability returned for a given level of capability for detection is significantly affected by the level of experience of the team members doing the code review, suggesting that organisations interested in maximizing the reliability pay-off from their code review investment should either pair their less experienced developers with more experienced reviewers—or structure the code review assignments such that the defects being detected were interpreted and addressed by an appropriate level of experience within the team.

## Conclusion

The research investigated the relationship between code review quality and software reliability. A mediator test was performed to ascertain the validity of code review for software reliability and a moderator test was performed to understand the influence of developer experience with code review on the relationship between code review quality and software reliability. The research was conducted in IT organizations involving 320 Software Engineers and QA Professionals. The results showed that the QCR is found to be significantly and positively related to the D and SR, defect detection is found to be significantly related to the SR and it partially mediates the relationship between the QCR and SR, and developer experience is

found to be significantly related to the relationship between the D and SR, with the relationship being found to be stronger among more experienced developers. These results validate the importance of modeling the impact of software engineering code review quality on software reliability in this way, as opposed to as a direct effect, and provide methodologically sound and substantiated results, based on the effect size approach and the metric of predictive relevance which are appropriate for PLS-SEM, to fill a software engineering technology body of knowledge that has often viewed these constructs as separate entities.

## **Recommendations**

Based on the very positive relationship between the quality of code reviews and the detection of defects and the reliability of the software, 1) IT organizations should invest in structured, formal code review systems rather than informal or ad hoc code review systems.

Expert Developer Reviewers should collaborate with less experienced Developers in the assignments of code reviews, with the notable exception that the gain in reliability from a particular rate of defect detection becomes many times greater with increased developer experience.

Organizations must consider defect detection as an important intermediate code review quality indicator in their code review processes (shown to be the most important of all, but not the only one, of the code review quality indicators that lead to software reliability, though not exclusively) and should follow rigorous practices of both reporting and evaluating the defect detection rate in their code review, based on current software engineering practices for defect-related prediction and measurement model validation (Tantithamthavorn, McIntosh, Hassan, & Matsumoto, 2017).

Beyond defect detection, organizations need to acknowledge and support code review's complement to defect detection: the knowledge-transfer and design-improvement aspect of code review also has a significant direct impact on reliability, apart from defect detection.

Professional development programs should focus on acquiring developer experience and familiarity with the code as well, including mentorship and continuity with code ownership, since experience has generated clear returns in reliability for effective defect detection.

Future studies should apply longitudinal designs to account for the temporal dynamics between the relationships observed in the current study and to test other possible moderators as codebase complexity or team size, as well as taking into account other possible study orientations, such as different programming paradigms or methodologies, e.g., agile versus waterfall, to evaluate the generalizability of the mediated and moderated relationships observed in the current study.

## **References**

1. Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. *Proceedings of the 35th International Conference on Software Engineering*, 712–721.
2. Basili, V. R., Briand, L. C., & Melo, W. L. (1996). A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10), 751–761.
3. Bavota, G., & Russo, B. (2015). Four eyes are better than two: On the impact of code reviews on software quality. *Proceedings of the 31st IEEE International Conference on Software Maintenance and Evolution*, 81–90.
4. Beck, K. (2003). *Test-driven development: By example*. Addison-Wesley.
5. Bird, C., Nagappan, N., Murphy, B., Gall, H., & Devanbu, P. (2011). Don't touch my code! Examining the effects of ownership on software quality. *Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 4–14.
6. Boehm, B., & Basili, V. R. (2001). Software defect reduction top 10 list. *Computer*, 34(1), 135–137.
7. Chin, W. W. (1998). The partial least squares approach to structural equation modeling. In G. A. Marcoulides (Ed.), *Modern methods for business research* (pp. 295–336). Lawrence Erlbaum Associates.
8. Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM*, 31(11), 1268–1287.
9. Hair, J. F., Risher, J. J., Sarstedt, M., & Ringle, C. M. (2019). When to use and how to report the results of PLS-SEM. *European Business Review*, 31(1), 2–24.
10. Hayes, A. F. (2017). *Introduction to mediation, moderation, and conditional process analysis: A regression-based approach* (2nd ed.). Guilford Press.
11. Henseler, J., Ringle, C. M., & Sarstedt, M. (2015). A new criterion for assessing discriminant validity in variance-based structural equation modeling. *Journal of the Academy of Marketing Science*, 43(1), 115–135.

12. Kitchenham, B., & Pfleeger, S. L. (1996). Software quality: The elusive target. *IEEE Software*, 13(1), 12–21.
13. McIntosh, S., Kamei, Y., Adams, B., & Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 192–201.
14. Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, 11(3), 309–346.
15. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). Wiley.
16. Rigby, P. C., & Bird, C. (2013). Convergent contemporary software peer review practices. *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, 202–212.
17. Runeson, P., Andersson, C., Thelin, T., Andrews, A., & Berling, T. (2006). What do we know about defect detection methods? *IEEE Software*, 23(3), 82–90.
18. Sjøberg, D. I. K., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N. K., & Rekdal, A. C. (2005). A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9), 733–753.
19. Tantithamthavorn, C., McIntosh, S., Hassan, A. E., & Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 43(1), 1–18.
20. Wohlin, C., Runeson, P., & Brantestam, J. (1994). An experimental evaluation of capture-recapture in software inspections. *Software Testing, Verification and Reliability*, 4(4), 213–232.



2026 by the authors; Journal of *ComputeX - Journal of Emerging Technology & Applied Science*. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (<http://creativecommons.org/licenses/by/4.0/>).