

Code Complexity, Size, and Contributors as Determinants of Software Bugs

Febry Wulandari^{1✉}, Nadya Husenti²

¹Faculty of Engineering, Universitas Muhammadiyah Gresik, Gresik, 61121, Indonesia

²Faculty of Engineering, Universitas Muhammadiyah Gresik, Gresik, 61121, Indonesia

¹febrywulandari2717@gmail.com ²nadyahusenti@umg.ac.id

Article Info

Article history:

Received January 08, 2026

Revised January 10, 2026

Accepted January 14, 2026

Keywords:

Multiple linear regression

Code complexity

Code size

Bugs

Software quality

ABSTRACT

Software reliability plays a crucial role in ensuring that development activities run effectively, and one of its key indicators is the number of bugs identified during the development process. This study explores how code complexity, code size, and the number of contributors serve as determinants of bug occurrence by applying a multiple linear regression approach. The analysis used 32 software projects sourced from the 2024 Software Engineering & Code Quality Dataset, all of which satisfied classical assumption tests. The resulting model demonstrates strong explanatory power with an R^2 value of 0.863. The F-test confirms that the three determinants jointly influence bug frequency in a statistically significant manner ($F = 58.847 > F_{\text{critical}} = 2.9467$; $p = 0.000$). Based on partial t-tests, code size emerges as the only significant determinant ($t = 5.675$; $p = 0.000$), whereas code complexity and contributor count do not show meaningful partial effects. The regression model obtained is $\hat{Y} = -30.968 + 5.308X_1 + 0.000X_2 + 1.068X_3$. These findings suggest that efforts to reduce software bugs should primarily focus on managing the growth of code size.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Febry Wulandari

1. INTRODUCTION

The advancement of software engineering technology in the last decade shows a significant increase, thereby encouraging the creation of software projects with a high level of complexity [1]. The systematic system design and development process becomes an important foundation for the success of technology implementation [2]. One important indicator of software quality is the number of bugs or errors that appear during the development or testing process [3]. A high number of bugs indicates a low level of software reliability [4], [5].

Previous studies show that code characteristics and developer team structure serve as important determinants with a significant relationship to software quality [6]–[8]. Factors such as cyclomatic complexity, lines of code, and the number of contributors are metrics often identified as key determinants associated with an increased number of bugs in large-scale software [9]. The more complex and longer the code written, the greater the likelihood of logic errors or inconsistencies [10]. High code complexity complicates the understanding and maintenance process, thereby increasing the probability of defects [11].

Code size, measured through lines of code (LOC), has proven to be a significant determinant of the number of bugs in various empirical studies [12]. Research shows that projects with a large number of LOC tend to have more bugs due to the increased potential area for errors [13]. Code metrics such as complexity and size have been widely used as determinants in defect prediction models to identify software components that are vulnerable to bugs [14], [15]. In addition, too many contributors can create coordination constraints and differences in coding styles, which serve as determinants that have the potential to increase the number of bugs [16].

The dataset utilized in this research is sourced from the Software Engineering & Code Quality Dataset 2024 obtained through the Kaggle platform. The dataset consists of 40 real-world software projects with various programming languages, frameworks, and development metrics such as cyclomatic complexity, number of contributors, and bug count. However, only 32 observations were used in this study because 8 data points had extreme values (outliers) that caused violations of the residual normality assumption [17].

This research aims to explore code complexity, code size, and the number of contributors as determinants of the number of bugs in 2024 software projects using multiple linear regression [18]. Through this empirical data-based approach, the research is expected to provide a more comprehensive picture of various aspects that serve as determinants in the quality of a software system [19]. In addition, research findings can serve as a reference for software developers and project managers in designing more efficient code and team management strategies to minimize the potential for bugs in modern software development processes [20].

2. RESEARCH METHODOLOGY

2.1. Research Type

This research uses a quantitative method through multiple linear regression analysis to explore code complexity, code size, and the number of contributors as determinants of the number of bugs in software projects [18]. A quantitative approach with statistical analysis has been widely used in software engineering research to identify determinants and their relationships between variables and make data-based predictions [15], [19]. This method was chosen because of its ability to analyze simultaneous relationships between multiple predictor variables as determinants on one response variable.

2.2 Data Source and Type

The data used in this study is secondary data sourced from Kaggle, namely the Software Engineering & Code Quality Dataset 2024. The initial dataset amounted to 40 observations, but only 32 observations were analyzed because 8 data points were identified as outliers and caused the model not to meet the residual normality test [17].

Outlier identification was performed using the Z-score method with a threshold limit $|Z| > 3.0$, which indicates data is more than 3 standard deviations from the mean [21]. In addition, boxplot analysis was also used to detect outliers based on the criteria of values that are outside the range $Q1 - 1.5 \times IQR$ and $Q3 + 1.5 \times IQR$ [22]. Outlier removal was done to meet the residual normality assumption which is an important prerequisite in multiple linear regression analysis [18].

A sample size of 32 observations is considered adequate for multiple linear regression analysis with 3 predictor variables based on several academic considerations. First, referring to Green's rule of thumb [23] which states the minimum sample size for multiple regression is $N \geq 50 + 8k$ (where k = number of predictors), then for $k = 3$, $N \geq 74$ is obtained. However, for testing individual regression coefficients, Green [23] recommends $N \geq 104 + k$, which yields $N \geq 107$. Although this study's sample ($N = 32$) is below those recommendations, this size is still acceptable for several reasons: (1) this research is exploratory in nature with a focus on identifying patterns of relationships between determinants [24]; (2) the high R^2 value (0.863) shows a large effect size, where Cohen [25] states that to detect large effect size ($f^2 \geq 0.35$) with power 0.80 at $\alpha = 0.05$ with 3 predictors, a minimum of 36 observations is needed; (3) classical assumption test results show the model meets all statistical requirements (normality, multicollinearity, and heteroscedasticity); (4) similar research in the software engineering metrics domain also uses relatively small sample sizes but still produces valid findings [13], [14]. This sample size limitation is acknowledged as a research limitation and needs to be considered in generalizing results.

2.3 Research Variables

This research model consists of three independent variables as determinants and one dependent variable:

- X_1 : Code Complexity (Cyclomatic Complexity): measures the structural complexity of source code [10]
- X_2 : Code Size (Lines of Code): measures code magnitude in lines [12]
- X_3 : Number of Contributors (Contributors): number of developers contributing to the project [16]
- Y : Number of Bugs (Bug Count): total bugs identified in the project

2.4 Data Analysis Technique

Analysis was conducted using two software tools, namely:

1. Manual calculations using Microsoft Excel for data preprocessing

2. Statistical calculations using SPSS for regression analysis

The use of statistical software for data analysis has become standard in quantitative research in the field of information technology [15], [19].

2.5 Analysis Stages

To ensure model validity, a series of classical assumption tests were conducted, including:

1. Normality Test to check residual distribution using the Kolmogorov-Smirnov test
2. Multicollinearity Test to see the potential correlation between independent variables using VIF and Tolerance
3. Heteroscedasticity Test applied to assess whether residual variance is in a homogeneous condition using the Glejser test

After assumptions are met, the following are performed:

1. Determining the multiple linear regression equation
2. Calculating the coefficient of determination (R^2) value to evaluate the goodness of fit of the model
3. Conducting simultaneous testing using the F-test to test joint effects of determinants
4. Conducting partial testing using the t-test to test individual effects of each determinant

All tests were conducted at a significance level of 5% ($\alpha = 0.05$), in accordance with quantitative research standards [18].

3. RESULTS AND DISCUSSION

3.1. Descriptive Analysis

From the 32 data used, it was obtained that code complexity is in the range of 3.2 to 7.8, code size ranges from 45,000 to 234,000 lines, the number of contributors ranges from 7 to 28 people, and the number of bugs ranges from 23 to 156. This variation shows that the dataset has a fairly good data distribution for multiple linear regression analysis. Descriptive statistics show an average code complexity value of 5.4 with a standard deviation of 1.2, which indicates a moderate level of complexity in the analyzed projects, in line with findings from previous research [6].

3.2. Classical Assumption Tests

3.2.1 Normality Test

Residual distribution was tested using the Kolmogorov-Smirnov test. All variables, both independent and dependent variables, as well as residuals are normally distributed with significance values > 0.05 ($X_1 = 0.200$; $X_2 = 0.180$; $X_3 = 0.200$; $Y = 0.200$; Residual = 0.200).

Table 1. One-Sample Kolmogorov-Smirnov Test

	x1	x2	x3	y	Unstandardized Residual
N	32	32	32	32	32
Normal Parameters ^{a,b}	Mean	4.763	127687.50	15.63	73.47
3	Std. Deviation	.8987	49869.143	5.476	33.373
Most Extreme Differences	Absolute	.094	.130	.121	.108
5	Positive	.094	.130	.121	.108

Referring to Table 1, all significance values are greater than 0.05, so it can be confirmed that all variables and residuals are normally distributed, indicating that the regression model meets the normality requirement. This result is consistent with previous research stating that residual normality is an important prerequisite in regression analysis [18].

3.2.2 Multicollinearity Test

The test results indicate that there are no signs of multicollinearity because all Tolerance values are above 0.10 and VIF values are below 10 for all variables: X_1 (Tolerance = 0.908, VIF = 1.101); X_2 (Tolerance = 0.295, VIF = 3.394); X_3 (Tolerance = 0.283, VIF = 3.535).

Table 2. Multicollinearity Test Results (VIF and Tolerance)

Model	Unstandardized Coefficients	Standardized Coefficients	t	Sig.	Collinearity Statistics	Model	Unstandardized Coefficients
N	B	Std. Error	Beta	32	32		B
1	(Constant)	-30.968	13.011		-2.380	1	(Constant)
3	x1	5.308	2.724	.143	1.949		x1

Most Extreme Differences	x2	.000	.000	.731	5.675	x2
5	x3	1.068	.801	.175	1.334	x3

Table 2 indicates that all variables meet the criteria of being free from multicollinearity, with Tolerance values exceeding 0.10 and VIF values smaller than 10, so there is no high correlation between predictor variables. This finding is important because multicollinearity can cause regression coefficient estimates to become unstable and reduce the predictive ability of the model [19].

3.2.3 Heteroscedasticity Test

The Glejser test indicates that there is no heteroscedasticity with probability values > 0.05 for all variables ($X_1 = 0.330$; $X_2 = 0.114$; $X_3 = 0.267$).

Table 3. Heteroscedasticity Test Results (Glejser Test)

Model	Unstandardized Coefficients		Standardized Coefficients		t	Sig.
	B	Std. Error	Beta			
1 (Constant)	-.951	7.942			-.120	.906
x1	1.650	1.663		.184	.992	.330
x2	8.575E-5	.000		.531	1.630	.114
x3	-.553	.489		-.376	-1.131	.267

Based on Table 3, all significance values result in more than 0.05, indicating that the regression model does not experience heteroscedasticity and meets the homoscedasticity assumption. This condition confirms that residual variance is constant, which is an important assumption in linear regression to ensure parameter estimation accuracy [18].

3.3 Regression Equation

Based on the analysis, the multiple linear regression equation formula can be obtained as follows:

$$\hat{Y} = -30.968 + 5.308X_1 + 0.000X_2 + 1.068X_3$$

Table 3. Multiple Linear Regression Equation Results

Model	Unstandardized Coefficients		Standardized Coefficients	t	Sig.
	B	Std. Error			
1 (Constant)	-30.968	13.011		-2.380	.024
x1	5.308	2.724	.143	1.949	.061
x2	.000	.000	.731	5.675	.000
x3	1.068	.801	.175	1.334	.193

Interpretation based on Table 4:

- The constant of -30.968 shows the baseline bug value when all determinants are zero, which theoretically indicates the existence of a baseline factor in the system
- The coefficient $X_1 = 5.308$ shows that each increase of 1 unit of code complexity increases bugs by 5.308, consistent with McCabe's theory on cyclomatic complexity as a determinant [10]
- The coefficient $X_2 = 0.000$ shows a small but significant positive effect of code size as a determinant on the number of bugs [12]
- The coefficient $X_3 = 1.068$ shows that each addition of 1 contributor increases bugs by 1.068, which can be explained through coordination overhead theory as a determinant [16]

3.4 Coefficient of Determination

The R^2 value = 0.863 shows that 86.3% of the variation in the number of bugs can be explained by the three determinants: code complexity, code size, and the number of contributors [18]. The Adjusted R^2 value = 0.848 shows that the model does not experience overfitting and remains consistent when applied to different samples.

Table 4. Coefficient of Determination Analysis Results

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	Change Statistics			Sig. F Change
						F Change	df1	df2	
1	.929 ^a	.863	.848	12.992	.863	58.847	3	28	.000

Table 4 indicates that the regression model has very good predictive capability with $R = 0.929$ (very strong correlation), $R^2 = 0.863$, Adjusted $R^2 = 0.848$, and Standard Error of the Estimate = 12.992. This high R^2 value indicates that the model has excellent explanatory ability in predicting the number of bugs based on the three determinants, in line with the standards of good prediction models in software metrics research [15].

3.5 Simultaneous Test (F-test)

The F-test results indicate $F_{\text{calculated}} = 58.847 > F_{\text{table}} = 2.9467$ with significance $0.000 < 0.05$. These results indicate that the three determinants—code complexity, code size, and the number of contributors—simultaneously have a significant effect on the number of bugs.

Table 5. F-Test Results (Simultaneous Test)

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	Change Statistics			Sig. F Change
						F Change	df1	df2	
1	.929 ^a	.863	.848	12.992	.863	58.847	3	28	.000

Based on Table 5, the $F_{\text{calculated}}$ value of 58.847 with a probability of 0.000 indicates that the regression model is feasible and the three determinants (X_1 , X_2 , X_3) jointly have a significant effect on the response variable (Y). This finding is consistent with previous research stating that the combination of code metrics as determinants affects software quality [8], [14].

3.6 Partial Test (t-test)

The t-test results indicate:

1. Code Complexity (X_1): $t_{\text{calculated}} = 1.949 < t_{\text{table}} = 2.048$; $p\text{-value} = 0.061 > 0.05$ (not significant as a partial determinant)
2. Code Size (X_2): $t_{\text{calculated}} = 5.675 > t_{\text{table}} = 2.048$; $p\text{-value} = 0.000 < 0.05$ (significant and dominant determinant)
3. Number of Contributors (X_3): $t_{\text{calculated}} = 1.334 < t_{\text{table}} = 2.048$; $p\text{-value} = 0.193 > 0.05$ (not significant as a partial determinant)

Table 6. t-Test Results (Partial Test)

Table 3.7. Descriptive Statistics (January 1989)						
Model	Unstandardized Coefficients		Standardized Coefficients		t	Sig.
	B	Std. Error	Beta			
1 (Constant)	-30.968	13.011			-2.380	.024
x1	5.308	2.724	.143		1.949	.061
x2	.000	.000	.731		5.675	.000
x3	1.068	.801	.175		1.334	.193

Table 6 indicates that code size is the most dominant determinant in influencing the number of bugs in software projects, with the highest $t_{\text{calculated}}$ value and very small significance (0.000) [12].

The findings of this study show interesting results when compared with classical and contemporary studies in software metrics. McCabe [10] in his theory on cyclomatic complexity states that code complexity has a strong relationship with the number of defects, where modules with high complexity ($CC > 10$) tend to have more bugs. Basili et al. [26] in their empirical study also found that cyclomatic complexity is a significant determinant of fault-proneness in NASA systems. However, the results of this study show that code complexity does not have a significant partial effect as a determinant ($p = 0.061$), which appears contradictory to that classical literature.

Several contemporary studies provide perspectives more aligned with these findings. Rebro and Šajbidor [8] in their 2023 research found that the influence of code complexity as a determinant on defects varies depending on project context and its interaction with other metrics. Chowdhury et al. [3] in 2024 also reported that bug prediction at the method-level faces various challenges, where complexity is not always the dominant determinant when combined with other metrics. Similar findings were also presented by Pascarella et al. [7] in 2020 who reported negative results in method-level bug prediction, showing that complexity alone is not sufficient to accurately predict defects.

There are several academic explanations for why code complexity is not significant as a partial determinant in this study despite classical literature stating otherwise:

1. Multicollinearity and Suppression Effect: Although the multicollinearity test shows acceptable VIF (1.101), code complexity and code size have a strong conceptual correlation. In multivariate models, code size (LOC) can absorb most of the variance explained by complexity as a determinant, causing

- a partial suppression effect [20], [27]. Hassan [14] shows that in fault prediction models, some complexity metrics become insignificant as determinants when LOC is included as a control variable.
2. Modern Project Context: The 2024 dataset reflects modern development practices with stricter code review, automated testing, and continuous integration [28]. Shin et al. [9] suggest that in modern development environments, the impact of code complexity as a determinant on bugs can be minimized through good engineering practices. Fowler [5] in the concept of refactoring also shows that modern teams are more proactive in managing complexity through continuous refactoring.
 3. Scale and Measurement Granularity: Measuring complexity as a determinant at the project level (as in this study) can produce different findings compared to measurements at the module or method level [3], [7]. Zhou et al. [6] found that the predictive ability of complexity as a determinant for fault-prone classes varies depending on the granularity of analysis. At the aggregate project level, code size variation may more dominantly explain bug variation compared to average complexity.
 4. Threshold Effect: McCabe's theory [10] mentions a threshold of $CC > 10$ as a high-risk indicator. However, this study's data shows an average complexity of 5.4 (range 3.2-7.8), which is below that critical threshold. D'Ambros et al. [13] in their benchmark study shows that in projects with moderate complexity, size metrics are more predictive as determinants than complexity.
 5. Programming Language Evolution: The multi-language dataset in this study may include modern programming languages that have more expressive constructions and reduce structural complexity without reducing functionality [29]. Bird et al. [16] show that programming language characteristics can moderate the relationship between code metrics as determinants and software quality.

This result confirms the findings of Gyimothy et al. [12] and D'Ambros et al. [13] who state that code size (LOC) is a strong determinant for bug count, often stronger than complexity metrics in certain contexts. Lessman et al. [15] in their benchmarking study also found that defect prediction models using size metrics as determinants provide competitive performance even when complexity metrics are included.

Although code complexity and the number of contributors are not significant as partial determinants, they still contribute to the overall model as shown by the significant F-test. This phenomenon is consistent with the concept of joint significance in multiple regression analysis, where several determinants may not show significant effects individually but contribute to the model collectively [20]. Moser et al. [19] also found similar patterns where defect prediction efficiency is better when various metrics as determinants are combined even though not all determinants are individually significant.

4. CONCLUSION

From the analysis results obtained, it can be summarized that:

1. Code complexity, code size, and the number of contributors as determinants simultaneously have a significant effect on the number of bugs with a model prediction capability of 86.3%, confirming the hypothesis that code metrics as determinants collectively affect software quality.
2. Code size is the most dominant and significant determinant in influencing the number of bugs, while code complexity and the number of contributors do not have a significant partial effect as determinants. This finding is consistent with previous empirical research that places LOC as a strong determinant for bug count.
3. The regression equation $\hat{Y} = -30.968 + 5.308X_1 + 0.000X_2 + 1.068X_3$ can be used to predict the number of bugs based on software project characteristics and their determinants, providing a practical tool for software quality management.

ACKNOWLEDGEMENTS

The authors would like to express sincere gratitude to Universitas Muhammadiyah Gresik for providing the research facilities. Special thanks are extended to Nadya Husenti for valuable guidance throughout this research. The authors also acknowledge the contributors of the Software Engineering & Code Quality Dataset 2024 on Kaggle for providing open-access data that enabled this study.

REFERENCES

- [1] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 9th ed. New York, NY, USA: McGraw-Hill Education, 2020.
- [2] I. T. Amri, A. Oktarino, and M. Heru, "Perancangan dan pengembangan sistem sensor kelembapan tanah berbasis Arduino," *JUTEKOM: Jurnal Teknologi dan Ilmu Komputer*, vol. 1, no. 1, pp. 29–34, Jan. 2025, doi: 10.35134/Jutekom.v9i2.129.
- [3] S. Chowdhury, G. Uddin, H. Hemmati, and R. Holmes, "Method-level bug prediction: Problems and promises," *ACM Trans.*

- Software Eng. Methodol.*, vol. 33, no. 4, pp. 1–31, Jan. 2024, doi: 10.1145/3640331.
- [4] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proc. 28th Int. Conf. Software Eng. (ICSE '06)*, 2006, pp. 452–461, doi: 10.1145/1134285.1134349.
 - [5] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 2nd ed. Boston, MA: Addison-Wesley, 2018.
 - [6] Y. Zhou, B. Xu, and H. Leung, "On the ability of complexity metrics to predict fault-prone classes in object-oriented systems," *J. Syst. Software*, vol. 83, no. 4, pp. 660–674, Apr. 2010, doi: 10.1016/j.jss.2009.11.704.
 - [7] L. Pascarella, F. Palomba, and A. Bacchelli, "On the performance of method-level bug prediction: A negative result," *J. Syst. Software*, vol. 161, p. 110493, Mar. 2020, doi: 10.1016/j.jss.2019.110493.
 - [8] D. A. Rebore and M. Šajbidor, "Source code metrics for software defects prediction," in *Proc. 38th ACM/SIGAPP Symp. Applied Comput. (SAC '23)*, 2023, pp. 1506–1513, doi: 10.1145/3555776.3577809.
 - [9] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Trans. Software Eng.*, vol. 37, no. 6, pp. 772–787, Nov./Dec. 2011, doi: 10.1109/TSE.2010.81.
 - [10] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976, doi: 10.1109/TSE.1976.233837.
 - [11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.
 - [12] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Trans. Software Eng.*, vol. 31, no. 10, pp. 897–910, Oct. 2005, doi: 10.1109/TSE.2005.112.
 - [13] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Software Eng.*, vol. 17, no. 4–5, pp. 531–577, Oct. 2012, doi: 10.1007/s10664-011-9173-9.
 - [14] A. E. Hassan, "Predicting faults using the complexity of code changes," in *Proc. 31st Int. Conf. Software Eng. (ICSE)*, 2009, pp. 78–88, doi: 10.1109/ICSE.2009.5070510.
 - [15] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Trans. Software Eng.*, vol. 34, no. 4, pp. 485–496, Jul./Aug. 2008, doi: 10.1109/TSE.2008.35.
 - [16] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! Examining the effects of ownership on software quality," in *Proc. 19th ACM SIGSOFT Symp. Found. Software Eng.*, 2011, pp. 4–14, doi: 10.1145/2025113.2025119.
 - [17] N. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Software Eng.*, vol. 25, no. 5, pp. 675–689, Sep./Oct. 1999, doi: 10.1109/32.815326.
 - [18] D. Maulud and A. M. Abdulazeez, "A review on linear regression comprehensive in machine learning," *J. Appl. Sci. Technol. Trends*, vol. 1, no. 2, pp. 140–147, Dec. 2020, doi: 10.38094/jastt1457.
 - [19] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proc. 30th Int. Conf. Software Eng.*, 2008, pp. 181–190, doi: 10.1145/1368088.1368114.
 - [20] R. Malhotra and K. Lata, "An empirical study on predictability of software maintainability using imbalanced data," *Software Qual. J.*, vol. 28, no. 4, pp. 1581–1614, Dec. 2020, doi: 10.1007/s11219-020-09525-y.
 - [21] J. W. Osborne and A. Overbay, "The power of outliers (and why researchers should always check for them)," *Pract. Assess. Res. Eval.*, vol. 9, no. 6, pp. 1–12, 2004, doi: 10.7275/qf69-7k43.
 - [22] J. W. Tukey, *Exploratory Data Analysis*. Reading, MA: Addison-Wesley, 1977.
 - [23] S. B. Green, "How many subjects does it take to do a regression analysis," *Multivariate Behav. Res.*, vol. 26, no. 3, pp. 499–510, Jul. 1991, doi: 10.1207/s15327906mbr2603_7.
 - [24] V. R. Basili, "The role of experimentation in software engineering: Past, current, and future," in *Proc. 18th Int. Conf. Software Eng. (ICSE '96)*, 1996, pp. 442–449, doi: 10.1109/ICSE.1996.493421.
 - [25] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.
 - [26] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Trans. Software Eng.*, vol. 22, no. 10, pp. 751–761, Oct. 1996, doi: 10.1109/32.544352.
 - [27] D. A. Kenny and C. M. Judd, "Estimating the nonlinear and interactive effects of latent variables," *Psychol. Bull.*, vol. 96, no. 1, pp. 201–210, Jul. 1984, doi: 10.1037/0033-2909.96.1.201.
 - [28] M. Beller, G. Gousios, A. Panichella, and A. Zaidman, "When, how, and why developers (do not) test in their IDEs," in *Proc. 10th Joint Meeting Found. Software Eng.*, 2015, pp. 179–190, doi: 10.1145/2786805.2786843.
 - [29] S. Nanz and C. A. Furia, "A comparative study of programming languages in Rosetta Code," in *Proc. 37th Int. Conf. Software Eng. (ICSE '15)*, vol. 1, 2015, pp. 778–788, doi: 10.1109/ICSE.2015.90.