

# Relationship between features volatility and bug occurrence rate to support software evolution

Tiara Rahmania Hadiningrum, Bella Dwi Mardiana, Siti Rochimah

Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

## Article Info

### Article history:

Received Dec 26, 2023

Revised Jun 10, 2024

Accepted Jun 16, 2024

### Keywords:

Bug occurrence rate

Correlation coefficient

Features volatility

Quality

Software evolution

Software stability

## ABSTRACT

Software evolution is an essential foundation in delivering technology that adapts to user needs and industry dynamics. In an era of rapid technological development, software evolution is not just a necessity, but a must to ensure long-term relevance. Developers are faced with major challenges in maintaining and improving software quality over time. This research aims to investigate the correlation between feature volatility and bug occurrence rate in software evolution, to understand the impact of dynamic feature changes on software quality and development process. The research method uses commit analysis on the dataset as a marker of bug presence, studying the complex relationship between feature volatility and bug occurrence rate to reveal the interplay in software development. Validated datasets are measured by metrics and correlations are measured by Pearson-product-moment analysis. This research found a strong relationship between feature volatility and bug occurrence rate, suggesting that an increase in feature changes correlates with an increase in bugs that impact software stability and quality. This research provides important insights into the correlation between feature volatility and bug occurrence rates, guiding developers and quality practitioners to develop more effective testing strategies in dynamic development environments.

*This is an open access article under the [CC BY-SA](#) license.*



## Corresponding Author:

Siti Rochimah

Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember

Surabaya, Indonesia

Email: siti@its.ac.id

## 1. INTRODUCTION

Software evolution is a critical aspect in building a technological foundation that can adapt to the development of user needs and industry dynamics [1]–[3]. In an era where the speed of technological change is accelerating, software evolution is not just a necessity, but a necessity to ensure that the applications developed can continue to be relevant and provide added value [4], [5]. Software developers are faced with significant challenges in maintaining and improving software quality over time [6], [7]. These challenges include demands to quickly respond to changing user needs, maintain compatibility with evolving technology environments, and still ensure security and optimal performance [8]–[11]. Thus, a deep understanding of the importance of software evolution and the difficulties faced by developers in the face of continuous change is a crucial basis for designing adaptive and sustainable development strategies.

Feature volatility, which indicates to what extent and how often changes are made to certain features in a software application over time, reflects the dynamic nature of software development [12], [13]. Understanding the constant evolution of such features is crucial, as this directly affects the adaptability and

robustness of software applications [14]–[17]. Understanding the nuances of feature changes throughout the development cycle is key for developers, as it empowers them to foresee and respond effectively to changing user needs and evolving industry trends [9], [10], [18], [19]. In the context of software evolution, where staying abreast of user demands is crucial, feature volatility emerges as a critical factor [20], [21]. The inherent dynamics of the feature set not only reflect the software system's response to user needs but also pose challenges that require careful attention [7], [22]. Thus, a careful understanding of feature volatility becomes a crucial point in navigating the complexities of software development and is integral to ensuring the sustainability and relevance of software applications in an ever-changing technological landscape.

The bug occurrence rate plays a crucial role as an important indicator to measure in the software development process [23], [24]. The bug occurrence rate, often expressed as a percentage or number of specific defects per test cycle, provides valuable insights into the stability of a software application [25], [26]. This indicator not only provides an overview of the quality of the application but can also be used as an important parameter in evaluating and measuring the overall stability level of a software application [27], [28]. This is because the bug incidence rate is a mirror of the overall quality of the developed software. The existence of bugs not only affects application performance but also has a direct impact on user experience [29]. A low bug occurrence rate indicates that the software is relatively stable and error-free and a high bug occurrence rate indicates that the software may have serious quality issues [30], [31].

Wang *et al.* [30] introduced an automated static analysis tool to identify critical configuration positions in complex source code, revealing a positive correlation between feature volatility and software error rates. Their study underscores the importance of understanding how feature volatility impacts software development. In a separate endeavor, Pilliang *et al.* [31] highlighted the advantages of automation in saving time and costs, particularly evident in their creation of automated regression suites using open-source tools for healthcare solutions. Their research, focusing on Kanggle.com's repository platform, demonstrated an 86% accuracy rate in a risk matrix model, showcasing the effectiveness of automation in improving software development processes. Furthermore, Handani *et al.* [20] quantitatively explored the relationship between feature volatility and software architecture design stability in object-oriented software. By utilizing Constantinou metrics, they analyzed architecture design stability alongside feature volatility across consecutive versions, providing valuable insights validated by expert evaluators. Additionally, Ruohonen *et al.* [32] investigated volatility modelling in time series within software development contexts, employing FreeBSD as a case study. Their research delved into volatility properties in bug tracking, development activities, and communication interactions, shedding light on challenges and empirical studies related to software evolution and time series volatility.

While previous studies offer insights into software development, none directly address the correlation between feature volatility and bug occurrence. Identifying this gap, our research explores this relationship to enhance understanding and mitigation of software errors, particularly within evolving systems. Uncovering this amidst the complexities of software evolution poses a significant challenge. Feature volatility can negatively impact software quality by making software more error-prone, harder to test, and more expensive to maintain [33], [34]. While it is known that feature volatility can introduce complexity and uncertainty in the development process, the exact relationship and mechanisms that affect bug occurrence rates still require further investigation [35]. By using commit as an indicator of bug presence, this study aims to fill this knowledge gap and provide deeper insights into how feature volatility affects software development quality. The results of this study are expected to provide valuable guidance to software developers and quality assurance practitioners in developing more effective testing strategies for dynamic development environments.

## 2. METHOD

In the research process, analysis methodology plays a central role in ensuring the success and accuracy of the findings. Through four structured stages, namely i) research design, ii) data collection, iii) measurement elements, and iv) quantitative analysis. Researchers are guided to carry out research with precise and directed steps.

### 2.1. Phase 1: Research design

This study uses an empirical research approach to investigate the impact of feature volatility on software quality. The research design includes a comprehensive analysis of the software development process in relation to feature volatility. To conduct this analysis, five different real projects were selected for simulation: 'Nelayanku', 'RajaGula', 'Loak.In', 'ServiceTrip', and 'SwapGoodFabric'. Developed between 2021 and 2023, each project offers four different versions, reflecting different stages of development and refinement. Notably, all projects are built using the Laravel framework, with PHP as the primary

programming language. This selection of projects and their respective characteristics provides a solid foundation for evaluating the effects of feature volatility on software quality.

## 2.2. Phase 2: Data collection

At this stage, it is explained how the data is modelled to fit the desired metrics. The data used comes from historical lists published on the official website of the project. The data features, as shown in Table 1 with five different data sets, include a wide range of modified features. In addition, the methods described also consider data customization techniques to ensure conformity with the desired metrics.

It is important to note that the selection of data from historical sources provides a solid basis for modelling. This historical data reflects the evolution of the project over time and provides the context necessary to understand the impact of feature changes on the metrics being measured. Each data set is described in detail to provide context for the feature variations that occur. This analysis includes significant feature changes, specific changes at certain levels, and other aspects that affect relevant metrics.

The next step after data modelling is to apply metrics to each data set. Thus, it can be evaluated whether the feature variations recorded in the data have a significant impact on the measured metrics. Overall, this stage provides an important foundation for understanding the relationship between feature changes and outcome metrics in this project. Advanced statistical analysis was used to evaluate these relationships, providing a deeper understanding of the dynamics of this project.

Table 1. Five datasets used

Loak.In application		
Version	Features	Description
LK01	General system modifications	
LK02	Add a product filter feature	
LK03	Add login option using google	
LK04	Add product search feature	
RajaGula application		
Version	Features	Description
RJ01	General system modifications	
RJ02	Add product search feature in the favorite menu	
RJ03	Add product filter feature by category	
RJ04	Add login option using google	
Nelayanku application		
Version	Features	Description
NL01	General system modifications	
NL02	Add login option using google	
NL03	Add product pre-order feature	
NL04	Add user address change feature	
ServiceTrip application		
Version	Features	Description
ST01	General system modifications	
ST02	Added search feature in all menus (city data, employee data, official travel data)	
ST03	Add login option using google	
ST04	Add generate pdf in business trip details	
SwapGoodFabric application		
Version	Features	Description
SG01	General system modifications	
SG02	Add login option using google	
SG03	Add print member transaction role feature	
SG04	Add delete complaint-member role feature	

## 2.3. Phase 3: Measurement

This measurement starts from the initial version to the final version of each project. Volatility results depend on the number of features modified and the total number of features commits from successive versions. The process of iterating commits for feature volatility involves creating repeated commits for different features, allowing for a thorough evaluation of the degree of fluctuation of those features over successive iterations. During these iterations, each feature modification is recorded and collected to form a history of the evolution of the project from the initial version to the final version. Measuring feature volatility not only looks at the frequency of feature changes but also considers the total commits involved in the process. This provides a more holistic perspective on feature changes that occur during project development.

Table 2 is a table recording changes and the number of commits in a dataset. This table provides details of changes that occurred during project development, including information about each commit made, including the type of change made and total number of commits. This logging forms the basis for further analysis of feature volatility, enabling a deep understanding of the project's evolution over time. By detailing each change and commit to this dataset, stakeholders can track and evaluate the dynamics of feature changes that may impact the overall volatility of the project. This iterative process allows for a deeper understanding of how each feature change contributes to overall volatility. By creating iterative commits for different features, the analysis can cover various contexts and dynamics that may influence feature fluctuations from version to version.

Table 2. Example of commits on Loak.In dataset

Commit Table Version			
Version 4			
	Filter	Authentication	Authentication
Commit 1	-	-	v
Commit 2	v	v	-
Commit 3	v	v	-
Commit 4	-	-	v
Commit 5	-	v	-
Commit 6	v	v	-
Commit 7	v	v	v

#### 2.4. Phase 4: Quantitative analysis

In statistical analysis, we have used the volatility metric to measure how much data changes. In the context of requirements, we use a requirements volatility metric called functional specification stability. As described in (1), this metric depends on the number of features that change. This metric can also be used to calculate the volatility of features. Volatility equal to zero indicates that all features change significantly. Conversely, volatility equal to one means that all features do not change.

$$Volatility(i, i + 1) = 1 - \frac{F(i+1)}{totF(i)} \quad (1)$$

In addition, we also use the Pearson product moment method to determine the dependency between two variables. This method, as shown in (2), is mainly used to model ratio data. To evaluate these results, we calculate the t value of the correlation coefficient or compare the r value with the r table. We also conducted a model fit test to ensure that the method used was appropriate for the characteristics of the observed data, thus strengthening the reliability of the analysis performed.

$$r_{xy} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}} \quad (2)$$

To conduct further quantitative analysis, we adopted the least squares regression method. Using data obtained from relevant software development sets, we built a regression model relating the independent variable (feature volatility) to the dependent variable (bug rate). Rigorous statistical measures, including significance and validity tests, were implemented to ensure the reliability of the results. This regression analysis, shown in (3), provides an in-depth understanding of the patterns of interconnectedness between the variables, enabling the identification of critical factors that could potentially affect bug occurrence rates. Where a is the slope of the line and b is the y-intercept of the line with the y-axis shown in (4) and (5). This quantitative approach provides a solid foundation for a rigorous and thorough empirical evaluation of the impact of feature volatility on software quality in the development lifecycle.

$$y = ax + b \quad (3)$$

$$a = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \quad (4)$$

$$b = \bar{y} - a\bar{x} \quad (5)$$

### 3. RESULTS AND DISCUSSION

We applied the evaluation mechanism to open-source projects built using the Laravel framework. We use 5 open-source projects that represent 5 datasets projects because we wanted to see if this mechanism could be applied in object-oriented projects. This was done to evaluate whether the mechanism could be applied to object-oriented projects. In addition, the selection of open-source projects that represent different aspects of software development in an object-oriented environment also aims to ensure the validity and generality of the findings obtained from the evaluation of these mechanisms.

#### 3.1. Result

Based on the results of volatility measurement and bug analysis on five different datasets, there is a correlation between the level of change in application features and the occurrence of bugs. The higher the level of modifications or changes made to an application feature, the greater the probability of bugs or malfunctions in the feature. This finding suggests that app developers should pay close attention to the volatility aspect of features and conduct rigorous testing whenever making changes to app features. Thus, the occurrence of bugs can be minimized and the quality of the application can be maintained.

In Table 3, it can be seen that the rate of change of application features in the RajaGula, Loak.In, ServiceTrip, SwapGoodFabric, and Nelayanku datasets are 2, 3, 2, 2, and 2, respectively. Meanwhile, the rate of occurrence of feature bugs in these datasets is 5, 8, 4, 9, and 7, respectively. From the data, it can be concluded that the higher the rate of change in application features, the greater the possibility of feature bugs. This is because changes to application features can cause changes to the program code, which can lead to errors or bugs. Through the execution of the least squares regression method on the corresponding graphs, we were able to construct a comprehensive calculation to determine the significant relationship between the number of bugs and feature volatility which can be seen in Figure 1.

Table 3. Result of measurement

Version	Dataset	Volatility	Bug
1 – 4	RajaGula Dataset	2	5
		2	8
		4	16
	Loak.In Dataset	3	8
		3	10
		3	11
	ServiceTrip Dataset	2	4
		2	8
		4	16
	SwapGoodFabric Dataset	2	9
		2	11
		4	15
Nelayanku Dataset	2	7	
	3	12	
	4	13	

As illustrated by the results on the RajaGula dataset, it was found that every one-unit increase in the number of bugs correlated with a two-unit increase in feature volatility. As such, this finding indicates a strong positive correlation between the two variables, reflecting that the more feature volatility there is, the higher the bug rate. After controlling for other variables such as code size or release time, additional analysis also confirmed the continued significance of the relationship between feature volatility and the number of bugs, strengthening the validity of the previous findings. This additional analysis provides additional support to the findings, confirming that the relationship between feature volatility and the number of bugs remains consistent and strong.

In exploring the implications of this finding, its relevance in the context of software development becomes clear. An increase in the number of bugs not only reflects a quantitative problem but also provides clues to the complexity and variety of possible impacts. For example, more and varied bugs may lead to increased complexity in software repair and management. Therefore, an in-depth understanding of the positive relationship between bug count and feature volatility can equip development teams with better insights, enabling a more proactive response to emerging issues. By basing these findings on concrete calculation results from the dataset, we can establish that bug volatility can be measured in more detail and estimated with more precision based on the number of bugs detected. This computer data is not only tangible evidence of the observed relationship but also an important tool in providing a solid foundation for more informed decision-making in software management and repair.

From the results of the previous graph analysis, it can be seen that there is a significant relationship between the number of bugs and feature volatility. In line with these findings, we also calculated Pearson-product moment correlation values to evaluate each of our datasets, to identify potential anomalous datasets. The high correlation, reaching a value of 0.85, is a strong indication that an increase in the number of bugs consistently correlates with an increase in bug volatility. While we were involved in this calculation to investigate each dataset, the findings provide additional confirmation of the close relationship between bug count and feature volatility, enriching our understanding of this dynamic in the context of software development.

In Table 4, datasets with high correlation values have important relevance in bug management, especially regarding the relationship between bug count and feature volatility. When the correlation value reaches 0, indicating no significant relationship between the two variables, attention to the effect of bug volatility due to an increase in the number of bugs may not be needed. On the contrary, a high correlation value, as in this case, is a warning against the potential risk of higher bug volatility as the number of bugs increases. In this context, it is emphasized that datasets that show a correlation value of 0 can be considered anomalies. Thus, the Loak.In dataset, which shows the lowest correlation among the five datasets, can be identified as an anomaly or data that does not follow the general pattern of the relationship between the number of bugs and feature volatility.

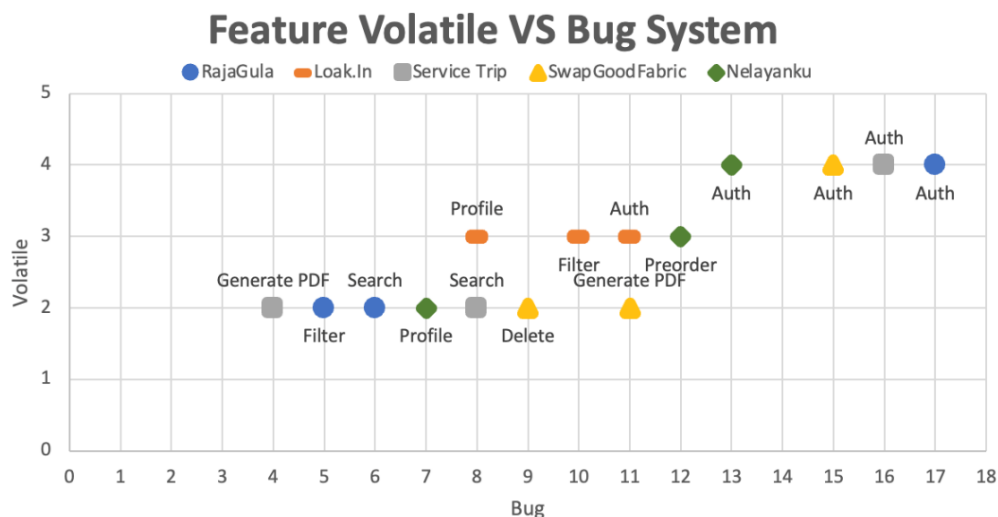


Figure 1. Feature volatility trends

### 3.2. Analysis and discussion

The overall aim of this research is to explore the correlation between requirements volatility and bug occurrence rate in the context of software development. Our approach focuses on the volatility of functional requirements in the requirements aspect and the bug occurrence rate in the architecture aspect. The analysis method we chose was Pearson-product moment correlation to provide clarity and objectivity. This approach strengthens the understanding of how changes in functional requirements can affect the occurrence of bugs in the software structure, providing valuable insights for software development practitioners.

Case studies on four datasets, RajaGula, ServiceTrip, SwapGoodFabric, and Nelayanku revealed a significant correlation between variables, suggesting that changes in functional requirements can affect system architecture, and vice versa. However, the Loak.In dataset showed a correlation coefficient of 0, indicating no correlation between requirements volatility and bug occurrence rate in the architecture. This confirms that changes in requirements in this dataset do not necessarily have a direct impact on system architecture. By incorporating the least squares regression method, the results reaffirmed the previous correlation findings, showing that the higher the level of feature volatility, the greater the number of bugs. However, it is important to note that the emergence of requirement changes that are considered anomalies does not always directly impact the architecture. The conclusion that can be drawn is that while the rate of change of application features can increase the likelihood of bugs, ideal software should be able to respond robustly to changing requirements, while still maintaining the stability of its architecture.

This analysis not only highlights the finding of a significant correlation between requirements volatility and bug occurrence rates but also emphasizes the importance of understanding the dynamics of requirements volatility in the context of software. This indicates that changing requirements can be a trigger for bug occurrence, which needs to be carefully considered in the software development lifecycle. While requirements volatility can contribute to an increase in the number of bugs, responses to requirements changes that are considered anomalous do not necessarily create a similar impact on the architecture. The practical implications of these findings can be directed towards the development of a more careful risk management strategy, where requirements changes can be identified, assessed and managed proportionally. As such, project management can be more effective in mitigating bug risks without excessively sacrificing architectural stability.

Previous research emphasizes that changes in architecture are a response to volatility features, which implies that the higher the level of volatility, the greater the likelihood of changes in architecture. However, recent research has found direct evidence that the more volatility there is, the higher the rate of bugs in software. These findings change the previous paradigm and highlight the strong correlation between feature volatility and bug rates, highlighting the importance of understanding and managing volatility effectively in software development. Overall, the findings provide a basis for a more holistic approach to software project management, emphasizing the complexity of the dynamics between requirements volatility, bug occurrence rate, and architectural stability. A deeper understanding of these relationships can make a positive contribution towards the development of software that is adaptive and responsive to inevitable environmental changes.

Table 4. Result of correlation analysis

Dataset	Re Volatility (Xi)	Bug (Yi)	$X_i^2$	$Y_i^2$	XY
RajaGula Dataset	2	5	4	25	100
	2	8	4	64	256
	4	16	16	256	4.096
<i>Sum</i>	8	29	24	345	4.452
		<i>Correlation Coefficient</i>		0.96	
Loak.In Dataset	3	8	9	64	576
	3	10	9	100	900
	3	11	9	121	1.089
<i>Sum</i>	9	29	27	285	2.565
		<i>Correlation Coefficient</i>		0	
ServiceTrip Dataset	2	4	4	16	64
	2	8	4	64	256
	4	16	16	256	4.096
<i>Sum</i>	8	28	24	336	4.416
		<i>Correlation Coefficient</i>		0.94	
SwapGoodFabric Dataset	2	9	4	81	324
	2	11	4	121	484
	4	15	16	225	3.600
<i>Sum</i>	8	35	24	427	4.408
		<i>Correlation Coefficient</i>		0.94	
Nelayanku Dataset	2	7	4	49	196
	3	12	9	144	1.296
	4	13	16	169	2.074
<i>Sum</i>	9	32	29	362	3.566
		<i>Correlation Coefficient</i>		0.49	

#### 4. CONCLUSION

This study highlights the significant relationship between feature volatility and the number of bugs in software applications. A thorough analysis shows that the higher the rate of change in application features, the greater the likelihood of bugs appearing. This finding suggests that feature volatility can be considered as a factor comparable to the bug occurrence rate in the software development cycle. In addition, from this study, it can be concluded that the coefficient values are correlated, as in the RajaGula Dataset with a value of 0.96, ServiceTrip with a value of 0.94, SwapGoodFabric with a value of 0.94, and Nelayanku 0.49. However, the results are slightly different in the case of volatility anomalies, resulting in uncorrelated coefficient values as in the Loak.In Dataset. The implications of this conclusion are significant for software developers and organizations looking to improve the quality and stability of their applications. By understanding the correlation between feature volatility and the number of bugs, developers can take preventative and proactive measures, such as improving testing on frequently changing features or strengthening change management strategies. This can help reduce the negative impact of bugs on user

experience and improve overall application reliability. In facing the challenges of software evolution, this research provides a strong foundation for software development practices that are more adaptive and responsive to change. By paying attention to and managing feature volatility, developers can optimize their efforts to design applications that are not only innovative but also stable and reliable over time. In addition, this research can also improve software quality by enabling more detailed measurements of development planning, thereby strengthening effectiveness and efficiency in dealing with constantly changing dynamics in the software development environment.

## REFERENCES




- [1] A. M. Ferrari, L. Volpi, D. Settembre-Blundo, and F. E. García-Muiña, "Dynamic life cycle assessment (LCA) integrating life cycle inventory (LCI) and Enterprise resource planning (ERP) in an industry 4.0 environment," *Journal of Cleaner Production*, vol. 286, p. 125314, Mar. 2021, doi: 10.1016/j.jclepro.2020.125314.
- [2] D. Budgen and P. Brereton, "Short communication: Evolution of secondary studies in software engineering," *Information and Software Technology*, vol. 145, p. 106840, May 2022, doi: 10.1016/j.infsof.2022.106840.
- [3] F. Gurcan, G. G. M. Dalveren, N. E. Cagiltay, D. Roman, and A. Soylu, "Evolution of software testing strategies and trends: semantic content analysis of software research corpus of the last 40 years," *IEEE Access*, vol. 10, pp. 106093–106109, 2022, doi: 10.1109/ACCESS.2022.3211949.
- [4] K. Bennett, "Software evolution: past, present and future," *Information and Software Technology*, vol. 38, no. 11, pp. 673–680, Nov. 1996, doi: 10.1016/0950-5849(96)01116-0.
- [5] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol. 80, pp. 457–479, Feb. 2014, doi: 10.1016/j.scico.2012.08.003.
- [6] A. Almogahed, H. Mahdin, M. Omar, N. H. Zakaria, G. Muhammad, and Z. Ali, "Optimized refactoring mechanisms to improve quality characteristics in object-oriented systems," *IEEE Access*, vol. 11, pp. 99143–99158, 2023, doi: 10.1109/ACCESS.2023.3313186.
- [7] F. N. Colakoglu, A. Yazici, and A. Mishra, "Software product quality metrics: a systematic mapping study," *IEEE Access*, vol. 9, pp. 44647–44670, 2021, doi: 10.1109/ACCESS.2021.3054730.
- [8] R. R. Althar and D. Samanta, "The realist approach for evaluation of computational intelligence in software engineering," *Innovations in Systems and Software Engineering*, vol. 17, no. 1, pp. 17–27, Mar. 2021, doi: 10.1007/s11334-020-00383-2.
- [9] T. Mens, S. Demeyer, M. Wermelinger, R. Hirschfeld, S. Ducasse, and M. Jazayeri, "Challenges in software evolution," in *International Workshop on Principles of Software Evolution (IW/PSE)*, 2005, vol. 2005, pp. 13–22, doi: 10.1109/IW/PSE.2005.7.
- [10] B. Vogel-Heuser, A. Fay, I. Schaefer, and M. Tichy, "Evolution of software in automated production systems: Challenges and research directions," *Journal of Systems and Software*, vol. 110, pp. 54–84, Dec. 2015, doi: 10.1016/j.jss.2015.08.026.
- [11] A. Salahirad, G. Gay, and E. Mohammadi, "Mapping the structure and evolution of software testing research over the past three decades," *Journal of Systems and Software*, vol. 195, Art. no. 111518, Jan. 2023, doi: 10.1016/j.jss.2022.111518.
- [12] G. Islam and T. Storer, "A case study of agile software development for safety-critical systems projects," *Reliability Engineering & System Safety*, vol. 200, p. 106954, Aug. 2020, doi: 10.1016/j.ress.2020.106954.
- [13] E. Siakas, H. Rahanu, E. Georgiadou, and K. Siakas, "Requirements volatility in multicultural situational contexts," in *Communications in Computer and Information Science*, vol. 1646 CCIS, 2022, pp. 633–655, doi: 10.1007/978-3-031-15559-8\_45.
- [14] S. M. A. Shah, D. Sundmark, B. Lindström, and S. F. Andler, "Robustness testing of embedded software systems: An industrial interview study," *IEEE Access*, vol. 4, pp. 1859–1871, 2016, doi: 10.1109/ACCESS.2016.2544951.
- [15] N. Luo and L. Zhang, "Chaos driven development for software robustness enhancement," in *Proceedings - 2022 9th International Conference on Dependable Systems and Their Applications, DSA 2022*, Aug. 2022, pp. 1029–1034, doi: 10.1109/DSA56465.2022.00154.
- [16] F.-C. Chang and H.-C. Huang, "A design approach for software robustness," in *2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech)*, Mar. 2021, pp. 428–431, doi: 10.1109/LifeTech52111.2021.9391977.
- [17] M. Salama, R. Bahsoon, and P. Lago, "Stability in software engineering: survey of the state-of-the-art and research directions," *IEEE Transactions on Software Engineering*, vol. 47, no. 7, pp. 1468–1510, Jul. 2021, doi: 10.1109/TSE.2019.2925616.
- [18] A. S. Nyamawe, H. Liu, N. Niu, Q. Umer, and Z. Niu, "Feature requests-based recommendation of software refactorings," *Empirical Software Engineering*, vol. 25, no. 5, pp. 4315–4347, Sep. 2020, doi: 10.1007/s10664-020-09871-2.
- [19] D. Kavitha and A. Sheshasaayee, "Requirements volatility in software maintenance," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, vol. 86, 2012, pp. 142–150, doi: 10.1007/978-3-642-27317-9\_15.
- [20] F. Handani and S. Rochimah, "Relationship between features volatility and software architecture design stability in object-oriented software: Preliminary analysis," in *2015 International Conference on Information Technology Systems and Innovation (ICITSI)*, Nov. 2015, pp. 1–5, doi: 10.1109/ICITSI.2015.7437736.
- [21] E. Cibir and T. E. Ayyildiz, "An empirical study on software test effort estimation for defense projects," *IEEE Access*, vol. 10, pp. 48082–48087, 2022, doi: 10.1109/ACCESS.2022.3172326.
- [22] G. Taentzer, M. Goedicke, B. Paech, K. Schneider, A. Schürr, and B. Vogel-Heuser, "The nature of software evolution," in *Managed Software Evolution*, Cham: Springer International Publishing, 2019, pp. 9–20, doi: 10.1007/978-3-030-13499-0\_2.
- [23] N. Almusharraf and H. Alotaibi, "An error-analysis study from an EFL writing context: Human and automated essay scoring approaches," *Technology, Knowledge and Learning*, vol. 28, no. 3, pp. 1015–1031, Sep. 2023, doi: 10.1007/s10758-022-09592-z.
- [24] H. Mahfoodh and Q. Obediat, "Software risk estimation through bug reports analysis and bug-fix time predictions," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies, 3ICT 2020*, Dec. 2020, pp. 1–6, doi: 10.1109/3ICT51146.2020.9312003.
- [25] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, Jul. 2000, doi: 10.1109/32.859533.
- [26] G. Hubert, S. Aubry, and J. A. Clemente, "Impact of ground-level enhancement (GLE) solar events on soft error rate for avionics," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 5, pp. 3674–3984, Oct. 2020, doi: 10.1109/TAES.2020.2977792.






- [27] C. Kin Keong, K. Tieng Wei, A. A. Abd. Ghani, and K. Y. Sharif, "Toward using software metrics as indicator to measure power consumption of mobile application: A case study," in *2015 9th Malaysian Software Engineering Conference (MySEC)*, Dec. 2015, pp. 172–177, doi: 10.1109/MySEC.2015.7475216.
- [28] E. Ismail, N. Utelieva, A. Balmaganbetova, and S. Tursynbayeva, "The choice of measures reliability of the software for space applications," in *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, Jun. 2020, pp. 1–5, doi: 10.1109/ICECCE49384.2020.9179411.
- [29] Q. U. Ain, T. Rana, and Aamana, "A study on identifying, categorizing and reporting usability bugs and challenges," in *2023 International Conference on Communication Technologies (ComTech)*, Mar. 2023, pp. 53–68, doi: 10.1109/ComTech57708.2023.10165169.
- [30] J. Wang, T. Baker, Y. Zhou, A. I. Awad, B. Wang, and Y. Zhu, "Automatic mapping of configuration options in software using static analysis," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 10044–10055, Nov. 2022, doi: 10.1016/j.jksuci.2022.10.004.
- [31] M. Pilliang, Munawar, M. A. Hadi, G. Firmansyah, and B. Tjahjono, "Predicting risk matrix in software development projects using BERT and K-Means," in *2022 9th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Oct. 2022, pp. 137–142, doi: 10.23919/EECSI56542.2022.9946637.
- [32] J. Ruohonen, S. Hyrynsalmi, and V. Leppänen, "Software evolution and time series volatility: an empirical exploration," in *Proceedings of the 14th International Workshop on Principles of Software Evolution*, Aug. 2015, pp. 56–65, doi: 10.1145/2804360.2804367.
- [33] Y. Zhao, Y. Hu, and J. Gong, "Research on international standardization of software quality and software testing," in *2021 IEEE/ACIS 20th International Fall Conference on Computer and Information Science (ICIS Fall)*, Oct. 2021, pp. 56–62, doi: 10.1109/ICISFall51598.2021.9627426.
- [34] A. Almogahed, M. Omar, N. H. Zakaria, G. Muhammad, and S. A. AlQahtani, "Revisiting scenarios of using refactoring techniques to improve software systems quality," *IEEE Access*, vol. 11, pp. 28800–28819, 2023, doi: 10.1109/ACCESS.2022.3218007.
- [35] K. Juhnke, M. Tichy, and F. Houdek, "Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality," *Software Quality Journal*, vol. 29, no. 1, pp. 39–100, Mar. 2021, doi: 10.1007/s11219-020-09523-0.

## BIOGRAPHIES OF AUTHORS






**Tiara Rahmania Hadiningrum**    successfully earned a bachelor's degree (S.Kom) in information systems from Telkom University in 2023. She is currently studying for a master's degree at the Department of Information Engineering, Institut Teknologi Sepuluh Nopember. Her research interests include aspects of software quality, traceability, and testing. Further information or contact can be obtained via email at 6025231079@student.its.ac.id.



**Bella Dwi Mardiana**    successfully earned a bachelor's degree (S.Kom) in informatics from Universitas Muhammadiyah Malang in 2023. She is currently studying for a master's degree at the Department of Information Engineering, Institut Teknologi Sepuluh Nopember, Surabaya. Her research interests include aspects of software development, testing, and traceability. Further information or contact can be obtained via email at 6025231032@student.its.ac.id.



**Siti Rochimah**    successfully earned a doctoral degree (PhD) in software engineering from Universiti Teknologi Malaysia in 2010. Currently, she serves as the head of the Software Engineering laboratory at the Department of Informatics Engineering, Institut Teknologi Sepuluh Nopember. Her work involves writing more than 80 articles related to software engineering. Her research interests include aspects of software quality, traceability, and testing. Further information or contact can be obtained via email at siti@its.ac.id.