Empirical investigation of the influence of continuous integration bad practices on software quality

Ruben Blenicio Tavares Silva*
Federal University of Ceará
Quixadá, Brazil
rubensilva@ufc.br

Carla Bezerra
Federal University of Ceará
Quixadá, Brazil
carlailane@ufc.br

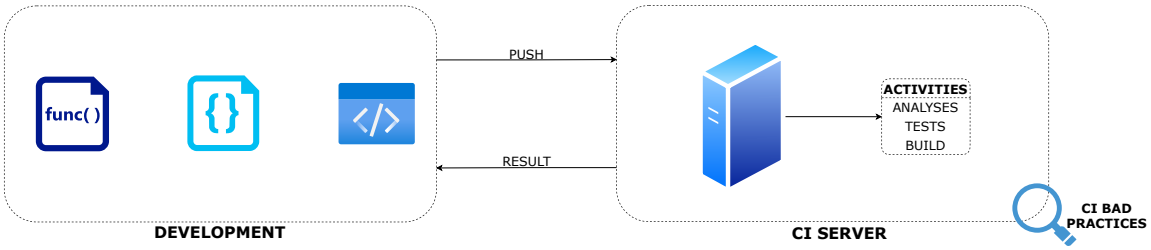

Figure 1: Identification of bad practices in the continuous integration process

ABSTRACT

Continuous Integration (CI) is a powerful tool to leverage software development in a safe, reliable, and efficient way. However, in day-to-day of software development, bad habits can arise in working with CI, which can make CI not reach its full potential in projects. These bad habits we call CI bad practices. This paper presented an exploratory study on closed-source projects to investigate how CI bad practices can affect software quality. We observe (1) the impact on the quality internal attributes after the implantation of CI, (2) the evolution of software quality indicators over time, and (3) the bad practices considered most harmful to the software quality by the development teams. Our results mean that projects affected by CI bad practices will not necessarily have their quality impaired. However, this does not mean that quality will be maintained or improved over time. Our analysis of the quality indicators has allowed us to observe that, generally, they have remained stable, and the number of quality issues reported has not decreased considerably either. Finally, the quality indicator most affected by the CI bad practices was maintainability, followed by reliability and security. In addition, the vast majority of the bad practices analyzed were classified as having a medium level of effort for resolution. In this sense, we also classify bad practices according to the level of effort/impact on the quality that can help prioritize their resolution.

KEYWORDS

continuous integration, bad practices, software quality

1 INTRODUCTION

Continuous integration (CI) is a technique that originated from Software Engineering, which emerged in the mid-'70s and has recently been applied in several software companies [3, 4]. The use of CI during the development of a system has the potential to provide numerous benefits, such as increasing the frequency of delivery of features [12, 22], improving the productivity of the team [16], detection and resolution of early bugs [3] and improved communication [7]. As a result, CI has been widely used in commercial and open-source projects through software tools that automate such processes and, consequently, collaborate with the good results related to CI [11].

However, applying CI in the scope of the software is not an easy task, especially in medium and high complexity software [18]. Therefore, all members involved in the project must go through a period of training and adaptation so that they can extract the maximum benefits that the technique provides and, consequently, escape the possible risks that CI can cause if implemented in an inappropriate form [21, 23]. Some studies in the literature have observed the effects of CI bad practices in software projects [8, 20]. Zampetti et al. [24] presents a catalog with 79 CI bad practices. Elazhary et al. [10] found in its study that it is possible to have bad practices in companies with a similar business model, demonstrating a significant variation in the way that CI is implemented.

Thus, this work conducts an study to investigate how CI bad practices can affect software quality. We conducted our research on 9 closed-source projects of our industry partners and analyzed quantitative data obtained (1) from the systems chosen, (2) from the API of the CI system used in the projects, and (3) from the perceptions of the members of the systems' development teams.

2 BACKGROUND

2.1 Continuous Integration Bad Practices

Duvall [9] alert to the fact that the CI needs to be well implemented to obtain the benefits and structure a series of bad practices related to the misuse of CI. Duvall [8] initially created a catalog with 50

*Both authors contributed equally to this research.

patterns (and their corresponding antipatterns) regarding several phases or relevant topics of the CI process. Zampetti et al. [24] derived a catalog of 79 CI bad smells organized into 7 categories spanning across the different dimensions of a CI pipeline management: Repository (REP), Build Process Organization (BPO), Quality Assurance (QA), Delivery Process (DP), Infrastructure Choices (INF), Build Maintainability (BM) and Culture (CUL). In this work, we use Zampetti's CI bad practices catalog [24] to investigate the use of CI in a company's closed-source projects. The Table 1 presents the CI bad practices used in the work that we identified in the study environment in a previous study [19].

Table 1: CI bad practices.

ID	BAD PRACTICE	CATEGORY
BP1	Some pipeline's tasks are started manually	BPO
BP2	Feature branches are used instead of feature toggles	REP
BP3	Lack of testing in a production-like environment	QA
BP4	Divergent Branches	REP
BP5	External tools are used with their default configurations	INF
BP6	Developers and operators are kept as separate roles	CUL
BP7	Failures notifications are only sent to teams/developers that explicitly subscribed	BPO
BP8	A build fails because of some flakiness in the execution, whereas it should not	BPO
BP9	Quality gates are defined without developers considering only what dictated by the customer	QA
BP10	The CI server hardware is used for different purposes other than running the CI framework	INF
BP11	Developers do not have a complete control of the environment	CUL
BP12	Lengthy build scripts	BM
BP13	Test cases are not organized in folders based on their purposes	REP
BP14	Missing tests on feature branches	QA

2.2 Quality Measurement

2.2.1 Internal Quality Attributes. Internal quality attributes allow measurement of software artifacts, such as system code. In this work, we will use to measure the internal quality of the code metrics that represent the attributes of cohesion, coupling, size, complexity and inheritance. To assess the systems quality investigated in this work, we used metrics well known from the literature presented in Table 2 Chidamber and Kemerer [5], Destefanis et al. [6], Lorenz and Kidd [13], McCabe [14]. These metrics are supported by the Understand tool[1].

Table 2: Metrics of the internal quality attributes [5, 6, 13, 14].

Attributes	Metric
Cohesion	Lack of Cohesion of Methods (LCOM2)[5]
Coupling	Coupling Between Objects (CBO)[5]
Complexity	Average Cyclomatic Complexity (ACC)[14]
	Sum Cyclomatic Complexity (SCC)[14]
	Nesting (MaxNest)[13]
	Essential Complexity (EVG)[14]
Inheritance	Number Of Children (NOC)[5]
	Depth of Inheritance Tree (DIT)[5]
	Bases Classes (IFANIN)[6]
Size	Lines of Code (LOC)[13]
	Lines with Comments (CLOC)[13]
	Classes (CDL)[13]
	Instance Methods (NIM)[13]

[1]https://scitools.com/

2.2.2 Quality Indicators. There are other ways to measure the quality level of software besides internal quality attributes. In this paper, we address three quality indicators that are actively used in the analyzed projects: (1) Software Reliability, (2) Software Security, and (3) Software Maintainability. According to Pham [17], Software Reliability can be defined as the probability that a software has of not failing for a given period of time and under specific conditions. In other words, it is the ability of software to operate properly within the limits for which it was designed. Thus, this is an important metric from both the developer's and customer's point of view. Software Security, on the other hand, can be defined as the ability of software to continue to function properly even under malicious attack [15]. The concept of software security covers both situations where the software may fail due to normal use and occasions when there is an intelligent agent willing to break the system. Finally, we can define Software Maintainability as the ease with which software can be modified [1]. Software maintainability is actually a grouping of several characteristics such as: readability of the source code, quality of the documentation, and understandability of the software [2].

2.3 Related Work

Some works in the literature have investigated the effects of CI bad practices in software projects. Felidré et al. [11] studied the side effects caused by bad CI practices in 1270 open source projects that use the TravisCI tool. As a result, the authors pointed out that ± 60% of the evaluated projects suffer from the low rate of commits performed, 85% of the analyzed systems have at least one damaged build, and that the projects with a smaller scope (with up to 1000 lines of code), were the ones that required the most time to handle inconsistent builds. In an initial study the authors Silva and Bezerra [19], analyzed CI bad practices in two closed-source projects. The study's main conclusions were identified that the most frequent CI bad practices are linked to repository management factors and the culture of the environment. Furthermore, it was also identified that the correction of build failures is delayed. The main negative effects of bad practices are related to project management and CI development and infrastructure. Zampetti et al. [24] investigated the impact of CI bad practices through semi-structured interviews with 13 experts and mining more than 2,300 Stack Overflow posts. The authors formulated a catalog that contains 79 CI bad smells, grouped into 7 categories inherent to the different dimensions of the CI pipeline process and management. Unlike these works, our work investigates the effects of CI bad practices on the internal quality attributes. We used the catalog of CI bad practices defined by Zampetti et al. [24] for our investigation.

3 STUDY SETTINGS

3.1 Goal and Research Questions

Our main objective in the present work is to analyze the effects of CI bad practices regarding software quality, thus we have designed the following research questions:

- **RQ$_1$:** *What is the impact on the internal quality attributes of closed-source systems after the implantation of CI?* Through this research question, we performed an initial analysis of the projects selected for the study to verify if there were any

differences in the internal quality attributes that might indicate the influence of bad practices in the analyzed software.
- **RQ_2:** *What can quality indicators over time reveal in projects affected by CI bad practices?* In our second research question, we investigate the behavior of the quality indicators used in the analyzed projects to see if the use of CI has succeeded in improving software quality over time.
- **RQ_3:** *Which CI bad practices have the highest priority for resolution for quality improvement?* Our final research question aims to obtain a prioritization of CI bad practices based on the level of effort to resolve and the impact on software quality so that quality improvement can be achieved more quickly and effectively.

3.2 Study Steps

Step 1: Select systems for analysis. We selected a total of 9 closed-source projects of our industry partners. Table 3 summarizes data from target systems. In the first column, we assign an identifier to each system, in the second column, we present a brief description regarding the purpose of the software, and in the remaining column, the total number of CI pipelines executed in the systems is evidenced target. We used the following acceptance criteria in the choice of systems: (1) systems with at least one year since the beginning of its development, and (2) systems that have CI configuration for at least 6 months. In the work of Silva and Bezerra [19], bad practices were identified in the study environment, so we did not include this factor in the acceptance criteria.

Table 3: General data of the target software systems.

SYSTEM	DOMAIN	PIPELINES
S1	Skills-based Organizational Management	1,764
S2	Student Assistance	731
S3	Foreign Language Reading Proficiency	2,135
S4	Management of Complementary Activities for Graduating Degrees	505
S5	Storeroom Management	106
S6	Organizational Risk Management	818
S7	Social Project Management	303
S8	Electronic Dental Records	1,218
S9	Academic Events Management	255

Step 2: Collect and analyze internal quality attributes. To answer the RQ_1, we conducted a preliminary analysis of the projects and separated those that did not adopt the CI since its inception. We collected the values of the internal quality attributes of the selected systems at two points, one before and one after the introduction of CI in the project. Metrics were collected through a non-commercial license of the Understand[2] tool. Our attributes analysis took place by comparing the values of the metrics before and after the introduction of CI in the projects.

Step 3: Collect and analyze history of quality indicators. To complement the analysis of internal quality attributes and answer our RQ_2, we collected the history of quality indicators from all selected systems. The quality indicators observed were: (1) Software Reliability, (2) Software Security and (3) Software Maintainability. The collection of indicators took place through a script we developed for this purpose[3], so it was possible to obtain the values of the quality indicators automatically from the CI system used in the projects. Invited experts validated the collection script and the data obtained. In addition to quality indicators, we also collect the history of issues raised by static analysis in projects. Our analysis of the quality indicators took place by comparing the values of the indicators in each of the projects.

Step 4: Develop, apply and analyze questionnaire. To answer the last RQ, we developed a questionnaire [4] aimed at the members of the development teams that worked on the analyzed projects. The form was composed of two objective questions where we sought to: (1) discover the relationships between CI bad practices and quality indicators and (2) understand the level of effort required to correct each bad practice in the projects. We carried out an initial application of the questionnaire with a specialist for validation purposes and, based on the feedback obtained, we refactored some points and created an explanatory material (which we refer to in the questionnaire) about poor CI practices and quality indicators to help in the understanding of the participants. We also collected characterization data from the participants: the study level, the role played in the project, and knowledge of CI and software quality. In total, we obtained 21 responses (out of 380 invitations sent) from people who were part of the development team of the analyzed systems.

We analyzed the responses as follows: first, we analyzed the relationship between CI bad practices and quality indicators to classify (1) bad practices according to their level of impact and (2) quality indicators by the level of impact suffered. Finally, we also rank CI bad practices according to the level of effort required to resolve them. For this, we used a strategy of weights associated with the available options for the level of resolution effort: weight 1 for the low level, weight 2 for the medium level and weight 3 for the high level. We calculated a score for each bad practice by multiplying the number of occurrences of each impact level by their respective weight. Finally, we also correlated the impact level on quality and effort level to resolve each bad practice and generated a prioritized list of bad practices that was translated into an effort versus impact matrix that can be useful in selecting bad practices, priority issues for resolution, aiming to improve software quality.

4 RESULTS

4.1 Impact on internal quality attributes after CI introduction (RQ_1)

To answer RQ_1, we proceeded with the static analysis of the source code of the five projects analyzed to obtain the values for the internal quality attributes. For this, we considered two versions of the code of each system: the first was based on the point immediately before the implementation of CI in the project. As a result, we could see an increase in the raw values of all internal quality attributes for each of the 5 systems. Table 4 shows the results of the data collected for each system.

Our analysis considers the Cohesion attribute to be inversely proportional to the other attributes (i.e., the higher the value, the

[2]https://www.scitools.com/

[3]Our artifacts are available at: https://github.com/ruben-silva-dev/VEM-2022

[4]https://forms.gle/KWg9P9CpGj8X7F668

Table 4: Impact of CI implantation in internal quality attributes of systems.

System	Cohesion	Complexity	Inheritance	Coupling	Size
S1	2290 ⇒ 3788 (↑ 65.41%)	718 ⇒ 955 (↑ 33%)	178 ⇒ 230 (↑ 29.21%)	172 ⇒ 290 (↑ 68.6%)	4008 ⇒ 5621 (↑ 40.24%)
S2	1399 ⇒ 1557 (↑ 11.29%)	331 ⇒ 370 (↑ 11.78%)	85 ⇒ 143 (↑ 68.23%)	46 ⇒ 73 (↑ 58.69%)	1670 ⇒ 2101 (↑ 25.8%)
S4	1247 ⇒ 1717 (↑ 37.69%)	355 ⇒ 497 (↑ 40%)	111 ⇒ 129 (↑ 16.21%)	69 ⇒ 102 (↑ 47.82%)	1774 ⇒ 2576 (↑ 45.20%)
S6	2608 ⇒ 2668 (↑ 2.3%)	886 ⇒ 1044 (↑ 17.83%)	257 ⇒ 258 (↑ 0.38%)	180 ⇒ 191 (↑ 6.11%)	5068 ⇒ 5729 (↑ 13.04%)
S8	4478 ⇒ 4846 (↑ 8.21%)	1559 ⇒ 1859 (↑ 19.24%)	353 ⇒ 390 (↑ 10.48%)	553 ⇒ 611 (↑ 10.48%)	8863 ⇒ 11541 (↑ 30.21%)

better). Thus, we could see significant improvements in the cohesion of all systems, especially in the S1 and S4 systems. As can be seen in Table 4 the values of all internal quality attributes increased after our partners adopted CI in the projects. It is important to note that it is expected the increase in internal attributes, such as complexity, during the development; however, another essential fact to be noted is that the cohesion of the code also increased in all cases, so the adoption of CI may have contributed to the maintenance of the cohesion of the code. In this regard, it is essential to note that our partners have implemented static analysis in all projects as part of the CI process. Since static analysis is a vital process for improving code quality, we can say that, indirectly, CI can improve software quality, especially about cohesion.

Finding 1: CI-aided development can help to increase the degree of cohesion of systems.

4.2 Analysis of the evolution of quality indicators (RQ_2)

We performed a historical analysis of the quality indicators for all the selected systems. From the raw data collected in the CI system of the projects, we generated graphs to represent the behavior of (1) the values of the quality indicators and (2) the number of issues reported by the static code analysis system used in the study environment (in this case, SonarQube[5]). We compiled the resulting graphics in Figure 2. The graphics represent the actual quantity of issues reported by the static analysis system. As can be observed, the quality indicators are associated with a scale ranging from 1 to 5, where a higher value implies a better quality for the system. In this case, we have divided the values into five categories according to the severity of each problem (blocker, critical, major, minor, info).

As can be seen, the graphs for each system have different behaviors, but in general, there is a relationship between issues and the level of quality indicators. Another interesting phenomenon to be observed is that, except for S8, all systems maintain some unresolved issues, indicating that the CI process, specifically the static analysis, is not adequately used throughout the organization. The existence of bad CI practices in the organization corroborates this observation, as it is possible to see, for example, (1) the deficient strategy for defining quality levels for projects and (2) the use of standard configuration tools that may not reflect the needs of the

[5]https://www.sonarqube.org/

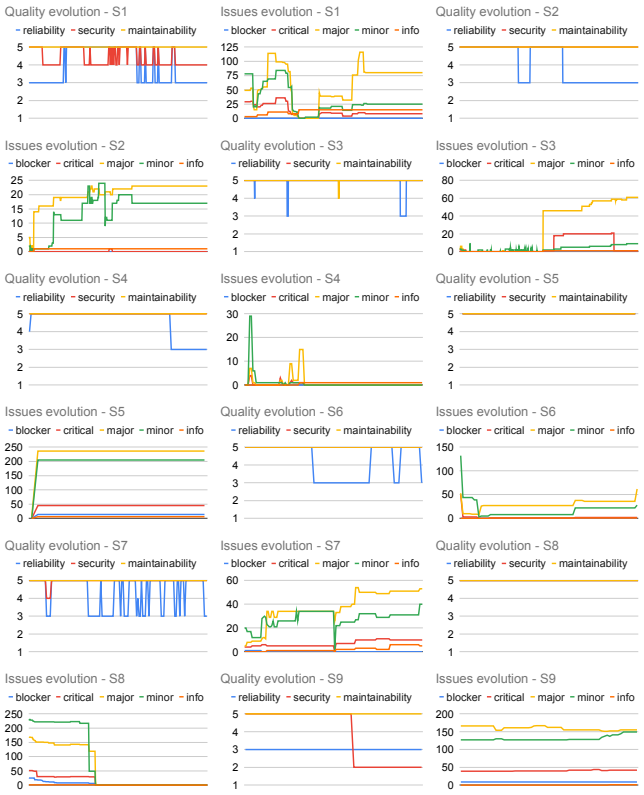

Figure 2: Quality indicators and issues history.

systems. Thus, we can say that bad CI practices can harm (even if indirectly) the level of quality indicators of a software system.

Finding 2: CI bad practices can indirectly harm the quality level of a software product.

4.3 Prioritizing the resolution of CI bad practices (RQ_3)

To finalize the study of the impact of CI bad practices on software quality in our study environment, we performed a correlation analysis between the level of impact and the level of effort to resolve the bad practices and assembled a prioritized view of the CI bad practices that the Figure 3 presents. As can be seen, we have divided the matrix into quadrants that represent the priority of resolution of the bad practices based on the level of impact on quality and the level of effort required to resolve them. Thus the bad practices in the first quadrant are those with the highest priority for resolution since they have a high impact on quality and do not require a high level of effort to correct. Then comes the second quadrant with the bad practices that also have a high impact but require much effort to correct. Finally, come two last quadrants contain the CI bad practices with low impact on quality but with a low and high level of effort (3rd and 4th, respectively).

Through the analysis of the Figure 3, it is possible to verify that the bad practices with the highest priority are related to the categories of Culture (BP6), Quality Assurance (BP3, BP14), and

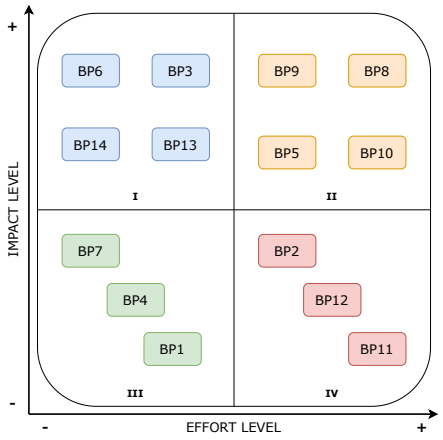

Figure 3: Difficulty and Effort Level Matrix.

Repository (BP13). Furthermore, based on the description of the bad practices, we can say that it is vital to implement a well-structured testing process and adopt a DevOps-oriented culture to achieve better results in improving software quality.

> **Finding 3**: A good test architecture and the implementation of DevOps are CI-related best practices that can improve software quality.

5 THREATS TO VALIDITY

As for the threats to the validity of our study, we can highlight: (1) the number of systems and team members considered in the study, which the closed-source nature of the projects can explain; however, we sought not to limit the data collected both from the API of the CI system and from the questionnaire that was directed even to ex-members of the organization; (2) a second threat is linked to the reliability of the data collection tools that we addressed through the use of a consolidated tool for the collection of the internal quality attributes and the validation of both the script and the questionnaire that we implemented by experts; (3) our last threat to validity consists in the generalizability of the results since we deal with a specific scenario; thus we consider that the results obtained here are applicable only in the context of small organizations, with little CI maturity and that are affected by CI bad practices.

6 FINAL REMARKS

In the present work, we conducted an empirical study to investigate the effects of CI bad practices concerning software quality. Our study was applied in a small industry organization characterized by the presence of CI bad practices in its development process and by the immaturity regarding CI practices. Our results indicate that: (1) the inclusion of CI in the software development process can help to maintain the cohesion of systems, (2) bad CI practices can, even if indirectly, harm software quality and (3)) a good test architecture and the implementation of a DevOps culture can significantly contribute to the improvement of software quality.

Our future works include: (1) the validation of the effort versus impact matrix to verify the quality improvement after solving the CI bad practices, (2) the analysis of the quality indicators in open-source projects with a more expressive set of data to enable a greater ability to generalize the results and (3) application of the study with a larger set of CI bad practices to broaden the view on its effects on software quality.

REFERENCES

[1] 1990. IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990* (1990), 1–84. https://doi.org/10.1109/IEEESTD.1990.101064

[2] K.K. Aggarwal, Y. Singh, and J.K. Chhabra. 2002. An integrated measure of software maintainability. In *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318)*. 235–241. https://doi.org/10.1109/RAMS.2002.981648

[3] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE.

[4] FP Brooks. 1978. The Mythical Man-Month: Essays on Softw. 1st.

[5] Shyam R Chidamber and Chris F Kemerer. 1994. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.* 20, 6 (1994), 476–493.

[6] Giuseppe Destefanis, Steve Counsell, Giulio Concas, and Roberto Tonelli. 2014. Software metrics in agile software: An empirical study. In *International Conference on Agile Software Development*. Springer, 157–170.

[7] John Downs, John Hosking, and Beryl Plimmer. 2010. Status communication in agile software teams: A case study. In *2010 Fifth International Conference on Software Engineering Advances*. IEEE.

[8] PM Duvall. 2018. Continuous Delivery Patterns and AntiPatterns in the Software LifeCycle. *WWW], Available (accessed on 25.7. 2022): https://dzone.com/refcardz/continuous-delivery-patterns* (2018).

[9] Paul M Duvall. 2010. *Continuous Integration: Patterns and Anti-Patterns*. DZone, Incorporated.

[10] Omar Elazhary, Colin Werner, Ze Shi Li, Derek Lowlind, Neil A. Ernst, and Margaret-Anne Storey. 2021. Uncovering the Benefits and Challenges of Continuous Integration Practices. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2021.3064953

[11] Wagner Felidré, Leonardo Furtado, Daniel A da Costa, Bruno Cartaxo, and Gustavo Pinto. 2019. Continuous Integration Theater. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*.

[12] David Goodman and Michael Elbaz. 2008. " It's Not the Pants, it's the People in the Pants" Learnings from the Gap Agile Transformation What Worked, How We Did it, and What Still Puzzles Us. In *Agile 2008 Conference*. IEEE.

[13] Mark Lorenz and Jeff Kidd. 1994. *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc.

[14] Thomas J McCabe. 1976. A complexity measure. *IEEE Trans. Softw. Eng.* 4 (1976), 308–320.

[15] G. McGraw. 2004. Software security. *IEEE Security Privacy* 2, 2 (2004), 80–83. https://doi.org/10.1109/MSECP.2004.1281254

[16] Ade Miller. 2008. A hundred days of continuous integration. In *Agile 2008 conference*. IEEE.

[17] Hoang Pham. 2000. *Software reliability*. Springer Science & Business Media.

[18] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2017. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access* (2017).

[19] Ruben Blenicio Tavares Silva and Carla I. M. Bezerra. 2020. Analyzing Continuous Integration Bad Practices in Closed-Source Projects: An Initial Study. In *Proceedings of the 34th Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 642–647. https://doi.org/10.1145/3422392.3422474

[20] Eliezio Soares, Gustavo Sizilio, Jadson Santos, Daniel Alencar da Costa, and Uirá Kulesza. 2022. The effects of continuous integration on software development: a systematic literature review. *Empirical Software Engineering* 27, 3 (2022), 1–61.

[21] Daniel Ståhl, Torvald Mårtensson, and Jan Bosch. 2017. The continuity of continuous integration: Correlations and consequences. *Journal of Systems and Software* (2017).

[22] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*.

[23] Carmine Vassallo, Fabio Palomba, Alberto Bacchelli, and Harald C Gall. 2018. Continuous code quality: are we (really) doing that?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*.

[24] Fiorella Zampetti, Carmine Vassallo, Sebastiano Panichella, Gerardo Canfora, Harald Gall, and Massimiliano Di Penta. 2020. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering* (2020).