Study of Failures in Continuous Integration Environment

Prameet Keshore Sinha
Department of Computer Science & Engineering
Sharda University
Greater Noida, Utter Pradesh, India
prameet.sinha@gmail.com

Parma Nand
Department of Computer Science and Engineering
Sharda University
Greater Noida, Utter Pradesh, India
parma.nand@sharda.ac.in

Sanjay Singh Chauhan
Department of Computer Science and Engineering
Sharda University
Greater Noida, Utter Pradesh, India
sanjay.chauhan2@sharda.ac.in

Smita Tiwari
Department of Computer Science and Engineering
Sharda University
Greater Noida, Utter Pradesh, India
smita.tiwari@sharda.ac.in

2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT) | 979-8-3503-7809-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICEECT61758.2024.10739077

***Abstract*—Continuous Integration (CI) and Continuous Delivery (CD) is a strategy for faster and high-quality software delivery. CI brings many advantages in software industries, quick detection of bugs when integrating code, faster integration and release time. However, it also comes up with its own challenges to meet organization expectation of being robust and continuously run throughout the project development cycle. It emphasizes that all failures in the CI process, whether genuine software bugs, test failures, or CI tools-related issues, play a critical role in increasing the delivery time of software. The basic aim is to raise awareness of these often-overlooked factors and argues for their equal importance in the context of CI and CD strategies for achieving faster, high-quality software delivery and help organization identify common issues in their CI/CD pipeline as well as make improvements to streamline the process and delivery of software in defined deadline. The paper presents a comprehensive analysis of the various non-software-related failures that occur within CI environments and propose a Pram model that shows promising improvement to mitigate such failures.**

keywords*—*continuous integration, testing, devops, build time.

I. Introduction

Different researchers define CI differently. [1] [2] defines it as practice, while Saidani et al. [3] and other defines it as a process, some says it as a technique [4]. All are true in their own context. CI is a combination of tools that allows teams to not just work collaboratively but also help them to integrate code seamlessly and deliver it on time. In today's fast-paced software development environment, the ability to deliver software functionalities quickly is crucial for the success of any project [5]. Customers demands new features and improvements on a regular basis, and prompt delivery of functionalities can help keep them satisfy while having valuable feedback. CI importance becomes more intuitive when a particular software project follows agile methodology, where continuous testing is a must. However, CI uses is not just constrained to a particular software methodology due to its ability to integrate code effortlessly and deliver it to respective stakeholders without human intervention.

The benefits of using CI are numerous. CI not only helps in the delivery of software but also helps in decreasing the effort of the developers [6]. It ensure rapid feedback among developers [7], which in turn improve code quality [8]. It significantly reduces integration problems [9]. Software applications are quite complex and usually consist of multiple packages and components written by different authors. These components may come from different project teams and can also be a combination of open and closed source software. While this approach has numerous benefits, it can also lead to build breaks as different users may require different adjustments of open-source software to cater to their specific needs [10]. In such scenarios as well, CI can be incredibly helpful in quickly integrating and validating the builds.

CI can be used differently for different phases of software development. It can be configured for making smaller code changes every day, which are monitored by a CI server. When a change is submitted, the CI server pull this change, builds it, and runs simple testing processes [3]. It can also be configured for module level integration where the team submits changes in larger chunks to work with different cross modules of the software and eventually make complete software. The process of CI becomes much more complex in this case as it starts taking more time and becomes very difficult to identify and fix failures. Besides, if any change is merged due to incorrect testing can completely shut down the CI eventually halting the entire development process which can jeopardize software release. Fig.1 shows typical CI work flow. It starts with the developer submitting changes in respective branches (git is widely used in this case). CI server either detects the changes and starts the CI process automatically or it runs at the scheduled time (typically every 4-6 hours) and picks all the changes to build (which include different steps from CI configuration to report generation) and forwards to the test team for automated testing. After successful completion of build and test, the binaries are created and copied either in dedicated filers where further stress testing is done or even in some cases directly delivered to respective stakeholders along with the build and test summary. CI is used amicably to run at specific time or desired number of times as it can also be a resource and power consuming system due to its usage of high performance build machines.
There are decent amount of cost involve in running and maintaining the CI. Company like Facebook, Microsoft, Google spend millions in their CI systems [11] [12]. When adopting CI, it is important to keep the time in building a project minimum. [13]. The goal of CI is to help developers integrate software changes quickly through a CI process. However, it's important to note that the build process can

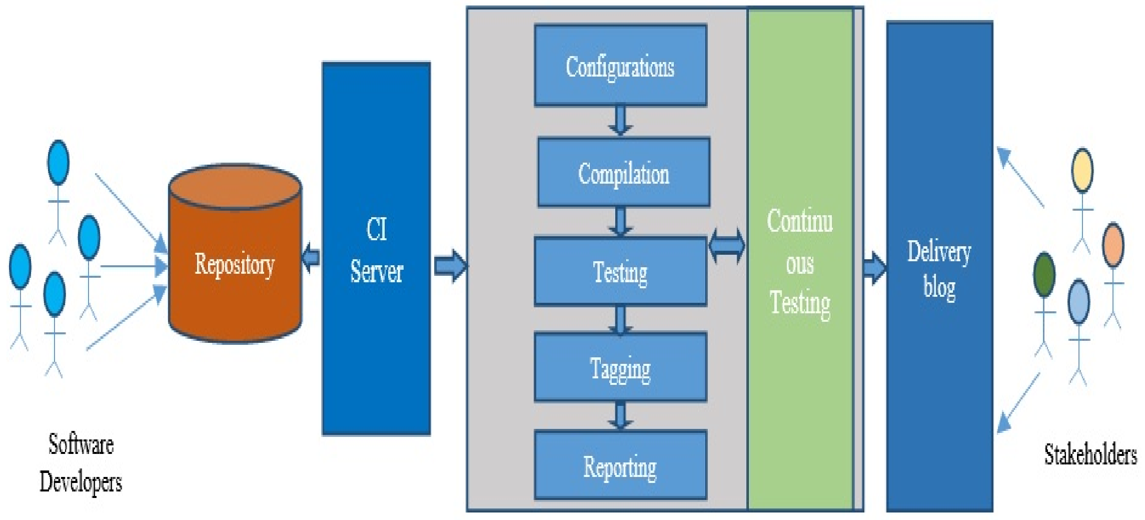

Fig. 1: TYPICAL CI FLOWCHART

be pretty time consuming, especially when running failed builds, which can take hours until to identify the root cause. The success of cross-module integration builds is crucial for software development, but when they fail, it can cause significant delays in project progress. Developers often face difficulties in determining whether a failure is caused by their code or not, or if it is a false positive generated by the CI tool itself. In order to prevent such issues, companies should prioritize producing comprehensive failure reports. These reports can provide developers with the necessary information to promptly identify and resolve any issues, enabling them to focus on their work and maintain the project's momentum. By ensuring that developers have access to the required resources, companies can streamline the development process and ensure that their projects are delivered on time [14].
Some of the common failure in CI is depicted in Table I.

TABLE I: MAJOR TYPES OF FAILURES IN CI ENVIRONMENT

SI.No.	Failure type	Failure reason
1.	Compilation error	Lack of unit testing
2.	Compilation error while integrating code with different module due to missing dependency	Development team Co-ordination error.
3.	Failure in functional testing of software code	Lack of test on device
4.	Failure in iterative testing of the software code	Lack of testing
5.	Random failure in testing (not reproduce in re test	Flaky test case
6.	Failure due to non-software bug	Failed in CI system

II. Literature Survey

CI/CD pipelines are becoming an increasingly popular tool in the software development industry. These pipelines provide a streamlined approach to automate the software development process. They involve integrating, testing, and deploying small code changes regularly, which helps to reduce errors and speed up the development process. In the current market, CI frameworks like Travis CI have gained immense popularity [15], with organizations adopting it to enhance their release process [16]. By utilizing the capabilities of such tools, organizations can build better software products that meet their users' needs. Open source developers have access to a broad range of compute cycles for CI, which are available for free on major code hosting sites like GitHub, GitLab, and Bitbucket [10] [17]. This access makes it easier for developers to build great software products and contribute to the software development community at the same time.
Table II shows research paper wise area of interest in optimizing CI while Fig.2 shows different papers published every year for the improvement of CI. There appears to be a notable disparity between the amount of research being conducted to improve CI tools versus the potential for optimization. Researcher have analyzed millions of build available in github for open source project and found various area of improvements. While the current level of research may be considered limited, it is important to acknowledge the efforts of those who have contributed to this field. Further research in this area would be a valuable investment in driving better outcomes and enhancing the efficacy of CI tools and at the same time help developer community to achieve their goal as per their schedule.
The papers sourced for the Table II and graph in Fig.2 were obtained from the scholarly databases:
1) IEEE Xplore (https://ieeexplore.ieee.org/Xplore/home.jsp)

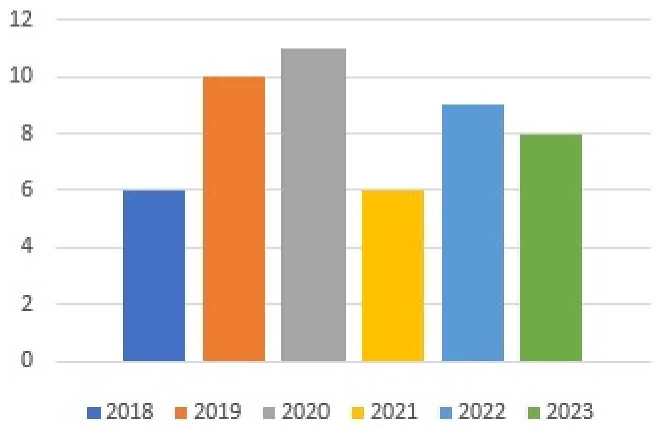

Fig. 2: RESEARCH PAPER IN CI

TABLE II: AREA OF RESEARCH IN CI

Ref. Paper	Year	Build Optimization	CItool Optimization	Test Optimization	Remarks
[3]	2022	✔	✗	✗	Use machine learning to provide suggestions on suspicious CI builds.
[5]	2023	✔	✗	✗	proposed to improve the decision making on Pull Request (PR) submissions.
[7]	2021	✗	✗	✔	Used RiskTopN and RiskBatch model to reduce the number of test cycle in CI by testing in batches thereby provide substantial saving of CI time.
[9]	2021	✗	✗	✔	Use of optimal Reinforcement Learning (pairwise-ACER) provide a solution for test case prioritization in CI environments.
[10]	2022	✔	✗	✔	Proposed solution is to move toward more automated,secure, flexible infrastructure.
[11]	2020	✔	✗	✗	SmartBuildSkip reduced build time by 30% with only 1 extra build in 15% of failures.
[13]	2019	✔	✗	✗	Employs mixed-effects logistic models to analyze the effect of various factors on build durations.
[18]	2023	✔	✗	✗	Used machine learning (ML) for software defect prediction (SDP) where a failing build is treated as a defect to fight against.
[19]	2022	✔	✗	✗	Proposed SKIPCI, an approach to automatically detect commit for CI Skip based on Strength-Pareto Evolutionary Algorithm SPEA-2.
[20]	2022	✔	✗	✗	KOTINOS improves CI efficiency by caching build environments so that only modified files are built in CI.
[21]	2019	✔	✗	✗	Proposed to automate the process of identifying which commits can be skipped by CI thereby helped in reducing the number of commits in CI by 18.16%.
[22]	2023	✔	✔	✔	Improve cost savings and safety by combining predictions from multiple techniques to decide on skipping builds or tests with stronger certainty. Provide cost savings of 11.3%
[23]	2023	✗	✗	✔	Developed and evaluated feature set for training ML models for test case prioritization (TCP) in CI. Model achieved average accuracy of 85%.
[24]	2022	✗	✗	✔	COLEMAN technique for test Case Prioritization (TCP) takes second to prioritize a test suite, contributes efficiently and effectively to address the problem of TCP in CI.
[25]	2020	✗	✗	✔	Prioritize test case based on XCS classifier systems (XCS).
[26]	2020	✗	✗	✔	Proposed technique to Maximize the number of faults discovered in available time during testing.
[27]	2023	✗	✗	✔	Proposed Flakify to minimize inconsistent test cases in CI.
[28]	2021	✔	✗	✔	Proposed FLAST to guide test rerunning and reduce flaky tests in CI.
[29]	2019	✗	✔	✗	Identifies a significant portion of build breakages as noise, caused by environmental factors, cascading errors from previous builds, or allowed failures

2) SpringerLink (https://link.springer.com/)
3) Google Scholar (https://scholar.google.com/)
4) Elsevier Science Direct (https://www.sciencedirect.com/)
5) ACM Digital Library (https://dl.acm.org/)

Different proposals made by researchers to make CI as efficient as possible [19] [30] [20] [21]. One such research is to reduce the computational cost of CI [22]. Developers are always looking for ways to optimize their workflow and boost productivity. One key method is to focus on executing (build or test) tasks that provide the most value, while minimizing time spent on lower-value tasks. To achieve this, many experts recommend skipping (build or test) executions that pass, and instead prioritizing those that fail. After all, a failed execution provides feedback that there is an issue to be resolved, while a successful one simply confirms that everything is working as expected. There are different proposal which have been made for early defect prediction [18] [31] [3] [32]. CI is increasingly adopting microservice architecture to achieve decoupling of solutions [33] [34]. Even research for optimization of testing is also numerous [7] [23] [24] [22] [1] [25] [26]. As software systems grow more complex, the testing process becomes increasingly challenging which is highly time constrained and produces a large volume of data from iterative code commits and testing runs. In this context, machine learning can leverage copious test data to identify test cases that can accelerate the detection of regression bugs introduced during code integration.

III. Discussion

In the fast-paced world of software development, every second counts. Any delays caused by compilation failures can have serious consequences for a project's timeline and success. Although developers are now taking more steps to ensure the quality of their code by compiling it on their own computers before submitting it to the CI system, compilation failures can still occur, for example, due to differences between the compiler version used locally and that of the CI system, or if necessary dependencies are not in place. It is therefore essential for developers to be aware of the potential costs of CI failure, and to be trained in how to work effectively with the CI system, including how to quickly resolve issues with their code in case of failure, particularly when it comes to module-level integration. This way, developers can avoid the costly delays associated with CI failure and help ensure the success of their projects. [35]. The common reason for CI failure is often attributed to flaky tests [27] [28]. Such errors are typically not related to code changes but instead arise due to resource unavailability, machine down, or other environmental factors [12]. Common failure seen in CI which does not occur due to the new changes coming for software integration but from the CI tools, such failure are also considered as noise [29] Some of the most generic failure which are not caused by software developers are depicted below.

- Machine in the pool went down after IT maintenance and CI could not detect and restart.
- Machine hanged while running integration build.
- Machine space full due to simultaneous failure and CI could not restart.
- CI system could not restart after detecting false positive failures.
- System could not auto detect for flaky test case and brake.
- System became slow while running parallel build with more thread.
- Change in CI configuration.
- Machine cache size increases to unacceptable level.
- Filer space for copy binaries get full.
- Resource wait time increases to threshold time in parallel build and hence CI is aborted.
- Regression in new code changes coming for CI software.

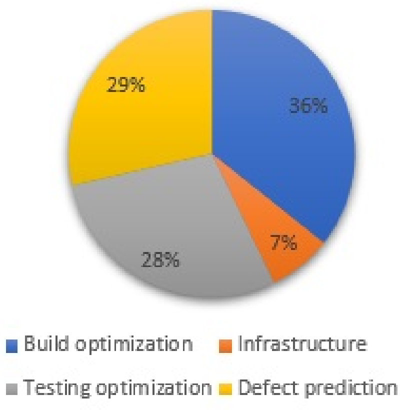

Fig. 3: CI OPTIMIZATION

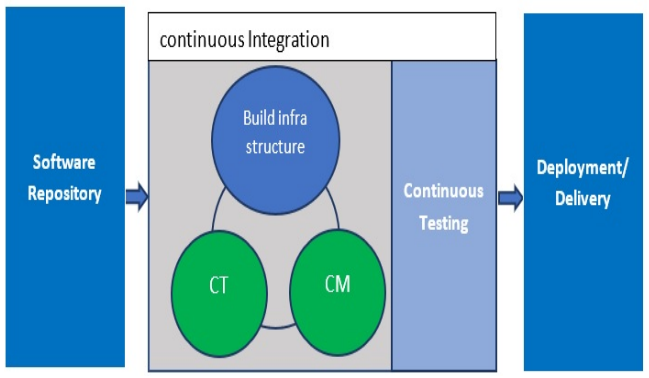

Fig. 4: Proposed Model

Maintaining the health of CI tools has become a major challenge. The number of builds being generated across different companies is staggering, leading to delays in software delivery. To overcome this challenge, companies have implemented various strategies to minimize build breakage during compilation and testing. However, we have found that little attention is given to failures caused by CI tools themselves, as compared to those caused by code changes from developers. We conducted an extensive review to uncover these failures and their solutions, and it's high time we focus on resolving these issues to ensure the smooth functioning of CI tools. Fig.3 shows there are very little being done to tackle failure

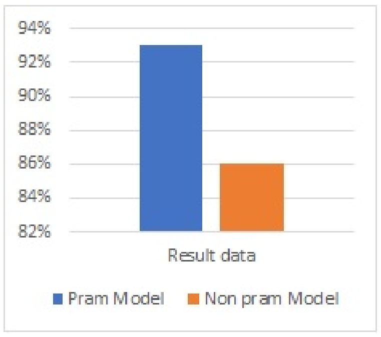

Fig. 5: Model Outcome

occurring by CI infrastructure (Out of 200+ articles searched only 7% out of 70 most relevant papers referred). This discovery has motivated us to explore and identify different types of CI tools related failures, with the goal of improving the overall performance and reliability of the system.

IV. Proposed Model

Designing a step or algorithm that will in advance detect the possible failures which might occurs, based on learning from the previous failures in software continuous integration. That is: Proposal for Pram Model to detect and provide early warning system (EWS) by going through a comprehensive CI step validation before starting integration of software builds.
1. Continuous Training[CT] of the system to handle integration failure. (Creating rules for integration system to handle and auto restart if system encounter new failure.)
2. Continuous Monitoring[CM] of the failures related to non-software issues as well along with genuine software failure. Fig.4 shows how CT and CM interact with CI system.

TABLE III: TRAVIS TORRENT DATA ANALYSIS.

Total build	Passed build	Failed build
1000	750	250

We evaluated data from Travis Torrent (Table III) in proposed pram model (Fig.6). The model contains two part, 1) Continuously training the model by analyzing the failure from CI through various logs file on large number of builds. These error are clustered into genuine software and other environment related issues. 2) Continuously monitoring for new failure which are in particular due to infrastructure and updating. Once the model is evaluated, we apply it back on new set of data to make sure CI predict the outcome and take the corrective action before the build is started. The outcome of this model provide us encouraging result (Fig.5). However we found the training of new failure increases CI time by approximately 2-5 minutes. We believe with more training of the model, the percentage of noise will further decrease

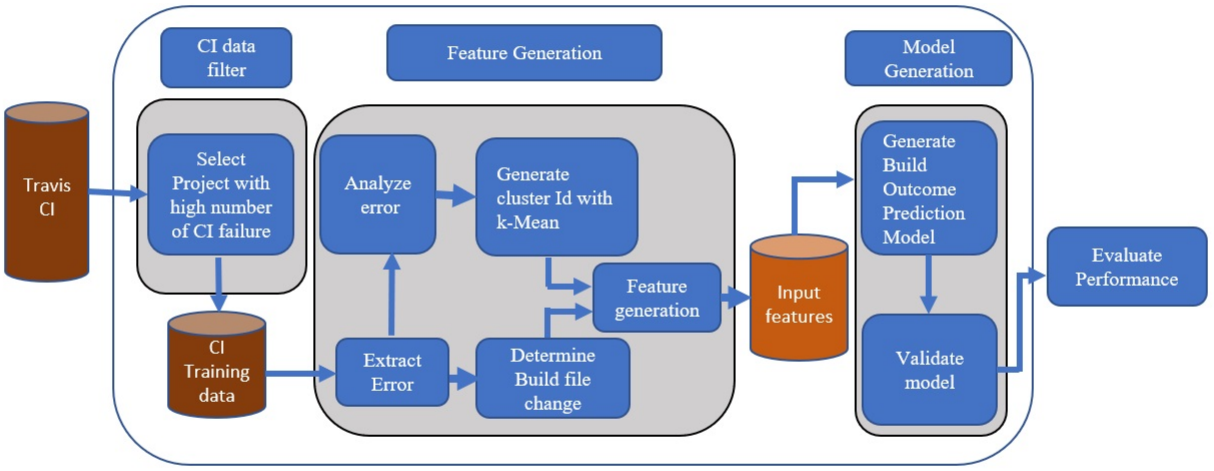

Fig. 6: Pram Model

to minimal acceptable level where as training time will also decrease thereby increasing the robustness of the CI tools.

V. Conclusion

There are number of research done on how to make robust CI system as more software industry are moving to continuous build the software code and detect defect at early phase of software development life cycle(SDLC). There are different proposal made on how to mitigate various failure occurs during compilation and testing due to software issues in an automated CI system across different research paper. This paper highlight major failures in CI which does not occur due to genuine software code (considered as noise) and highlight such failures are equally responsible to delay in CI/CD process. It also depicts the need to have research on how to make a robust CI system which can auto mitigate such failure by using machine learning and artificial intelligence-based solutions, and can unlock new possibilities for predictive identification and mitigation of such issues. The proposed pram model is one way to mitigate environment or infrastructure related failure thereby enabling developers and management to have complete trust in CI system.

References

[1] D. Marijan, M. Liaaen, and S. Sen, “Devops improvements for reduced cycle times with integrated test optimizations for continuous integration,” vol. 1. IEEE Computer Society, 6 2018, pp. 22–27.

[2] E. Klotins, T. Gorschek, K. Sundelin, and E. Falk, “Towards cost-benefit evaluation for continuous software engineering activities,” *Empirical Software Engineering*, vol. 27, no. 6, p. 157, 2022.

[3] I. Saidani, A. Ouni, and M. W. Mkaouer, “Improving the prediction of continuous integration build failures using deep learning,” *Automated Software Engineering*, vol. 29, no. 1, p. 21, 2022.

[4] E. Soares, G. Sizilio, J. Santos, D. A. da Costa, and U. Kulesza, “The effects of continuous integration on software development: a systematic literature review,” *Empirical Software Engineering*, vol. 27, no. 3, p. 78, 2022.

[5] J. H. Bernardo, D. A. da Costa, U. Kulesza, and C. Treude, “The impact of a continuous integration service on the delivery time of merged pull requests,” *Empirical Software Engineering*, vol. 28, 7 2023.

[6] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, “The work life of developers: Activities, switches and perceived productivity,” *IEEE Transactions on Software Engineering*, vol. 43, no. 12, pp. 1178–1193, 2017.

[7] M. J. Beheshtian, A. H. Bavand, and P. C. Rigby, “Software batch testing to save build test resources and to reduce feedback time,” *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2784–2801, 2021.

[8] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M. A. Storey, “Uncovering the benefits and challenges of continuous integration practices,” *IEEE Transactions on Software Engineering*, vol. 48, pp. 2570–2583, 7 2022.

[9] M. Bagherzadeh, N. Kahani, and L. Briand, “Reinforcement learning for test case prioritization,” *IEEE Transactions on Software Engineering*, vol. 48, no. 8, pp. 2836–2856, 2021.

[10] T. Gamblin, D. S. Katz, J. Carver, and K. Morris, “Overcoming challenges to continuous integration in hpc,” *Computing in Science and Engineering*, vol. 24, pp. 54–59, 11 2022.

[11] X. Jin and F. Servant, “A cost-efficient approach to building in continuous integration,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 13–25.

[12] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, “Taming google-scale continuous testing,” in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2017, pp. 233–242.

[13] T. A. Ghaleb, D. A. da Costa, and Y. Zou, “An empirical study of the long duration of continuous integration builds,” *Empirical Software Engineering*, vol. 24, pp. 2102–2139, 8 2019.

[14] M. R. Islam and M. F. Zibran, “Insights into continuous integration build failures,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 467–470.

[15] M. Beller, G. Gousios, and A. Zaidman, “Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017, pp. 447–450.

[16] R. Abdalkareem, S. Mujahid, and E. Shihab, “A machine learning approach to improve the detection of ci skip commits,” *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2740–2754, 2020.

[17] Z. Pan, W. Shen, X. Wang, Y. Yang, R. Chang, Y. Liu, C. Liu, Y. Liu, and K. Ren, “Ambush from all sides: Understanding security threats in open-source software ci/cd pipelines,” *IEEE Transactions on Dependable and Secure Computing*, 2023.

[18] M. Kawalerowicz and L. Madeyski, “Continuous build outcome prediction: an experimental evaluation and acceptance modelling,” *Applied Intelligence*, vol. 53, pp. 8673–8692, 4 2023.

[19] I. Saidani, A. Ouni, and M. W. Mkaouer, “Detecting continuous integration skip commits using multi-objective evolutionary search,” *IEEE Transactions on Software Engineering*, vol. 48, pp. 4873–4891, 12 2022.

[20] K. Gallaba, J. Ewart, Y. Junqueira, and S. McIntosh, “Accelerating continuous integration by caching environments and inferring dependencies,” *IEEE Transactions on Software Engineering*, vol. 48, pp. 2040–2052, 6 2022.

[21] R. Abdalkareem, S. Mujahid, E. Shihab, and J. Rilling, “Which commits can be ci skipped?” *IEEE Transactions on Software Engineering*, vol. 47, no. 3, pp. 448–463, 2019.

[22] X. Jin and F. Servant, “Hybridcisave: A combined build and test selection approach in continuous integration,” *ACM Transactions on Software Engineering and Methodology*, vol. 32, 5 2023.

[23] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand, “Scalable and accurate test case prioritization in continuous integration contexts,”

IEEE Transactions on Software Engineering, vol. 49, pp. 1615–1639, 4 2023.

[24] J. A. Lima and S. R. Vergilio, "A multi-armed bandit approach for test case prioritization in continuous integration environments," *IEEE Transactions on Software Engineering*, vol. 48, pp. 453–465, 2 2022.

[25] L. Rosenbauer, A. Stein, R. Maier, D. Pätzel, and J. Hähner, "Xcs as a reinforcement learning approach to automatic test case prioritization," in *Proceedings of the 2020 genetic and evolutionary computation conference companion*, 2020, pp. 1798–1806.

[26] T. Shi, L. Xiao, and K. Wu, "Reinforcement learning based test case prioritization for enhancing the security of software," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2020, pp. 663–672.

[27] S. Fatima, T. A. Ghaleb, and L. Briand, "Flakify: A black-box, language model-based predictor for flaky tests," *IEEE Transactions on Software Engineering*, 2022.

[28] R. Verdecchia, E. Cruciani, B. Miranda, and A. Bertolino, "Know you neighbor: Fast static prediction of test flakiness," *IEEE Access*, vol. 9, pp. 76 119–76 134, 2021.

[29] T. A. Ghaleb, D. A. da Costa, Y. Zou, and A. E. Hassan, "Studying the impact of noises in build breakage data," *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1998–2011, 2019.

[30] K. Gallaba, "Improving the robustness and efficiency of continuous integration and deployment," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 619–623.

[31] F. Hassan and X. Wang, "Change-aware build prediction model for stall avoidance in continuous integration," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 2017, pp. 157–162.

[32] J. Xia and Y. Li, "Could we predict the result of a continuous integration build? an empirical study," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2017, pp. 311–315.

[33] J. Xuan, T. Duan, Q. Guo, F. Gao, J. Li, X. Qiu, and S. Wu, "Microservice publishing technology based on devops architecture." Institute of Electrical and Electronics Engineers Inc., 10 2021, pp. 1310–1314.

[34] D. Pianini and A. Neri, "Breaking down monoliths with microservices and devops: An industrial experience report." Institute of Electrical and Electronics Engineers Inc., 2021, pp. 505–514.

[35] M. R. Wrobel, J. Szymukowicz, and P. Weichbroth, "Using continuous integration techniques in open source projects–an exploratory study," *IEEE Access*, 2023.