Reducing Cost in Continuous Integration with A Collection of Build Selection Approaches

Xianhao Jin
Virginia Tech
Blacksburg, Virginia, USA
xianhao8@vt.edu

ABSTRACT

Continuous integration (CI) is a widely used practice in modern software engineering. Unfortunately, it is also an expensive practice — Google and Mozilla estimate their CI systems in millions of dollars. To reduce CI computation cost, I propose the strategy of build selection to selectively execute those builds whose outcomes are failing and skip those passing builds for cost-saving. In my research, I firstly designed SmartBuildSkip as my first build selection approach that can skip unfruitful builds in CI automatically. Next, I evaluated SmartBuildSkip with all CI-improving approaches for understanding the strength and weakness of existing approaches to recommend future technique design. Then I proposed PreciseBuildSkip as a build selection approach to maximize the safety of skipping builds in CI. I also combined existing approaches both within and across granularity to be applied as a new build selection approach — HybridBuildSkip to save builds in a hybrid way. Finally, I plan to propose a human study to understand how to increase developers' trust on build selection approaches.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**; **Software testing and debugging**; **Maintaining software**.

KEYWORDS

continuous integration, build strategies, maintenance cost

ACM Reference Format:
Xianhao Jin. 2021. Reducing Cost in Continuous Integration with A Collection of Build Selection Approaches. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3468264.3473103

1 INTRODUCTION

Continuous integration (CI) is a popular practice in modern software engineering that encourages developers to build and test their software in frequent intervals [10]. While CI is widely recognized as a valuable practice, it also incurs a very high cost — mostly for the computational resources required to frequently run builds [16–18, 33, 40]. Overall, adopting CI can be very expensive. Google estimates the cost of running its CI system in millions of dollars [18], and Mozilla estimates theirs as $201,000 per month [23]. For smaller-budget companies that have not yet adopted CI, this high cost can pose a strong barrier.

There are many existing research approaches to save cost in CI, including techniques to make CI builds faster by accelerating preparation phase [6, 12] or running fewer tests [27]. In contrast, I propose to save CI cost by selectively executing a subset of builds among all builds — build selection. Practitioners are normally careful when triggering builds — existing work [4] finds that 82% of builds are passing. These passing builds are not able to provide actionable feedback but takes a long time for the result — developers have to wait more than 30 minutes to receive the test results [25]. Thus, detecting and selecting only failing builds to execute can provide much cost-saving and reduce time-to-feedback as well.

To achieve this goal, I proposed a collection of novel build selection strategies that focus on skipping builds that are predicted to pass to reduce the cost of CI. My goal is to **execute fewer builds**, while **running as many failing builds as early as possible**. The rationale behind my strategy is: skip builds that are predicted to pass and execute builds that are likely to fail. I posit that the value of CI lies in the observation of failures and its cost lies in the build executions because failing builds can provide actionable feedback.

However, build selection to save CI cost has many challenges. First of all, existing build outcome predictors cannot be used for build selection because their predictions rely on the result of previous build which may not be always available because the previous build can be skipped. To address this challenge, I proposed my first build selection approach, SmartBuildSkip. SmartBuildSkip focuses on predicting first failures without relying on any information from the last build and detect subsequent failures by continuously executing builds after failing builds until a passing build is observed instead of making predictions. This two-phase process ensures that SmartBuildSkip can save CI cost effectively.

Second, there are highly related existing fields to build selection such as test selection and build prioritization. Comparing all these approaches can be beneficial by understanding their own strengths and learning from each other. Despite that, these approaches are evaluated under different settings, making it a challenge to compare them. Thus, we proposed a first evaluation work to compare these CI-improving techniques in the same CI environment. This work aims to understand the strengths of various CI-improving techniques and can recommend future build selection design.

Another challenge for build selection lies in how to address mispredictions on failing builds. Failing builds are not desirable to skip and mistakenly predicting these failing builds to pass can

ESEC/FSE '21, August 23–28, 2021, Athens, Greece

ACM ISBN 978-1-4503-8562-6/21/08…$15.00
https://doi.org/10.1145/3468264.3473103

result in delaying failure observations, *i.e.,* build failures are wrongly skipped and thus cannot be observed on time. Many practitioners may prefer a build selection technique that maximizes its safety. Thus, I proposed PreciseBuildSkip that aims to maximize the observed build failures while saving CI cost to address this problem.

Besides, normally safe build selections that capture all failing builds are not able to save much cost. Thus, it is a challenge for build selection approaches to enlarge cost-saving without sacrificing the safety. To address this problem, I proposed HybridBuildSkip that combines existing selection approaches both within and across granularities. HybridBuildSkip combines selection approaches to make better predictions within their granularity and then combines them across granularity to earn more cost-saving by skipping both full and partial builds.

The final challenge for build selection approaches focuses on how to build developers' trust on these approaches. Practitioners may require more information about build selection decisions to adopt these techniques. Therefore, I plan to propose the first human study on understanding the barriers of adopting build selection approaches and how to increase developers' trust on these automated build selection recommendations.

To summarize, my research includes a collection of build selection approaches: SmartBuildSkip as my first build selection technique, PreciseBuildSkip that maximizes the safety and HybridBuildSkip that increases the cost-saving by combining existing approaches. My research also includes a comprehensive evaluation of CI-improving approaches and a human study about developers' opinions on build selection approaches. My work [20–22] have been published in ICSE'20 and ICSE'21.

2 MOTIVATION AND RELATED WORK

Characterizing Builds. There are many existing work aiming to characterize builds. Abdalkareem *et al.* [2] targeted characterizing CI-Skip builds which are likely to be skipped by developers. They proposed a human study to understand reasons why developers decide to skip builds and designed a rule-based technique based on their findings. Other works aimed to characterize builds to predict build outcomes. Hassan *et al.* found that features related to the previous build are most effective when predicting build outcomes [14]. Ni *et al.* [31] and Chen *et al.* [7] also found that historical features are the most useful features in predicting the build outcome.

Other studies investigated the reasons for build failures. Some studies [39] sort common build failures into compilation [42], unit test, static analysis [41], and server errors. Paixão *et al.* [32] studied the interplay between non-functional requirements and failing builds. Other studies found factors that contribute to build failures: architectural dependencies [36] and other more specific factors, such as the stakeholder role, the type of work item and build [24], or the programming language [4]. Other less obvious factors that could cause build failures are build environment changes or flaky tests [34]. Some work [34][20] also found that build failures tend to occur consecutively, which Gallaba *et al.* [11] describe as "persistent build breaks". Barrak *et al.* found that features related to code smell can be effective for predicting build outcomes [3]. Other studies found change characteristics that correlate with failing builds, such as: code churn [19, 34], build tool [19], and statistics on the last build and the history of the committer [31]. Hassan *et al.* [15] found that build scripts can result in build failures and proposed an approach to fix this kind of failing build automatically.

However, existing works that aim to characterize builds and predict their outcomes are not proposed to save the cost of CI. In contrast, my approaches are able to save CI computational cost by predicting the build outcome, *i.e.,* characterizing builds. My approaches also don't rely on the information of the previous build to make predictions especially when the previous build is skipped.

Approaches to Reduce the Cost of CI. A popular effort to reduce the cost of CI focuses on understanding what causes long build durations *e.g.,* [38]. Thus, most of the approaches skip tests within builds, *e.g.,* tests that historically failed less [9, 16], that have a long distance with the code changes [29], that test unchanged modules [37], or that are predicted to pass by a machine learning classifier [27]. These techniques are based on regression test selection (RTS) *e.g.,* [13, 35, 43, 44]. While these techniques focus on making every build cheaper, our work addresses the cost of CI differently: by reducing the total number of builds that get executed and reducing the number of tests in those executed builds. A related effort for improving CI aims at prioritizing its tasks to provide early fault observation. The most common approach in this direction is to apply test case prioritization (TCP) techniques *e.g.,* [8, 9, 26, 28, 30] so that builds fail faster. Another similar approach achieves faster feedback by prioritizing builds instead of tests [25] when there is a queue of builds waiting for executed under limited computation source. Prioritization-based techniques advance feedback but are not able to save cost, *i.e.,* all builds and tests still get executed. Finally, other existing efforts to reduce cost in CI make individual builds cheaper, by running less computation in them *e.g.,* [6][12].

The most related existing research to my work is proposed by Abdalkareem *et al.* [1, 2]. This prior work aims to predict and skip builds that are likely to be skipped by developers. However, predicting builds that may be decided by developers to skip can limit the cost-saving because it is not common for developers to skip builds manually. In contrast, my work targets saving more cost in CI by skipping unfruitful tasks, *i.e.,* only executing a subset of tasks. Also, my work aims at solving the main concern of adopting build selection approaches — skipping failing builds.

3 PUBLISHED WORK

I have already developed one novel build selection approach [20] and evaluated it with other CI-improving approaches [22].

3.1 SmartBuildSkip

Research Method. I created my first build selection approach — SmartBuildSkip to ensure that the build outcome predictor does not require information from the previous build when the last build is actually skipped. The design of SmartBuildSkip is based on a novel conceptual separation of build failures into first and subsequent failures, to improve the effectiveness of build prediction models. SmartBuildSkip differentiates between first failures and subsequent failures, following a two-phase process. First, SmartBuildSkip uses a machine-learning classifier to predict build outcomes to catch first failures. After it observes a first failure, it then determines that all subsequent builds will fail — until it observes a build pass and then changes its operation to predicting again. To motivate my design, I performed two empirical studies, of the prevalence of build passes over build failures, and of subsequent failures over first failures. I also studied the factors that predict first failures. Furthermore, I performed an evaluation of the extent to which SmartBuildSkip can save cost in CI while keeping most of its value, with the ability of customizing its cost-value balance. This work is based on TravisTorrent dataset [5] including 274,742 builds from 359 projects.

Result. From the empirical studies, I found that the passing builds represented a very large proportion (88% of all builds) for most projects and there are many of subsequent failing builds (52% of all failing builds). Both findings motivates the design of SmartBuildSkip. Regarding of features to predict first build failures, I found that features related to task complexity such as number of changed source lines are effective for first failure prediction. I also found that some kinds of projects (*e.g.,* older and bigger) are more likely to fail. I encoded these findings into SmartBuildSkip and evaluated it with the state-of-the-art build prediction technique [14]. In our experiments, SmartBuildSkip significantly improved the accuracy of the state-of-the-art build predictor — up to median 8% F-measure for first failures, and up to median 52% F-measure for all failures. Finally, SmartBuildSkip 's predictions resulted in high savings of build effort. In its most conservative configuration, SmartBuildSkip saved a median 30% of all builds by only incurring a median delay of 1 build in a median 15% build failures. In a more cost-saving-focused configuration, SmartBuildSkip saved a median 61% of all builds by incurring a 2-build delay for 27% of build failures.

3.2 Evaluate CI-Improving Approaches

Research Method. There are many existing approaches that are proposed to improve CI. It will be beneficial for build selection approaches to learn from those techniques. However, the existing CI-improving techniques have been evaluated under different settings, making it hard to compare them. Thus, I performed the first comprehensive evaluation of CI-improving techniques under the same environment to understand the strengths of these techniques for three dimensions: (D1) computational-cost reduction, (D2) missed failure observation, and (D3) early feedback and provide evidence for researchers to design future CI-improving techniques. The evaluation includes a replication of 14 variants of 10 CI-improving techniques from 4 technique families, representing the two goals (time-to-feedback and computational-cost reduction) and the two levels of granularity (build-level and test-level) for which such techniques have been proposed. I used a collection of metrics to measure the performance of CI-improving techniques across the three dimensions. To compare the techniques in the same environment, I extended the dataset used in §3.1 with: detailed test and commit information. The new dataset is named CIBench [21].

Result. I have multiple interesting observations in term of all these three dimensions that can recommend future build selection technique design. First, I observed that existing selection approaches have no preference on saving long duration executions, *i.e.,* they generally don't distinguish long duration builds or tests with short duration ones. Therefore, the first evidence of future technique design is that future selection approaches can be more targeted to save more long duration executions. Then I found that those approaches that aim to save cost safely can still achieve high savings but are not always safe, *i.e.,* they still have some missed failure observation. Thus, I recommend future selection approaches to have better definition of saving safe cost in CI — be safer in cost-saving.

Next, I found that build preparation phase is time consuming. This shows an opportunity for test-granularity techniques to incentivize skipping full builds to be able to also skip the build-preparation cost. On the other hand, skipping tests can still save some cost. This incentive revealed the promise of hybrid test and build selection techniques. Build selection techniques have the strength of skipping many full builds, and test selection technique shave the strength of skipping builds partially. Future hybrid techniques could achieve both goals. I also observed that higher savings means more missed failure observation, which also means a longer in term of feedback time. This shows a trade-off between cost-saving and failure-observation. Thus, future research could design better techniques to break this trade-off as well as better metrics to evaluate these approaches. Finally, I found that prioritization approaches advance time-to-feedback, but provides no computational cost reduction. As a result, I recommend future techniques to combine both selection and prioritization approaches in both granularities.

4 WORK IN PROGRESS

I have also developed two build selection approaches in a safe mode and hybrid mode. Both work are submitted and under review.

4.1 PreciseBuildSkip

Research Method. From the results of §3.2, I found that all build selection approaches suffer from undesirable skipped failing builds, even for those approaches that are supposed to skip builds safely. To minimize the side-effect of mispredictions of failing builds by build-selection approaches, I proposed PreciseBuildSkip as a technique that provides cost-savings in CI while capturing an overwhelming majority of failing builds. To achieve this goal, I firstly performed two empirical studies to understand which builds are safe to skip. I referred to CI-Skip rules [2] that are supposed to be safe and extended them with more rules. I also developed Exceptions for complementing CI-Skip rules. Additionally, I encoded the findings of the empirical studies in an automated build-selection technique, HybridBuildSkip and evaluated it with all existing build selection approaches. Finally, I designed two novel metrics that are able to compare build-selection techniques in a more comprehensive way as a response to the findings in §3.2 that researchers should design more comprehensive metrics for comparing build selection approaches. This work uses the same dataset as §3.2.

Result. I performed multiple observations in my studies. First, I observed that no CI-Skip rule is completely safe — all CI-Skip rules captured some builds that ended up failing. Generally, as CI-Skip rules provided higher potential cost savings, they also skipped more failing builds. Therefore, CI-Skip rules cannot be used as-is to safely skip builds. Second, I identified the four main Exceptions why builds under CI-Skip rules may fail: (1) changes in build scripts, (2) in configuration files, (3) subsequent failures, and (4) increasing platform numbers. In particular, the subsequent-failure Exception was the most common. Thus I used all these four Exceptions to complement CI-Skip rules. Third, my proposed safe approach to build selection, PreciseBuildSkip, provided both higher cost saving and failure observation rates than the state-of-the-art build-selection techniques. I designed PreciseBuildSkip with customizable tolerance to failure observation (higher rates of skipped build failures allow higher cost savings). When customized for highest safety, PreciseBuildSkip observed 100% build failures, while still saving 5.5% of build executions. I also performed a sensitivity analysis, in which I observed that our Exception features provided a strong contribution to PreciseBuildSkip 's cost savings and safety.

4.2 HybridBuildSkip

Research Method. According to my previous work (see §3.2), both build selection and test selection approaches miss some opportunities for cost-saving: test selection approaches seldom skip time-consuming build preparation phase and build selection approaches cannot save cost in executed builds. Thus, I proposed the first hybrid build selection approach — HybridBuildSkip that allows skipping both full and partial builds. HybridBuildSkip also aims to address the problem that PreciseBuildSkip is not able to provide high cost-savings when observing the majority of build failures. Besides, HybridBuildSkip also takes advantage of all existing approaches, *i.e.,* combines them in a hybrid way. I believe that hybridizing existing techniques can enlarge cost-saving while not sacrificing more missed failure observations. Since HybridBuildSkip requires multiple prediction processes of other techniques which may incur a concern that the actual execution time may cancel the its achieved cost-saving. Therefore, I additionally compare the total execution time of HybridBuildSkip with its saved duration to understand the actual achieved cost-reduction. This work also uses the same dataset as my previous work did (see §3.2).

Result. I observed that combining existing build selection techniques in a hybrid way can help save cost more efficiently — HybridBuildSkip outperforms all existing techniques by saving more cost while achieving same failure observations. I also found that the feature of whether the build is a subsequent failing build is the most effective feature for HybridBuildSkip. HybridBuildSkip can be customized to save 9% of build duration while observing 100% of failing builds. I identified that skipping both full and partial builds can improve the cost-saving productivity for CI build selection techniques while generally not sacrificing to skip more failing builds or tests — the saved duration can be increased from 9% to 14% while the observed failing builds remain the same 100% and the observed failing tests only drops from 100% to 99.4%. Additionally, I demonstrated that the total execution time of HybridBuildSkip is negligible compared to its saved build duration.

5 FUTURE WORK

In the future, I plan to perform a human study to understand how to fit my approaches in practical context.

5.1 Increase Trust on Build Selection

Research Method. Applying my approaches as-is may produce extra cost that barriers the adoption, *e.g.,* extra explanation on prediction result may be required to build the trust with developers. Thus, I plan to perform a human study to understand the gap between technique design and its adoption in the industrial context for build selection approaches including the extra cost produced after applying one technique and how to build the trust between techniques and developers. The result of this study can be used to guide future technique design in a practical way by increasing developers' trust on build selection approaches. I expect to learn what information of build selection technique is required for developers and what is their preference on the balance between cost saving and observation of failures. Besides, the human study can also involve insights on new features for build result prediction on different types of build failures.

Current Progress. We expect to understand how our build selection approaches can fit practitioners' usage context through this project. For example, through understanding the preference on the trade-off between cost-saving and mispredictions, we can get a sense of what is the most appropriate sensitivity to tune the technique and make comparison between techniques. We can also focus on predicting those failing builds that are more important to the developers. Also, we expect to understand the extra cost produced when applying a build selection technique. For example, developers may not trust the prediction result until we provide supporting information such as the most effective feature in this prediction. Finally, we want to collect the feedback about applying our technique into developers' industrial practice and design a technique that fits better in the practical scenario. For this task, we have elaborated a preliminary set of interview questions including questions of CI usage and how to build trust on build selection approaches and we have obtained IRB approval from Virginia Tech.

6 CONTRIBUTIONS

My dissertation work can provide multiple contributions. First, it provides different ways to save the computational cost in Continuous Integration by selectively executing builds. One of my work [20] has been published in ICSE'20. Second, my dissertation includes the first comprehensive evaluation of CI-improving approaches. This work provides evidence for researchers to design future CI-improving techniques. This work also arises some problems for researchers to address, *e.g.,* design better metrics for evaluation. This work [22] has been published in ICSE'21. Next, my dissertation proposes comprehensive metrics for evaluation cost-saving approaches in CI and hybridize existing works as a response to my prior work. Finally, my dissertation pioneers in many aspects in CI build selection, for example: My work is the first to propose the trade-off between cost-saving and failure-observation in build selection techniques and aim to address this problem by safe cost-reduction. My work is the first to differentiate first and subsequent failures to make predictors more practical and this has been demonstrated as an important feature for build selection approaches. My work is also the first work to combine build and test selection approaches to enlarge cost-saving.

REFERENCES

[1] Rabe Abdalkareem, Suhaib Mujahid, and Emad Shihab. 2020. A Machine Learning Approach to Improve the Detection of CI Skip Commits. *IEEE Transactions on Software Engineering (TSE)* (2020), To Appear.

[2] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. 2019. Which commits can be CI skipped? *IEEE Transactions on Software Engineering* (2019).

[3] Amine Barrak, Ellis E Eghan, Bram Adams, and Foutse Khomh. 2021. Why do builds fail?—A conceptual replication study. *Journal of Systems and Software* (2021), 110939.

[4] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, my tests broke the build: An explorative analysis of Travis CI with GitHub. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on.* IEEE, 356–367.

[5] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on.* IEEE, 447–450.

[6] Ahmet Celik, Alex Knaust, Aleksandar Milicevic, and Milos Gligoric. 2016. Build system with lazy retrieval for Java projects. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, 643–654.

[7] Bihuan Chen, Linlin Chen, Chen Zhang, and Xin Peng. 2020. BUILDFAST: History-Aware Build Outcome Prediction for Fast Feedback and Reduced Cost in Continuous Integration. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE).* IEEE, 42–53.

[8] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering* 28, 2 (2002), 159–182.

[9] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for improving regression testing in continuous integration development environments. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering.* 235–245.

[10] Martin Fowler and Matthew Foemmel. 2006. Continuous integration. *ThoughtWorks) http://www. thoughtworks. com/Continuous Integration. pdf* 122 (2006), 14.

[11] Keheliya Gallaba, Christian Macho, Martin Pinzger, and Shane McIntosh. 2018. Noise and heterogeneity in historical build data: an empirical study of Travis CI. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering.* ACM, 87–97.

[12] Alessio Gambi, Zabolotnyi Rostyslav, and Schahram Dustdar. 2015. Improving cloud-based continuous integration environments. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2.* IEEE Press, 797–798.

[13] Milos Gligoric, Lamyaa Eloussi, and Darko Marinov. 2015. Practical regression test selection with dynamic file dependencies. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis.* 211–222.

[14] Foyzul Hassan and Xiaoyin Wang. 2017. 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, 157–162.

[15] Foyzul Hassan and Xiaoyin Wang. 2018. HireBuild: An automatic approach to history-driven repair of build scripts. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE).* IEEE, 1078–1089.

[16] Kim Herzig, Michaela Greiler, Jacek Czerwonka, and Brendan Murphy. 2015. The art of testing less without sacrificing quality. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 483–493.

[17] Michael Hilton, Nicholas Nelson, Timothy Tunnell, Darko Marinov, and Danny Dig. 2017. Trade-offs in continuous integration: assurance, security, and flexibility. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering.* ACM, 197–207.

[18] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.* ACM, 426–437.

[19] Md Rakibul Islam and Minhaz F Zibran. 2017. Insights into continuous integration build failures. In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on.* IEEE, 467–470.

[20] Xianhao Jin and Francisco Servant. 2020. A Cost-efficient Approach to Building in Continuous Integration. In *Proceedings of the 42th International Conference on Software Engineering.* 13–25.

[21] Xianhao Jin and Francisco Servant. 2021. CIBench: A Dataset and Collection of Techniques for Build and Test Selection and Prioritization in Continuous Integration. In *Proceedings of the 43th International Conference on Software Engineering, Artifact Evaluation track.* 2 pages, To appear.

[22] Xianhao Jin and Francisco Servant. 2021. What helped, and what did not? An Evaluation of the Strategies to Improve Continuous Integration. In *Proceedings of the 43th International Conference on Software Engineering.* To appear.

[23] John O'Duinn . 2013. The financial cost of a checkin. https://oduinn.com/2013/12/13/the-financial-cost-of-a-checkin-part-2/ [Online; accessed 25-January-2019].

[24] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. 2014. Why do automated builds break? an empirical study. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on.* IEEE, 41–50.

[25] Jingjing Liang, Sebastian Elbaum, and Gregg Rothermel. 2018. Redefining prioritization: continuous prioritization for continuous integration. In *Proceedings of the 40th International Conference on Software Engineering.* 688–698.

[26] Qi Luo, Kevin Moran, Denys Poshyvanyk, and Massimiliano Di Penta. 2018. Assessing test case prioritization on real faults and mutants. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 240–251.

[27] Mateusz Machalica, Alex Samylkin, Meredith Porth, and Satish Chandra. 2019. Predictive test selection. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP).* IEEE, 91–100.

[28] Dusica Marijan, Arnaud Gotlieb, and Sagar Sen. 2013. Test case prioritization for continuous regression testing: An industrial case study. In *2013 IEEE International Conference on Software Maintenance.* IEEE, 540–543.

[29] Atif Memon, Zebao Gao, Bao Nguyen, Sanjeev Dhanda, Eric Nickell, Rob Siemborski, and John Micco. 2017. Taming google-scale continuous testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP).* IEEE, 233–242.

[30] Shaikh Mostafa, Xiaoyin Wang, and Tao Xie. 2017. PerfRanker: prioritization of performance regression tests for collection-intensive software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis.* 23–34.

[31] Ansong Ni and Ming Li. 2017. Cost-effective build outcome prediction using cascaded classifiers. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR).* IEEE, 455–458.

[32] Klérisson VR Paixão, Crícia Z Felício, Fernanda M Delfim, and Marcelo de A Maia. 2017. On the interplay between non-functional requirements and builds on continuous integration. In *Proceedings of the 14th International Conference on Mining Software Repositories.* IEEE Press, 479–482.

[33] Gustavo Pinto, Marcel Rebouças, and Fernando Castor. 2017. Inadequate testing, time pressure, and (over) confidence: a tale of continuous integration users. In *Proceedings of the 10th International Workshop on Cooperative and Human Aspects of Software Engineering.* IEEE Press, 74–77.

[34] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. 2017. An empirical analysis of build failures in the continuous integration workflows of Java-based open-source software. In *Proceedings of the 14th International Conference on Mining Software Repositories.* IEEE Press, 345–355.

[35] Gregg Rothermel and Mary Jean Harrold. 1997. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 6, 2 (1997), 173–210.

[36] Hyunmin Seo, Caitlin Sadowski, Sebastian Elbaum, Edward Aftandilian, and Robert Bowdidge. 2014. Programmers' build errors: a case study (at google). In *Proceedings of the 36th International Conference on Software Engineering.* ACM, 724–734.

[37] August Shi, Suresh Thummalapenta, Shuvendu K Lahiri, Nikolaj Bjorner, and Jacek Czerwonka. 2017. Optimizing test placement for module-level regression testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE).* IEEE, 689–699.

[38] Michele Tufano, Hitesh Sajnani, and Kim Herzig. 2019. Towards Predicting the Impact of Software Changes on Building Activities. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)* (Montral, Canada) *(ICSE '19)*. 4 pages.

[39] Carmine Vassallo, Gerald Schermann, Fiorella Zampetti, Daniele Romano, Philipp Leitner, Andy Zaidman, Massimiliano Di Penta, and Sebastiano Panichella. 2017. A tale of CI build failures: An open source and a financial organization perspective. In *2017 IEEE international conference on software maintenance and evolution (ICSME).* IEEE, 183–193.

[40] David Gray Widder, Michael Hilton, Christian Kästner, and Bogdan Vasilescu. 2019. A conceptual replication of continuous integration pain points in the context of Travis CI. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* ACM, 647–658.

[41] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How open source projects use static code analysis tools in continuous integration pipelines. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR).* IEEE, 334–344.

[42] Chen Zhang, Bihuan Chen, Linlin Chen, Xin Peng, and Wenyun Zhao. 2019. A Large-Scale Empirical Study of Compiler Errors in Continuous Integration. (2019).

[43] Lingming Zhang. 2018. Hybrid regression test selection. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE).* IEEE, 199–209.

[44] Chenguang Zhu, Owolabi Legunsen, August Shi, and Milos Gligoric. 2019. A framework for checking regression test selection tools. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* IEEE, 430–441.