Continuous Software Performance Assessment: Detecting Performance Problems of Software Libraries on Every Build

Christoph Laaber
Department of Informatics
University of Zurich
Zurich, Switzerland
laaber@ifi.uzh.ch

ABSTRACT

Degradation of software performance can become costly for companies and developers, yet it is hardly assessed continuously. A strategy that would allow continuous performance assessment of software libraries is software microbenchmarking, which faces problems such as excessive execution times and unreliable results that hinder wide-spread adoption in continuous integration. In my research, I want to develop techniques that allow including software microbenchmarks into continuous integration by utilizing cloud infrastructure and execution time reduction techniques. These will allow assessing performance on every build and therefore catching performance problems before they are released into the wild.

CCS CONCEPTS

• **Software and its engineering → Software performance**; **Software testing and debugging**.

KEYWORDS

performance testing, continuous integration, cloud

ACM Reference Format:
Christoph Laaber. 2019. Continuous Software Performance Assessment: Detecting Performance Problems of Software Libraries on Every Build. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3293882.3338982

1 PROBLEM STATEMENT

Software performance assessment is a crucial task in software engineering to ensure that a new software release does not impair user-perceived performance. Performance degradation can surface in various forms, such as high response time latency, low throughput, or excessive resource utilization. Such a degradation can have detrimental effects on the revenue and reputation of a company as well as on the necessary time and effort to fix the issue. For example, Google reported that an increase of 500ms response time in its search service costs them about 20% revenue [22]. Similarly, Amazon loses 1% revenue if the latency during checkout grows by 100ms [21]. Compared to functional bugs, performance problems stay undiscovered much longer [15] and are harder to fix [11].

Although these arguments would suggest that performance should be assessed regularly and arguably on every change, recent studies on continuous integration (CI) and software builds did not find any indication that this is the case [3, 12, 26, 31]. Performance testing is a solution to assess performance that would fit into the context of CI, but there are multiple reasons why it is not standard practice yet. This includes, but is not limited to, extensive test execution times; unreliable results; required in-depth knowledge about compiler/runtime internals, measurement biases, and statistics; and lack of adequate tooling [4, 9, 13, 16, 17, 19, 24, 30].

Performance testing comes in two flavors: load testing and software microbenchmarking. Load testing requires a deployed and running system where defined application programming interfaces (APIs) (e.g., REST endpoints) are performance tested, similar to functional system/integration tests. On the other hand, software microbenchmarking is the performance-testing-equivalent of functional unit tests where small software components (e.g., methods) are tested. Software libraries, such as collections or logging, cannot be deployed as a system which is required for load tests, therefore software microbenchmarking is the technique of choice.

2 PROPOSED RESEARCH

I envision that *software performance problems can be detected already at build time by executing software microbenchmark (SMB) in CI*, even in high-code-velocity environments. This would identify performance changes before new software releases and therefore (ideally) not exposing consumers to performance bugs. Among the many challenges outlined above, this mainly demands solutions for two of the problems: (1) long execution times and (2) uncertain results due to unreliable execution environments (e.g., cloud infrastructure).

In my PhD, I want to design solutions for reducing the time spent in performance testing of software libraries by prioritizing/selecting the tests to run on every build and utilize cloud infrastructure for test executions that yield reliable results. These solutions will lower the complexity of rigorous performance test execution with cheap and highly-available resources, utilize valid statistical techniques for change assessment, and prioritize and select the tests to be run. Consequently, the time-to-feedback on the performance is reduced.

In order to assess this thesis' goals, I will investigate and answer the following research questions.

RQ 1 How comprehensive are existing software microbenchmark suites, and do they lend themselves to rigorous software performance evaluation?

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07…$15.00
https://doi.org/10.1145/3293882.3338982

First, I studied [17] the current state of software microbenchmarking, understand whether an inclusion in CI is feasible, and identify problems that hinder adoption. This includes a quantitative study on the extent of SMB suites (i.e., sizes and execution times) and a preliminary slowdown detection study of the individual benchmarks in bare-metal and cloud environments.

RQ 2 How can we run software microbenchmarks in cloud environments and deal with the uncertainty that comes with it?

Cloud instances are cheap, on-demand execution resources which are maintained by a provider (e.g., Amazon Web Services (AWS)) and therefore easy to use by software developers. Performance engineering best practices require performance measurements (in my case: tests) to be executed in controlled environments in order to have as little interference as possible by, e.g., background tasks or co-located tenants. Unfortunately, these requirements are contrary to what cloud instances offer. As hosted CI servers are typically deployed on cloud instances, and the goal is to enable continuous performance testing for everyone; I investigated [18] which cloud providers and instance types deliver acceptable performance test results, which deployment strategies are preferable, and which statistical tests are best for slowdown detection. This addresses the problem of performance test result uncertainty.

RQ 3 Are traditional test case prioritization and regression test selection techniques good at reducing the execution time of software microbenchmark suites, ...

RQ 3a with respect to the importance of API features?

RQ 3b with respect to the quality of their results in certain execution environments?

Running performance tests on every build faces the major obstacle of extensive runtimes [13]. Hence, I want to address this problem by devising novel approaches to reducing the time spent in performance testing activities by focusing on the relevant tests to execute. For the third research question, I want to evaluate time-reduction techniques from functional/unit testing research (i.e., test case prioritization (TCP) and regression test selection (RTS)) and their applicability for SMB. I envision two approaches that extend these traditional techniques to better suit software microbenchmarking of libraries: (1) I want to focus on SMB that test more important functions of the library for API consumers, and (2) I want to favor SMB with better result quality (i.e., stabler results and smaller detectable slowdowns) in a defined execution environment. These two techniques will address the problems of long runtimes and performance test result uncertainty.

RQ 4 How frequently can we execute software microbenchmarks as part of continuous integration, and are we able to capture performance problems faster?

With the insights and techniques from RQ 1, RQ 2, and RQ 3, I will have tackled the two outlined problems, i.e., long execution times and uncertain test results, but one still misses a quantification of how frequently we can run SMB (as part of CI), and therefore whether one is able to detect performance problems faster and before a new software version release. Consequently, I plan to combine the techniques from RQ 1, RQ 2, and RQ 3to answer the forth research question.

3 RQ 1: CI-INCLUSION FEASIBILITY

As a first step in [17], we conducted an initial investigation whether it would be possible to include SMBs into CI and devised a methodology developers can follow to assess their own SMB suite for CI-readiness. This study looked at three aspects of SMBs and outlined the properties which are challenging for CI inclusion. These three aspects are: (1) the extent of existing SMB suites in form of number of individual benchmarks and total runtime, (2) SMB result stability across multiple identical executions in bare-metal and cloud environments, and (3) whether SMB suites are able to detect actual slowdowns. To assess the three aspects, we executed the full SMB suites of 10 open-source software (OSS) projects written in two languages (i.e., Java and Go) multiple times on a bare-metal server and a cloud instance from Google Compute Engine (GCE).

Based on these executions we assessed the mean runtime across multiple executions and the number of executed benchmarks, which took between 12 minutes and 8 hours and 45 minutes and ranged from 16 to 983 individual benchmarks. These runtimes confirm that *some* suites run for too long to be executed on every build while others might be small enough to be run in its entirety.

We further studied the variability of SMB suites between identical executions in a particular environment, which gives an indication whether a benchmark could be used for slowdown detection. The results confirm the intuition that bare-metal environments produce more stable results than cloud environments. Nonetheless, they also indicate that running performance test in the cloud might be feasible as most benchmarks show an increased but acceptable variability compared to bare-metal environments.

Based on the variability insights, we finally studied whether benchmark suites are able to find injected slowdowns in often-used parts of its project's API. Depending on the study subject, SMB suites are able to detect slowdowns when executed in cloud environments. The limiting factor is the size of the slowdown to be detected, which ranges for the studied subjects from 10% to 260% relative slowdown size.

These results suggest that executing performance tests in cloud environments is feasible if the size of the slowdown is large. The biggest issue is the result variability originating from the execution on cloud instances. Regarding inclusion in CI, the bigger challenge is the excessive test suite runtimes.

4 RQ 2: SOFTWARE MICROBENCHMARKING IN THE CLOUD

In [18], we explored SMB executions in cloud environments on a broader spectrum, studied benchmark variability, and investigated techniques that work best for slowdown detection.

We selected 20 benchmarks from two Java projects (log4j2, RxJava) and two Go projects (bleve, etcd), where each project contributed five benchmarks ranging from very stable to very unstable, based on the variability results from Section 3. As execution environments, we selected 3 cloud providers (i.e., AWS, Microsoft Azure, and GCE) with 3 instance types each and a hosted bare-metal environment from IBM Bluemix. We executed the benchmarks repeatedly on multiple levels: (1) within the same execution, (2) within the same instance, and (3) across multiple instances of the same type.

In contrast to the previous study from Section 3, we based our variability analysis and the slowdown detection on standard statistical measures and techniques. We measured the benchmark variability with the coefficient of variation (CV). For slowdown detection we relied on Wilcoxon rank-sum with effect sizes and bootstrapped confidence intervals of the mean, which are both state-of-the-art techniques for performance analysis. With the increased numbers of samples across multiple levels of repetition, we found that result variability of the studied benchmark environment combinations varies drastically from 0.03% to 100.68%. This finding is in-line with the previous results. Interestingly though, AWS appears to be much stabler than Azure and GCE, and even on-par with the bare-metal environment. With regards to the detectable slowdowns, when executing test and control group on the same instances, slowdowns below 10% are detectable most of the time. Testing for performance changes with Wilcoxon rank-sum (with effect sizes) is superior compared to confidence intervals when dealing with results from unreliable environments.

These findings show that relatively small slowdowns are detectable in cloud environments when test and control group are executed on the same cloud instance and repeated multiple times within and across cloud instances. With the goal of including SMB executions into CI, these results support the thesis statement and empirically validate that utilizing cloud environments is indeed feasible.

5 RQ 3: REDUCING BENCHMARK EXECUTION TIME

SMBs are different from unit tests, e.g., they test the "usual" path of a program rather then the exceptional, are highly parameterized which results in multiple SMBs having the same call path, and have a result distribution compared to a binary outcome. Research has focused on whether a commit should be performance tested at all [13, 28], or it employed performance impact prediction as a driver for prioritization and selection [5, 23]. It is unclear though how traditional TCP/RTS techniques perform for SMBs and whether performance impact is the *only* important goal to optimize for.

In this part, I will answer **RQ 3** and address the problems of excessive runtimes and unreliable results, by empirically studying the applicability of TCP/RTS techniques for SMBs and devising novel approaches to prioritize/select SMBs that test the most important features first and have better result quality.

5.1 RQ 3a: Importance-Based Benchmark Prioritization

Importance-based benchmark prioritization considers the "importance" of a benchmark for its rank and consequently whether it gets selected for execution. A benchmark has higher importance if it directly or indirectly calls API features that are more often used by *other* programs. The benchmarks that are prioritized to be run after a software change (e.g., after a new commit) depend on two main factors: (1) whether a benchmark is *affected* by a source code change and (2) the *importance* of the functions the benchmark calls.

A benchmark is *affected* if it directly or indirectly calls a part of the software that has been changed. Secondly, the *importance* of a benchmark can be defined in various ways. The above mentioned papers consider the predicted performance change impact as the importance criteria. I propose a different strategy that assumes importance of an API method is defined by the amount of API consumers (*other* projects) using the software being developed (and which is performance tested), i.e., a slowdown introduced in a method that is most-used has higher severity because it potentially slows down more consumers than a method which is hardly used. For this, I will extract fine-grained API usages of the benchmark projects employing the approach by Sawant and Bacchelli [29] and assign weights to benchmarks based on the importance of the called methods. Bringing the two aspects together, the importance-based benchmark prioritization ranks the most important benchmarks, defined by whether they are affected by a change and their impact on API consumer software.

5.2 RQ 3b: Result-Quality-Aware Benchmark Prioritization

With the same goal in mind as above — reducing the time spent in performance testing activities — I want to shift the focus in this part to SMB result quality as a driver for prioritization. The result quality is defined as the variability of a benchmark executed in a particular environment (as reported by the paper from Section 4). SMB results with low variability lend themselves better to detect performance changes than highly variable results. Hence, I plan on devising a technique that incorporates benchmark properties that relate to *higher result quality* into the prioritization of benchmarks, i.e., favoring benchmarks that are better at finding performance changes for a particular software change and executed in a defined environment (e.g., cloud).

I plan on investigating the following properties that could drive the prioritization for ranking benchmarks of *higher quality* with respect to performance change detection capabilities: (1) variability of benchmarks in a particular environment (e.g., coefficient of variation or confidence interval widths); (2) historical performance detection rate and sizes of benchmarks; (3) source code properties and changes that are related to unreliable benchmark results (e.g., IO operations and concurrency); (4) predicted performance profile based on system level benchmarks [33]. Because of the complex parameter space, I plan to define the benchmark prioritization problem as a multi-objective search-based problem that incorporates these parameters.

5.3 RQ 3: Evaluation

The evaluation of both prioritization approaches will use prototyping and case study evaluations based on multiple OSS projects, where I will compare the importance-based and result-quality-aware prioritizations to a number of baseline approaches, i.e., traditional TCP techniques. The planned evaluation will take the following steps: (1) Select a set of OSS projects available on GitHub with long-running SMB suites. (2) Execute test suites for multiple adjacent commits/version and detect performance change sizes (based on bootstrapped confidence intervals of the mean ratios) in a bare-metal and cloud (only for RQ 3b) environment . These provide an ordering of importance based on slowdown sizes between all version pairs. (3) Prioritize/select benchmarks with the two approaches. (4) Compare this research's prioritization approaches to traditional

TCP techniques (e.g., total and additional approaches) with standard metrics such as APFD-P and nDCG [23] as well as adaptions of these that assess the rankings with respect to the importance (RQ 3a) and result quality (RQ 3b).

6 RQ 4: HIGH-VELOCITY SOFTWARE MICROBENCHMARKING IN CONTINUOUS INTEGRATION

In the final part of my PhD, I want to conduct a holistic evaluation of the approaches and techniques introduced for RQ 1, RQ 2, and RQ 3. The goal of this evaluation and therefore the answer to **RQ 4** is to examine whether my techniques enable CI integration of SMBs with software changes arriving at high velocity. In particular, the evaluation should quantify the frequency with which we can execute SMBs and the gain of detecting performance problems on every commit rather than on every new software release. For this, I want to build a CI plug-in which enables continuous performance assessment, deployment of tests in different environments (i.e., bare-metal, virtualized, and cloud), and prioritization/selection of tests to be run on every commit.

I will conduct a large-scale quantitative empirical evaluation of the simulated histories (e.g., a year) of multiple case study subjects to show the benefits of including SMBs in CI. Moreover, I will perform a sensitivity analysis on the available time budgets for performance testing.

7 RELATED WORK

Work related to this research evolves around performance testing, performance measurements, and test prioritization and selection.

Performance Testing: Performance testing has traditionally received research attention in form of system testing such as load and stress testing [14, 34], with more recent research focusing on industrial applicability [8] and reducing the execution time [1, 10]. Academic studies on software microbenchmarking, the unit test equivalent for performance testing, have not received as much attention as studies on load testing, though Stefan et al. [30] and Leitner and Bezemer [19] quantitatively and qualitatively studied microbenchmarking practices in Java OSS recently. Stefan et al. [30] considered projects with benchmark suite runtimes of below 4 hour in their subject selection, which they justify because longer runtimes are not feasible for continuous testing. As the results from Section 3 show, there are projects with longer runtimes that require solutions for continuous performance assessment.

Performance Measurements: Measuring software performance correctly, applying the necessary statistical analyses, and dealing with measurement bias has been studied extensively. This is a hard task to get right and much can be done wrong, especially if unreliable environments such as cloud instances are utilized. Mytkowicz et al. [24] study different measurement biases and show that many researchers have drawn wrong conclusions because of that. Georges et al. [9] and Kalibera and Jones [16] outline approaches and statistical techniques to rigorously assess the performance of software systems. Nevertheless, all of these studies expect an as-stable-as-possible environment to run performance experiments on. More recently, Arif et al. [2] studied the effect of virtual environments on load tests, showing a discrepancy between physical and virtual environments, and Wang et al. [33] proposed an approach to assess whether running performance tests in the cloud meets certain goals.

So far and to the best of my knowledge, no one has studied existing SMBs and their results when executed in cloud environments. In Section 4 and to a lesser degree in Section 3, we investigated result variability, statistical techniques that lend themselves best to slowdown detection, and the magnitude of detectable slowdowns.

Test Prioritization and Selection: Applying test case prioritization (TCP) and regression test selection (RTS) is a way to reduce the time spent in performance testing activities and consequently bringing it to CI. TCP and RTS techniques for unit testing has been widely researched since the turn of the 21st century [6, 27]. Especially in contexts where high code velocity is key, approaches that are time-aware [32, 35] and consider CI [7, 20] have received increased attention in recent years. The approaches outlined in Sections 5 and 6 can particularly draw inspiration from these.

Performance test regression selection and prioritization research so far explored testing only performance-critical commits [13], or focused on particular types such as collection-intensive software [23] and concurrent classes [25]. De Oliveira et al. [5] propose selection of individual benchmarks based on static and dynamic data that assess whether a code change affects the performance of each benchmark. In my research, I want to explore other properties that are essential in performance test prioritization and selection such as importance and and result quality.

8 CONCLUSIONS AND RESEARCH PROGRESS

The goal of my PhD is to devise techniques and gather empirical evidence that support bringing software microbenchmarking to CI. Especially, I want to tackle the problems of excessive benchmark execution times and unreliable results. This, therefore, enables detecting performance problems for every new software version (i.e., commit) already at build time.

The contributions of my research are: (1) An approach to assess the quality of an existing SMB suite and their feasibility for CI inclusion in a certain environment, which is published at the 15th International Conference on Mining Software Repositories [17]. (2) Empirical evidence on result variability, false-alarm-rate, and detectable slowdowns of SMB executions in different cloud and bare-metal environments, which has been published in Empirical Software Engineering [18]. (3) Two prioritization/selection approaches with proof-of-concept implementations to reduce performance test execution time. (4) Empirical evidence about the inclusion of real-world SMB suites in CI. Particular focus is put on the temporal frequency, i.e., how often we can execute performance tests in high-code-velocity environments, and what we gain in terms of earlier detection of performance degradation.

ACKNOWLEDGMENTS

I express my gratitude to Philipp Leitner and Harald Gall for supervision and support. This research received funding from the Swiss National Science Foundation (SNF) under project MINCA – Models to Increase the Cost Awareness of Cloud Developers (no. 165546). Conference attendance is partially supported by CHOOSE (Swiss Group for Software Engineering).

REFERENCES

[1] H. M. Alghmadi, M. D. Syer, W. Shang, and A. E. Hassan. 2016. An Automated Approach for Recommending When to Stop Performance Tests. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 279–289. https://doi.org/10.1109/ICSME.2016.46

[2] Muhammad Moiz Arif, Weiyi Shang, and Emad Shihab. 2017. Empirical study on the discrepancy between performance testing results from virtual and physical environments. *Empirical Software Engineering* (03 Oct 2017). https://doi.org/10.1007/s10664-017-9553-x

[3] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR '17)*. IEEE Press, Piscataway, NJ, USA, 356–367. https://doi.org/10.1109/MSR.2017.62

[4] Stephen M. Blackburn, Amer Diwan, Matthias Hauswirth, Peter F. Sweeney, José Nelson Amaral, Tim Brecht, Lubomír Bulej, Cliff Click, Lieven Eeckhout, Sebastian Fischmeister, Daniel Frampton, Laurie J. Hendren, Michael Hind, Antony L. Hosking, Richard E. Jones, Tomas Kalibera, Nathan Keynes, Nathaniel Nystrom, and Andreas Zeller. 2016. The Truth, The Whole Truth, and Nothing But the Truth: A Pragmatic Guide to Assessing Empirical Evaluations. *ACM Trans. Program. Lang. Syst.* 38, 4, Article 15 (Oct. 2016), 20 pages. https://doi.org/10.1145/2983574

[5] Augusto Born De Oliveira, Sebastian Fischmeister, Amer Diwan, Matthias Hauswirth, and Peter F Sweeney. 2017. Perphecy: Performance Regression Test Selection Made Simple but Effective. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 103–113. https://doi.org/10.1109/ICST.2017.17

[6] Sebastian Elbaum, Alexey G. Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: a family of empirical studies. *IEEE Transactions on Software Engineering* 28, 2 (Feb 2002), 159–182. https://doi.org/10.1109/32.988497

[7] 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 (FSE 2014)*. ACM, New York, NY, USA, 235–245. https://doi.org/10.1145/2635868.2635910

[8] King Chun Foo, Zhen Ming (Jack) Jiang, Bram Adams, Ahmed E. Hassan, Ying Zou, and Parminder Flora. 2015. An Industrial Case Study on the Automated Detection of Performance Regressions in Heterogeneous Environments. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 159–168. http://dl.acm.org/citation.cfm?id=2819009.2819034

[9] Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications (OOPSLA '07)*. ACM, New York, NY, USA, 57–76. https://doi.org/10.1145/1297027.1297033

[10] Mark Grechanik, Chen Fu, and Qing Xie. 2012. Automatically Finding Performance Problems with Feedback-Directed Learning Software Testing. In *Proceedings of the 34th International Conference on Software Engineering (ICSE '12)*. IEEE Press, Piscataway, NJ, USA, 156–166. http://dl.acm.org/citation.cfm?id=2337223.2337242

[11] Christoph Heger, Jens Happe, and Roozbeh Farahbod. 2013. Automated Root Cause Isolation of Performance Regressions During Software Development. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13)*. ACM, New York, NY, USA, 27–38. https://doi.org/10.1145/2479871.2479879

[12] 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 (ASE 2016)*. ACM, New York, NY, USA, 426–437. https://doi.org/10.1145/2970276.2970358

[13] Peng Huang, Xiao Ma, Dongcai Shen, and Yuanyuan Zhou. 2014. Performance Regression Testing Target Prioritization via Performance Risk Analysis. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 60–71. https://doi.org/10.1145/2568225.2568232

[14] Z. M. Jiang and A. E. Hassan. 2015. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Transactions on Software Engineering* 41, 11 (Nov 2015), 1091–1118. https://doi.org/10.1109/TSE.2015.2445340

[15] Guoliang Jin, Linhai Song, Xiaoming Shi, Joel Scherpelz, and Shan Lu. 2012. Understanding and Detecting Real-world Performance Bugs. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '12)*. ACM, New York, NY, USA, 77–88. https://doi.org/10.1145/2254064.2254075

[16] Tomas Kalibera and Richard Jones. 2012. *Quantifying Performance Changes with Effect Size Confidence Intervals*. Technical Report 4–12. University of Kent. 55 pages. http://www.cs.kent.ac.uk/pubs/2012/3233

[17] Christoph Laaber and Philipp Leitner. 2018. An Evaluation of Open-source Software Microbenchmark Suites for Continuous Performance Assessment. In *Proceedings of the 15th International Conference on Mining Software Repositories (MSR '18)*. ACM, New York, NY, USA, 119–130. https://doi.org/10.1145/3196398.3196407

[18] Christoph Laaber, Joel Scheuner, and Philipp Leitner. 2019. Software microbenchmarking in the cloud. How bad is it really? *Empirical Software Engineering* (17 Apr 2019), 40. https://doi.org/10.1007/s10664-019-09681-1

[19] Philipp Leitner and Cor-Paul Bezemer. 2017. An Exploratory Study of the State of Practice of Performance Testing in Java-Based Open Source Projects. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE '17)*. ACM, New York, NY, USA, 373–384. https://doi.org/10.1145/3030207.3030213

[20] 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 (ICSE '18)*. ACM, New York, NY, USA, 688–698. https://doi.org/10.1145/3180155.3180213

[21] Greg Linden. 2006. Make Data Useful. http://glinden.blogspot.com/2006/12/slides-from-my-talk-at-stanford.html. Stanford - Data Mining CS345.

[22] Marissa Meyer. 2009. In Search of... A better, faster, stronger Web. http://assets.en.oreilly.com/1/event/29/Keynote%20Presentation%202.pdf. Velocity'09.

[23] 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 (ISSTA 2017)*. ACM, New York, NY, USA, 23–34. https://doi.org/10.1145/3092703.3092725

[24] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. 2009. Producing Wrong Data Without Doing Anything Obviously Wrong!. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XIV)*. ACM, New York, NY, USA, 265–276. https://doi.org/10.1145/1508244.1508275

[25] Michael Pradel, Markus Huggler, and Thomas R. Gross. 2014. Performance Regression Testing of Concurrent Classes. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA '14)*. ACM, New York, NY, USA, 13–25. https://doi.org/10.1145/2610384.2610393

[26] 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 (MSR '17)*. IEEE Press, Piscataway, NJ, USA, 345–355. https://doi.org/10.1109/MSR.2017.54

[27] Gregg Rothermel and Mary Jean Harrold. 1998. Empirical studies of a safe regression test selection technique. *IEEE Transactions on Software Engineering* 24, 6 (June 1998), 401–419. https://doi.org/10.1109/32.689399

[28] Juan Pablo Sandoval Alcocer, Alexandre Bergel, and Marco Tulio Valente. 2016. Learning from Source Code History to Identify Performance Failures. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE '16)*. ACM, New York, NY, USA, 37–48. https://doi.org/10.1145/2851553.2851571

[29] Anand Ashok Sawant and Alberto Bacchelli. 2017. Fine-GRAPE: Fine-Grained APi Usage Extractor – an Approach and Dataset to Investigate API Usage. *Empirical Software Engineering* 22, 3 (01 Jun 2017), 1348–1371. https://doi.org/10.1007/s10664-016-9444-6

[30] Petr Stefan, Vojtech Horky, Lubomir Bulej, and Petr Tuma. 2017. Unit Testing Performance in Java Projects: Are We There Yet?. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering (ICPE '17)*. ACM, New York, NY, USA, 401–412. https://doi.org/10.1145/3030207.3030226

[31] 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)*. 183–193. https://doi.org/10.1109/ICSME.2017.67

[32] Kristen R. Walcott, Mary Lou Soffa, Gregory M. Kapfhammer, and Robert S. Roos. 2006. Time-Aware Test Suite Prioritization. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis (ISSTA '06)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/1146238.1146240

[33] Wei Wang, Ningjing Tian, Sunzhou Huang, Sen He, Abhijeet Srivastava, Mary Lou Soffa, and Lori Pollock. 2018. Testing Cloud Applications under Cloud-Uncertainty Performance Effects. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 81–92. https://doi.org/10.1109/ICST.2018.00018

[34] Elaine J. Weyuker and Filippos I. Vokolos. 2000. Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study. *IEEE Transactions on Software Engineering* 26, 12 (Dec. 2000), 1147–1156. https://doi.org/10.1109/32.888628

[35] Lu Zhang, Shan-Shan Hou, Chao Guo, Tao Xie, and Hong Mei. 2009. Time-aware Test-case Prioritization Using Integer Linear Programming. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis (ISSTA '09)*. ACM, New York, NY, USA, 213–224. https://doi.org/10.1145/1572272.1572297