Enhancing Performance Testing with N-Version Testing: An Industrial Experience Report

Críssia Marcelino
Projeto Samsung
Brazil
csm4@fadeorg.onmicrosoft.com

Breno Miranda
Federal University of Pernambuco
Recife, Brazil
bafm@cin.ufpe.br

Abstract

Performance regression is a critical challenge for the software industry, particularly in Continuous Integration (CI) environments, where frequent changes may degrade response time, resource consumption, and user experience. However, automating the detection of such regressions remains limited by the Oracle Problem, i.e., the difficulty of defining objective criteria to validate performance testing results in systems whose performance requirements are not formally specified. This work proposes an approximate oracle for performance testing based on an adaptation of the N-version testing (NVT) technique. Unlike traditional functional redundancy, the proposed approach uses successful historical versions of the same system as a temporal reference to establish a performance baseline and an adaptive tolerance margin. The solution incorporates a sliding window mechanism to continuously update historical data and exclude versions with confirmed regressions, reducing bias and false negatives. The approach was implemented and evaluated in a proprietary industrial image-editing software system, where it successfully detected performance regressions previously confirmed by the development team. The results indicate that the technique reduces subjectivity in performance analysis, increases the diagnostic value of automated testing, and supports performance regression testing throughout software evolution. In addition to the experimental validation, the approach was assessed by industrial software test engineers using the System Usability Scale (SUS), achieving a score of 86.9, which indicates high acceptance, ease of use, and practical applicability in real-world software engineering environments.

Keywords

Performance, Test Oracle, N-version Testing

1 Introduction

Industrial software requires continuous improvements to remain useful and attractive to users. However, constant updates can interfere with software performance. Recent studies warn that performance-related failures currently outnumber functional bugs [3]. To minimize these impacts, regular testing is essential to detect errors early. However, in many industrial projects, running test cycles for each new implementation is not feasible.

As an alternative, automated tests are run to ensure that the main functionalities are covered. However, in many cases, the output data from automated tests pose challenges to confirm the success or failure of the System Under Test (SUT) Barr et al. [1]. In performance testing, the expected outputs can classify the results incorrectly, since attributes such as CPU, memory processing and RAM can interfere with the results.

A practice that helps in correctly defining the expected outputs is the use of test oracles. A test oracle is a document or software that allows testers to assess whether the execution of a given test case passed or failed, i.e., whether the output produced corresponds to the expected output (defined in the oracle) [2]. Therefore, a fundamental challenge in applying test oracles in automated testing is to ensure that the output complies with expectations.

If the creation/automation of test oracles is already a challenge for functional testing, it is logical that for non-functional aspects, such as performance, the output definitions are even more complex. Therefore, one of the main challenges in modern software projects with continuous integration is to ensure that the quality of performance is preserved, as well as to establish mechanisms to detect output deviations caused by performance regression. A common practice in such projects is to rely on testers to certify the software, using the test results to build evidence that legitimizes the performance of the system.

Given this scenario, it is essential to adopt an approach that allows greater objectivity in the execution of tests, reducing personnel effort and improving the accuracy of the results.

We recently introduced the concept of using n-version testing to define approximate performance oracles, as presented in [6]. In that work, we focused on outlining the methodology and illustrating the proposed approach through a series of examples. Although Section 3 provides a brief overview of the approach previously introduced to maintain coherence within this paper, the focus of this work is different. Our primary aim here is to present the practitioner's perspective and share the feedback we received from the industrial context in which our approach was adopted. Additionally, we discuss the lessons we learned after evaluating our proposed approach in a real-world industrial setting.

The remainder of this paper is structured as follows: Section 2 presents the context and motivation behind the study. In Section 3, we provide an overview of the methodology introduced in [6]. Section 4 showcases an example of the results obtained by applying our approach with our industrial partner. Section 5 reveals the perception of testing professionals regarding the approach. Section 6 discusses the lessons learned from this application, and Section 7 concludes the work.

2 Context and Motivation

In our company, we perform tests that aim to ensure the quality of the software when the product is launched, as well as in its subsequent updates. The system has several features related to image editing. With 1,364 test cases (manual and automated), only 60 are related to performance. However, in the context of the project,

performance tests are only prioritized when there are significant and direct changes in the performance of some functionality.

The Problem When a functionality is created/updated, the functional tests related to this functionality are performed based on the functional specification that defines the expected outputs. The expected outputs for the performance tests are at the discretion of the testers. It is their feeling/experience that defines whether the functionality's performance is adequate. This "informality" leads to inconsistency among the team itself, which has different points of view. To illustrate, we identified in the project's bug management tool the record of a response time error, where a feature took around 13 seconds to open. According to the tester, the time in the previous version was 5 seconds. However, there was no documented reference for the correct response time. In this particular case, the tester subjectively defined 5.6 seconds as the maximum acceptable response time, without providing any clear rationale for this threshold.

The Solution To minimize the possibility of inconsistencies in identifying performance errors, we suggest that the company should adopt performance test oracle, with an approximate definition of the expected results according to the historical data collected through n-version testing.

Our Approach We propose using performance test oracles as a reference point for evaluating test runs. Specifically, our approach involves comparing the performance results of the current version with those from previous versions to identify regressions or improvements. To perform our experiments, we defined a process with the following steps:

- Since software releases are generated daily, it is essential to maintain a repository to track the performance of features across versions.
- When a new version is released, an approximate performance oracle is derived from the historical performance data of the corresponding feature. This allows testers to determine whether the performance of the current version deviates from previously observed behavior. The details of our approach are presented in the following sections.

3 Methodology

In this section, we provide a brief overview of the approach introduced in [6] to ensure the paper remains self-contained.

Performance metrics can be measured in different ways, even for the same metric, and the measurement method directly influences how the data is described and analyzed. In the literature [4, 5, 7], two primary approaches to measuring performance metrics are identified: Point-in-Time Measurement and Time-Series Measurement.

- **Point-in-Time Measurement:** Captures the value of a metric at a specific moment in time, typically used to assess the system's state after a specific action or event.
  Example: "Memory usage was 1.3 GB right after the file was loaded."
- **Time-Series Measurement:** Involves collecting a sequence of metric values at regular or irregular intervals over time, enabling the analysis of trends, patterns, and dynamic behavior.
  Example: "CPU usage was monitored every second for 5 seconds during the test, resulting in a time-series plot."

Regardless of the measurement type (point-in-time or time-series), our approach, leveraging **n-version testing** to define **approximate performance oracles**, follows the same four-step process. An overview of the proposed approach is shown in Figure 1.

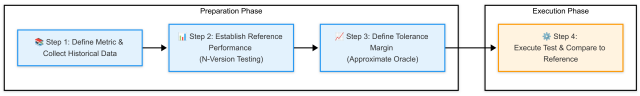


Figure 1: Four-step approach for defining approximate performance oracles based on N-Version testing

Step 1 – Preparation: Define Metric and Collect Historical Data The performance metric of interest is specified, and historical data is collected across n prior versions of the SUT. The performance expert selects how many versions to include (e.g., the last 10 or 20).

Step 2 – Preparation: Establish Reference Performance Using the collected historical data, a **Reference Performance** baseline is computed.

For point-in-time metrics, this could be a single value (e.g., average or median) or a range (e.g., minimum and maximum values).

For time-series metrics, a curve can be derived representing the average or median value at each time point across the n versions; minimum and maximum curves can also be computed to represent the range of behavior at each time step.

This step incorporates principles from N-version Testing, treating each prior version as an independent source of performance behavior.

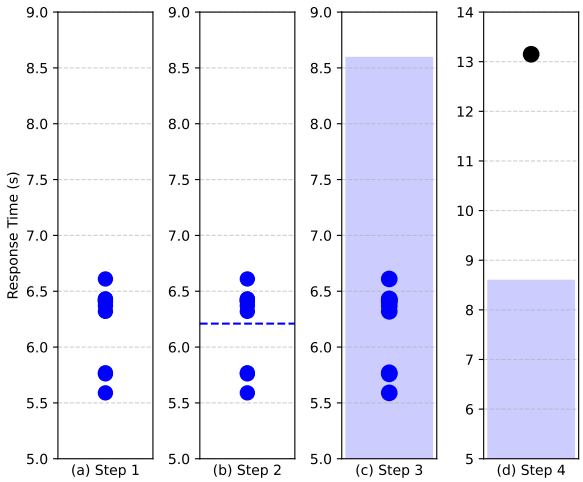
Step 3 – Preparation: Define Tolerance Margin A **Tolerance Margin** is applied to the reference performance to form an approximate oracle. This margin can be defined using standard deviation, fixed thresholds, or percentages over the baseline (mean, median, min, or max). The margin specifies acceptable deviation before signaling a performance regression.

Step 4 – Execution: Run Test and Compare Results The test case is executed on the current version of the SUT using the same measurement method applied to previous versions. The resulting metric is then compared against the reference performance and its tolerance margin:

- If the observed value falls **within** the margin, the system is assumed to behave as expected.
- If the value is **outside** the margin, an alert is raised, indicating a potential performance regression requiring further investigation.

4 Results

Figure 2 illustrates the performance regression identified in one of our studie. The four-panel structure systematically progresses from raw data collection (a) to statistical benchmarking (b-c) and



Applying the four-step approach to point-in-time Response Time in a real application with known performance bug. The dashed blue line represents the average, and the shaded purple area indicates the tolerance margin.

Figure 2: Identification of a real performance bug using approximate performance oracles in an industrial context

anomaly detection (d), effectively operationalizing the concept of metrics-based quality control.

Figure 2(a) provides a visual representation of the historical distribution of response times, enabling the identification of variability and performance trends. This step is crucial for establishing a baseline, a reference performance pattern derived from empirical data.

Figure 2(b) presents the calculation of the overall mean response time. This value serves as a global reference metric, representing the typical behavior of the system throughout the evaluated versions.

Figure 2(c) introduces a tolerance margin based on input from the performance expert. In this case, a 30% tolerance is applied to the maximum value observed in previous versions (6.61). By visually emphasizing this tolerance margin, the figure illustrates an acceptable variation range in response times, reflecting the natural capability of the development process. This approach prevents false alarms due to minor fluctuations and focuses attention on significant deviations, thus promoting a more robust performance verification process.

Figure 2(d) illustrates the introduction of a new performance measurement (V11), directly compared against the previously established tolerance range. The V11 value lies outside the tolerance margin, indicating a possible performance regression.

Figure 2 clearly demonstrates how the definition of approximate oracles can support technical decision-making in software evolution. Using performance benchmarks and tolerance margins as quantitative criteria not only facilitates automated performance testing, but also strengthens governance over software quality.

This study supports the use of tolerance margins for comparative assessments. A well-defined tolerance margin serves as an effective gauge for: (1) prioritizing issues, (2) evaluating inconsistency severity, (3) enabling preemptive resolutions, and (4) reducing unplanned corrective costs.

5 Usability Evaluation

In addition to the experimental validation of the proposed approach, a usability evaluation was conducted with software test engineers using the System Usability Scale (SUS) to assess practitioners' perceptions regarding ease of use, understandability, and the potential adoption of the solution in industrial environments. The evaluation involved professionals experienced in software testing processes, after presenting the approach and its results within the context of the evaluated industrial software system. The obtained results indicated an average SUS score of 86.9, classifying the approach as having high acceptability and excellent usability perception. The SUS scores for each participant are shown in the right mostcolumn of the Table 1. Since the questionnaire was anonymous, each participant is designated as "P" + a number.

Table 1: Demographic information of the participants and System Usability Scores based on questionnaire responses

Part.	Gender	Age	SW Exp. (yrs)	Perf. Exp. (yrs)	Score
P1	Female	33	13	10	92.5
P2	Female	44	20	3	97.5
P3	Female	47	18	5	82.5
P4	Male	40	12	2	72.5
P5	Male	24	3	2	85.0
P6	Female	36	5	3	75.0
P7	Female	34	13	5	100.0
P8	Female	49	20	5	97.5
P9	Female	31	5	3	80.0
				Average System Usability Score:	**86.9**

Participants positively highlighted the objectivity provided by the automatic definition of the tolerance margin, the reduction of subjectivity in performance regression analysis, and the possibility of integrating the technique into real testing workflows. These findings reinforce not only the technical feasibility of the proposal, but also its practical adoption potential in the software industry, particularly in scenarios characterized by the absence of formal performance requirements and the continuous evolution of software systems.

Applicability and Limitations. The proposed approach is particularly suitable for software systems that maintain historical performance data across multiple releases and execute comparable regression tests over time. In contrast, its applicability may be limited in projects with highly unstable execution environments, insufficient historical data, or major architectural changes that significantly alter performance behavior. Furthermore, the tolerance margin should not be interpreted as a definitive indicator of defects. Instead, values outside the acceptable range should be treated as indicators for further investigation, supporting testers rather than replacing engineering judgment.

6 Discussion and Lessons Learned

- **Lesson #1**: Thanks to our study, it became clear that subjective perceptions of performance tests were causing false positives in relation to expected results and delaying the identification of critical performance errors.
- **Lesson #2**: Defining a performance test oracle requires constant calibration to maintain the tolerance margin.
- **Lesson #3**: After listing the appropriate metrics to define the tolerance margin, the maintenance of the oracle is minimal compared to the efficiency and assertiveness it proposes.
- **Lesson #4**: The results of our experiments provided useful information to the test managers of the project where we conducted our studies, as they positively influence strategic definitions.

With these lessons in mind, it became evident that the adoption of the proposed approach could serve as the main artifact to guide performance testing, improving the company's testing strategy. Integrating this process into the testing cycles would be essential to optimize the overall testing workflow and achieve more assertive results. Beyond these lessons, we observed practical trade-offs during adoption. The approach requires an initial effort to organize historical performance data and define appropriate metrics and tolerance margins. However, once this baseline is established, the maintenance effort becomes low compared with the benefits of obtaining more objective and consistent performance assessments. Based on our experience, organizations intending to adopt this approach should first standardize the execution of performance regression tests and ensure that historical measurements are collected under comparable conditions.

7 Conclusion

The results obtained in our study raise a discussion about the importance and effectiveness of test oracles in detecting performance problems, especially in companies that do not have defined metrics for non-functional requirements. The oracle is a tool that helps the tester in analyzing scenarios, serving as a guide, with technical basis and not with subjective assumptions that depend on the expertise/seniority of the tester. We emphasize that not all results outside the tolerance range are errors, but it signals that such inconsistencies justify a detailed analysis before potential defects are identified too late. Finally, based on the SUS results, it is clear that applying the approach in an industrial context has brought a promising and viable outlook, as experts were in favor of the proposal. Although additional evaluations in different industrial domains are still necessary, the reported experience provides initial evidence that historical-performance-based approximate oracles constitute a practical alternative for supporting performance regression testing in industrial software projects.

Artifact Availability

We provide the artifacts used in our work at: https://github.com/csm33/Artifact-Availability. Due to a non-disclosure agreement, we excluded all information related to the proprietary application used in our evaluation.

Acknowledgments

This work was supported by Samsung Eletronica da Amazônia Ltd., through the Collaborative Project between SiDi and the Centre of Informatics, Federal University of Pernambuco (CIn/UFPE), through the auspices of the Brazilian Federal Law of Informatics, under Grant 8248/91.

References

[1] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[2] Ilene Burnstein. 2006. *Practical software testing: a process-oriented approach* (1st ed.). Springer Science Business Media.

[3] Jinfu Chen and Weiyi Shang. 2017. An Exploratory Study of Performance Regression Introducing Code Changes. In *2017 IEEE International Conference on Software Maintenance and Evolution*. 341–352. https://doi.org/10.1109/ICSME.2017.13

[4] Henrik Ingo and David Daly. 2020. Automated system performance testing at MongoDB. In *Proceedings of the workshop on Testing Database Systems*. 1–6.

[5] Mark Leznik, Md Shahriar Iqbal, Igor Trubin, Arne Lochner, Pooyan Jamshidi, and André Bauer. 2022. Change point detection for MongoDB time series performance regression. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*. 45–48.

[6] Críssia Marcelino and Breno Miranda. 2025. Leveraging N-Version Testing to Define Approximate Oracles for Performance Testing. In *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software* (Recife/PE). SBC, Porto Alegre, RS, Brasil.

[7] Luca Traini, Federico Di Menna, and Vittorio Cortellessa. 2024. AI-driven Java Performance Testing: Balancing Result Quality with Testing Time. In *Proceedings of the 39th IEEE/ACM Inter. Conference on Automated Software Engineering*. 443–454.