UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Críssia de Santana Marcelino

DEFINIÇÃO DE ORÁCULOS APROXIMADOS PARA TESTES DE DESEMPENHO COM BASE EM DADOS HISTÓRICOS

Recife

2026

Críssia de Santana Marcelino

DEFINIÇÃO DE ORÁCULOS APROXIMADOS PARA TESTES DE DESEMPENHO COM BASE EM DADOS HISTÓRICOS

Trabalho apresentado ao Programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Engenharia de Software e Linguagens de Programação

Orientador (a): Breno Alexandro Ferreira de Miranda

Recife

2026

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Marcelino, Crissia de Santana.
Definição de Oráculos Aproximados para Testes de Desempenho com Base em Dados Históricos / Crissia de Santana Marcelino. - Recife, 2026.
87f.: il.

Dissertação (Mestrado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2026.
Orientação: Breno Alexandro Ferreira de Miranda.
Inclui Referências.

1. Testes de Performance; 2. Oráculo Aproximado para Testes; 3. n-version testing. I. Miranda, Breno Alexandro Ferreira de. II. Título.

UFPE-Biblioteca Central

Críssia de Santana Marcelino

“Definição de Oráculos Aproximados para Testes de Desempenho com Base em Dados Históricos”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia de Software Linguagens de Programação.

Aprovado em: 27/03/2026.

BANCA EXAMINADORA

_____

Prof. Dr. Jamilson Ramalho Dantas
Centro de Informática/UFPE

_____

Profa. Dra. Silvana Morita Melo
Faculdade de Ciências Exatas e Tecnológicas/UFGD

_____

Prof. Dr. Breno Alexandro Ferreira de Miranda
Centro de Informática/UFPE
(**orientador**)

Para mim.

AGRADECIMENTOS

Sempre sou grata a Deus por tudo que me proporciona. Pela saúde, pela capacidade do pensar e questionar, pelas pessoas que coloca em minha vida. Considero-me privilegiada pelo carinho, apoio e amor que recebo dos meus. O mestrado foi mais uma fase em que recebi incentivos dos familiares e amigos, como também criei novos vínculos (colegas e professores) que me impulsionaram a avançar a cada semestre e me ajudaram direta/indiretamente nesta dissertação.

O maior agradecimento de minha vida é para meus pais. Pessoas simples, esforçadas e que sempre fizeram o seu melhor. Eles podem não entender a importância que o mestrado tem para mim, mas sempre me apoiaram, contribuindo para que eu avançasse nos estudos, sendo minha rede de apoio em diversos finais de semana, em que receberam minha filha para que eu pudesse me dedicar à pesquisa da dissertação. Por falar em rede de apoio, quero agradecer a Márcio, pelas inúmeras noites em que ficou com Clarice para que eu estudasse. Esse apoio foi valioso para que eu mantivesse a constância na pesquisa. Nunca vou esquecer.

Quero agradecer a minha filha Clarice, ela ainda não sabe, mas foi o impulso que eu precisava para iniciar o mestrado. Ser mãe despertou em mim a necessidade de ser mais para mim mesma, de me desafiar e de continuar realizando meus objetivos.

Agradeço aos amigos leais Cristiane, Steffane, Betânia, Edivânia e Emanoel que sempre estiveram ao meu lado, sendo meu divã quando eu precisava desabafar, desopilar, me incentivando e sendo o fôlego que eu precisei em muitos momentos para seguir. Em especial, quero agradecer ao amigo Felipe, que além de ter feito tudo que mencionei acima, também revisou a dissertação diversas vezes, deu feedbacks valiosos que sempre me induziam a melhorar. Felipe, esse apoio foi fundamental!

Quero agradecer ao colega Saulo Henrique Aguiar pela introdução no Google Colab. Ajudando-me na programação em python e na geração dos primeiros gráficos. Agradeço também aos colegas Lais Felipe e Davi Freitas pelos suportes no Overleaf. Agradeço também aos colegas de profissão que se disponibilizaram a participar do questionário para avaliar a abordagem, as opiniões de vocês me fizeram acreditar que o propósito do trabalho foi alcançado.

Quero agradecer imensamente ao meu orientador Breno Miranda. Desde o primeiro contato você demonstrou interesse e confiança em mim, na proposta da pesquisa e no trabalho. Foram anos de parceria, trilhando um caminho próspero. Obrigada pela paciência e pelos

ensinamentos.

Agradeço também ao projeto Samsung, que possibilitou que eu cursasse o mestrado com qualidade. A liberação para as aulas, a compreensão nos momentos em que eu precisei de um tempo a mais para desenvolver a pesquisa. Sem o incentivo à pós-graduação que vocês têm eu não conseguiria conciliar trabalho e mestrado. O suporte de vocês foi essencial para que eu alcançasse esse objetivo. Muito obrigada!

"O teste de software é e continuará sendo uma atividade fundamental da engenharia de software: apesar dos avanços revolucionários na forma como é construído e empregado (ou talvez exatamente por causa deles), o software sempre precisará ser eventualmente testado e monitorado.- Antonia Bertolino (BERTOLINO, 2007)

ABSTRACT

The constant evolution of software and the adoption of accelerated practices, such as Continuous Integration (CI), introduce a significant risk in contemporary Software Engineering: performance regression. Regression occurs when a system starts consuming more resources or takes longer to execute actions, compromising user satisfaction and increasing operational costs. Studies indicate that most failures found today are related to performance, rather than functional errors. In this scenario, the objective identification of these regressions is hindered by the methodological challenge known as the Oracle Problem: the difficulty in automating the definition of the correct/expected output of a test. For performance tests, throughput metrics are inherently sensitive to environmental variability and often depend on the tester's informal estimates, which can lead to false positives and insufficient system coverage. Therefore, the main objective of this work was to define an approximate oracle for performance tests based on a tolerance margin. The research proposed an innovative solution by adapting the N-version testing (NVT) method. Instead of traditional functional redundancy, the approach utilizes Temporal Redundancy, treating N historical and successful versions of the same software as a redundant and reliable database. The new version is validated through quantitative evaluation, comparing it with the historical distribution within an adaptive tolerance margin. The specific objectives included a literature review, obtaining historical data values from versions, defining the approach for creating the oracle for the studied applications, and automatically prototyping the approach. The inductive and exploratory methodology was applied in four stages: (1) Metric Definition and Historical Data Collection; (2) Establishment of the Performance Baseline; (3) Definition of the Tolerance Margin; and (4) Execution and Comparison of Results. To ensure adaptability to software evolution, the concept of a Sliding Window was incorporated, continuously updating the historical data and recalibrating the tolerance margin. Crucially, versions where performance degradation is confirmed are excluded from the baseline calculation to prevent bias and the occurrence of false negatives. The approach was validated through experimental scenarios, carried out on proprietary industrial image editing software, covering point-in-time and temporal measurements. The empirical results validated the effectiveness and soundness of the approach, proving that the Approximate Performance Test Oracle, based on the adaptation of N-version testing, is capable of defining an objective tolerance margin for detecting regressions, thus meeting the main objective of the dissertation. The work demonstrated that the technique has high diagnostic value, anticipating failures and reducing the

subjectivity of the human tester in defining and verifying performance outputs. In addition to the experimental validation, the approach was also subjected to a usability evaluation using the System Usability Scale (SUS), applied to software test engineers with experience in the field. The obtained result, with a score of 86.9, indicates a high level of acceptability and a positive perception regarding the ease of use and practical applicability of the proposed approach.

Keywords: performance tests; approximate oracle for test; n-version testing.

RESUMO

A constante evolução de software e a adoção de práticas aceleradas, como a Integração Contínua (CI), introduzem um risco significativo na Engenharia de Software contemporânea: a regressão de performance. A regressão ocorre quando um sistema passa a consumir mais recursos ou despende mais tempo para executar ações, comprometendo a satisfação do usuário e aumentando os custos operacionais. Estudos indicam que a maioria das falhas encontradas atualmente está relacionada à performance, e não mais a erros funcionais. Neste cenário, a identificação objetiva dessas regressões é dificultada pelo desafio metodológico conhecido como Oracle Problem: a dificuldade em automatizar a definição da saída correta/esperada de um teste. Para testes de desempenho, as métricas de rendimento são inerentemente sensíveis à variabilidade ambiental e frequentemente dependem de estimativas informais do testador, o que pode levar a falsos positivos e cobertura insuficiente do sistema. Diante disso, o objetivo principal deste trabalho foi definir um oráculo aproximado para testes de performance a partir de uma margem de tolerância. A pesquisa propôs uma solução inovadora, adaptando o método *N-version testing* (NVT). Em vez da redundância funcional tradicional, a abordagem utiliza a Redundância Temporal, tratando N versões históricas e bem-sucedidas do mesmo software como uma base de dados redundante e confiável. A nova versão é validada por avaliação quantitativa, comparando-se com a distribuição histórica dentro de uma margem de tolerância adaptativa. Os objetivos específicos incluíram a revisão de literatura, a obtenção dos valores dos dados históricos das versões, a definição da abordagem para criação do oráculo para as aplicações estudadas e a prototipação automática da abordagem. A metodologia indutiva e exploratória foi aplicada em quatro etapas: (1) Definição de Métrica e Coleta de Dados Históricos; (2) Estabelecimento da Referência de performance (baseline); (3) Definição da Margem de Tolerância; e (4) Execução e Comparação de Resultados. Para garantir a adaptabilidade à evolução do software, o conceito de Janela Móvel foi incorporado, atualizando continuamente o histórico de dados e recalibrando a margem de tolerância. Crucialmente, versões onde o erro de performance é confirmado são excluídas do cálculo da baseline para evitar o viés e a ocorrência de falsos negativos. A abordagem foi validada através de cenários experimentais, realizados em um software industrial proprietário de edição de imagens, abrangendo medições pontuais e temporais. Os resultados empíricos validaram a eficácia e a solidez da abordagem, comprovando que o Oráculo Aproximado para Testes de Performance, baseado na adaptação do *N-version testing*, é capaz de definir uma margem de tolerância objetiva para a detecção

de regressões, atendendo assim ao objetivo principal da dissertação. O trabalho demonstrou que a técnica possui alto valor diagnóstico, antecipando falhas e reduzindo a subjetividade do testador humano na definição e verificação das saídas de desempenho. Além da validação experimental, a abordagem também foi submetida a uma avaliação de usabilidade por meio do *System Usability Scale* (SUS), aplicada a Engenheiros de testes de software com experiência na área. O resultado obtido, com pontuação de 86.9, indica elevado nível de aceitabilidade e percepção positiva quanto à facilidade de uso e aplicabilidade prática da abordagem proposta.

Palavras-chave: testes de performance, oráculo aproximado para testes, n-version testing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CD	Continuous Delivery/Deployment
CI	Continuous Integration
CPU	Central Processing Unit
GB	Gigabyte
JEST	JavaScript Engines and Specification Tester
KPIs	Key Performance Indicator
MB	Megabyte
NVT	N-version testing
PODS	Projeto Sobre Software Diversificado
RNFs	Requisitos não funcionais
SUS	System Usability Scale
SUT	System Under Test
TI	Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O aperfeiçoamento contínuo de software é essencial para que sistemas permaneçam úteis, competitivos e atrativos aos usuários. Manter um software ativo e relevante no mercado exige atualizações cada vez mais frequentes, seja para incorporar novas funcionalidades, corrigir defeitos ou adaptar-se a mudanças tecnológicas. No entanto, essas evoluções não podem comprometer atributos de qualidade já consolidados, especialmente o desempenho.

Nesse contexto, estudos como os de (XIE; NOTKIN, 2005) e (CHEN; SHANG, 2017) alertam que atualizações frequentes trazem consigo um risco recorrente: a regressão de desempenho. Esse fenômeno ocorre quando uma nova versão do sistema passa a demandar mais tempo de execução ou maior consumo de recursos para realizar operações que anteriormente eram executadas de forma mais eficiente. Evidências recentes indicam que grande parte das falhas identificadas em sistemas modernos está associada a problemas de performance, e não apenas a erros funcionais (LIAO et al., 2022). No ambiente industrial, observa-se ainda que muitos sistemas não atendem satisfatoriamente aos requisitos de desempenho estabelecidos (KUDRJAVETS et al., 2022).

Identificar regressões de desempenho é fundamental, mas sua correção apresenta desafios significativos. A percepção de um desempenho adequado pode ser subjetiva, variando conforme o contexto de uso e as expectativas das partes interessadas. Por isso, a definição de Requisitos não funcionais (RNFs), bem como a especificação precisa de entradas, saídas e métricas de avaliação, torna-se imprescindível. Estabelecer padrões objetivos de qualidade exige conhecimento especializado, tempo e planejamento. Entretanto, na prática, restrições de prazo e recursos frequentemente limitam a atenção dedicada a esses aspectos durante o desenvolvimento.

Uma estratégia amplamente adotada para mitigar tais problemas é a execução sistemática de testes ao longo do processo de desenvolvimento. A prática de testar continuamente permite a identificação precoce de defeitos, reduz a propagação de falhas entre versões e minimiza custos decorrentes de retrabalho. Apesar de seus benefícios, a atividade de teste demanda esforço significativo da equipe e pode representar um custo elevado para o projeto.

Nesse cenário, a automatização de testes surge como alternativa para tornar o processo mais eficiente. Projetos que investem em automação buscam reduzir a dependência de esforço

manual e aumentar a confiabilidade e a repetibilidade das verificações. O principal artefato desse processo é o caso de teste, tradicionalmente estruturado em três componentes: (1) dados de entrada, (2) ações de execução e (3) resultados esperados. Contudo, a definição e a validação dos resultados esperados, especialmente em testes de performance, frequentemente representam um desafio, dificultando a determinação objetiva do sucesso ou fracasso do sistema sob teste.

Segundo (BARR et al., 2015), a automatização da geração das saídas esperadas permanece menos explorada e resolvida do que a geração automática de dados de entrada. Essa limitação evidencia uma fragilidade estrutural dos testes automatizados: mesmo quando entradas e execuções são sistematicamente produzidas, a determinação objetiva do sucesso ou fracasso do teste pode permanecer dependente de julgamento humano. Em testes de desempenho, essa dificuldade é ainda mais acentuada, pois métricas como tempo de resposta, uso de CPU, consumo de memória e tráfego de rede podem variar em função do ambiente de execução, ocasionando falsos positivos ou falsos negativos na identificação de regressões.

Uma prática que busca mitigar essa fragilidade é a utilização de oráculos de teste. Conforme definido por (BURNSTEIN, 2006), um oráculo é um mecanismo, documento ou software, que permite avaliar se a execução de um caso de teste foi aprovada ou reprovada, comparando a saída produzida com a saída esperada previamente definida. Entretanto, garantir que essa saída esperada seja correta, completa e automaticamente verificável constitui um dos principais desafios da área, conhecido como *Oracle Problem* (BARR et al., 2015). Esse problema refere-se à dificuldade de automatizar a definição do comportamento correto do sistema em meio a múltiplos resultados possíveis e variações contextuais.

Se a criação de oráculos já representa um desafio para testes funcionais, em propriedades não funcionais, como desempenho, a complexidade é ampliada. Em testes de performance, as métricas de rendimento podem evidenciar degradações sutis, como aumento progressivo de latência ou instabilidade sob carga concorrente (MALIK et al., 2013). Além disso, a variabilidade inerente ao ambiente de execução torna inadequada a adoção de limiares fixos e absolutos como critério universal de aceitação.

No contexto industrial, a verificação desses aspectos ainda depende fortemente da atuação humana. Em muitos cenários, o testador é responsável por analisar manualmente o comportamento do sistema diante de um conjunto extenso de casos de teste (AFSHAN et al., 2013). Esse processo, além de custoso, está sujeito à subjetividade e à inconsistência, especialmente quando envolve propriedades não funcionais. Ademais, é improvável que o testador antecipe

todos os impactos indiretos que uma modificação pode provocar em componentes não diretamente alterados. Sob restrições de tempo e infraestrutura, a atividade de teste tende a concentrar-se nas partes modificadas, reduzindo a cobertura efetiva e aumentando o risco de efeitos colaterais não detectados.

Diante desse cenário, torna-se fundamental adotar abordagens que reduzam a subjetividade do processo de teste, ao mesmo tempo em que demandem menor esforço humano e ofereçam maior confiabilidade na identificação de regressão de performance. Uma estratégia promissora para mitigar o *Oracle Problem* em testes de performance consiste na utilização de mecanismos comparativos entre versões sucessivas do sistema. Nesse contexto, este trabalho propõe a criação e aplicação de um oráculo aproximado de testes voltado à detecção de erros de performance, fundamentado no método de N-version testing (NVT). O NVT consiste na execução de múltiplas versões funcionalmente equivalentes de um sistema, desenvolvidas ou configuradas de forma independente, cujos resultados são comparados com o objetivo de identificar divergências comportamentais. A combinação entre comparação interversões e inferência orientada a dados enquadra-se na categoria de oráculos derivados, os quais utilizam artefatos como execuções anteriores, versões alternativas ou propriedades observáveis do sistema como referência (BARR et al., 2015).

Para a construção do oráculo proposto, adotam-se suítes de testes de regressão como mecanismo estruturante. Testes de regressão têm por objetivo identificar se modificações recentes comprometem comportamentos previamente estabelecidos (YOO; HARMAN, 2012), partindo da premissa de que versões anteriores podem servir como referência para o comportamento esperado. Neste contexto, dados históricos do sistema assumem papel central, pois permitem monitorar sua evolução ao longo do tempo (SERBOUT et al., 2025) e detectar regressões de performance entre versões sucessivas (PEER, 2024; KALYAN et al., 2023). A análise desses dados tem sido explorada por pesquisadores para identificar padrões de evolução, revelar tendências e apoiar o desenvolvimento de ferramentas voltadas ao monitoramento e à melhoria contínua da qualidade e da manutenibilidade do software (SERBOUT et al., 2025).

A partir da análise comparativa entre versões sucessivas, o oráculo opera de forma adaptativa, estimando e atualizando seus limites de aceitação com base na variabilidade histórica observada. Em vez de depender exclusivamente de limiares fixos, o mecanismo proposto infere comportamentos esperados de maneira incremental e orientada a dados, acompanhando a evolução natural do desempenho do software.

Assim, a proposta busca estabelecer saídas esperadas mais coerentes e contextualizadas,

reduzindo o custo associado ao oráculo humano. O objetivo central é diminuir, tanto quantitativa quanto qualitativamente, o esforço necessário para validar resultados de testes de performance, mantendo ou ampliando a cobertura e a confiabilidade do processo de teste. Dessa forma, esta dissertação insere-se no campo de pesquisa sobre oráculos de testes ao propor uma abordagem que combina regressão, comparação entre versões e inferência quantitativa adaptativa, contribuindo para o enfrentamento do *Oracle Problem* em cenários de regressão de desempenho.

Além dos desafios técnicos relacionados à definição de oráculos aproximados para testes de performance, também é necessário considerar a viabilidade prática e a aceitação da abordagem por profissionais da área. Em contextos industriais, soluções metodologicamente consistentes podem apresentar baixa adoção caso sejam percebidas como excessivamente complexas ou difíceis de integrar ao fluxo de trabalho existente. Diante disso, esta pesquisa também incorporou uma avaliação de usabilidade da abordagem proposta por meio do System Usability Scale (SUS), um instrumento amplamente utilizado para mensurar a percepção de facilidade de uso e aceitabilidade de sistemas e processos computacionais. A utilização do SUS neste estudo permitiu complementar a validação técnica da proposta com uma perspectiva centrada na experiência dos engenheiros de testes de software, contribuindo para avaliar não apenas a eficácia da abordagem na detecção de regressões, mas também seu potencial de adoção em cenários reais de desenvolvimento e testes contínuos.

1.2 PROBLEMA

O desempenho de um software pode variar significativamente entre versões sucessivas. Funcionalidades previamente consideradas estáveis podem apresentar degradações ou falhas em versões posteriores, cujas causas nem sempre são facilmente identificáveis, especialmente em cenários de testes automatizados. Na ausência de mecanismos capazes de regular e interpretar variações de desempenho ao longo da evolução do software, confirmar a ocorrência de regressões de desempenho torna-se um desafio. Nesse contexto, a adoção de práticas como a integração contínua tende a intensificar esse problema, uma vez que acelera o ciclo de mudanças no sistema. Uma alternativa para mitigar esses impactos é a utilização de oráculos de testes, os quais auxiliam na avaliação da correção do comportamento observado. Contudo, quando esses oráculos não são definidos a partir de um processo metodologicamente estruturado, a decisão sobre a correção do comportamento do sistema permanece fortemente

dependente da interpretação humana.

A literatura sobre oráculos de testes introduziu técnicas para automação de oráculo, incluindo modelagem, especificações, desenvolvimento orientado a contratos e testes metamórficos (construídos a partir de relacionamentos conhecidos entre comportamento desejado) (BARR et al., 2015). Entretanto, quando essas técnicas são aplicadas de forma inadequada ou incompleta, a definição das saídas corretas dos testes continua sendo delegada ao testador, muitas vezes fundamentada em orientações informais, especificações incompletas ou conhecimento tácito. Diante desse cenário, torna-se necessário estabelecer mecanismos que aumentem a objetividade e a confiabilidade dos oráculos de testes, especialmente no contexto de testes de desempenho.

Alguns trabalhos acadêmicos (PEER, 2024; KALYAN et al., 2023) defendem a aplicação do *N-version testing* como uma abordagem promissora para a identificação de regressão de desempenho. A utilização desse metódo para criação de oráculos aproximados de testes pode contribuir para a detecção mais assertiva de regressões de performance, bem como para a ampliação da cobertura do sistema testado. Diante disso, este trabalho buscou compreender como um oráculo aproximado para testes de performance pode ser definido através do *N-version testing*. Buscando avaliar sua eficácia na identificação de regressões, sua capacidade de oferecer saídas coerentes com o comportamento esperado e seu potencial para antecipar cenários de erro ao longo da evolução do software.

1.3 JUSTIFICATIVA

Testar a performance de sistemas é um processo intrinsecamente complexo, pois os testes executados frequentemente são específicos para cada tipo de software e altamente sensíveis ao ambiente de execução. Um mesmo sistema sob teste (System Under Test (SUT)) pode apresentar variações significativas nos resultados quando executado em máquinas com configurações distintas, tornando desafiadora a definição de critérios universais de aceitação. Além disso, em muitos projetos, requisitos de desempenho não são formalmente documentados; em vez disso, baseiam-se em percepções informais ou no consenso tácito entre membros da equipe.

Embora a experiência de especialistas possa mitigar ambiguidades, a ausência de formalização de RNFs, como desempenho, pode gerar inconsistências e impasses decisórios. Como estimar a performance esperada para uma funcionalidade específica? Qual métrica define se

uma regressão é aceitável ou crítica? A inexistência de critérios objetivos tende a ampliar discussões subjetivas, elevar custos e comprometer a reprodutibilidade das decisões.

Nesse contexto, a utilização de oráculos de testes surge como alternativa para estabelecer critérios mais sistemáticos de avaliação. Conforme discutido por (BARR et al., 2015), os oráculos podem ser classificados como especificados, derivados ou implícitos. Oráculos especificados dependem de requisitos formais previamente definidos; oráculos derivados inferem o comportamento esperado a partir de artefatos existentes, como execuções anteriores ou versões alternativas; e oráculos implícitos assumem propriedades gerais de correção. Embora essas categorias tenham sido amplamente investigadas, ainda há escassez de abordagens estruturadas voltadas especificamente à construção de oráculos derivados adaptativos para detecção de regressões de performance em ambientes com ausência de especificações formais.

Diante da realidade de muitos projetos industriais, nos quais não há indicadores de desempenho (Key Performance Indicator (KPIs)) formalizados nem limiares de desempenho claramente estabelecidos, os oráculos derivados apresentam-se como alternativa viável e pragmática. A utilização de dados históricos e testes de regressão como base para inferência do comportamento esperado permite reduzir a dependência de limiares arbitrários e do julgamento exclusivamente humano.

Para validar essa proposta, foi utilizada uma aplicação industrial com registros reais de regressão de performance. O contexto analisado, caracterizado por evolução contínua do sistema e ausência de critérios formais de aceitação de desempenho, representa um cenário recorrente na prática de desenvolvimento de software. A abordagem proposta, ao estruturar um mecanismo de oráculo aproximado orientado a dados históricos e comparação entre versões, busca reduzir a subjetividade, aumentar a confiabilidade das análises e apoiar decisões mais fundamentadas na detecção de regressões.

Assim, esta pesquisa justifica-se tanto pela relevância prática do problema, amplamente presente na indústria, quanto pela necessidade de avançar na construção de mecanismos adaptativos para oráculos aproximados para testes de performance, contribuindo para o enfrentamento do *Oracle Problem* em contextos de regressão de desempenho.

1.4 OBJETIVOS

O objetivo principal deste trabalho é propor uma abordagem adaptativa para definição de um oráculo aproximado para testes de performance, fundamentada em *N-version testing* e

análise de dados históricos, visando apoiar a detecção de regressões de desempenho e reduzir a subjetividade na validação de resultados.

Para alcançar o objetivo geral, estruturou-se um conjunto de objetivos específicos que organizam o desenvolvimento da pesquisa desde a fundamentação teórica até a validação experimental da abordagem. Esses objetivos visam garantir coerência metodológica, alinhamento com a lacuna identificada na literatura e aplicabilidade prática da solução proposta. São eles:

- Realização de uma revisão de literatura sobre regressão de performance, oráculos de testes, *N-version testing* e abordagens relacionadas ao *Oracle Problem*;
- Estruturação do uso de dados históricos de versões sucessivas como referência para inferência de comportamento esperado;
- Definição de uma abordagem adaptativa baseada em margem de tolerância dinâmica para construção de oráculo aproximado para testes de performance;
- Desenvolvimento de um protótipo para automatizar a aplicação da abordagem proposta;
- Avaliação experimental da abordagem em um contexto industrial real, analisando sua capacidade de identificar regressões de desempenho e seu potencial de redução da subjetividade na validação dos testes.

Em conjunto, esses objetivos buscam estruturar uma contribuição metodológica para o enfrentamento do *Oracle Problem* em testes de regressão de performance, articulando fundamentação teórica, modelagem da abordagem, implementação prática e avaliação em contexto real. Assim, a pesquisa pretende não apenas propor um mecanismo técnico, mas também oferecer evidências empíricas sobre sua viabilidade e potencial de aplicação em ambientes industriais.

1.5 METODOLOGIA DE PESQUISA

Esta pesquisa foi conduzida a partir de uma abordagem indutiva e exploratória, combinando análise qualitativa e quantitativa para investigar a utilização do *N-version testing* na definição de oráculos aproximados para testes de performance (GIL, 1999), (GIL, 2008).

Quanto à natureza, o estudo caracteriza-se como uma pesquisa aplicada, uma vez que busca propor e validar uma solução prática para o problema da detecção de regressões de

desempenho em ambientes reais de desenvolvimento de software (SILVA; MENEZES, 2001). Em relação aos objetivos, a pesquisa possui caráter exploratório, visando aprofundar a compreensão sobre o uso de dados históricos e comparação entre versões como mecanismo de inferência do comportamento esperado em testes de performance (LAKATOS; MARCONI, 2010).

Os procedimentos metodológicos adotados envolveram pesquisa bibliográfica e análise de estudos de casos (SELLTIZ; WRIGHTSMANN, 1967). A pesquisa bibliográfica foi utilizada para fundamentar os conceitos relacionados a performance, regressão de performance, oráculos de testes e *N-version testing*, além de apoiar a identificação de lacunas e abordagens correlatas na literatura. Já a análise do estudo de caso permitiu avaliar a viabilidade da abordagem proposta em um contexto industrial real.

O desenvolvimento da pesquisa foi estruturado em três etapas principais: análise de conteúdo, construção da abordagem e validação.

Na etapa de análise de conteúdo, realizou-se o levantamento e análise da literatura relacionada aos temas centrais da pesquisa, permitindo compreender limitações existentes no contexto de oráculos para testes de performance e identificar possibilidades de adaptação do *N-version testing* para esse domínio.

A etapa de construção concentrou-se na definição da abordagem de oráculo aproximado baseada em dados históricos de versões sucessivas do software. Nessa fase, foram estabelecidos os mecanismos de coleta de métricas, definição de *baseline*, cálculo da margem de tolerância adaptativa e utilização da janela móvel para acompanhamento da evolução do sistema.

Por fim, a etapa de validação consistiu na execução de avaliações dos estudos de caso em um software industrial proprietário de edição de imagens, utilizando medições pontuais e temporais de desempenho. Os resultados obtidos foram analisados com o objetivo de verificar a capacidade da abordagem em identificar regressões de performance e reduzir a subjetividade na validação dos testes. Complementarmente, foi realizada uma avaliação de usabilidade por meio do SUS, aplicada a profissionais da área de testes de software, visando analisar a percepção de aceitabilidade e aplicabilidade prática da abordagem.

1.6 ESTRUTURA DA DISSERTAÇÃO

O restante do conteúdo desta dissertação está organizado da seguinte forma:

Capítulo 2. Fundamentação Teórica: traz os conceitos que envolve as principais áreas

de estudo deste trabalho, como performance, teste de regressão, regressão de performance, oráculos de testes e *N-version testing*. Também apresenta a revisão de literatura e os trabalhos relacionados aos objetivos deste trabalho, dicutindo as principais contribuições e lacunas das diferentes abordagens e aplicações do *N-version testing*.

Capítulo 3. Expõe a abordagem de oráculo aproximado para testes de performance: descreve a utilização do *N-version testing* e da janela móvel para definição da margem adaptativa, apresentando seu conjunto de princípios, processos, atividades, resultados e métricas.

Capítulos 4 e 5. Avaliação: destaca os resultados dos estudos de caso realizados em um software industrial com o objetivo de identificar a viabilidade, eficiência, completude e assertividade da abordagem proposta.

Capítulo 6. Automação da lógica do Oráculo: Definição dos requisitos e os parâmetros necessários para criação de um protótipo automatizado e como ele reduz a subjetividade humana na detecção de regressão de performance.

Capítulo 7. Avaliação de Usabilidade: Aplicação do SUS sobre a abordagem para profissionais experientes em testes de softwares e avaliação da posição da abordagem na escala de classificação de (BANGOR et al., 2008).

Capítulo 8. Ameaças à Validade: Expõe os artefatos externos que podem influenciar nos resultados obtidos e as estratégias adotadas para minimizar ou dissipar essas influências.

Capítulo 9. Contribuições da Dissertação: apresenta as conclusões do estudo, resume as principais contribuições, assim como suas limitações e aponta sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

2.1 PERFORMANCE

Em geral, a performance é um requisito não funcional e refere-se a atributos de qualidade do software. Ela mede a capacidade de um sistema ou aplicativo de responder às solicitações dos usuários de maneira eficiente, considerando critérios como tempo de resposta, uso de recursos computacionais e escalabilidade. Para (LIAO, 2023) a performance mede a eficácia de um sistema de software a partir de diferentes métricas, tais como tempo de resposta, rendimento e utilização da CPU. Sendo assim, entende-se que independente do objetivo para o qual determinado software foi desenvolvido, espera-se que ele apresente respostas rápidas, utilize de forma eficiente os recursos de hardware disponíveis e seja capaz de escalar conforme o aumento da demanda, garantindo a satisfação do usuário e a sustentabilidade do sistema.

Estudos indicam que aspectos não funcionais são mais propensos a apresentar falhas em ambientes reais de uso, especialmente em software de grande escala, quando comparados aos aspectos funcionais (CHEN; SHANG, 2017). Dentre esses aspectos, o desempenho demanda atenção especial devido ao impacto direto que exerce sobre a percepção do usuário e sobre o sucesso ou fracasso de uma aplicação. Nesse sentido, a análise de performance deve ser considerada desde as fases iniciais do desenvolvimento de software, uma vez que intervenções tardias voltadas à otimização tendem a ser mais custosas e menos eficazes (BASS L., 2012).

Por isso, investir em testes de performance é fundamental para manter um software funcionalmente eficaz. Estes testes são responsavéis por analisar características como velocidade, estabilidade, confiabilidade e escalabilidade do sistema, permitindo identificar componentes ou funcionalidades que apresentem comportamento inadequado sob diferentes condições de carga. Ignorar o teste de desempenho significa que seu sistema não foi totalmente testado, especialmente das perspectivas de risco e perfil operacional (KAUSHIK; FAGERIA, 2014). Entre os principais tipos de testes de performance destacam-se os testes de carga, estresse e resistência, cada um voltado a avaliar aspectos específicos do comportamento do sistema.

Para a execução desses testes, utilizam-se ferramentas especializadas capazes de medir o tempo necessário para a realização de determinadas tarefas, bem como avaliar a estabilidade do sistema sob diferentes volumes de requisições. Idealmente, esses testes devem ser conduzidos de forma realista, aproximando-se o máximo possível das condições do ambiente de produção e sendo executados de maneira periódica ao longo da evolução do software. No entanto,

embora seja desejável realizar testes de performance a cada nova versão, a viabilidade dessa prática é frequentemente limitada por custos elevados e restrições de tempo e infraestrutura (HANNIGAN, 2018).

Como consequência dessas limitações, problemas de desempenho podem permanecer latentes por várias versões, manifestando-se apenas após determinadas modificações no sistema. Um dos problemas mais recorrentes nesse contexto são as regressões de performance, caracterizadas por situações em que uma nova versão do software mantém sua funcionalidade correta, porém apresenta uma degradação na experiência do usuário, como aumento do tempo de resposta ou maior consumo de recursos computacionais, em comparação com versões anteriores. Essas regressões são particularmente críticas, pois podem passar despercebidas em avaliações pontuais e gerar impactos negativos significativos, como redução da satisfação do usuário, aumento de custos operacionais e falhas em ambientes de produção (LIAO, 2023).

Diante desse cenário, torna-se evidente que avaliar desempenho de forma isolada não é suficiente, sendo necessário observar o comportamento do sistema ao longo de sua evolução. Essa necessidade reforça a importância de mecanismos sistemáticos capazes de comparar versões sucessivas do software, permitindo identificar variações negativas de desempenho introduzidas por modificações no código. Nesse contexto, os testes de regressão emergem como uma prática fundamental para garantir que alterações realizadas no sistema não comprometam características previamente estabelecidas.

2.2 TESTES DE REGRESSÃO

Os testes de regressão são um tipo de teste de software que verifica se alterações no código (como novas funcionalidades, correções de bugs ou refatorações) não introduziram falhas em partes previamente funcionais do sistema. Ele se baseia na suposição implícita de que a versão anterior pode servir como um oráculo para a funcionalidade existente. Segundo (SOMMERVILLE, 2011), os testes de regressão garantem que mudanças no software não impactem negativamente o comportamento previamente validado do sistema.

De forma geral, os testes de regressão são compostos por casos de testes que descrevem cada passo que deve ser executado e o comportamento esperado em um determinado contexto. Um caso de teste consiste em duas partes: uma entrada de teste para exercitar o sistema e um oráculo de teste para verificar a exatidão da execução do teste (XIE, 2006). No âmbito funcional, essa verificação costuma ser objetiva, uma vez que os resultados esperados são bem definidos.

Contudo, quando os testes de regressão são estendidos para propriedades não funcionais, como desempenho, a definição do comportamento esperado torna-se mais complexa.

Em testes de regressão funcional, especialmente no caso de modificações corretivas, assume-se que o comportamento desejado permanece inalterado. Dessa forma, o oráculo de teste utilizado em uma versão pode ser reaproveitado para validar versões subsequentes. Essa estratégia tem se mostrado eficaz para detectar falhas introduzidas inadvertidamente durante o processo de manutenção do software (GU et al., 2010). Entretanto, essa suposição não se aplica de maneira direta aos atributos de desempenho, os quais podem variar de forma legítima ou indesejada em função de alterações aparentemente benignas no código.

No contexto de performance, o resultado da execução de um ciclo de testes de regressão atua como um indicador da estabilidade do sistema ao longo de sua evolução. Uma nova versão pode manter sua correção funcional e, ainda assim, apresentar degradações sutis de eficiência, tais como aumento no tempo de resposta, maior consumo de recursos computacionais ou redução da escalabilidade. Essas variações nem sempre violam requisitos formais, mas podem comprometer a experiência do usuário e elevar os custos operacionais.

Assim, ao estender testes de regressão para propriedades de performance, torna-se necessário observar não apenas a correção do comportamento, mas também a variação do desempenho entre versões sucessivas. Essas variações caracterizam o fenômeno conhecido como regressão de performance, no qual o software continua operando corretamente do ponto de vista funcional, porém com eficiência reduzida. A compreensão desse tipo específico de regressão e seus impactos é fundamental para a definição de estratégias adequadas de detecção e análise, conforme discutido da seção seguinte.

2.3 REGRESSÃO DE PERFORMANCE

A regressão de performance ocorre quando uma nova versão de um software apresenta desempenho pior do que uma versão anterior em termos de tempo de resposta, que é o intervalo entre uma solicitação e sua respectiva resposta, rendimento, que mede a quantidade de operações concluídas por unidade de tempo, consumo de recursos ou escalabilidade. Uma regressão de performance acontece quando modificações no software resultam em uma piora na eficiência, mesmo sem alterações funcionais aparentes (SMITH; WILLIAMS, 2002). Em ambientes de desenvolvimento evolutivo, tais degradações podem ser introduzidas de forma gradual, tornando sua identificação menos evidente do que falhas funcionais tradicionais.

Diferentemente dos defeitos funcionais, a regressão de performance não se caracteriza por comportamentos incorretos explícitos, mas por variações quantitativas em métricas de desempenho previamente aceitáveis. Essas variações podem ser influenciadas por diversos fatores, como alterações no código-fonte, inclusão de novas funcionalidades, mudanças em bibliotecas externas ou variações no ambiente de execução. Conforme destacado por (SMITH; WILLIAMS, 2002), a ausência de critérios objetivos para avaliar essas variações dificulta a distinção entre flutuações naturais do sistema e regressões efetivas de desempenho.

Nesse contexto, a análise comparativa entre versões assume papel fundamental. A utilização de dados históricos de desempenho possibilita observar a evolução do sistema ao longo do tempo, identificar tendências e detectar desvios que possam indicar regressões. Estudos recentes (XIE, 2006) (LIU et al., 2021) reforçam que o uso sistemático desses dados contribui para uma avaliação mais consistente do desempenho do software, especialmente em cenários de integração contínua, nos quais novas versões são disponibilizadas com alta frequência (SANTOS et al., 2025).

Entretanto a simples coleta de dados históricos não é suficiente para garantir a identificação automática de regressões de performance. A interpretação dos resultados ainda depende, em muitos casos, do julgamento humano, que precisa decidir se uma variação observada compromete ou não o desempenho esperado do sistema. Essa dependência introduz subjetividade no processo de teste, limita sua reprodutibilidade e dificulta a automação em larga escala.

Diante dessas limitações, evidencia-se a necessidade de mecanismos capazes de apoiar a análise objetiva dos resultados de testes de performance. Nesse cenário, os oráculos de testes tornam-se essenciais ao estabelecer critérios que permitam determinar se o comportamento observado é aceitável ou representa uma regressão de desempenho. Assim, a definição de oráculos adequados configura-se como um elemento central para a automação e a confiabilidade dos testes de performance.

2.4 ÓRACULO DE TESTES

A utilização de oráculos de testes começou a ser explorada no final dos anos 70 (BARR et al., 2015) apud (HOWDEN, 2006). De forma geral, um oráculo define critérios que permitem determinar se a saída produzida por um sistema, para uma dada entrada, é correta ou aceitável. No contexto de testes de performance, esses critérios podem assumir a forma de limites aceitáveis para tempo de resposta, consumo de recursos ou throughput. Por exemplo, se um

sistema web deve responder em menos de 200ms para 95% das requisições, o oráculo pode ser programado para considerar qualquer valor acima disso como falha no teste de performance.

Oráculos de testes também são fundamentais para a automação e o monitoramento contínuo, podendo ser integrados a pipelines de integração contínua (Continuous Integration (CI)) e entrega contínua (Continuous Delivery/Deployment (CD)) para evitar a liberação de versões com degradações significativas de desempenho. Além disso, ao estabelecer parâmetros esperados para métricas de performance, os oráculos auxiliam na identificação de gargalos e anomalias no comportamento do sistema.

Segundo (BARR et al., 2015), os oráculos de testes podem ser classificados em 3 categorias principais: oráculos de testes especificados, oráculos de testes derivados e oráculos de testes implícitos. De forma objetiva, segue as definições de cada categoria. Os oráculos especificados consideram todos os aspectos comportamentais de um sistema com relação a uma especificação formal dada, ou seja, se baseiam na documentação. Já os oráculos derivados são obtidos a partir de artefatos existentes, como execuções anteriores ou versões alternativas do sistema. Por sua vez, o oráculo implícito está relacionado à detecção de falhas evidentes, como travamentos, exceções ou violações genéricas de propriedades do sistema. Além dessas categorias clássicas, a literatura apresenta abordagens especializadas para lidar com cenários em que a definição explícita da saída correta é complexa ou inviável. Entre elas, destacam-se os oráculos metamórficos e os oráculos baseados em *Machine Learning*.

Os oráculos metamórficos fundamentam-se na definição de relações metamórficas, isto é, propriedades esperadas entre diferentes entradas e saídas do sistema. Em vez de validar diretamente se uma saída é correta, essa técnica verifica se determinadas transformações aplicadas às entradas produzem relações coerentes entre os resultados obtidos (CHEN; SHANG, 2017). Por exemplo, em um sistema de processamento de imagens, aumentar proporcionalmente o brilho de uma entrada pode exigir que a saída preserve determinadas propriedades visuais esperadas. Essa abordagem é particularmente útil em contextos nos quais não existe um resultado esperado claramente definido, como aplicações científicas, algoritmos de otimização e sistemas inteligentes. Entretanto, a construção de relações metamórficas válidas depende de conhecimento especializado sobre o domínio da aplicação, o que pode dificultar sua adoção em larga escala.

Outra abordagem relevante consiste nos oráculos baseados em Machine Learning, que utilizam modelos treinados a partir de dados históricos para inferir padrões esperados de comportamento do sistema (BARR et al., 2015). Nesses cenários, algoritmos de aprendizado podem

ser empregados para detectar anomalias, prever tendências de desempenho ou classificar execuções como aceitáveis ou suspeitas. Em testes de performance, por exemplo, modelos podem aprender padrões normais de consumo de CPU, memória ou tempo de resposta e identificar desvios potencialmente associados a regressões. Apesar de oferecerem elevado potencial adaptativo, essas abordagens frequentemente dependem de grandes volumes de dados rotulados, treinamento contínuo e ajustes de hiperparâmetros, além de apresentarem limitações relacionadas à interpretabilidade dos critérios utilizados na tomada de decisão.

Considerando que este trabalho busca identificar regressões de performance por meio da comparação entre versões de software, os oráculos derivados mostram-se particularmente adequados, uma vez que utilizam versões anteriores como referência para inferir o comportamento esperado do sistema. A abordagem proposta nesta dissertação aproxima-se dessa categoria ao utilizar dados históricos reais de versões anteriores para estabelecer margens adaptativas de aceitação. Diferentemente dos oráculos metamórficos, a proposta não depende da definição manual de relações entre entradas e saídas. Da mesma forma, distingue-se dos oráculos baseados em *Machine Learning* por não exigir treinamento de modelos complexos ou grandes conjuntos de dados rotulados. Em vez disso, fundamenta-se na análise quantitativa da variabilidade histórica observada ao longo da evolução do software.

Neste trabalho, define-se um oráculo aproximado para testes de performance como um mecanismo de validação derivado capaz de inferir o comportamento esperado de desempenho de um sistema a partir da análise quantitativa de versões históricas previamente validadas, utilizando margens adaptativas de tolerância para identificar possíveis regressões de performance. Diferentemente de oráculos tradicionais baseados em saídas exatas, o oráculo aproximado considera a variabilidade inerente às métricas de desempenho e utiliza dados históricos como referência probabilística para tomada de decisão sobre a aceitabilidade dos resultados observados. Nesse contexto, técnicas que exploram múltiplas versões do sistema, como o *n-version testing*, tornam-se especialmente relevantes, pois oferecem um mecanismo comparativo capaz de apoiar a inferência do comportamento esperado a partir da redundância entre execuções e versões históricas do software.

2.5 N-VERSION TESTING

O *N-version testing* é uma técnica de teste de software baseada na execução e comparação de múltiplas implementações independentes de uma mesma especificação funcional, com

o objetivo de aumentar a confiabilidade do sistema e mitigar o problema do oráculo. Nessa abordagem, diferentes versões são submetidas aos mesmos dados de entrada, e discrepâncias entre seus resultados são interpretadas como indícios de falhas, podendo-se empregar mecanismos como a votação majoritária para determinar a resposta considerada correta (PATEL; HIERONS, 2018); (MANOLACHE; KOURIE, 2001); (BARR et al., 2015).

Originalmente proposto para lidar com situações em que o resultado correto de uma execução não é previamente conhecido, o *N-version testing* utiliza versões alternativas ou de referência como oráculos implícitos para verificar comportamentos divergentes. Formalmente, considerando S como o sistema sob teste (SUT) e $\mathcal{S}_R$ como uma implementação de referência, ambos são executados com um mesmo caso de teste. Caso as saídas produzidas por S e $\mathcal{S}_R$ divirjam, registra-se uma falha. Quando há múltiplas versões de referência, o processo pode ser repetido para cada uma delas.

Essa técnica fundamenta-se na premissa de que falhas introduzidas de maneira independente tendem a se manifestar de forma distinta, reduzindo a probabilidade de que múltiplas versões apresentem o mesmo erro para uma mesma entrada. Embora a independência completa de falhas nem sempre é alcançada na prática (KNIGHT; LEVESON, 1985), essa suposição permanece como um dos pilares teóricos que justificam a efetividade do *N-version testing* como técnica de verificação baseada em redundância.

Por sua natureza, o *N-version testing* é uma técnica de teste de caixa-preta, concentrando-se exclusivamente nas entradas e saídas observáveis das versões testadas, sem considerar aspectos internos de implementação (PATEL; HIERONS, 2018). Dentre os contextos em que essa técnica é tradicionalmente aplicada, destacam-se: Sistemas críticos e de alta confiabilidade, onde falhas podem acarretar consequências severas, como em sistemas aeronáuticos e de controle industrial; Testes de regressão entre versões concorrentes ou alternativas, onde se deseja garantir que diferentes implementações preservem comportamentos funcionais consistentes; Sistemas com problemas de oráculo, como software científico e de aprendizagem de máquina (*Machine Learning*), em que o resultado correto pode não ser trivial de definir sem um modelo de referência; Comparações de compiladores ou interpretadores de linguagens, especialmente quando não há uma especificação executável única e completa.

A partir dessas caracterísitcas, o teste diferencial pode ser compreendido como uma generalização prática do *N-version testing*. Nessa abordagem, em vez de desenvolver múltiplas implementações independentes, comparam-se automaticamente diferentes versões, configurações ou plataformas de um mesmo software, executadas com entradas idênticas, a fim de

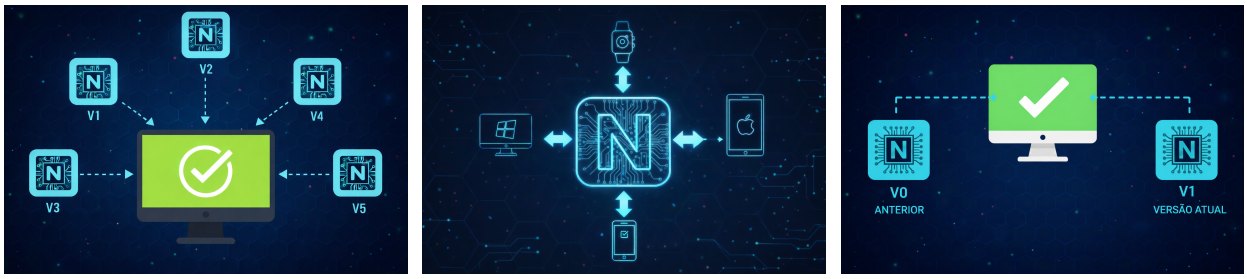
identificar divergências de comportamento observável, sem a necessidade de um oráculo explícito (MCKEEMAN, 1998).

O teste diferencial se caracteriza nos seguintes tipos: (i) Comparação de múltiplas implementações (*multi-backends*), executa a mesma entrada em várias versões e observar discrepâncias nas saídas; (ii) N+1-version (teste diferencial),que inclui explicitamente uma versão associada à especificação; e (iii) *Benchmarking* de regressão, no qual versões anteriores ou versões consideradas estáveis são utilizadas como referência para avaliar versões modificadas.

A Figura 1 ilustra como o teste diferencial é aplicado dentro do contexto do *N-version testing*.

Figura 1 – N-version testing dentro do contexto de Teste diferencial

(a) Múltiplas implementações (b) Versão de referência (c) Teste de Regressão



Fonte: Elaborada pela autora (2026)

Tradicionalmente, tanto o *N-version testing* quanto o teste diferencial têm sido aplicados com foco predominante em requisitos funcionais, nos quais discrepâncias entre resultados podem ser resolvidas por meio de decisões binárias ou votação majoritária, assumindo-se que a resposta correta corresponde ao comportamento adotado pela maioria das versões.

Entretanto, a aplicação direta desses mecanismos tradicionais mostra-se inadequada para a avaliação de RNFs, em especial performance. Métricas como tempo de resposta, vazão e consumo de recursos estão sujeitas a variações ambientais e flutuações naturais de execução, o que inviabiliza a definição de um único valor correto ou a exclusão simples de versões divergentes. Decisões pontuais baseadas em comparações diretas podem, nesse contexto, distorcer os resultados e comprometer a credibilidade da análise.

Dessa forma, ao tratar atributos não funcionais, torna-se mais apropriado analisar conjuntos de valores provenientes de múltiplas execuções, capturando a variabilidade natural do sistema e permitindo uma avaliação mais consistente do comportamento esperado. Sob essa perspectiva, os princípios do *N-version testing* configuram um arcabouço conceitual promissor para análises comparativas entre versões de software, particularmente no contexto de testes de regressão de performance, no qual a definição objetiva de um comportamento esperado permanece um

desafio. A partir dessa base teórica, torna-se viável investigar adaptações da técnica para apoiar a definição de oráculos aproximados para testes de performance. Nesse sentido, o capítulo seguinte destaca como diferentes trabalhos têm aplicado o *N-version testing* em variados contextos.

2.6 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou os principais conceitos que fundamentam a área de testes de desempenho, incluindo métricas de performance, testes de regressão e regressão de performance, além de discutir o papel dos oráculos de teste e do *N-version Testing* na validação de sistemas. Também foram abordadas limitações práticas relacionadas à definição de critérios objetivos para avaliação de desempenho, especialmente em contextos onde há variabilidade nas medições. Como contribuição, o capítulo fornece uma visão estruturada do estado da arte, organizando conceitos essenciais e evidenciando os principais desafios e abordagens existentes na literatura, servindo como base teórica para a compreensão do problema investigado.

3 TRABALHOS RELACIONADOS

O trabalho de (HATTON, 1997) constitui um marco crítico na discussão sobre a Programação N-Versões ao questionar empiricamente a suposição de que múltiplas implementações independentes de uma mesma especificação conduzem, de forma confiável, à redução de falhas. O autor investiga o paradigma clássico de N-Versões a partir da comparação entre múltiplas implementações reais, desenvolvidas de maneira independente, com o objetivo de avaliar a ocorrência de falhas coincidentes e a efetividade do voto majoritário como mecanismo de validação. Os resultados apresentados indicam que falhas correlacionadas são frequentes, especialmente em cenários nos quais ambiguidades ou complexidades da especificação levam diferentes equipes a interpretações semelhantes. No contexto do teste de software, o estudo demonstra que a concordância entre versões não pode ser interpretada como evidência suficiente de correção funcional, uma vez que múltiplas versões podem produzir resultados idênticos e incorretos. A partir dessas observações, Hatton argumenta que, em muitos casos, o investimento no desenvolvimento e teste rigoroso de uma única versão, a one good version, pode ser mais eficaz do que a adoção indiscriminada de múltiplas versões com diversidade apenas aparente. Esse trabalho estabelece, portanto, uma base crítica fundamental para abordagens posteriores que buscam reinterpretar ou reformular o uso da diversidade no teste de software.

As críticas formuladas por (HATTON, 1997) ao uso do teste baseado em N-Versões são diretamente relevantes para a proposta desta dissertação, uma vez que o autor demonstra que a concordância entre múltiplas implementações não garante, por si só, a correção do software, especialmente quando todas as versões compartilham interpretações semelhantes da especificação. A abordagem proposta neste trabalho responde a essas limitações ao abandonar a necessidade de múltiplas implementações paralelas e, consequentemente, o risco de falhas correlacionadas entre versões distintas. Em vez disso, a diversidade explorada é de natureza temporal e amostral, baseada no comportamento histórico do próprio sistema ao longo de sua evolução. O oráculo aproximado aqui definido não se fundamenta na concordância entre versões independentes, mas na identificação de desvios quantitativos em relação a padrões de desempenho previamente observados. Assim, diferentemente do teste de N-Versões criticado por Hatton, a proposta desta dissertação substitui a diversidade estrutural entre implementações por uma diversidade empírica derivada de múltiplas execuções reais, oferecendo um mecanismo de validação mais alinhado a cenários de evolução contínua e menos suscetível

às limitações apontadas no contexto clássico da Programação N-Versões. Em síntese, ao substituir a diversidade entre versões pela diversidade ao longo da evolução do sistema, esta dissertação redefine o uso de oráculos aproximados e oferece uma resposta direta, prática e conceitualmente alinhada às limitações evidenciadas por Hatton.

Em contraste com a perspectiva crítica de (HATTON, 1997), (MANOLACHE; KOURIE, 2001) propõem uma reinterpretação dos princípios da Programação N-Versões no contexto específico do teste de software. Os autores introduzem a técnica de *M-mp testing*, na qual um programa principal é testado em conjunto com um conjunto de Programas Modelo desenvolvidos de forma independente a partir da mesma especificação. Diferentemente do uso tradicional de N-Versões para tolerância a falhas em tempo de execução, essa abordagem utiliza múltiplas implementações como um oráculo diferencial, voltado à detecção de discrepâncias durante o processo de teste. Os Programas Modelo não são concebidos como versões completas do sistema, mas como abstrações funcionais que capturam aspectos relevantes da especificação, permitindo a comparação automática de saídas sem a necessidade de previsão manual de resultados esperados. Ao evitar o uso de voto majoritário e tratar divergências como indícios diagnósticos, o trabalho responde parcialmente às críticas de Hatton, deslocando o foco da concordância entre versões para a identificação de inconsistências. Ainda que permaneça dependente da diversidade estrutural entre implementações, o *M-mp testing* inaugura uma linha de pesquisa que explora a diversidade como mecanismo de oráculo aproximado, e não como garantia absoluta de correção.

A proposta apresentada por (MANOLACHE; KOURIE, 2001) estabelece uma base conceitual relevante para este trabalho ao explorar o uso de múltiplas implementações independentes como mecanismo de validação automática, mitigando o problema do oráculo de teste por meio da adaptação dos princípios da Programação N-Versões. Embora ambas as abordagens se fundamentem no princípio da diversidade como mecanismo de aumento de confiança na verificação de resultados, diferem substancialmente quanto à natureza dessa diversidade e às estratégias adotadas. Enquanto o *M-mp testing* depende da construção explícita de programas modelo funcionais e semanticamente equivalentes ao sistema sob teste, exigindo desenvolvimento paralelo e comparações diretas de saídas, a abordagem proposta nesta dissertação concentra-se na utilização de dados históricos de desempenho e em critérios quantitativos para a identificação de regressões, dispensando a necessidade de múltiplas implementações completas da funcionalidade. Além disso, o trabalho de (MANOLACHE; KOURIE, 2001) foca predominantemente na correção funcional, ao passo que a presente proposta direciona-se à

detecção de regressões de desempenho, tratando o comportamento temporal do software como um oráculo aproximado. Dessa forma, esta dissertação amplia o conceito de oráculo baseado em múltiplas perspectivas ao deslocar o eixo da diversidade de implementações para a diversidade temporal e quantitativa do comportamento do sistema, mantendo a premissa de redução da subjetividade defendido pelo (MANOLACHE; KOURIE, 2001), porém com menor custo de desenvolvimento e maior aderência a cenários de evolução contínua do software.

O Projeto Sobre Software Diversificado (PODS), apresentado por (BISHOP et al., 2012), amplia o debate ao tratar a diversidade de software não como um pressuposto, mas como um fenômeno empírico a ser medido, analisado e compreendido. Inserido no contexto de sistemas críticos, o trabalho investiga sistematicamente os benefícios e limitações da diversidade introduzida por múltiplas versões desenvolvidas a partir da mesma especificação. A metodologia do projeto envolve a construção de versões diversas utilizando diferentes linguagens, ferramentas e estratégias de desenvolvimento, seguidas de execuções controladas sobre conjuntos comuns de testes. O teste de N-Versões é empregado como instrumento analítico para observar padrões de falhas compartilhadas e avaliar o grau de correlação entre erros, e não como um mecanismo operacional de validação por consenso. Dessa forma, o PODS consolida empiricamente as críticas levantadas por (HATTON, 1997), ao mesmo tempo em que reforça a utilidade do teste comparativo entre versões como meio de compreensão do comportamento do software. O trabalho aproxima-se de (MANOLACHE; KOURIE, 2001) ao utilizar discrepâncias como fonte de informação, mas mantém um foco explícito na avaliação da diversidade estrutural e de suas limitações práticas.

A abordagem apresentada por (BISHOP et al., 2012) estabelece um elo conceitual importante com a proposta desta dissertação ao tratar a diversidade não como um pressuposto, mas como uma propriedade observável e mensurável, derivada da análise empírica do comportamento do software. Assim como no projeto PODS, a presente dissertação abandona a dependência de múltiplas implementações paralelas e do voto majoritário como critério de validação, reconhecendo explicitamente os riscos associados às falhas correlacionadas. No entanto, enquanto o PODS explora a diversidade estrutural entre versões desenvolvidas independentemente, este trabalho desloca esse princípio para a diversidade temporal, fundamentada na evolução do desempenho do próprio sistema ao longo de sucessivas versões. Em ambos os casos, o teste baseado em múltiplas observações é utilizado como mecanismo de redução da subjetividade no processo de validação, porém a abordagem desta dissertação amplia essa perspectiva ao aplicar critérios quantitativos para a detecção de regressões de performance, oferecendo uma

alternativa alinhada às limitações empíricas identificadas no contexto clássico da Programação N-Versões.

Mais recentemente, (PARK et al., 2021) propôs o JavaScript Engines and Specification Tester (JEST), uma abordagem de teste diferencial em larga escala que estende o paradigma de N-Versões ao introduzir explicitamente a especificação como uma versão adicional, caracterizando o teste N + 1 versões. O trabalho aplica essa técnica ao domínio de motores JavaScript, executando automaticamente casos de teste em múltiplas implementações reais e em um modelo executável derivado da especificação ECMAScript. Diferentemente do paradigma clássico criticado por (HATTON, 1997), o JEST não utiliza voto majoritário nem assume que a concordância entre versões implica correção. Ao contrário, qualquer divergência é tratada como informação relevante, podendo indicar defeitos de implementação ou ambiguidades na própria especificação. Essa abordagem consolida uma evolução conceitual importante: a diversidade deixa de ser apenas estrutural e passa a incluir a relação entre implementação e especificação como fonte ativa de oráculos.

A abordagem proposta por (PARK et al., 2021) apresenta forte afinidade conceitual com a proposta desta dissertação ao utilizar a diversidade como mecanismo central de validação automática, sem depender de um oráculo perfeito ou de especificações completamente precisas. No entanto, enquanto o JEST explora a diversidade estrutural e semântica entre múltiplas implementações e a especificação para detectar defeitos funcionais, a presente dissertação desloca esse princípio para o domínio dos RNFs, em especial a regressão de performance. Além disso, diferentemente do uso simultâneo de múltiplas versões independentes, a abordagem aqui proposta baseia-se na diversidade temporal do próprio sistema, utilizando dados históricos de execução para construir um oráculo adaptativo fundamentado em critérios quantitativos. Nesse sentido, ambos os trabalhos compartilham a premissa de que discrepâncias são mais informativas do que concordâncias, mas diferem quanto à natureza da diversidade explorada e ao tipo de comportamento analisado, reforçando a aplicabilidade do paradigma de oráculos aproximados em contextos distintos do teste de software.

SÍNTESE DOS TRABALHOS RELACIONADOS

Para consolidar a evolução dos conceitos discutidos, a Tabela 1 apresenta uma comparação entre os trabalhos analisados e a proposta desta dissertação.

Tabela 1 – Relação entre os trabalhos analisados e a proposta da Dissertação

Trabalho	Foco	Natureza da Diversidade	Papel das N-Versões	Oráculo
(HATTON, 1997)	Avaliação crítica da N-versões	Estrutural (múltiplas implementações)	Instrumento de análise empírica	Oráculo derivado
(MANOLACHE; KOURIE, 2001)	Mitigação do problema do oráculo	Estrutural e funcional (Principal + Modelo)	Mecanismo de comparação (Teste diferencial)	Oráculo derivado
(BISHOP et al., 2012)	Investigação empírica da diversidade	Estrutural (linguagens e equipes)	Instrumento para medir correlação de falhas	Oráculo derivado
(PARK et al., 2021)	Teste diferencial em larga escala	Estrutural e semântica	Teste diferencial N + 1 versão	Oráculo derivado
Dissertação	Regressão de performance	Temporal e empírica (histórico do sistema)	Redefinição do conceito de diversidade	Oráculo derivado aproximado (RNFs)

Fonte: Elaborada pela autora (2026).

3.1 CONSIDERAÇÕES FINAIS DO CAPÍTULO

Este capítulo apresentou uma análise cronológica e conceitual das principais abordagens relacionadas ao uso da Programação N-Versões e de técnicas de teste diferencial no contexto do teste de software. A partir da crítica seminal de (HATTON, 1997), que evidencia as limitações empíricas da diversidade estrutural entre implementações, observou-se uma evolução gradual das propostas no sentido de reinterpretar o papel da diversidade como mecanismo de validação. Trabalhos posteriores, como os de (MANOLACHE; KOURIE, 2001) e (BISHOP et al., 2012), reforçam a importância de tratar discrepâncias como fonte de informação diagnóstica, ao mesmo tempo em que destacam a necessidade de evidências empíricas para avaliar a efetividade da diversidade introduzida. Mais recentemente, a abordagem JEST (PARK et al., 2021) consolida essa evolução ao empregar o teste diferencial em larga escala, incorporando a especificação como entidade ativa no processo de validação. Nesse contexto, a proposta desta dissertação posiciona-se como uma continuidade natural dessa linha de pesquisa, ao deslocar

o conceito de diversidade para o domínio temporal e quantitativo do desempenho do software, oferecendo uma alternativa alinhada às limitações identificadas na literatura e adequada a cenários de evolução contínua.

4 ABORDAGEM DE ORÁCULO APROXIMADO PARA TESTES DE PERFORMANCE COM MARGEM ADAPTATIVA

As métricas de desempenho podem ser medidas de diferentes maneiras, até mesmo para a mesma métrica, e o método de medição influencia diretamente a forma como os dados são descritos e analisados. Na literatura (INGO; DALY, 2020; TRAINI et al., 2024; LEZNIK et al., 2022), são identificadas duas abordagens principais para medir as métricas de desempenho: Medição pontual e Medição de séries temporais.

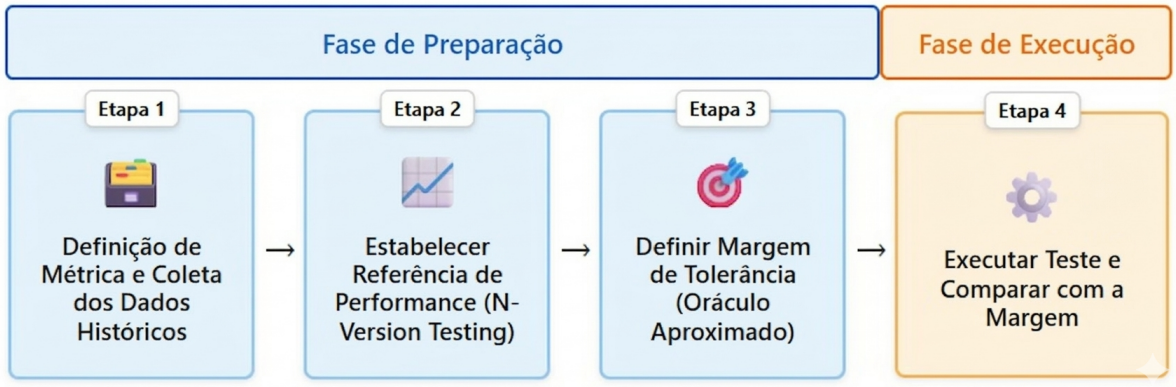
- Medição pontual: captura o valor de uma métrica em um momento específico no tempo, normalmente usado para avaliar o estado do sistema após uma ação ou evento específico. Exemplo: "O uso da memória foi de 1,3 Gigabyte (GB) logo após o arquivo ter sido carregado."

- Medição de série temporal: envolve a coleta de uma sequência de valores de métricas em intervalos regulares ou irregulares ao longo do tempo, permitindo a análise de tendências, padrões e comportamento dinâmico.

  Exemplo: "O uso da memória foi monitorado a cada segundo por 5 segundos durante o teste, resultando em um gráfico de série temporal."

Independentemente do tipo de medição (pontual ou temporal), nossa abordagem, que aproveita o teste de n-versões para definir oráculos aproximados para testes de performance, segue o mesmo processo de quatro etapas. Uma visão geral da abordagem proposta é mostrada na Figura 2.

Figura 2 – Abordagem em quatro etapas para definir oráculos aproximados de performance com base em testes de N versões



Fonte: (MARCELINO; MIRANDA, 2025)

Etapa 1 - Preparação: Definir métrica e coletar dados históricos. A métrica de desempenho de interesse é especificada e os dados históricos são coletados em n versões anteriores do SUT. O especialista em desempenho seleciona quantas versões serão incluídas (por exemplo, as últimas 10 ou 20).

Etapa 2 - Preparação: Estabelecer a referência de performance. Usando os dados históricos coletados, é calculada uma linha de base de desempenho de referência.

- Para métricas pontuais, isso pode ser um valor único (por exemplo, média ou mediana) ou um intervalo (por exemplo, valores mínimo e máximo).
- Para métricas de série temporal, é possível derivar uma curva que represente o valor médio ou mediano em cada ponto de tempo nas versões n; as curvas mínima e máxima também podem ser calculadas para representar o intervalo de comportamento em cada etapa de tempo.

Esta etapa incorpora os princípios do teste de N versões, tratando cada versão anterior como uma fonte independente de comportamento de desempenho.

Etapa 3 - Preparação: Definir margem de tolerância. Uma margem de tolerância é aplicada a referência de performance para formar um oráculo aproximado. Essa margem pode ser definida usando desvio padrão, limites fixos ou porcentagens sobre a linha de base (média, mediana, valor mínimo ou máximo). A margem especifica o desvio aceitável antes de sinalizar uma regressão de desempenho.

Etapa 4 - Execução: Executar teste e comparar resultados. O caso de teste é executado na versão atual do SUT usando o mesmo método de medição aplicado às versões anteriores. A métrica resultante é então comparada com o desempenho de referência e sua margem de tolerância:

- Se o valor observado estiver dentro da margem, presume-se que o sistema se comporte conforme o esperado.

- Se o valor estiver fora da margem, é emitido um alerta, indicando uma possível regressão de desempenho que requer investigação adicional.

A abordagem proposta neste estudo, embora detalhada em suas etapas, fundamenta-se nos princípios teóricos discutidos na revisão da literatura. A lógica de nosso oráculo é diretamente inspirada nos conceitos de redundância e comparação do *N-version testing*, mas de uma forma recontextualizada para o domínio de testes de performance. A utilização de versões históricas

do sistema para estabelecer uma linha de base é a materialização do princípio de redundância temporal. Da mesma forma, a margem de tolerância adaptativa e os cálculos matemáticos aplicados às métricas de desempenho representam a adaptação do princípio de comparação, substituindo a validação binária por uma avaliação probabilística. Ao fazer essa conexão, nossa abordagem transforma uma técnica clássica em uma solução inovadora para um problema contemporâneo, fornecendo uma base teórica sólida para o oráculo de teste aproximado e adaptativo.

Em seguida, demonstramos esse processo de quatro etapas com dois exemplos ilustrativos: um usando uma métrica pontual e outro usando uma métrica temporal. Observe que as possíveis combinações de parâmetros (ou seja, o número de versões históricas, a referência da performance e a margem de tolerância) são diversas. Essas combinações dependem de fatores como o tipo de sistema, a dinâmica do projeto e o cenário específico que está sendo testado. Portanto, enfatizamos a importância de envolver um especialista com experiência nos requisitos e nas características de desempenho do software.

4.1 EXEMPLO COM MEDIÇÃO PONTUAL

Este exemplo ilustra como a nossa abordagem de quatro etapas é aplicada ao lidar com métricas de desempenho pontuais, especificamente o uso de memória para o nosso exemplo (veja a Figura 3).

Na Etapa 1 (Definir métrica e coletar dados históricos), o uso da memória foi selecionado como a métrica de desempenho de interesse. Os dados foram coletados em um ponto específico no tempo, imediatamente após uma operação de software definida (por exemplo, carregar um arquivo), para 10 versões anteriores do SUT (Figura 3a).

Na Etapa 2 (Estabelecer a referência da performance), um valor de referência foi computado, calculando-se o uso médio de memória nas 10 versões históricas. Esse valor representa o consumo de memória esperado naquele ponto específico da operação (Figura 3b).

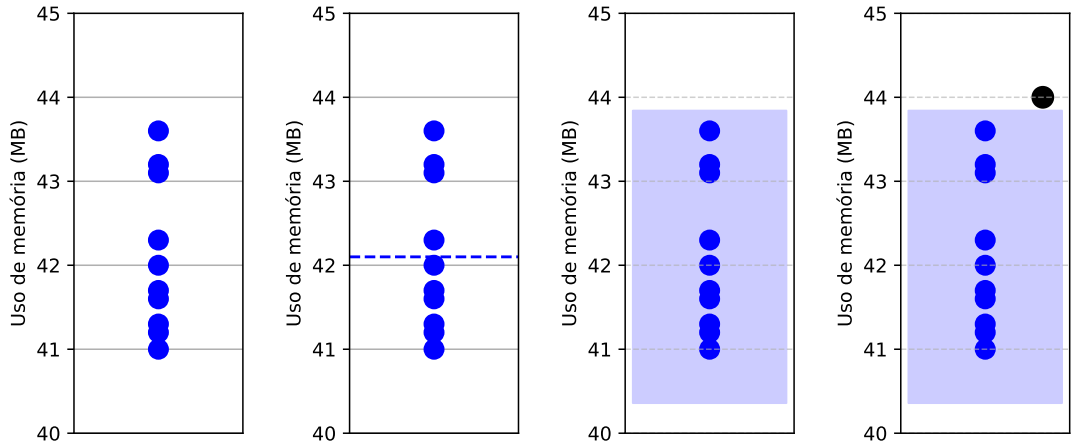
Na Etapa 3 (Definir margem de tolerância), um intervalo de tolerância foi obtido aplicando-se uma margem de dois desvios padrão acima e abaixo da média de referência. Esse intervalo define a variabilidade aceitável e atua como um oráculo aproximado de desempenho (Figura 3c).

Na Etapa 4 (Executar teste e comparar resultados), o uso da memória foi medido para a versão atual do SUT (versão 11) usando o mesmo método. O valor observado foi comparado

com o intervalo de tolerância definido anteriormente. Como o resultado excedeu o limite superior do intervalo, foi emitido um alerta, indicando uma possível regressão de desempenho no consumo de memória que justifica uma investigação mais aprofundada (Figura 3d).

Figura 3 – Exemplo da aplicação da abordagem em quatro etapas a dados de uso de memória em um ponto no tempo em 10 versões de software.

(a) Performance das versões anteriores (b) Referência da performance (c) Margem de tolerância (d) Avaliação da performance atual



A linha azul tracejada representa a média, a área sombreada indica a margem de tolerância e o ponto preto no gráfico (d) indica a performance na versão 11.

Fonte: (MARCELINO; MIRANDA, 2025)

4.2 EXEMPLO COM MEDIÇÃO DE SÉRIE TEMPORAL

Este exemplo ilustra como nossa abordagem de quatro etapas é aplicada ao lidar com métricas de desempenho de séries temporais, especificamente a utilização da CPU para o nosso exemplo (veja a Figura 4).

Na Etapa 1 (Definir métrica e coletar dados históricos), a utilização da CPU foi escolhida como a métrica de desempenho de interesse. As medições foram feitas a cada 0,5 segundo em um período de 5 segundos, em 10 versões anteriores do SUT (Figura 4a).

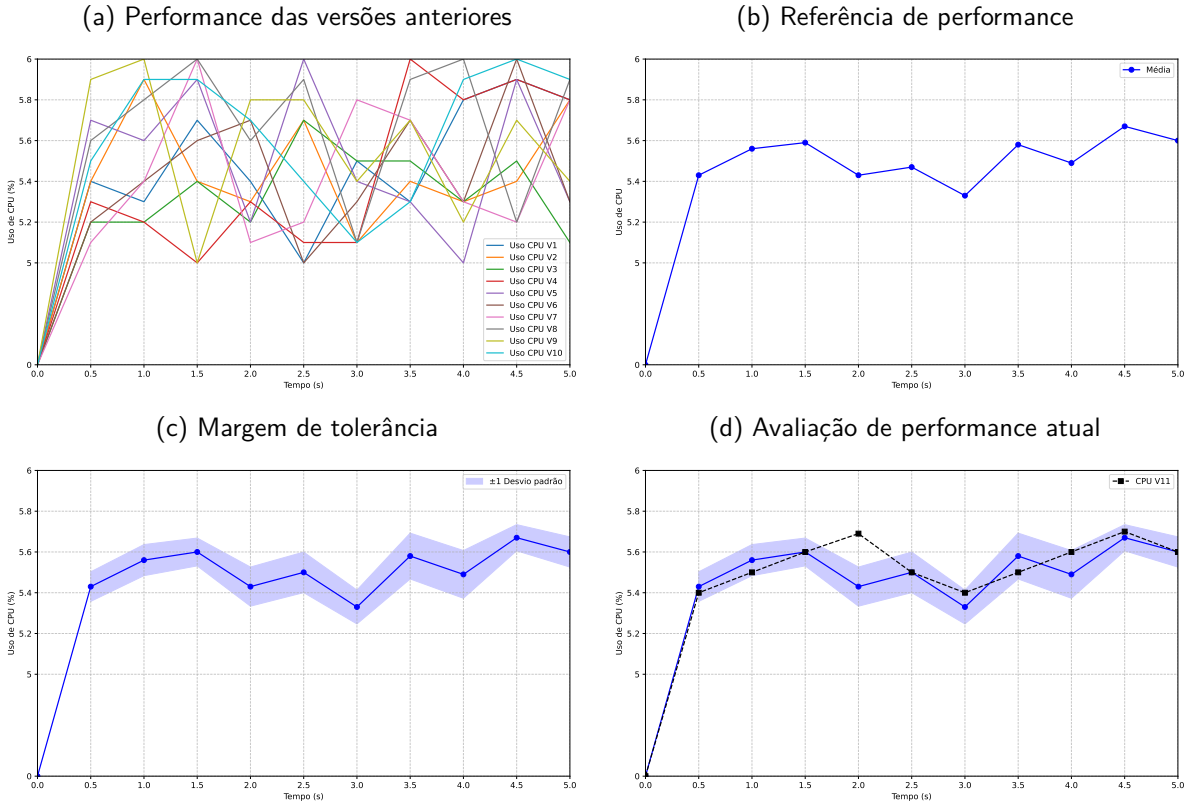
Na Etapa 2 (Estabelecer referência da performance), uma curva de referência foi computada, calculando-se o uso médio da CPU em cada intervalo de tempo nas 10 versões. Essa curva de referência representa o comportamento esperado ao longo do tempo. Embora também seja possível definir limites mínimos e máximos, neste exemplo, optamos por usar apenas a curva média como desempenho de referência (Figura 4b).

Na Etapa 3 (Definir margem de tolerância), a área de tolerância foi construída adicionando

e subtraindo 1 desvio padrão da curva de referência em cada ponto de tempo. Isso cria uma área delimitada que representa a variabilidade aceitável com base no comportamento histórico (Figura 4c).

Por fim, na Etapa 4 (Executar teste e comparar resultados), o mesmo teste foi executado na versão atual do SUT (versão 11), coletando o uso da CPU nos mesmos intervalos de tempo. A curva resultante foi comparada com a área de tolerância. Os pontos de dados que estão fora dos limites definidos indicam uma possível regressão de desempenho e sinalizam a necessidade de uma investigação mais aprofundada para determinar a causa do desvio (Figura 4d).

Figura 4 – Exemplo de aplicação da abordagem de quatro etapas aos dados de uso da CPU em séries temporais em 10 versões de software.



Fonte:(MARCELINO; MIRANDA, 2025)

4.3 JANELA MÓVEL - LIDANDO COM A EVOLUÇÃO DE SOFTWARE

Os exemplos acima ilustram o fluxo principal da abordagem, onde a definição das 3 primeiras etapas regem o quão fora do estipulado na margem está o desempenho da aplicação testada. Com o tempo, o software evolui, então as versões antigas deixam de representar o comportamento atual. Assim, o valor da margem não pode ser vitalício, ou seja, o valor obtido a partir das dez versões testadas é flexível e evolui a cada nova implementação. O histórico

de dados coletados é atualizado a cada versão que é integrada no sistema. E o cálculo da referência de performance desse novo histórico depende dos valores obtidos a cada novo teste executado. Adota-se assim o conceito de janela móvel na definição da margem de tolerância, já que com a evolução constante do software não se pode definir um valor fixo como aceitável em todo ciclo de vida da aplicação.

A proposta da janela móvel caracteriza-se da seguinte forma: a margem de tolerância (etapa 3) é definida baseada no cálculo dos valores obtidos pelo histórico de versões testadas antes da identificação da anomalia (etapa 1). A versão em que a anomalia é identificada e o erro confirmado não entra no cálculo da referência de performance (etapa 2), pois é um resultado atípico. Com a correção implementada na versão seguinte, o novo valor gerado é integrado ao histórico de dados, assim os valores da referência de performance e da margem de tolerância são atualizados para cada versão. Logo, o valor da margem de tolerância da última versão pode ser diferente do que foi calculado para penúltima versão e assim por diante.

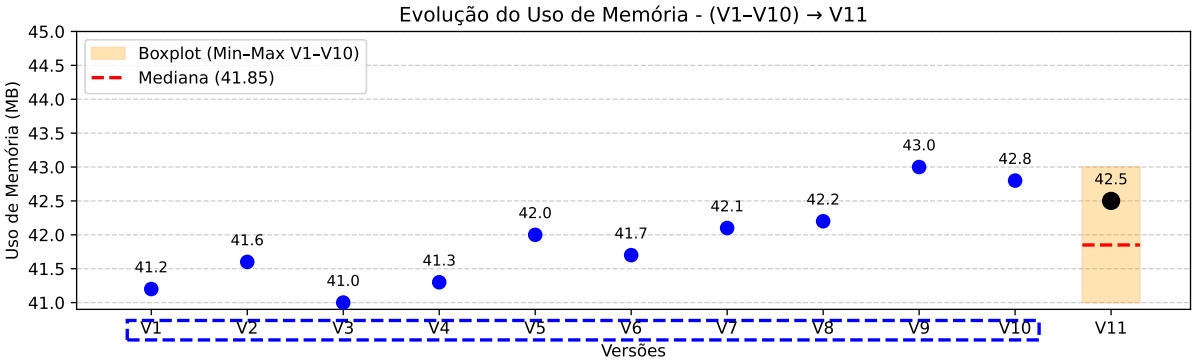
A Janela Móvel garante que apenas as versões mais recentes e estáveis sejam consideradas na baseline. E tem como objetivo acompanhar a evolução do software a cada versão entregue para testes e identificar possíveis pontos de regressão de performance. O acompanhamento dos resultados para cada versão testada possibilita um monitoramento consistente e pode antecipar potenciais riscos. Vide ilustração (Figura 5) para compreensão da janela móvel.

Dado os seguintes parâmetros: Histórico de dados = 10 versões; Referência de Performance = cálculo da mediana do histórico de dados; Margem de tolerância= Valores mínimo e máximo do histórico de dados.

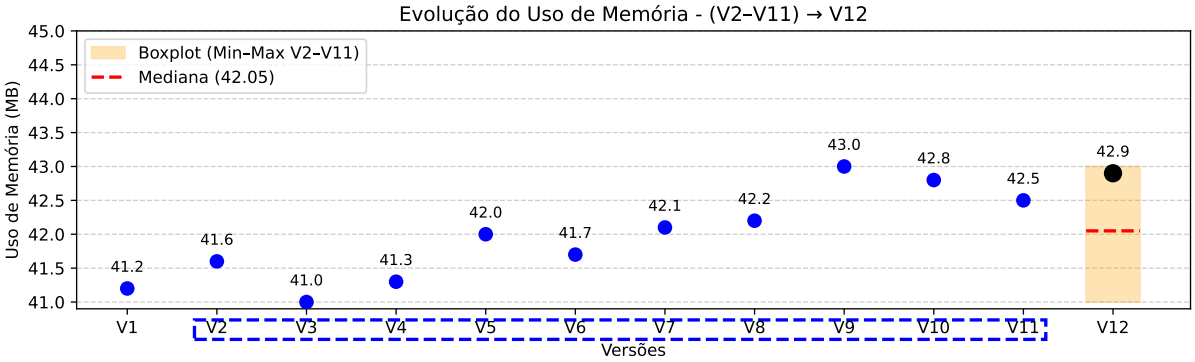
As Figuras 5a, 5b, 5c e 5d mostram os valores de consumo de memória coletados em 11 versões diferentes de um software na execução de uma mesma ação. Onde o valor da última versão é comparado com a margem de tolerância estimada de acordo com os resultados das 10 versões anteriores.

Figura 5 – Janela móvel do consumo de memória.

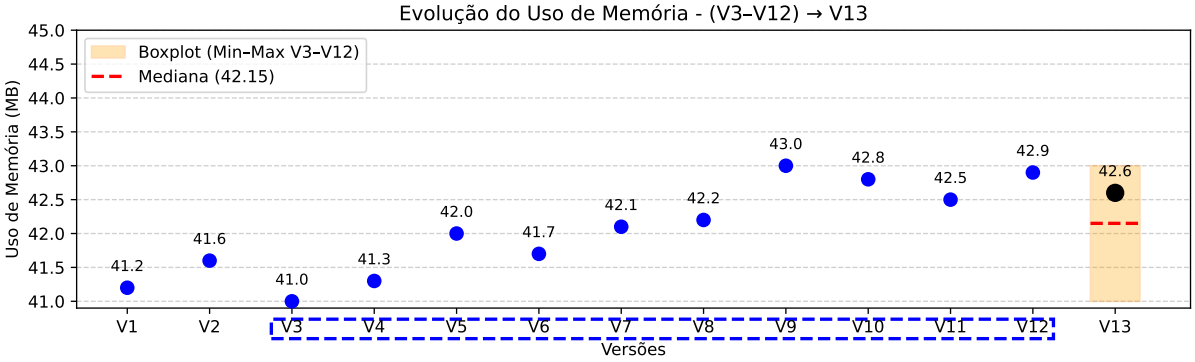
(a) Consumo de memória entre as versões 1 a 10 e comparação da versão 11.



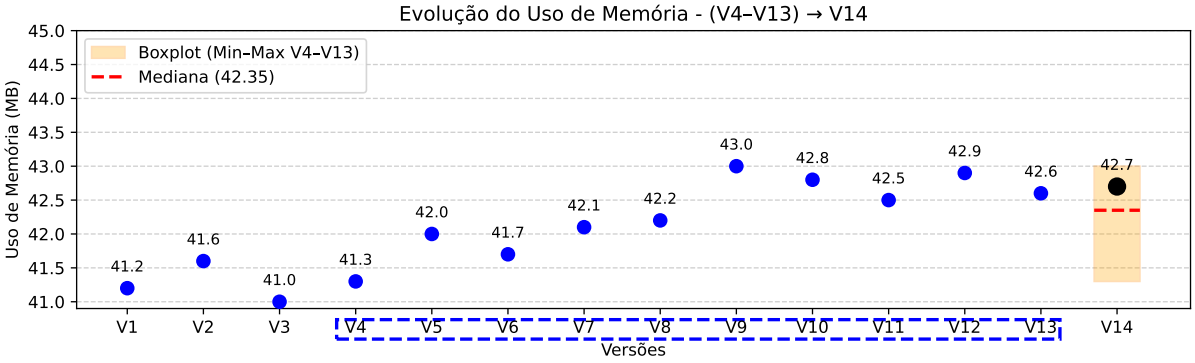
(b) Consumo de memória entre as versões 2 a 11 e comparação da versão 12.



(c) Consumo de memória entre as versões 3 a 12 e comparação da versão 13.



(d) Consumo de memória entre as versões 4 a 13 e comparação da versão 14.



Fonte: Elaborada pela autora (2026)

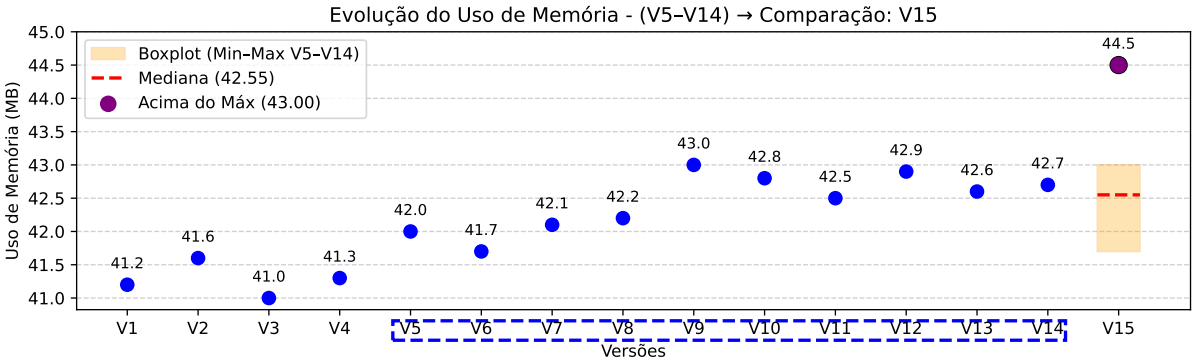
O contexto da janela móvel propõe uma margem de tolerância adaptativa a cada nova versão, ou seja, ela evolui juntamente com o software, proporcionando um controle contínuo sobre os resultados obtidos. Se o resultado da versão atual estiver dentro da margem, ele é considerado como um comportamento aceitável e integrará o histórico de dados para a próxima versão, caso contrário uma análise mais detalhada deve ser realizada para entender a modificação de comportamento.

4.4 JANELA MÓVEL - LIDANDO COM A PRESENÇA DE VERSÕES COM ERROS

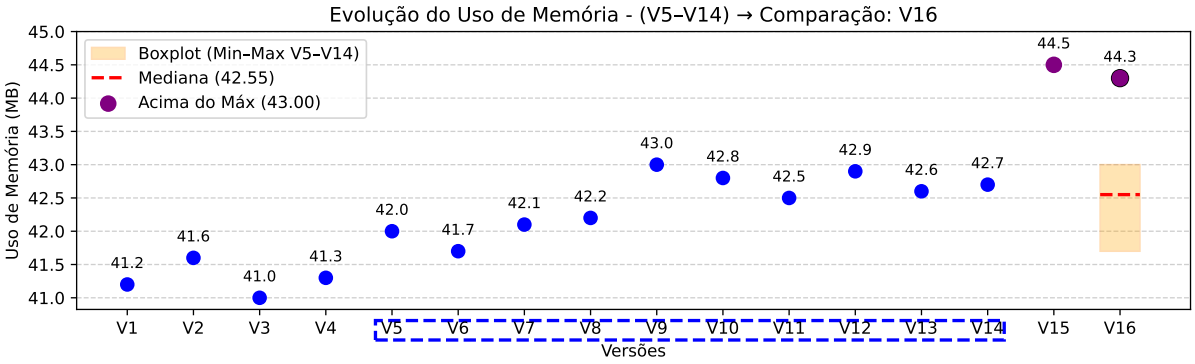
A janela móvel possibilita um monitoramento atualizado da evolução do software e dos impactos sofridos pela inserção/remoção/edição de código. Esse acompanhamento em tempo real facilita a identificação de anomalias e suas possíveis causas. Caso a versão atual apresente um resultado fora da margem estipulada, a equipe de desenvolvimento é acionada para analisar o cenário, caso a análise confirme que esse resultado caracteriza erro, essa versão não entra no cálculo da janela móvel, como ilustram as subfiguras (6a, 6b, 6c e 6d) da Figura 6:

Figura 6 – Confirmação de erro de performance nas versões 15, 16, 17 e 18.

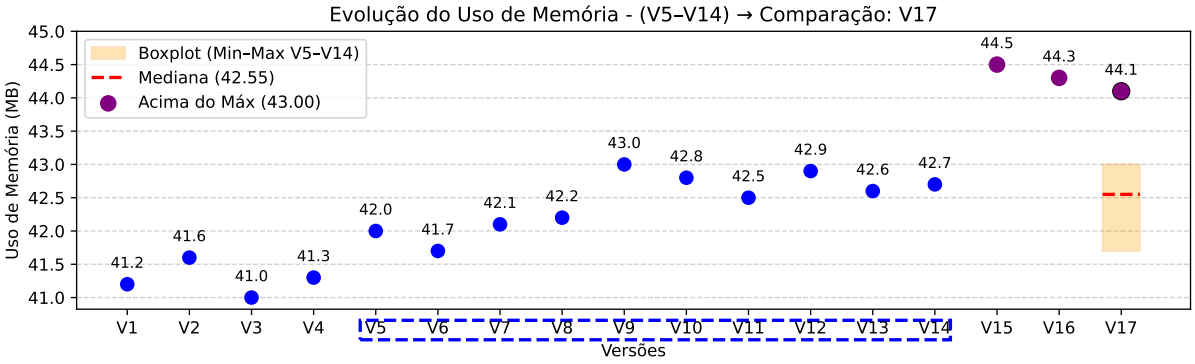
(a) Confirmação de erro de performance na versão 15.



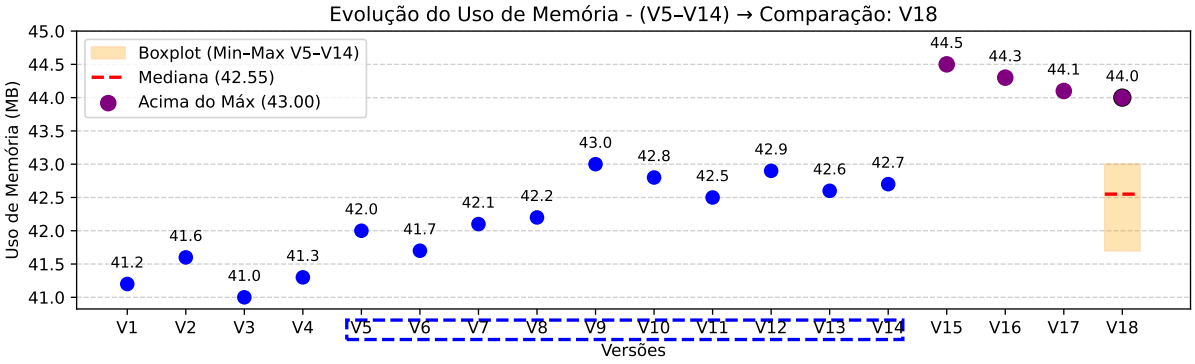
(b) Confirmação de erro de performance na versão 16.



(c) Confirmação de erro de performance nas versões 17.



(d) Confirmação de erro de performance nas versões 18.



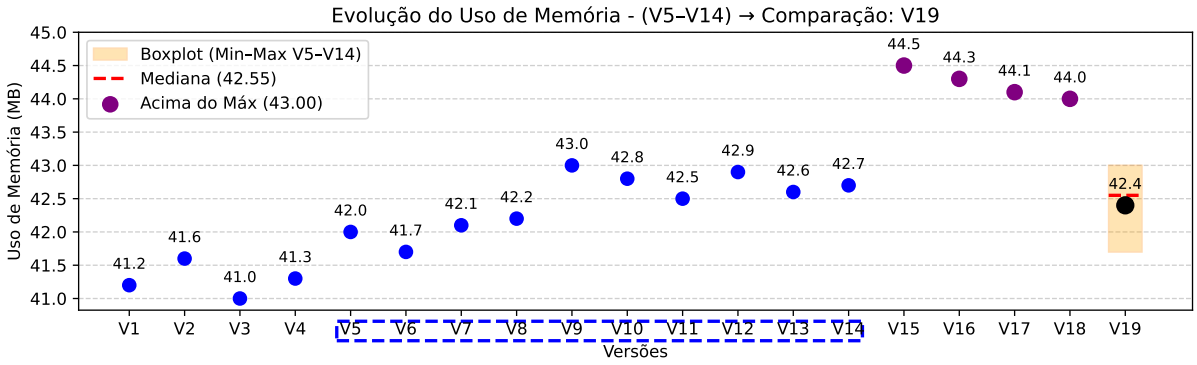
Fonte: Elaborada pela autora (2026)

Quando um erro é identificado, a margem de tolerância permanece sendo o último intervalo considerado estável, no exemplo da Figura 6 consideramos o intervalo entre as versões 5 a 14 como correto, e a margem resultante do cálculo entre essas versões (limite mínimo e máximo) é a referência para as versões seguintes. Os resultados com erros não serão utilizados para definição de uma nova margem de tolerância, pois a introdução desse valor no cálculo geraria uma margem inapropriada. O cálculo da margem deve ser retomado após correção das inconsistências, como mostra a Figura 7.

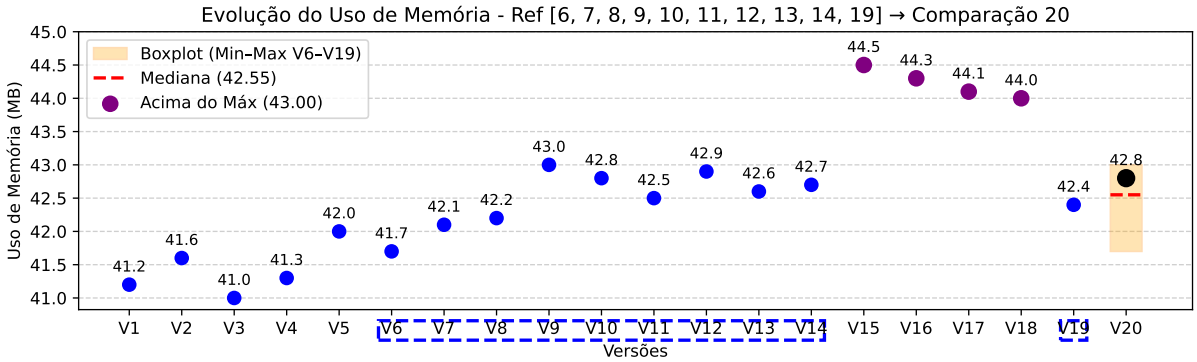
Percebe-se que a correção foi implementada na versão 19 (Figura 7a). A partir dessa confirmação, a versão 19 entra nos dados da janela móvel. Assim, para definição da margem, passa a ser contabilizada da versão 6 até a versão 19 (exceto as versões 15, 16, 17 e 18 que apresentaram erros), como ilustra a Figura 7b:

Figura 7 – Correção de inconsistência e retomada do cálculo da janela móvel.

(a) Correção da inconsistência de performance na versão 19.



(b) Retomada do cálculo da janela móvel onde a versão 20 é comparada com a margem definida entre as versões 6 a 14 + versão 19.



Fonte: Elaborada pela autora (2026)

A aplicação da janela móvel também evita a ocorrência de falsos-positivos (quando um comportamento aceitável é erroneamente sinalizado como regressão) ou falsos negativos (quando uma regressão real não é detectada), pois se apenas as primeiras versões dos dados históricos

fossem consideradas para definir a margem de tolerância de todas as versões subsequentes, ela ocultaria a evolução do software e trataria qualquer valor fora da margem definida como um erro.

Assim, na abordagem proposta, os falsos positivos são mitigados através da margem de tolerância adaptativa. Ao invés de uma comparação binária rígida, a abordagem utiliza um intervalo númerico (por exemplo, baseado no desvio padrão) para definir o que é considerado um comportamento aceitável. Isso permite que o oráculo ignore a variabilidade natural e aleatória que ocorre nas medições de desempenho, garantindo que apenas desvios significativos sejam sinalizados como anomalias. Já a detecção de falsos negativos é minimizada pela adoção de uma perspectiva holística sobre o desempenho. Em vez de se basear em uma única métrica, a abordagem monitora e compara um conjunto de métricas não-funcionais correlacionadas, como tempo de resposta, uso de CPU e consumo de memória. Conforme demonstrado no estudo, uma regressão pode ser sutil em uma métrica (como um pequeno aumento no tempo de resposta) mas evidente em outra (como um pico no consumo de memória). A combinação da análise de múltiplas métricas aumenta a probabilidade de capturar uma regressão que poderia passar despercebida por um oráculo unidimensional.

Por fim, acreditamos que o conceito de janela móvel reflete um escopo mais adequado aos softwares de integração contínua. Calcular as métricas a cada nova implementação, além de manter a margem de tolerância atualizada, permite um acompanhamento evolutivo do impacto das novas implementações nas funcionalidades existentes.

Seguimos, no próximo capítulo, com os testes práticos e confirmamos a eficiência da abordagem de janela móvel na detecção de anomalias de performance.

5 AVALIAÇÃO DO ESTUDO DE CASO COM MEDIÇÃO PONTUAL

Nesta seção apresentamos os resultados obtidos nas avaliações de medição pontual aplicadas ao software escolhido, bem como a interpretação de seus resultados.

5.1 PRIMEIRA AVALIAÇÃO COM MEDIÇÃO PONTUAL

O sistema utilizado no estudo de caso foi um software proprietário de edição de imagens. Esse software foi selecionado por fazer parte do ecossistema industrial ao qual a autora deste trabalho faz parte. A aplicação apresenta uma arquitetura modular composta por diferentes subsistemas funcionais responsáveis pelo gerenciamento, processamento e visualização de mídias digitais. Embora a proprietária da aplicação não disponibilize informações oficiais sobre a arquitetura interna ou quantidade de linhas de código da aplicação, observa-se que sua complexidade funcional envolve múltiplos componentes especializados, processamento assíncrono, gerenciamento de metadados e integração com serviços externos, evidenciando uma estrutura arquitetural compatível com aplicações multimídia modernas de médio a grande porte.

Para simular os cenários de regressão de performance, foram desenvolvidos quatro scripts automatizados em Python, cada um representando um cenário de erro previamente identificado pela equipe de testes. Os scripts executavam ações controladas no aplicativo com o objetivo de reproduzir, de forma padronizada, os comportamentos associados às degradações analisadas. Os experimentos foram realizados em um ambiente com sistema operacional Windows 11, 256 GB de armazenamento, 8 GB de memória RAM, placa de vídeo de 128 MB e processador Intel Core i7, configuração compatível com as máquinas utilizadas no ambiente real de testes da organização.

Ao todo, foram coletadas 40 versões de desenvolvimento do software no período entre agosto de 2024 e agosto de 2025. A coleta foi realizada de forma incremental, acompanhando a disponibilização de novas versões após alterações no código-fonte. Dessa forma, o intervalo entre as coletas variava de acordo com o fluxo de desenvolvimento do projeto, podendo ocorrer múltiplas coletas no mesmo dia ou intervalos maiores entre versões consecutivas, conforme a liberação de novas implementações pela equipe de desenvolvimento. Dentre as versões coletadas, foram identificadas duas falhas referentes ao acesso a duas funcionalidades diferentes do sistema. Os dois erros estão relacionados ao tempo de resposta, sendo aplicadas métricas

diferentes na definição da margem de tolerância do Oráculo. O intuito em estabelecer métricas diferentes é deixar claro que é o especialista do software o calibrador dos parâmetros e que as métricas são adaptativas ao contexto testado.

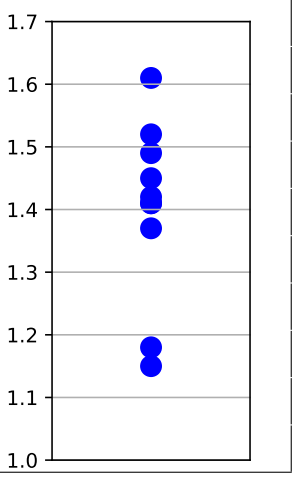
O 1º erro foi registrado na ferramenta de gerenciamento, relatando um *delay* entre o clique e o acesso à tela de edição da imagem. No registro, o tempo de resposta estava em torno de 15 segundos. Subjetivamente, o especialista que relatou o erro definiu como comportamento esperado que o tempo correto seria entre 2 e 3 segundos.

Como não há requisitos de performance definidos no projeto, o resultado esperado é considerado como estimativa. Assim, iniciamos a exploração de nossa abordagem neste cenário, aplicando métricas técnicas para comprovar se a estimativa do especialista estava coerente.

Definimos que as 10 versões anteriores a versão defeituosa seria uma coletânea histórica interessante para inferirmos métricas estáveis e realistas. Consideramos também que a referência de performance seria calculada pela média dos 10 valores obtidos. E concluímos que 2 desvios padrão para cima, em cima do valor da média, resultaria em uma margem de tolerância aceitável.

A Tabela 2 mostra os resultados obtidos nos 10 testes iniciais e ao lado o gráfico com as medições pontuais.

Tabela 2 – Tempo de Resposta por Versão.

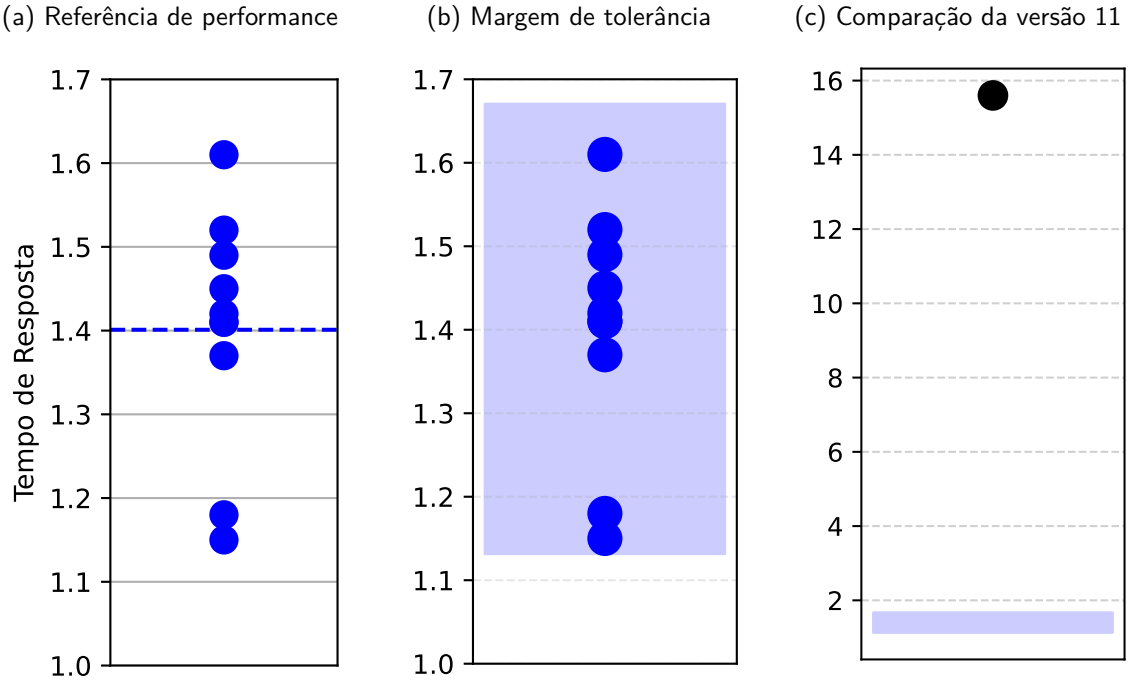
Versões	**Tempo de Resposta**	**Gráfico**
Versão 1	1.15 s	
Versão 2	1.37 s	
Versão 3	1.52 s	
Versão 4	1.45 s	
Versão 5	1.61 s	
Versão 6	1.41 s	
Versão 7	1.49 s	
Versão 8	1.41 s	
Versão 9	1.42 s	
Versão 10	1.18 s	

Fonte:(MARCELINO; MIRANDA, 2025)

Assim, a referência de performance desses valores é de 1.40 (média)(Figura 8a). O cálculo do desvio padrão é de 0.14. Em seguida, calculando a média mais 2 desvio padrão para cima (1.40 + 2* 0.14) obtivemos a margem de tolerância 1.68 (Figura 8b). A partir desse cálculo,

entende-se que todo resultado igual/menor do que 1.68 é considerado como comportamento aceitável. Por fim, comparamos o resultado da versão 11 (com erro) com a margem definida (figura 8c), como mostra a Figura 8.

Figura 8 – Definição da Margem de tolerância ao tempo de resposta em uma aplicação real com erro de performance conhecido.



Fonte: (MARCELINO; MIRANDA, 2025)

Embora a margem de tolerância obtida automaticamente esteja próxima ao valor empiricamente definido pelo especialista (situado entre 2 e 3 segundos), o resultado da avaliação 4.1 evidencia a relevância de fundamentar tais limites em métricas tecnicamente objetivas. A abordagem proposta utiliza dados históricos reais para estimar a variabilidade natural do sistema, permitindo a definição de uma margem de tolerância consistente mesmo na presença de valores atípicos (*outliers*). Esse aspecto é particularmente importante, pois reduz a dependência de julgamentos subjetivos e evita a adoção de limiares excessivamente permissivos ou restritivos. Como consequência, o oráculo aproximado torna-se mais consistente na distinção entre variações normais de desempenho e degradações reais, mitigando a ocorrência tanto de falsos positivos quanto de falsos negativos na identificação de regressões de performance.

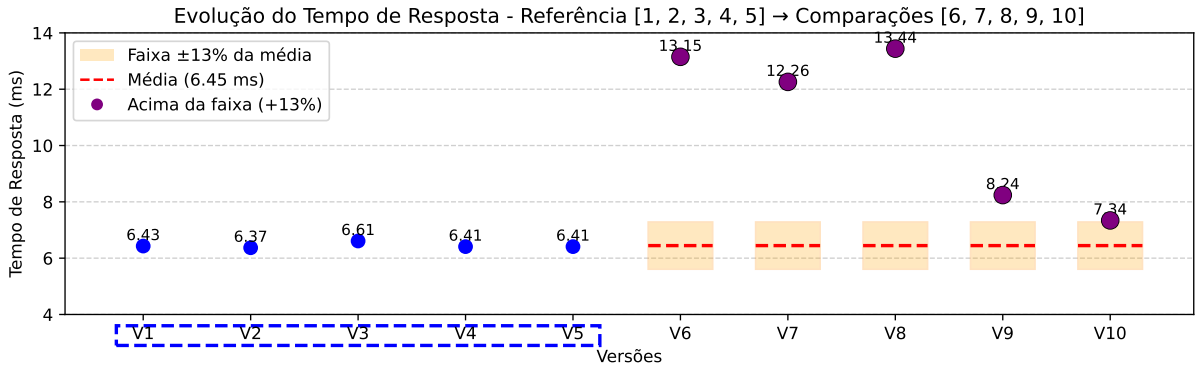
5.2 SEGUNDA AVALIAÇÃO COM MEDIÇÃO PONTUAL

O segundo teste também derivou de um registro de erro de performance relacionada ao tempo de resposta para acessar uma funcionalidade do sistema. Nesta, o especialista relatou que estava levando 13 segundos para acessar uma funcionalidade de aplicação de efeitos, que antes respondia entre 5/6 segundos.

Neste caso, para compor a margem de tolerância, consideramos que 5 versões seriam suficientes para representar os dados históricos. A referência de performance permaneceu sendo a média entre esses resultados. Porém, a margem de tolerância será calculada considerando 13% para mais e para menos da média.

Neste cenário, conseguimos acompanhar todas as versões que apresentaram o erro, ou seja, 5 versões foram consideradas aptas para constituir os dados históricos, o erro foi identificado na 6ª versão e continuou ocorrendo até a 10ª versão. Os parâmetros combinados nesse cenário confirmaram que a nossa abordagem conseguiu identificar o erro em todas as versões defeituosas. Como ilustra a Figura 9:

Figura 9 – Versões fora da margem de tolerância

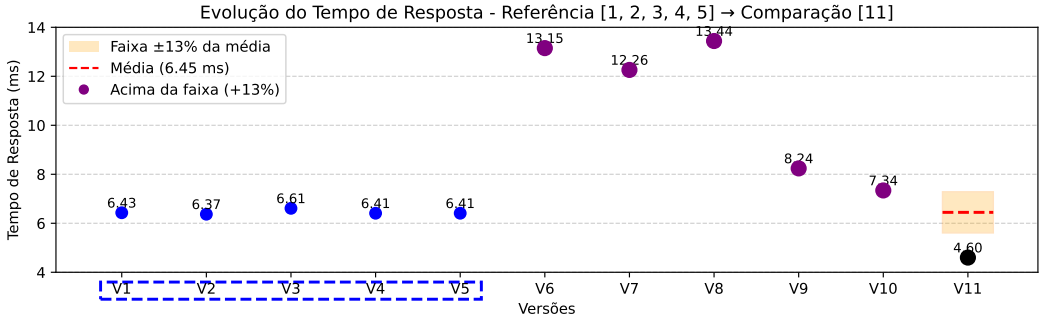


Fonte: Elaborada pela autora (2026)

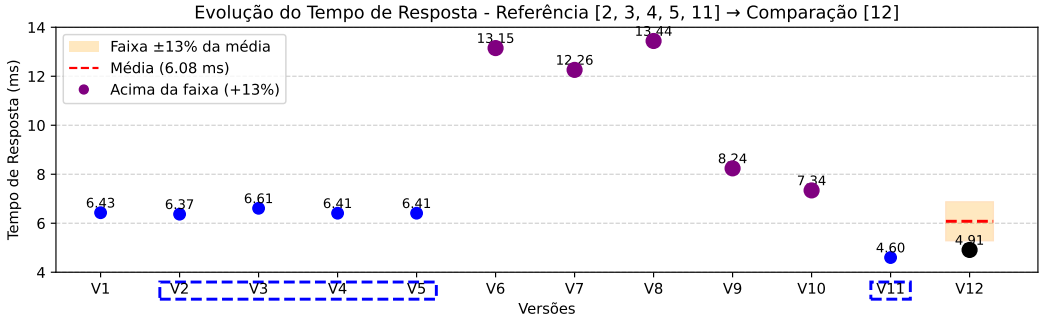
A correção foi implementada na 11ª versão, e o tempo de resposta performou melhor do que os valores dos dados históricos. Após a confirmação de que o erro foi corrigido, retomou-se a utilização da janela móvel para definição da margem de tolerância para as versões seguintes (Figura 10).

Figura 10 – Correção de erro de performance a partir da versão 11

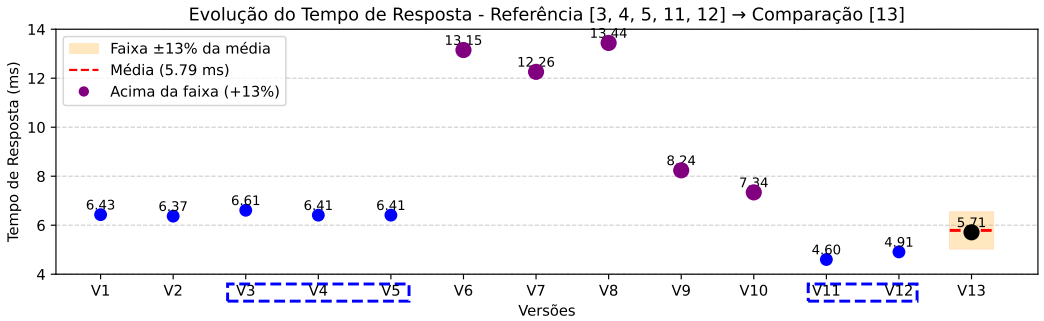
(a) Correção do erro de performance na versão 11.



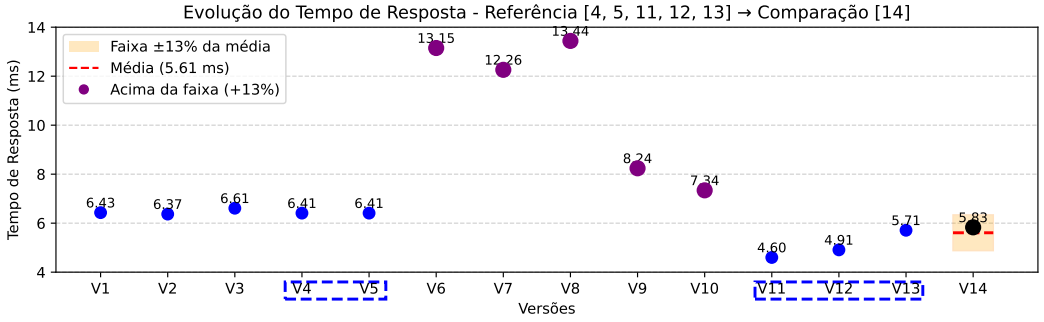
(b) Comportamento da versão 12 perante a margem de tolerância.



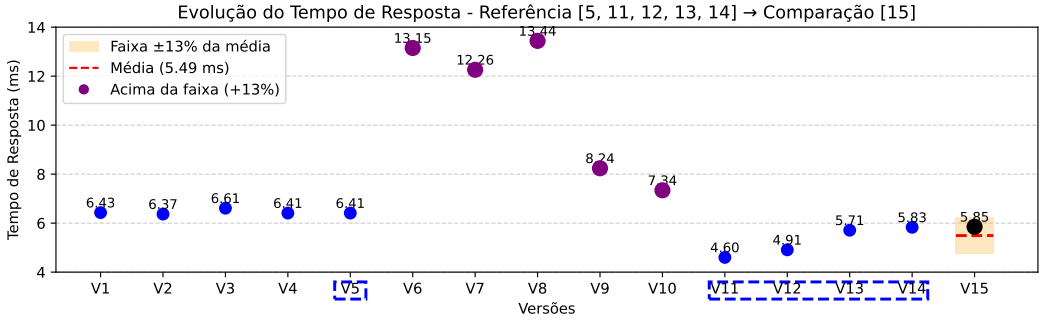
(c) Comportamento da versão 13 perante a margem de tolerância.



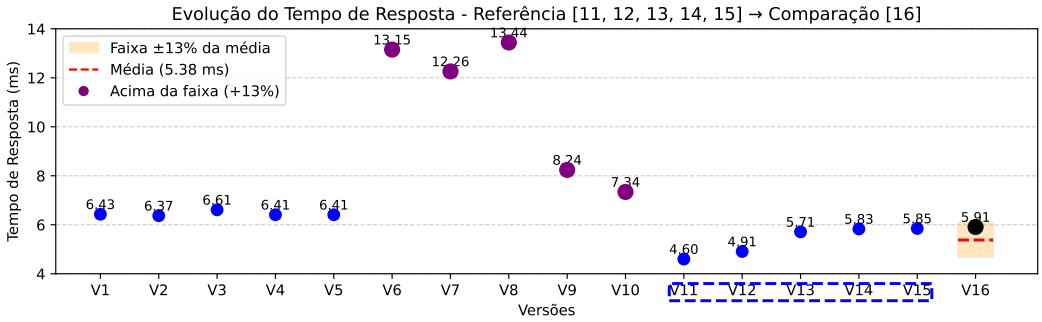
(d) Comportamento da versão 14 perante a margem de tolerância.



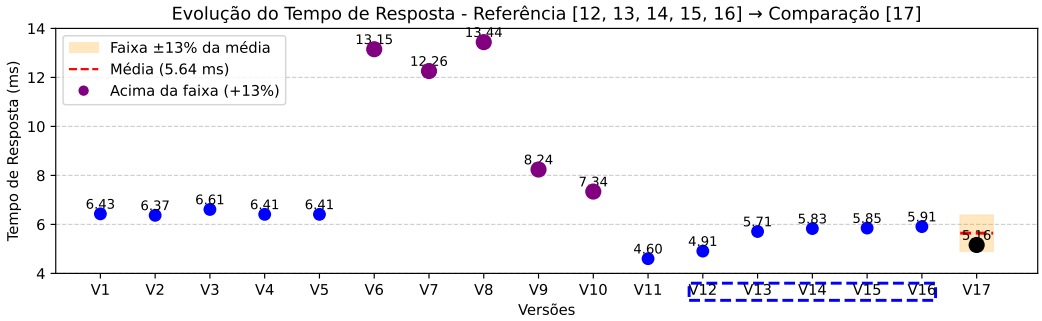
(e) Comportamento da versão 15 perante a margem de tolerância.



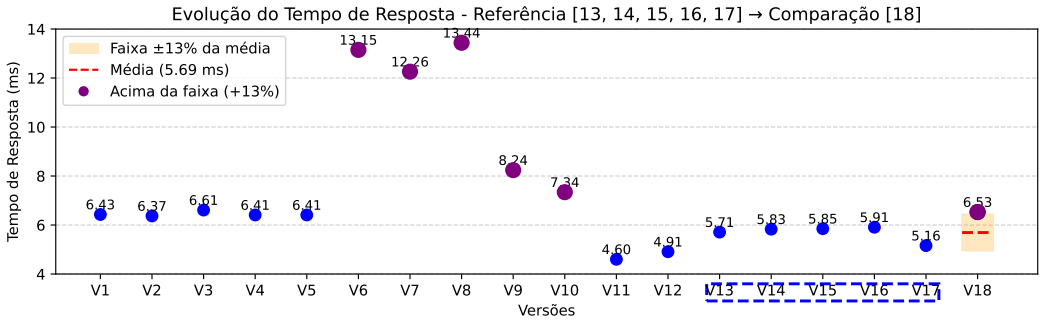
(f) Comportamento da versão 16 perante a margem de tolerância.



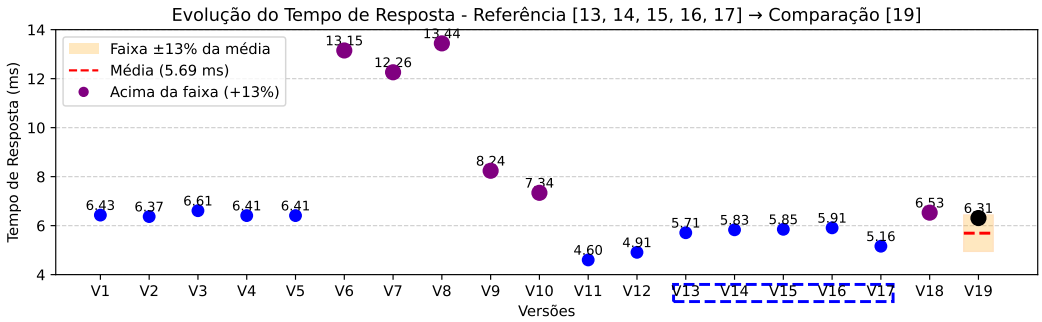
(g) Comportamento da versão 17 perante a margem de tolerância.



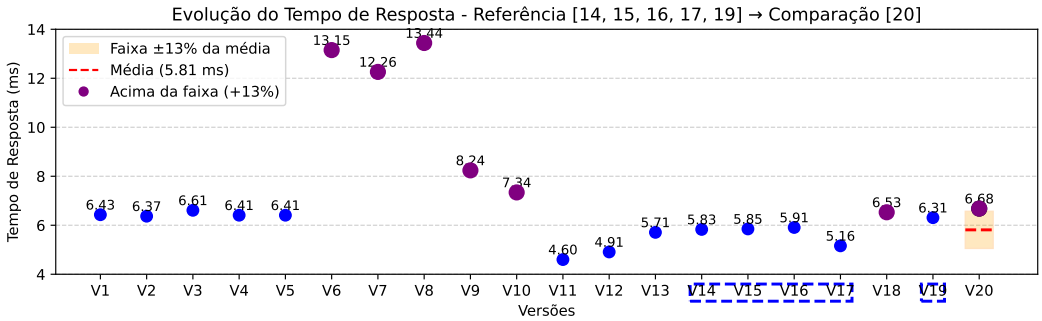
(h) Comportamento da versão 18 perante a margem de tolerância.



(i) Comportamento da versão 19 perante a margem de tolerância.



(j) Comportamento da versão 20 perante a margem de tolerância.



Fonte: Elaborada pela autora (2026)

Com o objetivo de realizar uma análise longitudinal e abrangente do comportamento do sistema, a evolução do software foi monitorada e avaliada em todas as versões coletadas, estendendo-se até a versão 20. O foco inicial deste exame recai sobre a variação dos resultados em relação ao limite de tolerância estabelecido, particularmente no período subsequente a uma intervenção corretiva de performance. Observa-se que, imediatamente após a correção, as duas primeiras iterações (V11: 4.60 ms e V12: 4.91 ms, representadas pelas Figuras 10a e 10b apresentaram um tempo de resposta que se situou abaixo do limite inferior de tolerância (5.05 ms). Este resultado configura um ganho substancial de performance, indicando que a correção implementada resultou em um desempenho significativamente superior ao baseline histórico. Não obstante, a partir da V13 (Figuras 10c, 10d, 10e, 10f e 10g) verificou-se uma estabilização e convergência dos tempos de resposta, os quais passaram a se situar, de forma consistente, dentro da margem de variação numericamente aceitável, denotando a eficácia inicial da correção implementada no restabelecimento do baseline de desempenho.

Contudo, a análise detalhada das iterações mais recentes revela uma instabilidade de desempenho que merece inspeção. Observa-se, de forma notável, uma instabilidade de desempenho no range de versões V18 a V20. Especificamente, a V18 e a V20 (Figuras 10h e 10j) demonstraram um desvio no tempo de resposta, resultando em anomalias de performance que extrapolam a faixa de tolerância estabelecida (média $\pm 13\%$). Em contrapartida, a V19 (Figura 10i) manteve-se conforme o esperado, situando-se dentro dos limites aceitáveis. Este padrão de oscilação inter-versões, caracterizado pela recorrência de um evento de regressão de performance na V20 após um período de estabilidade na V19, corrobora a capacidade e sensibilidade da abordagem proposta em identificar e sinalizar proativamente essas anomalias. O aspecto mais crucial reside no fato de que as regressões detectadas nas versões V18 e V20, apesar de sutis e de magnitudes próximas ao baseline histórico, não foram reportadas no ambiente de produção do projeto real. Tal ausência de percepção pela equipe reforça o pressuposto de que a discrepância no desempenho não atingiu o limiar de detecção humana ou de monitoramento convencional. A identificação precisa dessas flutuações pela nossa metodologia evidencia seu alto valor preditivo e efeito preventivo, permitindo o diagnóstico antecipado de degradações de performance antes que estas evoluam para estados de criticidade que comprometam a qualidade de serviço e a experiência do usuário.

5.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

As avaliações apresentadas neste capítulo tiveram como objetivo avaliar, em cenários de medições pontuais, a capacidade da abordagem proposta de identificar regressões de desempenho a partir de dados históricos e margens de tolerância adaptativas. A utilização de versões anteriores como baseline permitiu estabelecer limites quantitativos para o comportamento esperado do sistema, transformando estimativas subjetivas dos especialistas em critérios mensuráveis de avaliação. Nos dois cenários analisados, a abordagem identificou corretamente as versões que apresentaram degradação no tempo de resposta, evidenciando sua eficácia na detecção de regressões e sua flexibilidade na definição de parâmetros de tolerância conforme o contexto do sistema avaliado.

6 AVALIAÇÃO DO ESTUDO DE CASO COM MEDIÇÃO DE SÉRIE TEMPORAL

Nesta seção apresentamos dois resultados obtidos nas avaliações de séries temporais aplicadas ao software escolhido, bem como a interpretação de seus resultados.

6.1 PRIMEIRA AVALIAÇÃO COM MEDIÇÃO DE SÉRIE TEMPORAL

Neste cenário foi relatado um problema de performance ao navegar consecutivamente entre as imagens na funcionalidade de visualização de mídias. O relator percebeu que após navegar alguns minutos entre as mídias e parar, o consumo de memória não declinou.

Seguindo a premissa do levantamento dos dados históricos das versões para estabelecer um valor referencial da performance e posteriormente definir uma margem de tolerância para este cenário, utilizamos 10 versões estavéis como base histórica e comparamos o resultado da 11ª (com erro de performance confirmado) sob a margem definida.

Originalmente, o cenário de erro foi relatado destacando 3 pontos de verificação do sistema (Antes de iniciar a navegação -> Depois de concluir a navegação -> 1 minuto após o fim da navegação). Porém, ao analisar o escopo de teste, percebemos a relevância em acrescentar mais 1 ponto de verificação. E assim, acrescentamos ao nosso script a verificação de memória no momento em que as imagens eram clicadas. A nossa sequência para reprodução do erro ficou assim: Antes de iniciar a navegação -> No início da navegação -> Depois de concluir a navegação -> 1 minuto após o fim da navegação. Conforme descrito nos seguintes passos:

1. Abrir o visualizador de mídias;
2. Executar 2 cliques por segundo, durante 30 segundos, na seta que alterna a visualização de imagens no visualizador;
3. Finalizar a iteração no 30º segundo;
4. Aguardar 1 segundo para iniciar a coleta do consumo de memória (O valor da memória foi captado 6 vezes até que 1 minuto de espera fosse atingido).

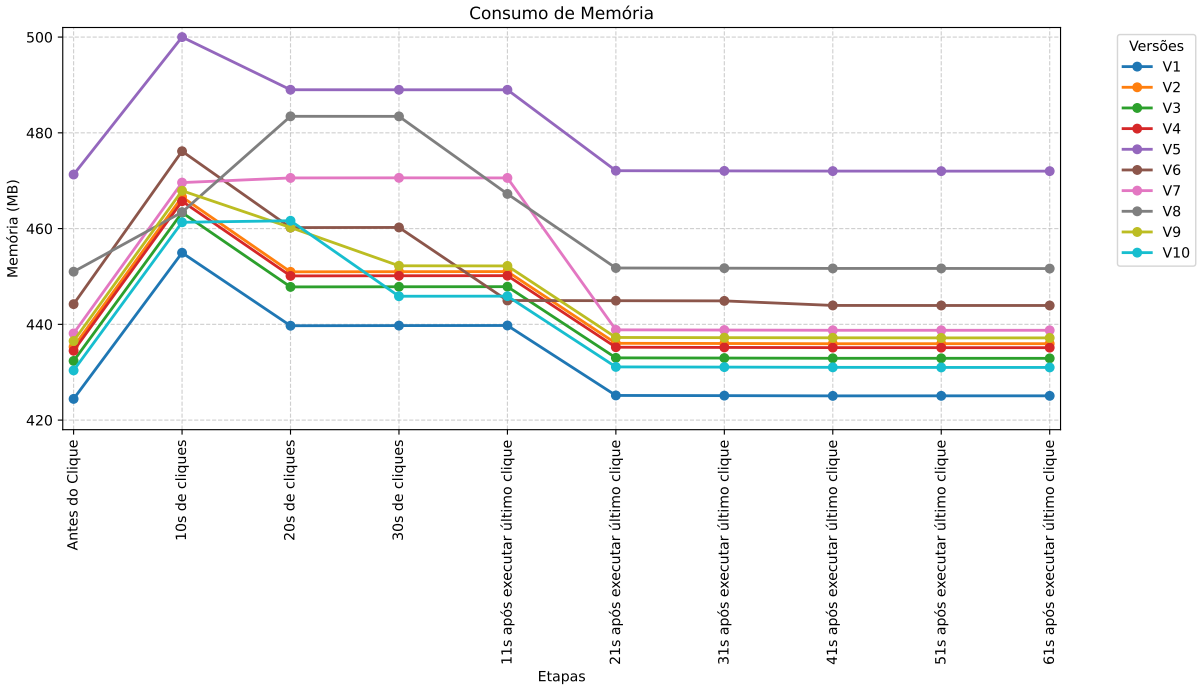
Assim, estruturamos as verificação de consumo de memória em 10 pontos:

- Com a tela do visualizador aberta, mas sem iniciar a navegação entre as mídias;

- Após 10 segundos de cliques;
- Após 20 segundos de cliques;
- Após 30 segundos de cliques; (Espera 1 segundo após fim da iteração.)
- Capta a memória após 11 segundos do fim da interação;
- Capta a memória após 21 segundos do fim da interação;
- Capta a memória após 31 segundos do fim da interação;
- Capta a memória após 41 segundos do fim da interação;
- Capta a memória após 51 segundos do fim da interação;
- Capta a memória após 61 segundos do fim da interação;

Os valores obtidos neste teste estão representados na Figura 11:

Figura 11 – Consumo de memória entre as 10 versões avaliadas durante 1 minuto.

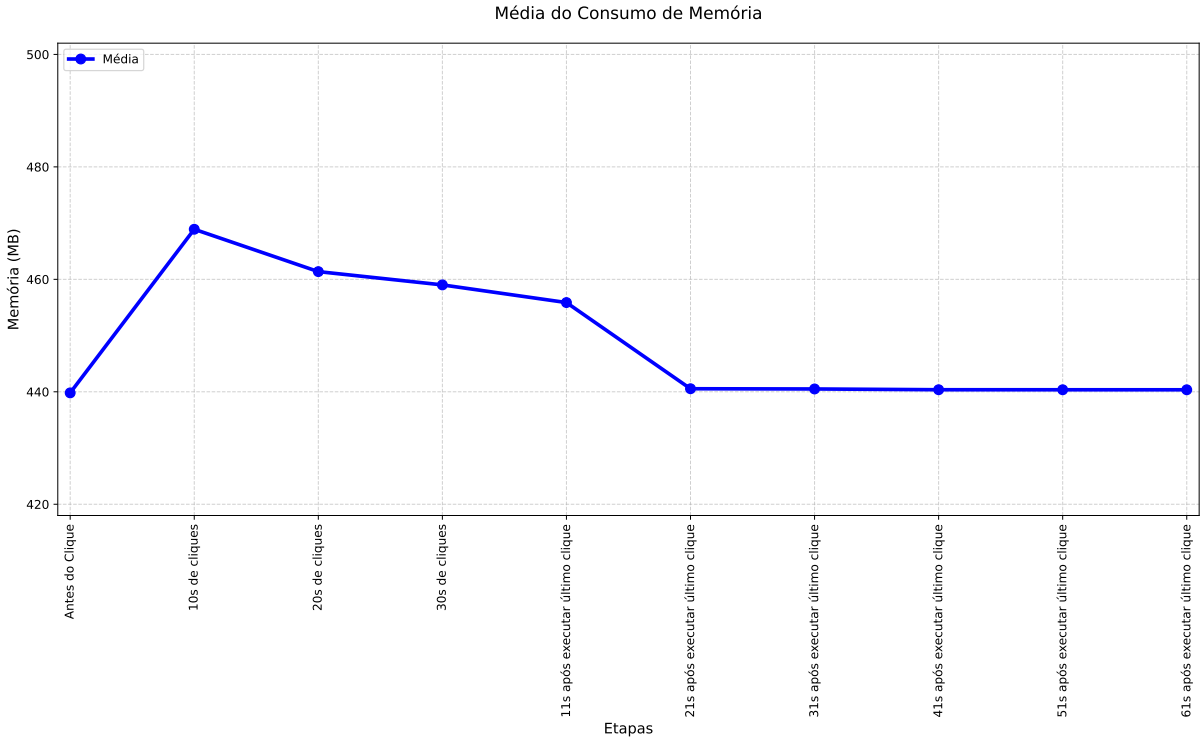


Fonte: Elaborada pela autora (2026)

Nesta ilustração, percebe-se um aumento no consumo de memória enquanto as etapas '10s de cliques', '20s de cliques' e '30s de cliques' são executadas. Já ao fim dos 11 segundos de espera, após finalizar a ação, o consumo começa a cair, aproximando-se dos valores registrados antes de iniciar os cliques.

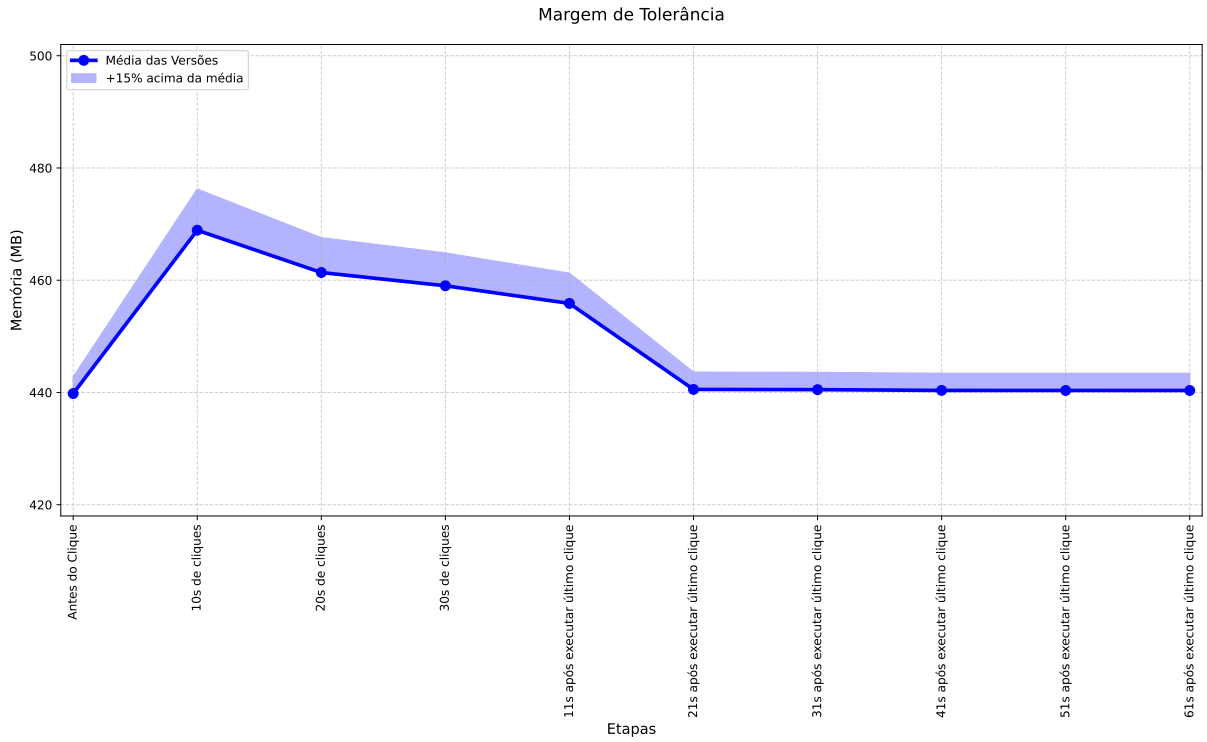
Em seguida, calculou-se a referência de performance dos 10 resultados de cada versão (Figura 12) e definiu-se a margem de tolerância, considerando 15% sobre a média de cada etapa (Figura 13):

Figura 12 – Definição da referência de performance



Fonte: Elaborada pela autora (2026)

Figura 13 – Definição da margem de tolerância, considerando 15% sob a referência de performance.

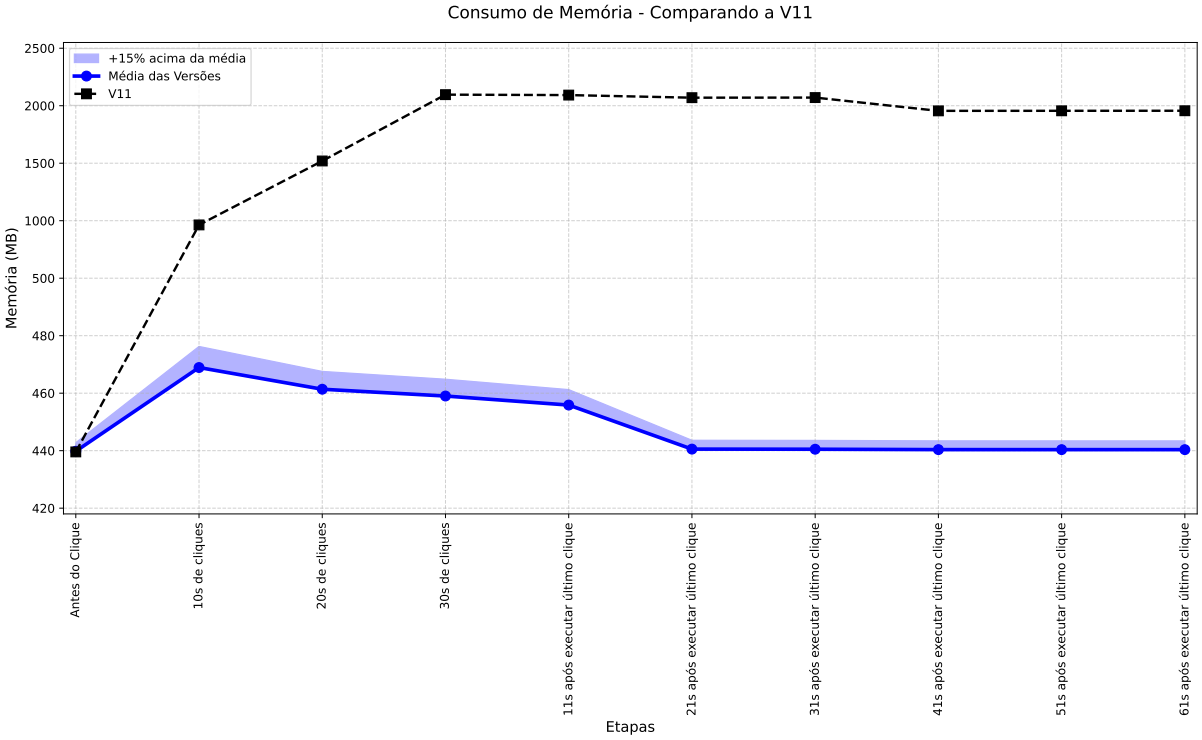


Fonte: Elaborada pela autora (2026)

Na definição da refêrencia de perfomance (Figura 12) , percebemos que o ápice de consumo de memória acontece na etapa 2 (10s de cliques) e se mantém maior do que o ponto inicial nas etapas seguintes (‘20s de cliques’ e ‘30s de cliques’), esse aumento é aceitável já que é o período de interação com a aplicação. Após isso o consumo começa a declinar e retorna ao mesmo valor que antecede o início da interação.

Por fim, plotamos o resultado da versão 11 (defeituosa) sobre a área de tolerância definida e analisamos o quão ela se distancia do limite tolerável (vide Figura 14).

Figura 14 – Comparação entre a versão 11 e a margem de tolerância definida.



Fonte: Elaborada pela autora (2026)

A partir da análise da Figura 14, observa-se que a regressão de desempenho manifesta-se já no segundo ponto de coleta, correspondente a 10 segundos de interação, momento em que o consumo de memória da versão 11 ultrapassa de forma significativa a margem de tolerância definida. Em comparação com o comportamento histórico apresentado na Figura 11, nota-se um crescimento abrupto e desproporcional do uso de memória, caracterizado por um aumento superior a 100% em relação ao estágio inicial. Esse padrão se intensifica ao longo da execução, atingindo aproximadamente 2000 Megabyte (MB) após 30 segundos de cliques, o que corresponde a um incremento médio da ordem de 500MB a cada intervalo de 10 segundos. Tal comportamento evidencia uma degradação progressiva e persistente, incompatível com variações esperadas ou aceitáveis em cenários normais de uso. Destaca-se, ainda, que essa anomalia antecede o erro originalmente relatado pelo testador na ferramenta de registro de bugs, demonstrando a capacidade da abordagem em antecipar a detecção de falhas de performance.

Diferentemente da avaliação com medição pontual, no qual a margem de tolerância foi utilizada principalmente para quantificar o afastamento dos valores observados em relação ao limite aceitável, nesta avaliação com medição temporal a abordagem possibilitou a identificação de um novo erro de performance, em um ponto distinto e anterior ao previamente

reportado. Como resultado, foram caracterizadas duas regressões de desempenho independentes, ampliando significativamente o diagnóstico inicialmente realizado pela equipe de testes.

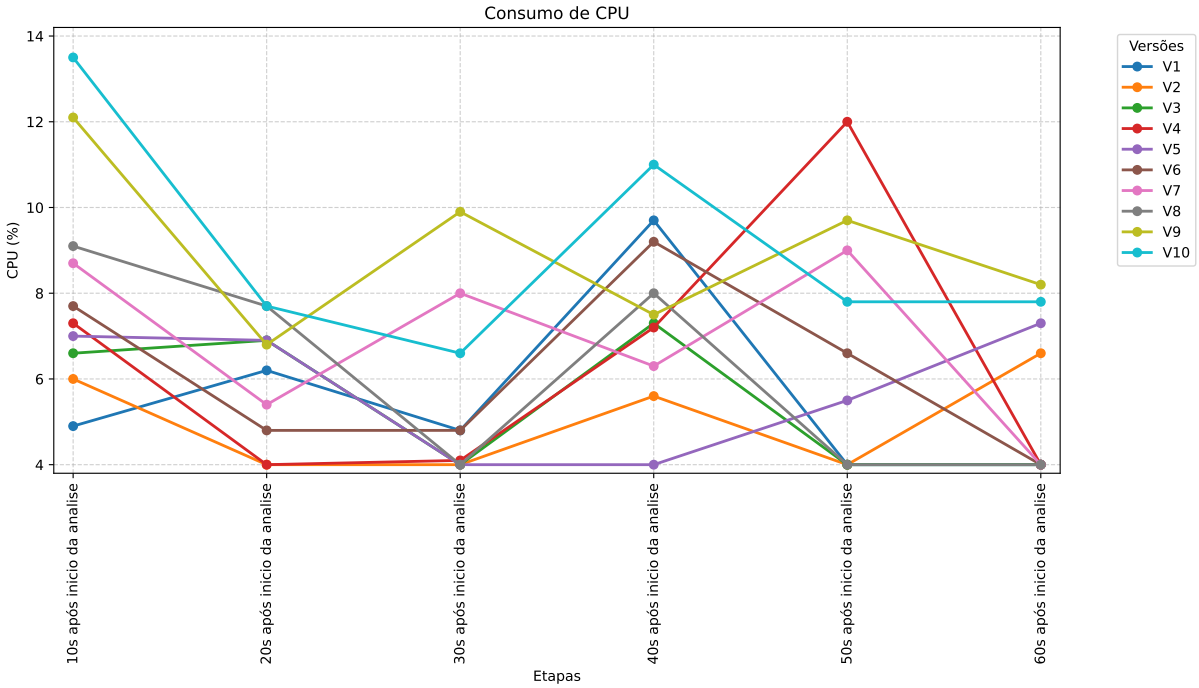
Esse resultado reforça a eficácia da adaptação do *N-version testing* em conjunto com o oráculo aproximado para testes de performance, evidenciando sua capacidade não apenas de estabelecer uma margem de tolerância objetiva e funcional em contextos onde requisitos explícitos de desempenho não estão formalmente definidos, mas também de revelar anomalias ainda não identificadas no processo tradicional de teste. A utilização de versões históricas como base de redundância temporal demonstra consistência metodológica, flexibilidade e capacidade de generalização da abordagem, permitindo a construção de uma referência inicial confiável, ainda que aproximada, para a avaliação contínua do desempenho ao longo da evolução do software.

6.2 SEGUNDA AVALIAÇÃO COM MEDIÇÃO DE SÉRIE TEMPORAL

Este cenário refere-se ao processo de indexação de 190 mídias na aplicação, uma funcionalidade relevante do ponto de vista operacional. Registros na ferramenta de gerenciamento de bugs indicaram que, durante a execução dessa atividade, o consumo de CPU ultrapassa 80%, caracterizando um comportamento fora dos limites esperados para esse tipo de processamento. Em um contexto de software industrial, esse nível de utilização é considerado crítico, pois implica elevada pressão sobre os recursos computacionais da máquina, podendo comprometer a estabilidade do sistema e a execução concorrente de outras funcionalidades.

Dando continuidade à abordagem adotada nas avaliações prévias, foram selecionadas dez versões anteriores à versão com erro, utilizadas como base histórica para a avaliação do comportamento de desempenho nesse cenário específico. O código de teste foi estruturado para coletar o consumo de CPU em intervalos regulares de 10 segundos, iniciando-se no momento em que o processo de indexação das imagens é disparado e finalizando após 1 minuto de processamento. A Figura 15 apresenta a ilustração dos valores de consumo de CPU obtidos ao longo das execuções.

Figura 15 – Consumo de CPU entre as 10 versões avaliadas durante 1 minuto

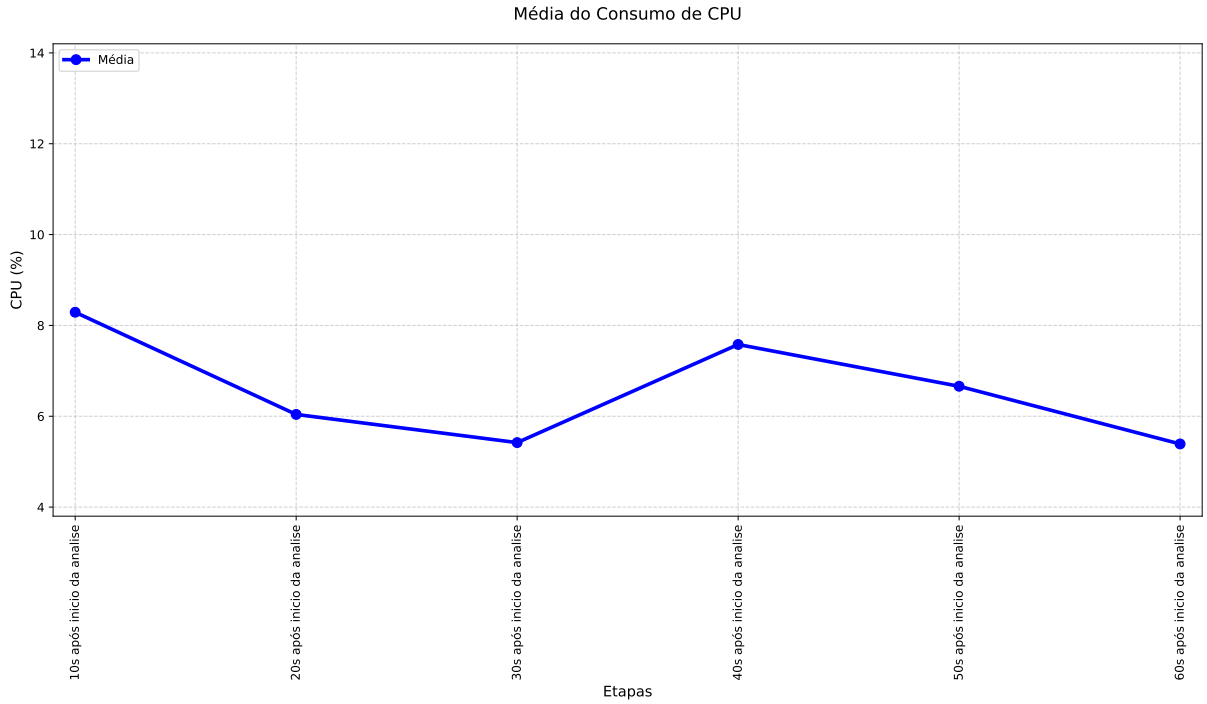


Fonte: Elaborada pela autora (2026)

A Figura 15 apresenta o comportamento histórico do consumo de CPU entre as versões analisadas ao longo das etapas de execução. Observa-se que, especialmente nos 10 segundos após o início da análise, ocorre um aumento no consumo de CPU entre as versões 7 e 10, cujo pico evolui aproximadamente de 8,7% para 13,5%. Esse acréscimo progressivo poderia representar um indício inicial de regressão de desempenho, pois sugere uma tendência de crescimento no uso de recursos ao longo das versões. No entanto, durante a validação manual realizada pelo especialista, esse comportamento não foi interpretado como um problema de desempenho, sendo os dados considerados como parte de um cenário estável de referência. Esse resultado evidencia uma limitação da análise baseada exclusivamente em inspeção manual, na qual variações graduais podem não ser percebidas. Caso a abordagem proposta já estivesse em uso, a análise baseada em métricas permitiria identificar automaticamente essa tendência de aumento no consumo de CPU, possibilitando a detecção antecipada de potenciais regressões de desempenho.

Diante desse comportamento, calculou-se a referência de performance do cenário, obtida a partir da média do consumo de CPU em cada ponto analisado. Essa referência estabelece a linha base para a análise comparativa do desempenho ao longo do tempo, conforme ilustrado na Figura 16.

Figura 16 – Definição da referência de performance

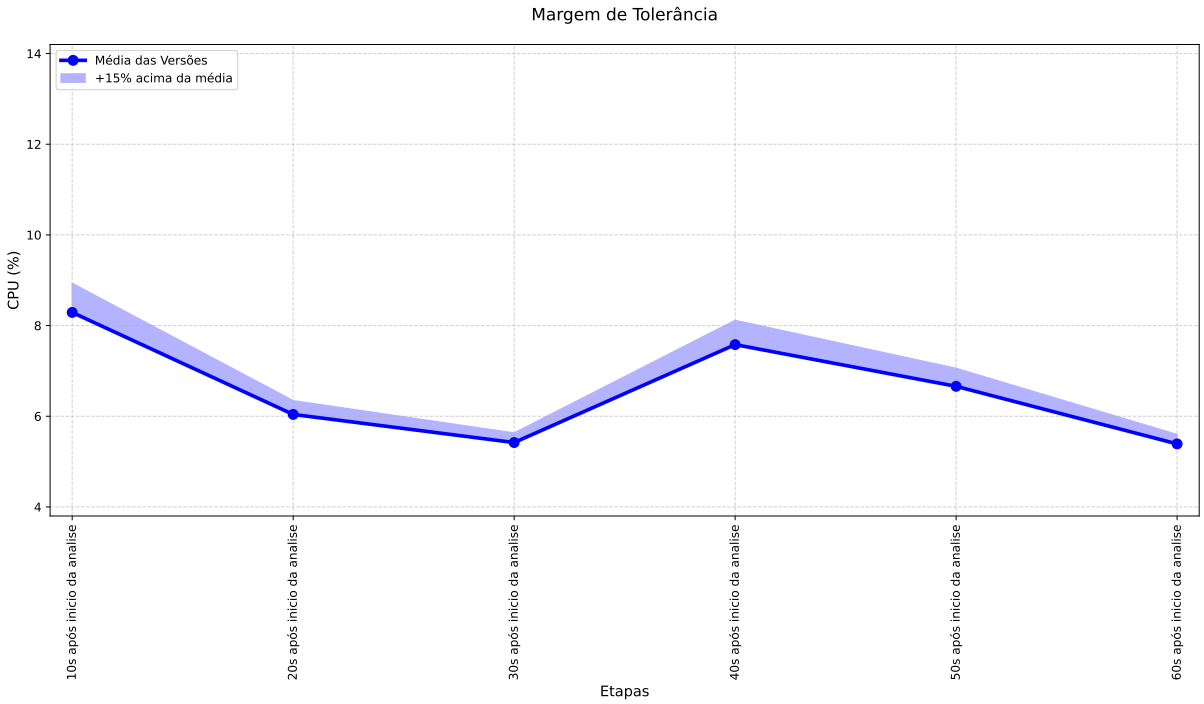


Fonte: Elaborada pela autora (2026)

A partir dos dados históricos, foi estabelecida a referência de performance, conforme ilustrado na Figura 16, por meio do cálculo da média do consumo de CPU em cada etapa da execução. Nota-se que o pico médio de consumo concentra-se nos primeiros 10 segundos de processamento, seguido por uma redução gradual e estabilização ao longo do tempo. Um leve acréscimo observado por volta dos 40 segundos é justificado pela presença de versões específicas (1, 6 e 10) que apresentaram consumo superior a 9%, destacando-se em relação ao comportamento predominante. Ainda assim, a média resultante preserva uma representação adequada do desempenho esperado, servindo como base objetiva para a definição da margem de tolerância.

Com base na referência de performance estabelecida, definiu-se a margem de tolerância aplicando-se um acréscimo de 15% sobre o valor médio de cada etapa, conforme apresentado na Figura 17. Essa margem foi adotada como um limite superior adaptativo, capaz de acomodar flutuações esperadas decorrentes de variabilidade ambiental e de execução, sem comprometer a sensibilidade do método à identificação de regressões reais. A área delimitada pela margem representa, portanto, o intervalo de desempenho considerado aceitável para versões subsequentes do software, funcionando como o núcleo do oráculo aproximado para testes de performance.

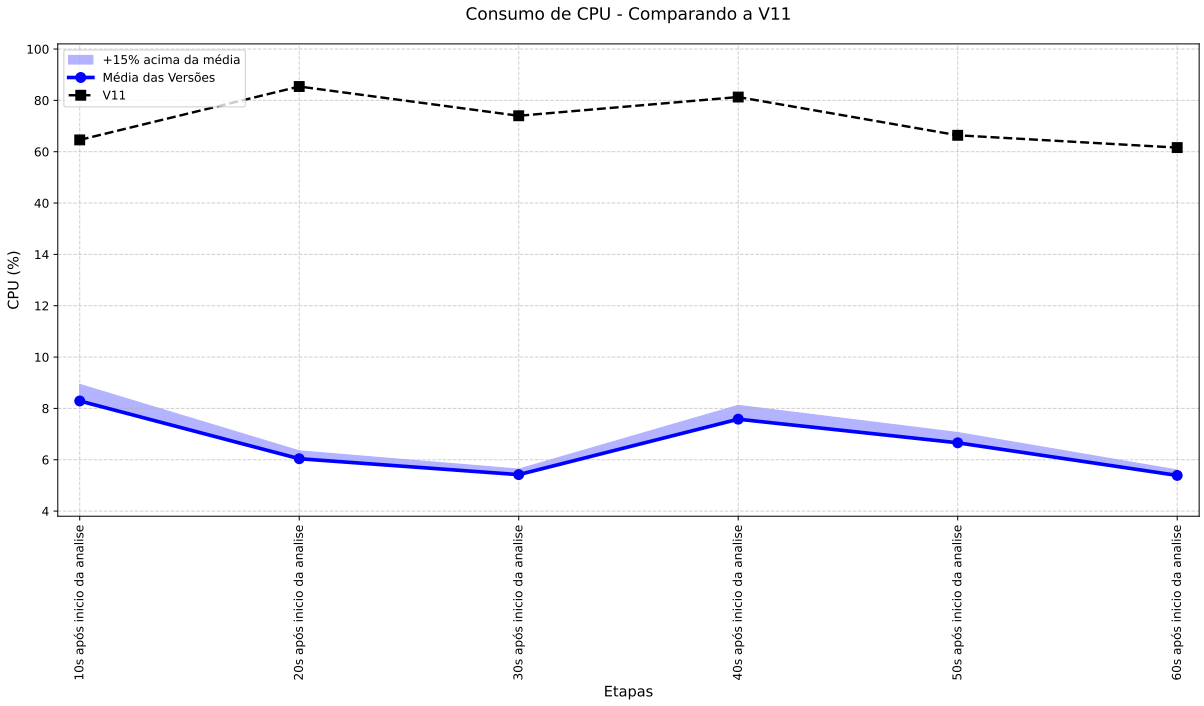
Figura 17 – Definição da margem de tolerância, considerando 15% sob a referência de performance.



Fonte: Elaborada pela autora (2026)

Por fim, comparamos os resultados obtidos no teste realizado com a versão 11, que apresenta o erro de alto consumo de CPU, e avaliamos o quão fora da área de tolerância definida ela está (Figura 18).

Figura 18 – Comparação entre a versão 11 e a margem de tolerância definida.



Fonte: Elaborada pela autora (2026)

A Figura 18 evidencia de forma inequívoca que o comportamento da versão 11 diverge sistematicamente da margem de tolerância, permanecendo fora dos limites aceitáveis em todas as etapas da medição temporal. Diferentemente das versões anteriores, nas quais variações graduais de consumo de CPU foram observadas e classificadas como comportamento aceitável, a versão 11 apresenta um deslocamento consistente e de grande magnitude ao longo de todo o intervalo analisado. Esse padrão indica uma regressão de desempenho persistente, e não uma flutuação pontual ou ruído inerente ao ambiente de execução. Nesse contexto, a margem de tolerância mostrou-se fundamental para quantificar objetivamente o grau de degradação de desempenho, permitindo não apenas a identificação da regressão, mas também a avaliação de seu impacto ao longo do tempo. A aplicação do oráculo aproximado para testes de performance elimina a dependência de julgamentos subjetivos do testador e possibilita a detecção precoce de anomalias que, se identificadas apenas por inspeção manual ou testes pontuais, poderiam ser diagnosticadas tardiamente. Assim, este resultado reforça a relevância da abordagem proposta para o monitoramento contínuo da evolução do software, especialmente em cenários de integração contínua, nos quais regressões de desempenho podem se propagar rapidamente entre versões sucessivas.

6.3 CONSIDERAÇÕES FINAIS DO CAPÍTULO

A avaliação baseada em medições de série temporal ampliou a capacidade analítica da abordagem ao permitir observar o comportamento das métricas de desempenho ao longo da execução do sistema. Diferentemente das medições pontuais, essa análise possibilitou identificar padrões persistentes de degradação, evidenciando regressões que se manifestam de forma contínua durante o tempo de execução. Os resultados mostraram que a comparação entre curvas históricas de referência e a série temporal da versão analisada permitiu detectar desvios sistemáticos no consumo de recursos, caracterizando regressões de desempenho de maneira objetiva.

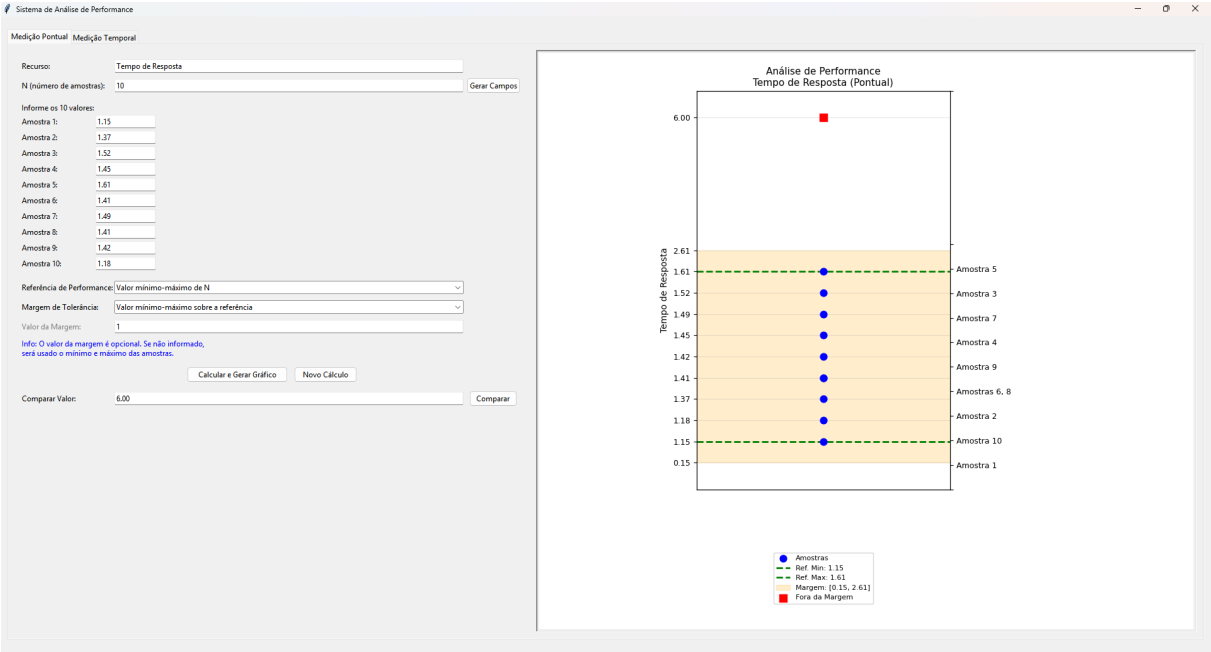
7 AUTOMAÇÃO DOS PARÂMETROS DO ÓRACULO

Neste capítulo, vamos apresentar o protótipo criado para otimizar o trabalho da equipe de testes na obtenção das margens de tolerância. Esse protótipo realiza as medições pontuais e temporais, gerando como saídas gráficos com o agrupamentos dos valores obtidos nos dados históricos e a comparação com a versão atual.

O protótipo foi desenvolvido com linguagem Python. Para que o script seja executado corretamente, é necessário executar o arquivo *requerements.txt* no VS Code com as dependências matplotlib>=3.5.0, numpy>=1.21.0 e pandas>=1.3.0. A interface foi projetada de forma clara, e com campos que correspondem às etapas da abordagem proposta, com duas abas paralelas, uma para Medição Pontual e outra para Medição de Série Temporal.

Na aba 'Medição Pontual' o usuário pode informar o **Recurso** que será avaliado (memória, tempo de resposta, velocidade de rede, entre outros), o **Número de amostras** (quantas versões serão utilizadas). Após definir a quantidade e clicar no botão **Gerar Campos**, os campos para informar os valores de cada versão são criados. A **Referência de Performance**, que pode ser a Média, Mediana ou os valores mínimo e máximo das amostras informadas. A **Margem de Tolerância**, que pode ser Desvio Padrão, Porcentagem ou Valor mínimo e máximo das amostras, o **Valor da Margem**, que pode ser um número inteiro ou um percentual, de acordo com o tipo de cálculo selecionado no campo Margem de Tolerância. E o campo **Comparar Valor**, onde é informado o valor da versão atual que está sendo comparada. Como mostra a Figura 19.

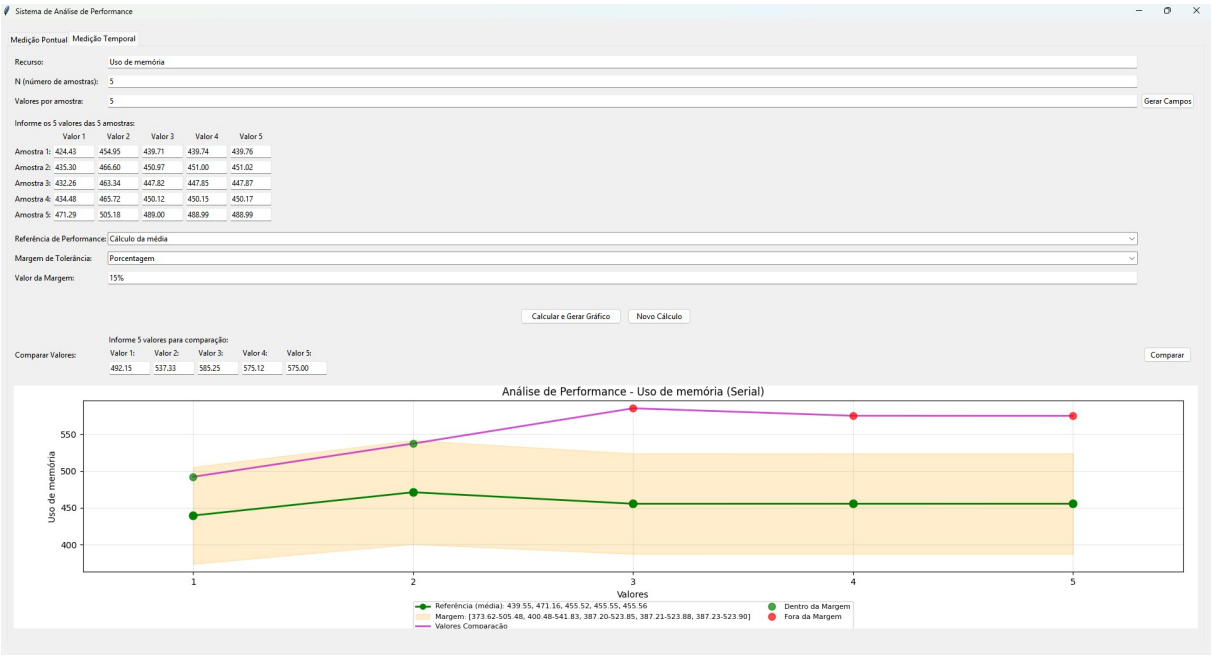
Figura 19 – Medição Pontual para Tempo de Resposta



Fonte: Elaborada pela autora (2026)

Na aba 'Medição Temporal' o usuário pode informar o **Recurso** que será avaliado, o **Número de amostras** (quantas versões serão utilizadas). O **Valores por amostra**, esse campo define quantos pontos por versão serão mensurados. Assim que N e Valores por amostras são informados, os campos correspondentes são criados na aba. Exemplo: Se o usuário informar 5 amostras e 5 valores por amostra, será considerado 5 pontos por versão. A **Referência de Performance**, que pode ser a Média, Mediana ou os valores mínimo e máximo das amostras informadas. A **Margem de Tolerância**, que pode ser Desvio Padrão, Porcentagem ou Valor mínimo e máximo das amostras, o **Valor da Margem**, que pode ser um número inteiro ou um percentual, de acordo com o tipo de cálculo selecionado no campo Margem de Tolerância. E o campo **Comparar Valores**, onde são informados os valores da versão atual que está sendo testada (vide Figura 20).

Figura 20 – Medição Temporal para uso de Memória



Fonte: Elaborada pela autora (2026)

Como a abordagem é flexível e a combinação de parâmetros depende das percepções do especialista, o campo **Valor da Margem** torna-se opcional quando utilizado o Valor mínimo-máximo na **Margem de Tolerância**. Assim, visualizamos na Tabela 3 as possíveis combinações de cálculo:

Tabela 3 – Combinações entre a Referência de Performance e a Margem de Tolerância.

Refer. de Performance	**Medida para Margem de Tolerância**	**Valor da Margem**	**Resultado calculado**
Média	Desvio Padrão	Nº inteiro. Exemplo: 2	Calculado 2 desvios padrão para mais ou para menos em relação a média dos dados históricos.
Média	Percentual	Exemplo: 10 %	Calculado 10% para mais ou para menos em relação a média dos dados históricos.
Média	Valor min-max	Sem valor, Inteiro, Percentual	Sem valor: O valor mínimo e máximo é a margem de tolerância. Inteiro: É aplicado o valor sob a referência de performance. Percentual:É aplicado o percentual sob a referência de performance.
Mediana	Desvio Padrão	Nº inteiro. Exemplo: 2	Calculado 2 desvios padrão para mais ou para menos em relação a mediana dos dados históricos.
Mediana	Percentual	Exemplo: 10%	Calculado 10% para mais ou para menos em relação a mediana dos dados históricos.
Mediana	Valor min-max	Sem valor, Inteiro, Percentual	Sem valor: O valor mínimo e máximo é a margem de tolerância. Inteiro: É aplicado o valor sob a referência de performance. Percentual:É aplicado o percentual sob a referência de performance.
Valor min-máx	Desvio Padrão	Nº inteiro. Exemplo: 1	Calculado 1 desvio padrão para mais ou para menos em relação ao valor min-máx dos dados históricos.
Valor min-máx	Percentual	Exemplo: 15%	Calculado 15% para mais ou para menos em relação ao valor min-máx dos dados históricos.
Valor min-máx	Valor min-max	Sem valor, Inteiro, Percentual	Sem valor: O valor mínimo e máximo é a margem de tolerância. Inteiro: É aplicado o valor sob a referência de performance. Percentual:É aplicado o percentual sob o valor mínimo-máximo.

Fonte: Elaborada pela autora (2026)

Essas combinações permitem que a abordagem seja aplicada a diferentes sistemas, ajustando-se às necessidades de projetos com dinâmicas específicas, ao mesmo tempo em que mantém

um respaldo quantitativo confiável nos resultados obtidos. O protótipo evidencia a facilidade de aplicação da abordagem, apresentando-se como um recurso simples e ágil, capaz de transformar as etapas do processo em dados claros e de produzir resultados objetivos, passíveis de interpretação e reutilização em ciclos contínuos de teste.

8 AVALIAÇÃO DE USABILIDADE

Embora os resultados empíricos discutidos nesta dissertação demonstrem a eficácia técnica da abordagem na detecção de regressões de desempenho, a aceitação e a usabilidade de qualquer nova abordagem ou metodologia de software, por parte de profissionais da área, são fatores cruciais para sua adoção na indústria.

Neste capítulo, delineia-se a metodologia de avaliação da usabilidade percebida da abordagem proposta, utilizando um instrumento amplamente reconhecido pela academia e pela indústria: o SUS.

8.1 O SYSTEM USABILITY SCALE (SUS)

O SUS é um questionário simples, de fácil aplicação, que foi desenvolvido por John Brooke em 1986 na *Digital Equipment Corporation* (BROOKE et al., 1996) com o objetivo de fornecer uma medida rápida e confiável da usabilidade subjetiva de um sistema.

8.2 DEFINIÇÃO E ESTRUTURA

O SUS consiste em uma escala Likert de 10 itens. Cada item apresenta uma afirmação sobre o sistema ou produto sendo avaliado, e os usuários respondem em uma escala de cinco pontos, variando de 1 (Discordo Totalmente) a 5 (Concordo Totalmente). As questões alternam entre afirmações positivas e negativas (por exemplo, "Eu achei o sistema fácil de usar"e "Eu achei o sistema desnecessariamente complexo") para evitar o viés de concordância do respondente (LEWIS, 2018). Adaptando as questões para nosso contexto, temos as seguintes afirmações:

1. Eu acho que gostaria de usar esta abordagem com frequência.
2. Eu achei a abordagem desnecessariamente complexa.
3. Eu achei a abordagem fácil de usar.
4. Eu acho que precisaria de suporte técnico para conseguir usar esta abordagem.
5. Eu achei as etapas dessa abordagem bem integradas.
6. Eu achei que esta abordagem tinha muita inconsistência.

7. Eu imagino que a maioria das pessoas aprenderia a usar esta abordagem rapidamente.
8. Eu achei a abordagem muito complicada de usar.
9. Eu me senti muito confiante usando a abordagem.
10. Eu precisei aprender muitas coisas antes de conseguir usar esta abordagem.

O resultado do SUS é um escore final que varia de 0 a 100, permitindo que a usabilidade de diferentes sistemas seja comparada de forma padronizada. Pesquisas subsequentes demonstraram a alta confiabilidade do instrumento e sua capacidade de mensurar dois fatores principais: a Usabilidade e a Capacidade de Aprendizagem (*Learnability*) do sistema (BANGOR et al., 2008).

O principal objetivo do SUS é mensurar a usabilidade percebida do usuário em três aspectos fundamentais da interação humano-computador: Efetividade (*Effectiveness*): A capacidade do usuário de completar tarefas e atingir seus objetivos com o sistema. Eficiência (*Efficiency*): O esforço e os recursos despendidos para completar essas tarefas. Satisfação (*Satisfaction*): O nível de conforto e satisfação subjetiva do usuário com a experiência (SAURO; LEWIS, 2016). Ao aplicar o SUS, busca-se obter um índice global e objetivo de quão amigável e aceitável o sistema (neste caso, a abordagem do oráculo aproximado) é para os seus usuários-alvo.

8.3 METODOLOGIA DE AVALIAÇÃO

Para validar a aceitação da abordagem do oráculo aproximado para testes de performance, aplicou-se o SUS como um instrumento de avaliação somativa. A avaliação focou na utilidade e facilidade de uso da abordagem desenvolvida, que define a margem adaptativa a partir dos dados de NVT.

8.4 PARTICIPANTES

O questionário SUS foi administrado a 9 especialistas da área de Testes de Software. A seleção desses participantes visa garantir que a percepção de usabilidade seja capturada por profissionais com experiência prática e familiaridade com testes de performance.

Foram recrutados 9 Engenheiros de Testes de diferentes níveis em três projetos diferentes. Todos os participantes concordaram voluntariamente em participar do nosso estudo e avaliar

a abordagem. A Tabela 4 apresenta dados demográficos dos participantes do estudo, todos com formação superior em Tecnologia da Informação (TI).

Entre os participantes, 7 se identificam como mulheres e 2 como homens, com idades variando de 24 a 49 anos. A experiência em testes de software abrange de 3 a 20 anos, representando um espectro que vai de profissionais iniciantes a experientes. Além disso, cada participante possui experiência prévia em testes de performance, com especialização variando de 2 a 10 anos

8.5 PROCEDIMENTOS

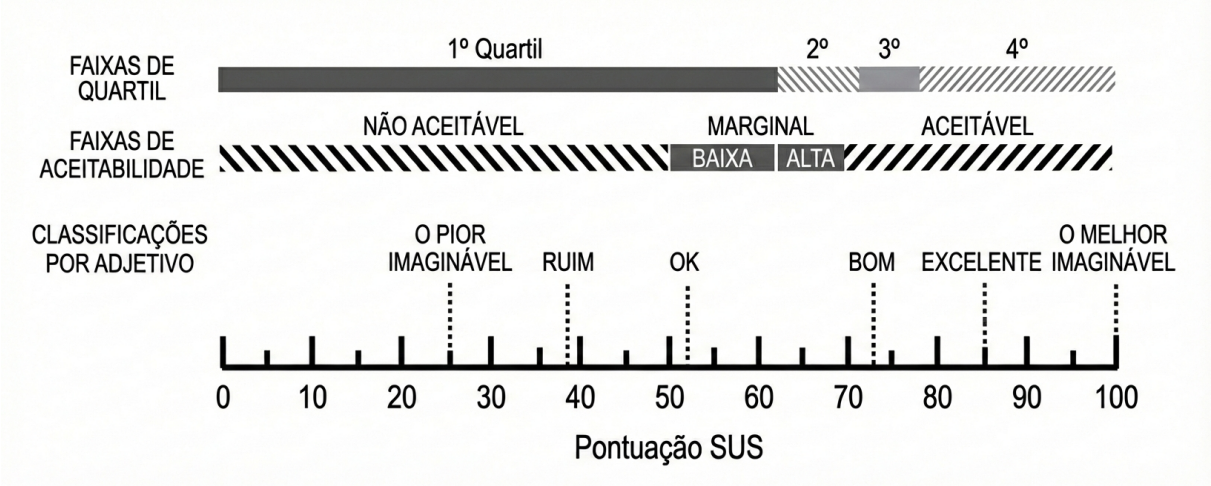
Uma apresentação sobre a abordagem foi feita aos participantes, onde foi explicado as etapas da abordagem, bem como sua aplicação em um cenário de erro de performance real registrado no projeto. Uma demonstração de utilização do protótipo também foi apresentada, possibilitando que os usuários tirassem dúvidas sobre seu funcionamento. Após a demonstração os participantes foram convidados a preencher o questionário SUS, fornecendo suas impressões e *feedbacks* sobre o uso da abordagem proposta para estabelecer métricas de performance mais coerentes ao contextos dos projetos.

8.6 RESULTADOS E DISCUSSÕES

Após os participantes concluírem a avaliação por meio do questionário SUS, as pontuações são calculadas da seguinte forma: a pontuação de cada item (afirmação) varia de 0 a 4. Para cada item ímpar, a contribuição da pontuação é a posição na escala menos 1. Para cada item par, a contribuição é 5 menos a posição na escala. Em seguida, as pontuações são somadas e multiplicadas por 2,5 para obter o valor geral do SUS. A pontuação final do SUS estará na faixa de 0 a 100 (BROOKE et al., 1996).

Para interpretar os resultados do SUS, adotamos a escala de classificação e classificação de aceitabilidade de (BANGOR et al., 2008) conforme Figura 21.

Figura 21 – Classificação de Aceitabilidade



Fonte: (BANGOR et al., 2008)

Para pontuações abaixo de 50, a usabilidade do sistema avaliado situa-se na faixa não aceitável e pode ser classificada como a pior imaginável ou ruim. Por outro lado, sistemas com pontuações de 70 ou mais situam-se na faixa aceitável e sua classificação pode variar entre bom, excelente e o melhor imaginável.

As pontuações do SUS para cada participante são mostradas na coluna mais à direita da Tabela 4. Como o questionário era anônimo, cada participante é denominado como "P"+ número.

Tabela 4 – Dados dos Participantes e Pontuação SUS.

Participantes	**Gênero**	**Idade**	**Anos em testes de software**	**Anos em testes de performance**	**Pontuação**
P1	Feminino	33	13	10	92.5
P2	Feminino	44	20	3	97.5
P3	Feminino	47	18	5	82.5
P4	Masculino	40	12	2	72.5
P5	Masculino	24	3	2	85.0
P6	Feminino	36	5	3	75.0
P7	Feminino	34	13	5	100.0
P8	Feminino	49	20	5	97.5
P9	Feminino	31	5	3	80.0
Pontuação média de usabilidade do sistema					**86.9**

Fonte: Elaborada pela autora (2026)

A média aritmética da pontuação SUS obtida foi de 86.9, posicionando a abordagem proposta em um patamar elevado de usabilidade, classificado entre excelente e o melhor nível

possível, segundo a escala padrão do instrumento. As pontuações individuais variaram entre 72.5, ainda enquadrada como boa usabilidade, e 100.0, indicando percepção máxima de facilidade de uso por parte de alguns participantes.

8.7 CONSIDERAÇÕES FINAIS DO CAPÍTULO

O resultado do SUS indica uma percepção amplamente positiva por parte dos participantes, independentemente do nível de experiência em testes de software ou testes de performance, e pode ser associado à clareza da abordagem, à baixa complexidade de configuração e à utilização de métricas objetivas baseadas em dados históricos. Além disso, a redução da subjetividade na definição de limites de desempenho e a aderência ao fluxo de trabalho já consolidado em projetos de software contribuíram para que os profissionais reconhecessem a abordagem como prática, compreensível e aplicável. Nesse sentido, a elevada pontuação de usabilidade sugere que a abordagem não foi apenas percebida como academicamente válida, mas também como útil e viável para o uso cotidiano em projetos reais de software, especialmente em ambientes industriais que demandam soluções automatizadas, reprodutíveis e de fácil integração aos processos existentes.

9 AMEAÇAS À VALIDADE

Nesta seção abordamos os pontos que podem influenciar os resultados obtidos de cada cenário avaliado.

9.1 VALIDADE EXTERNA

Ambiente de reprodução dos testes: Os erros avaliados no estudo foram registrados por diferentes especialistas, com diferentes máquinas, assim foi percebido que a replicação dos cenários não resultou em valores idênticos aos relatados na ferramenta de gerenciamento de erros. **Minimizada:** Apesar dos resultados não serem iguais, os valores obtidos nos resultados foram aproximados aos relatados, o que não comprometeu a eficácia na aplicação da abordagem, bem como confirmou a regressão de performance relatada.

Performance da rede de internet: Além de diferentes máquinas, a velocidade da rede do especialista que relatou o erro na ferramenta e a velocidade da rede da máquina utilizada na replicação do cenário, podem influenciar nos resultados. **Minimizada:** Acredita-se que a diferença da rede não influenciou nos resultados, pois os valores obtidos na máquina de replicação do cenário alcançaram os demonstrados no registro do erro.

9.2 VALIDADE INTERNA

Avaliação dos resultados: Por questões de tempo, motivação ou atenção, o avaliador dos resultados pode deixar passar dados relevantes da avaliação. **Minimizada:** A avaliação foi realizada em par com o orientador, para garantir que os resultados sejam interpretados em concordância. Além de captação de *feedbacks* de alguns membros da equipe do projeto.

9.3 VALIDADE DA CONSTRUÇÃO

Criação da abordagem: Definição inapropriada da base teórica para justificar a utilização da abordagem proposta e do processo de etapas na construção do oráculo aproximado para testes de performance. **Neutralizada:** Para neutralizar essa ameaça foi realizada uma avaliação minuciosa dos trabalhos relacionados com a proposta desta dissertação. Os trabalhos

selecionados que mais convergem com a temática formaram a base teórica deste estudo e regeram o processo de aplicação e avaliação da abordagem.

9.4 VALIDADE DA CONCLUSÃO

Análise e Interpretação probabilística: Utilização dos métodos matemáticos de forma inapropriada durante a análise e interpretação dos dados coletados. **Minimizada:** Para minimizar essa ameaça os cálculos e interpretações foram realizados juntamente com o orientador e também alguns membros da equipe do projeto foram consultados para confirmar a eficácia lógica das equações utilizadas.

10 CONCLUSÃO

Este capítulo traz um breve resumo desta dissertação, apresentando como os objetivos propostos por esse estudo foram atingidos. Em seguida, na seção 9.1, serão elencadas as lições aprendidas no desenvolvimento deste trabalho, as possibilidades de melhorias e as novas oportunidades de estudo.

Testes de performance são inerentes a projetos de softwares, em âmbito acadêmico e principalmente industrial, muitos projetos, sem abordagens bem definidas para executar esses tipos de testes, utilizam oráculos de testes. Porém, um problema para o oráculo é definir objetivamente a saída correta para testes de performance, ocasionando um gargalo para a automação e confiabilidade.

A presente dissertação propôs uma solução inovadora para este problema, culminando na definição e validação de um oráculo aproximado para testes de performance com margem adaptativa, utilizando uma adaptação da técnica de NVT. A abordagem substituiu a redundância funcional tradicional pela redundância temporal, tratando n versões históricas e bem-sucedidas do software como uma base de dados redundante e confiável.

Os resultados empíricos, obtidos através dos cenários experimentais em um software industrial, validaram a eficácia e a solidez da abordagem. A análise detalhada confirmou a adequação do uso de intervalos de tolerância, baseados em parâmetros como o desvio padrão e limites percentuais, como um mecanismo objetivo para detecção de regressões.

O objetivo principal deste trabalho: propor uma abordagem para a definição de um oráculo aproximado para testes de performance, fundamentado na análise de dados históricos e inspirado nos princípios do *N-version testing*, foi integralmente alcançado. A abordagem proposta permitiu a criação de um oráculo calibrado e adaptável, capaz de definir uma margem de tolerância objetiva para a detecção de regressões de desempenho. Essa margem, ao acomodar a variabilidade inerente aos ambientes de teste, evita falsos positivos e traduz percepções subjetivas em limites tecnicamente fundamentados. Os objetivos específicos também foram atendidos, com destaque para prototipação do processo da abordagem, reduzindo o esforço humano e reforçando a viabilidade e a importância da automação para a aplicação prática da abordagem.

A abordagem desenvolvida demonstrou ser capaz de utilizar dados históricos de versões anteriores para estabelecer um comportamento de referência, permitindo a identificação de

desvios significativos de desempenho sem a necessidade de valores absolutos previamente definidos. Os resultados obtidos indicaram que o oráculo aproximado proposto foi eficiente na detecção de regressões de desempenho, apresentando coerência com o comportamento observado do sistema e sensibilidade suficiente para destacar degradações relevantes, mesmo em cenários sujeitos a variações ambientais.

Além disso, os estudos de casos evidenciaram que a utilização de conjuntos históricos de dados contribui para uma avaliação mais potente do desempenho, reduzindo falsos positivos e aumentando a confiabilidade dos resultados. Esses achados reforçam a viabilidade da abordagem como suporte a atividades de teste em ambientes de desenvolvimento contínuo, nos quais a rápida evolução do software dificulta a manutenção de oráculos tradicionais.

Por fim, como objeto de estudo, foi definido um processo confiavél para estabelecer um oráculo aproximado para testes de performance. A abordagem foi submetida a avaliação, por meio do SUS, com especialistas da área de testes. Com base nos resultados dessa pesquisa, percebeu-se que a aplicação da abordagem trouxe uma expectativa promissora e viável, ao passo que especialistas foram favoráveis à proposta, conforme mostrado na Figura 21 de Classificação de aceitabilidade de (BANGOR et al., 2008). A avaliação também proporcionou identificar algumas direções para melhorias e as limitações desta pesquisa.

Dessa forma, esta dissertação contribui para a área de testes de software ao apresentar uma alternativa prática e conceitualmente fundamentada para a definição de oráculos aproximados para testes de performance, ampliando o escopo de aplicação do *N-version testing* para além da verificação funcional. A abordagem proposta mostrou-se promissora para apoiar equipes de desenvolvimento e testes na detecção de regressões de desempenho, especialmente em contextos onde dados históricos estão disponíveis e a definição exata do comportamento esperado é inviável.

10.1 LIÇÕES APRENDIDAS E TRABALHOS FUTUROS

A principal contribuição do trabalho caracteriza-se pela definição de um oráculo aproximado para testes de performance que permite a detecção de regressões de desempenho. E as principais lições aprendidas dessa contribuição são:

- Adaptação do *N-version testing*: O trabalho demonstrou que a técnica de *N-version testing*, tradicionalmente usada para tolerância a falhas funcionais, pode ser adaptada

com sucesso para o domínio de testes de desempenho, utilizando versões históricas do mesmo software como "n versões"para estabelecer uma linha de base. Essa abordagem é particularmente útil em ambientes de integração contínua, onde a rápida identificação de problemas de desempenho é crucial.

- A importância do oráculo aproximado: O estudo ressalta que as percepções de desempenho são frequentemente subjetivas e informais. A criação de um oráculo aproximado para testes de performance oferece um mecanismo objetivo para validar os resultados, reduzindo o esforço humano na definição e verificação das saídas esperadas. Ao definir uma margem de tolerância, a abordagem permite uma avaliação mais objetiva e consistente do desempenho.

- Validação da abordagem: A dissertação validou a abordagem através de avaliações em uma aplicação industrial. Nos cenários de testes com medição pontual e temporal, o oráculo proposto foi capaz de identificar regressões de desempenho. Notavelmente, o trabalho identificou erros que não haviam sido previamente mapeados pela equipe, demonstrando o potencial da abordagem em ir além da detecção de falhas já conhecidas.

Em termos práticos, a pesquisa fornece um mecanismo objetivo e baseado em evidências para lidar com a subjetividade e informalidade que frequentemente cercam a definição de requisitos de performance. Ao reduzir o custo do oráculo humano e fornecer limites de aceitação quantificáveis, a dissertação oferece uma abordagem consistente para o monitoramento contínuo da performance em pipelines de CI.

Com base nos resultados e nas lições aprendidas, sugere-se os seguintes trabalhos futuros para ampliar o escopo e a aplicabilidade da abordagem:

1. Generalização e adaptação a ambientes heterogêneos: Aprofundar a discussão sobre como a abordagem mitiga ou se adapta à variabilidade ambiental (diferenças de hardware, carga do sistema operacional, rede), especialmente quando os dados históricos não correspondem perfeitamente ao ambiente de teste atual.

2. Avaliação em softwares de código aberto e em grande escala: Replicar os experimentos em um conjunto mais diversificado de softwares, incluindo projetos de código aberto e sistemas com diferentes perfis de carga e escalabilidade, para validar a generalização da abordagem.

3. Integração com ferramentas de CI/CD: Implementar a abordagem diretamente em um pipeline de CI/CD industrial para avaliar a latência e o impacto operacional de sua execução em tempo real, fornecendo métricas de desempenho sobre o próprio oráculo.

REFERÊNCIAS

AFSHAN, S. et al. Evolving readable string test inputs using a natural language model to reduce human oracle cost. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. [S.l.: s.n.], 2013. p. 352–361.

BANGOR, A. et al. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, Taylor & Francis, v. 24, n. 6, p. 574–594, 2008.

BARR, E. T. et al. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, v. 41, n. 5, p. 507–525, 2015.

BASS L., C. P. . K. R. *Software Architecture in Practice*. 3rd. ed. [S.l.]: Addison-Wesley, 2012.

BERTOLINO, A. Software testing research: Achievements, challenges, dreams. In: *2007 Future of Software Engineering*. USA: IEEE Computer Society, 2007. (FOSE '07), p. 85–103. ISBN 0769528295. Disponível em: <https://doi.org/10.1109/FOSE.2007.25>.

BISHOP, P. G. et al. Pods—a project on diverse software. *IEEE Transactions on Software Engineering*, IEEE, n. 9, p. 929–940, 2012.

BROOKE, J. et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, London, England, v. 189, n. 194, p. 4–7, 1996.

BURNSTEIN, I. *Practical software testing: a process-oriented approach*. 1st. ed. [S.l.]: Springer Science Business Media, 2006.

CHEN, J. et al. An exploratory study of performance regression introducing code changes. In: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.: s.n.], 2017. p. 341–352.

GIL, A. C. *Métodos e técnicas de pesquisa social. Fundamentos da metodologia científica*. [S.l.: s.n.], 1999.

GIL, A. C. *Como elaborar projetos de pesquisa*. 5. ed. [S.l.]: Atlas, 2008.

GU, Z. et al. Has the bug really been fixed? In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. [S.l.: s.n.], 2010. p. 55–64.

HANNIGAN, K. *An Empirical Evaluation of the Indicators for Performance Regression Test Selection*. [S.l.]: Rochester Institute of Technology, 2018.

HATTON, L. N-version design versus one good version. *IEEE Software*, IEEE, v. 14, n. 6, p. 71–76, 1997.

HOWDEN, W. E. Theoretical and empirical studies of program testing. *IEEE Transactions on Software Engineering*, IEEE, n. 4, p. 293–298, 2006.

INGO, H. et al. Automated system performance testing at mongodb. In: *Proceedings of the workshop on Testing Database Systems*. [S.l.: s.n.], 2020. p. 1–6.

KALYAN, A. et al. Prioritization of dynamic test cases based on historical data for use in software testing. *International Journal of Food and Nutritional Sciences*, v. 12, n. 4, 2023. Disponível em: <http://www.ijfans.org/uploads/paper/ba030df42bed62d10c7e8db60cdae7d7.pdf>.

KAUSHIK, D. M. et al. Performance testing tools: A comparative study. *International Journal of Innovative Science, Engineering & Technology*, v. 1, n. 4, p. 264–267, 2014.

KNIGHT, J. et al. Correlated failures in multi-version software1. *IFAC Proceedings Volumes*, v. 18, n. 12, p. 159–165, 1985. ISSN 1474-6670. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474667017601009>.

KUDRJAVETS, G. et al. The evolving landscape of software performance engineering. In: *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2022. (EASE '22), p. 260–261. ISBN 9781450396134. Disponível em: <https://doi.org/10.1145/3530019.3534977>.

LAKATOS, E. M. et al. *Fundamentos da metodologia científica. Fundamentos da metodologia científica.* 5th. ed. [S.l.]: Editora Atlas S.A, 2010.

LEWIS, J. R. The system usability scale: past, present, and future. *International Journal of Human–Computer Interaction*, Taylor & Francis, v. 34, n. 7, p. 577–590, 2018.

LEZNIK, M. et al. Change point detection for mongodb time series performance regression. In: *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*. [S.l.: s.n.], 2022. p. 45–48.

LIAO, L. Addressing performance regressions in devops: Can we escape from system performance testing? In: *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. [S.l.: s.n.], 2023. p. 203–207.

LIAO, L. et al. Locating performance regression root causes in the field operations of web-based systems: An experience report. *IEEE Transactions on Software Engineering*, v. 48, n. 12, p. 4986–5006, 2022.

LIU, L. et al. An empirical study on underlying correlations between runtime performance deficiencies and "bad smells" of microservice systems. In: IEEE. *2021 IEEE International Conference on Web Services (ICWS)*. [S.l.], 2021. p. 751–757.

MALIK, H. et al. Automatic detection of performance deviations in the load testing of large scale systems. In: *2013 35th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2013. p. 1012–1021.

MANOLACHE, L. et al. Software testing using model programs. *Software: Practice and Experience*, Wiley Online Library, v. 31, n. 13, p. 1211–1236, 2001.

MARCELINO, C. et al. Leveraging n-version testing to define approximate oracles for performance testing. In: SBC. *Simpósio Brasileiro de Engenharia de Software (SBES)*. [S.l.], 2025. p. 818–824.

MCKEEMAN, W. M. Differential testing for software. *Digital Technical Journal*, v. 10, n. 1, p. 100–107, 1998.

PARK, J. et al. Jest: N+ 1-version differential testing of both javascript engines and specification. In: IEEE. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. [S.l.], 2021. p. 13–24.

PATEL, K. et al. A mapping study on testing non-testable systems. *Software Quality Journal*, v. 26, n. 4, p. 1373–1413, 2018. Disponível em: <https://doi.org/10.1007/s11219-017-9392-4>.

PEER, D. *Predicting Software Performance Using Historical Data Trends*. 2024. Disponível em: <https://peerdh.com/blogs/programming-insights/predicting-software-performance-using-historical-data-trends>.

SANTOS, L. B. R. et al. Performance regression testing initiatives: a systematic mapping. *Information and Software Technology*, v. 179, p. 107641, 2025. ISSN 0950-5849. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0950584924002465>.

SAURO, J. et al. *Quantifying the user experience: Practical statistics for user research*. [S.l.]: Morgan Kaufmann, 2016.

SELLTIZ, C. et al. *Planejamento de pesquisa: estudos exploratórios e descritivos. Métodos de Pesquisa nas Relações Sociais*. [S.l.]: Ed. Herder e Editora da Universidade de São Paulo, 1967.

SERBOUT, S. et al. Evoscat: Exploring software change dynamics in large-scale historical datasets. *arXiv preprint arXiv:2508.10852*, 2025. Disponível em: <https://arxiv.org/abs/2508.10852>.

SILVA, E. L. d. et al. *Metodologia da pesquisa e elaboração de dissertação.* 3rd. ed. [S.l.]: ed.rev atual, 2001.

SMITH, C. U. et al. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. 1st. ed. [S.l.]: Addison-Wesley, 2002.

SOMMERVILLE, I. *Software Engineering*. 9th. ed. [S.l.]: Addison-Wesley, 2011.

TRAINI, L. et al. Ai-driven java performance testing: Balancing result quality with testing time. In: *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2024. p. 443–454.

XIE, T. Augmenting automatically generated unit-test suites with regression oracle checking. In: THOMAS, D. (Ed.). *ECOOP 2006 – Object-Oriented Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 380–403.

XIE, T. et al. Checking inside the black box: Regression testing by comparing value spectra. *IEEE Transactions on software Engineering*, IEEE, v. 31, n. 10, p. 869–883, 2005.

YOO et al. Regression testing minimization, selection and prioritization: A survey. *Software Testing, Verification and Reliability*, v. 22, n. 2, p. 65–120, 2012. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/stvr.430>.