Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Analisando variações de desempenho em diferentes versões de um produto de um software como serviço-SaaS, em uma organização privada brasileira: um estudo observacional

Autores: Eduardo Afonso Dutra Silva
Rafael Cleydson da Silva Ramos
Orientador: Prof. Msc. Hilmer Rodrigues Neri

Brasília, DF
2023

Eduardo Afonso Dutra Silva
Rafael Cleydson da Silva Ramos

Analisando variações de desempenho em diferentes versões de um produto de um software como serviço-SaaS, em uma organização privada brasileira: um estudo observacional

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Brasília, DF
2023

Eduardo Afonso Dutra Silva
Rafael Cleydson da Silva Ramos

Analisando variações de desempenho em diferentes versões de um produto de um software como serviço-SaaS, em uma organização privada brasileira: um estudo observacional/ Eduardo Afonso Dutra Silva
Rafael Cleydson da Silva Ramos. – Brasília, DF, 2023-
139 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Msc. Hilmer Rodrigues Neri

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. análise da eficiência de desempenho. 2. métricas e medidas de desempenho. 3. qualidade de software. I. Prof. Msc. Hilmer Rodrigues Neri. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Analisando variações de desempenho em diferentes versões de um produto de um software como serviço-SaaS, em uma organização privada brasileira: um estudo observacional

CDU 02:141:005.6

Eduardo Afonso Dutra Silva
Rafael Cleydson da Silva Ramos

Analisando variações de desempenho em diferentes versões de um produto de um software como serviço-SaaS, em uma organização privada brasileira: um estudo observacional

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 20 de dezembro de 2023:

Prof. Msc. Hilmer Rodrigues Neri
Orientador

Profa. Dra. Milene Serrano
Convidado 1

Prof. Dr. Renato Coral Sampaio
Convidado 2

Brasília, DF
2023

Agradecimentos

de Eduardo Afonso Dutra Silva

Agradeço ao Prof. Hilmer Rodrigues Neri pela confiança e dedicação em aceitar o grande e repentino desafio de nos orientar no desenvolvimento deste trabalho.

Agradeço ao meu caro amigo Rafael Cleydson por todos esses anos de colaboração, noites em claro e experiência trocadas desde o primeiro semestre da graduação. Ele sempre foi uma inspiração no âmbito acadêmico, profissional e pessoal.

Agradeço aos meus fiéis amigos Nicole Damasceno, Pablo Nepomuceno e Phelipe Luiz pelos longos anos de amizade sincera e por acompanhar e celebrar cada uma das minhas conquistas como se fossem deles.

Agradeço à Virginia Cardoso por ser meu porto seguro e a melhor companheira que eu poderia desejar. Desde 2017, ela me apoia, conforta e motiva nos momentos de fraqueza e, também comemora, compartilha e proporciona momentos de felicidade.

E agradeço principalmente à Antonia Dutra Silva por fazer o impossível para que eu pudesse chegar onde hoje estou. A ela devo tudo, e sei que toda conquista minha é também mérito dela.

Agradecimentos

de Rafael Cleydson da Silva Ramos

Gostaria de agradecer a Deus, em primeiro lugar, por todas as bençãos que vem acontecendo em minha vida, e principalmente pela saúde que me permite ir atrás dos meus sonhos.

Agradeço à minha família pelo suporte e amor incondicional recebido em toda minha vida. Por sempre compartilhar as conquistas e momentos como se fossem únicos, com muita união e alegria.

Agradeço à Ana Ferreira Sousa, minha companheira e maior apoiadora nesses últimos tempos. Uma mulher maravilha que está comigo nos momentos bons e ruins, sempre com seu alto astral contagiante iluminando o ambiente. Com você, consigo alçar voos mais altos.

Agradeço ao meu amigo, Eduardo Dutra Silva, por todos os momentos e experiências compartilhadas, por ser essa pessoa carismática que sempre trata tudo da melhor maneira possível. Obrigado por ser uma inspiração desde os primeiros semestres da faculdade. Agradeço também aos meus amigos Jhonson, João Pedro e Thiago, por todo o apoio, confiança e amizade.

Agradeço ao prof. Hilmer Neri, que é um homem muito aplicado e profissional, além de ser uma pessoa incrível. Obrigado por todo conhecimento e todas as experiências trocadas. E também pela confiança que nos foi concedida.

"Mas não precisamos saber pra onde vamos,
nós só precisamos ir. Não queremos
ter o que não temos, nós só queremos viver.
(Engenheiros do Hawaii)

Resumo

A não observância às restrições de recursos computacionais pode acarretar prejuízos financeiros para organizações que lidam com o desenvolvimento e a sustentação de produtos. O monitoramento e a análise da característica de qualidade de eficiência e desempenho, realizados de forma sistemática, provêm a capacidade de identificação de oportunidades de melhoria e por conseguinte, auxilia desenvolvedores e gestores técnicos a tomarem decisões orientadas a dados, em relação a este fator da qualidade. Considerando esse contexto, neste trabalho de conclusão de curso, realizou-se uma revisão estruturada da literatura, a fim de compreender a aplicação de técnicas que auxiliam a análise de desempenho e que podem apoiar a tomada de decisão sobre a aceitação de versões de produto de software. A partir dessa compreensão, conduziu-se uma investigação experimental, por meio da execução de um estudo observacional, em uma organização da indústria brasileira. Essa organização possui, em seu portfólio, um produto SaaS (*Software as a Service*) que enfrenta desafios relacionados ao desempenho. O processo de coleta e análise de dados de desempenho foi automatizado, aplicando as técnicas de gráfico de controle e florestas aleatórias em diferentes versões do caso observado, buscando auxiliar a identificação da variação de desempenho e eficiência de forma a auxiliar a tomada de decisão de aceitação de *releases* entre diferentes versões do produto de software.

Palavras-chave: eficiência de desempenho, qualidade do produto de software, métricas de desempenho, estudo observacional, análise de desempenho.

Abstract

Failure to observe the restrictions on computational resources can result in financial losses for organizations that deal with the development and support of products. The monitoring and analysis of the quality characteristic of efficiency and performance carried out systematically, provide the ability to identify opportunities for improvement and therefore help developers and technical managers to make data-driven decisions regarding this quality factor. Considering this context, in this course completion work, a structured literature review was carried out to understand the application of techniques that help performance analysis and support decision-making on the acceptance of software product versions. From this understanding, it was conducted an experimental investigation, through the execution of an observational study, in a Brazilian industry organization. This organization has, in its portfolio, a SaaS product (*Software as a Service*) that faces challenges related to performance. The process of collecting and analyzing performance data was automatized using the control chart and random forest techniques, in different versions of the observed case, evaluating if it could help identify performance and efficiency variation to help decision-making on acceptance of releases between different software product versions.

Key-words: performance efficiency, software product quality, performance metrics, observational study, performance analysis.

Lista de ilustrações

Lista de tabelas

Lista de quadros

Lista de códigos

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BPMN	*Business Process Model and Notation*
CPU	*Central Processing Unit*
DB	*Database*
DE	*Dados Excessivos*
EPS	*Engenharia de Produto de Software*
FGA	*Faculdade do Gama*
GB	*GigaBytes*
GQM	*Goal Question Metric*
ISO	*International Organization for Standardization*
LC	*Limite de consulta*
MAPE	*Mean Average Percentage Error*
ORM	*Object-Relational Mapping*
PICO	*Patient Intervention Comparison Outcome*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
SaaS	*Software as a Service*
SO	*Sistema Operacional*
TCC	Trabalho de conclusão de curso

Sumário

1 Introdução

Este capítulo visa contextualizar a área de domínio deste trabalho, apresentando conceitos importantes referentes à: qualidade de software e análise de desempenho do produto. Na sequência, serão abordados o Problema, a Questão de Pesquisa, os Objetivos, Geral e Específicos e a Organização dessa monografia. Por fim, tem-se Cronograma e fluxo de atividades do trabalho.

1.1 Contexto

Um dos princípios de gestão da qualidade estabelecidos na norma ISO 9000 é a tomada de decisão baseada em evidências. Esse princípio enfatiza que decisões eficazes devem ser embasadas na análise de dados e informações (ISO 9000, 2015). A ISO/IEC/IEEE 15939 (2017) define um processo, no contexto da medição de software, que envolve a definição de um conjunto apropriado de medidas que atendam as necessidades específicas de informação. No entanto, a norma não especifica o conjunto de medidas a ser utilizado como parte do plano de medição.

A ISO/IEC 25010 (2011) foca em aspectos de qualidade de produto e qualidade de uso de *software*, e define um conjunto de características e subcaracterísticas que corroboram na aferição dessa qualidade, considerando propriedades estáticas e dinâmicas do produto. Dentre as propriedades dinâmicas, é importante ressaltar a característica de eficiência de desempenho, a qual é subsidiada por subcaracterísticas como tempo de resposta, utilização dos recursos e capacidade do produto. Entretanto, a ISO não provê uma definição de como se conduzir o processo de medição necessário para aferir a qualidade do ponto de vista de eficiência de desempenho.

De acordo com Jiang et al. (2009), a análise dos dados de uso de recursos computacionais coletados ao longo de um período de execução do sistema pode ser uma tarefa desafiadora e demorada. Isso pode ocorrer devido à alta quantidade de dados gerados ou à experiência necessária para realizar uma análise eficiente. Nesse sentido, a utilização de uma abordagem sistemática e automatizada para auxiliar a realização dessa tarefa pode ser benéfica e auxiliar a interpretação do resultado dos testes de maneira mais assertiva.

Devido à inerente preocupação dos sistemas embarcados com a precisão do tempo necessário para executar uma tarefa, é comum realizar a análise do desempenho buscando encontrar o melhor e o pior cenário para garantir que o sistema atenda aos requisitos do contexto em que será utilizado (ALSHEIKHY; HAN; AMMAR, 2015). Por outro lado, no contexto *web*, existem empresas de grande escala, atuando em diferentes ramos como

lojas virtuais ou redes sociais, fornecendo serviços para quantidades enormes de usuários diariamente. Nesse sentido, um problema de desempenho pode prejudicar diretamente a reputação e o lucro da empresa, apresentando assim a necessidade de identificar esses problemas antes que o usuário final seja afetado (LIAO et al., 2020).

Existem diversas técnicas para realizar a análise de desempenho de produtos de *software*. Geralmente, o objetivo dessas técnicas é detectar se quedas significativas no desempenho computacional ocorrem entre versões distintas do mesmo produto. Em algumas, é necessário o conhecimento da lógica interna do produto, sua estrutura arquitetural e comunicação entre componentes. Já para outras, somente é necessário acesso à carga aplicada ao sistema e a métrica de desempenho que se deseja analisar (GAO et al., 2016).

1.2 Problema

Segundo Nguyen et al. (2012), a execução de testes convencionais de desempenho coloca o *software* em uma carga de campo por um determinado período de tempo, durante o qual as métricas de uso de recursos computacionais, que serão utilizadas para aferição da qualidade de eficiência de desempenho do produto, são coletadas. Esses testes de carga são geralmente conduzidos no final do ciclo de desenvolvimento de *software*, próximo ao prazo de entrega, o que muitas vezes resulta em pouco tempo disponível para executar e analisar os resultados dos testes.

No entanto, negligenciar os testes de carga e a análise de uso de recursos computacionais pode acarretar prejuízos financeiros, especialmente em produtos de *software* que atendem a milhões de usuários diariamente, pois qualquer queda no desempenho dessa categoria de produtos pode afetar negativamente a experiência do usuário (LIAO et al., 2020).

Por outro lado, os dados gerados pelos testes de carga podem ser analisados visando identificar melhorias no desempenho do uso de recursos computacionais. Em determinados contextos, equipes de desenvolvimento podem buscar aprimorar a eficiência computacional para reduzir custos com infraestrutura ou para atender clientes com máquinas com menor capacidade computacional.

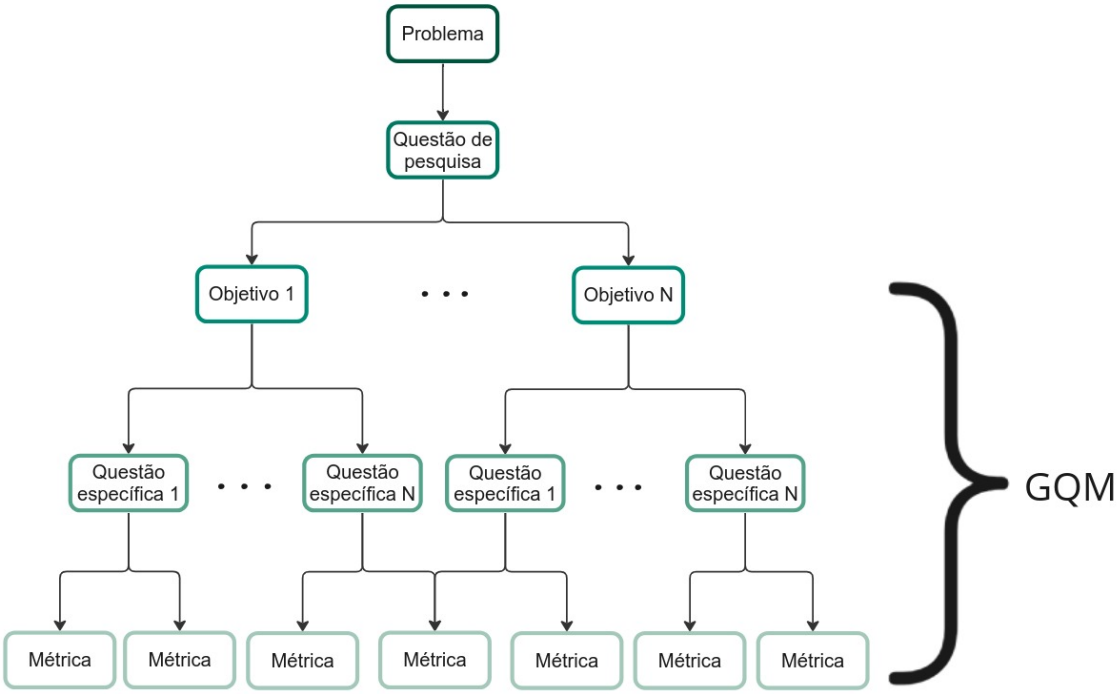
Jiang et al. (2009) apresentam como um dos desafios encontrados na análise dos resultados de testes de carga, a definição de critérios de sucesso e de falha. Ao contrário dos testes funcionais, em que os critérios são geralmente mais claros e objetivos, a interpretação de métricas e medidas que descrevem a qualidade de eficiência de desempenho pode ser menos precisa. Deste modo, a **falta de clareza sobre como interpretar métricas e medidas que descrevem a característica de qualidade de eficiência de desempenho afeta a identificação de restrições de recursos computacionais**. Isso acarreta desperdício de recursos financeiros e, em última instância, compromete a

estratégia das organizações.

1.3 Questão de Pesquisa

Para definir a questão de pesquisa do estudo, adaptou-se a técnica *Goal Question Metric* (GQM), que consiste em uma forma de estruturar hierarquicamente objetivos, questões e métricas, para que seja possível planejar e mensurar projetos de maneira eficiente (BASILI; CALDIERA; ROMBACH, 1994). A estrutura apresentada na Figura 1 representa de maneira hierárquica como estruturar um estudo observacional, baseando-se na técnica GQM.

Figura 1 – Estrutura adaptada do GQM para estudo observacional



Fonte: Adaptado de Basili, Caldiera e Rombach (1994)

Dentro dessa estrutura, Basili, Caldiera e Rombach (1994) definem algumas características que devem ser descritas para desenvolver o objetivo de maneira clara, sendo elas: **propósito**, **problema**, **objeto de estudo** e **ponto de vista**. Para descrever a questão de pesquisa, essas características foram adaptadas e podem ser observadas no Quadro 1.

Visando compreender a análise e a interpretação de dados relacionados ao desempenho de um produto de *software*, formulou-se a seguinte questão de pesquisa, principal norteadora deste trabalho:

Como analisar dinamicamente a característica de qualidade de eficiência de desempenho

Quadro 1 – Características de Definição da Questão de Pesquisa

Característica	**Valor**
Propósito	Caracterizar
Foco	Qualidade da eficiência de desempenho
Objeto de estudo	Produto de software
Ponto de vista	Pesquisadores
Contexto	Ambiente *web*

Fonte: Adaptado de Basili, Caldiera e Rombach (1994)

para apoiar a tomada de decisão sobre a implantação de versões de produto de software em ambiente web?

1.4 Objetivos

Este trabalho tem como objetivo geral analisar o uso de recursos computacionais de diferentes versões do software de forma automatizada visando identificar a variação de desempenho utilizando diferentes técnicas estatísticas. Para alcançar o objetivo geral, foi importante a divisão em objetivos específicos, sendo ele:

- Levantar fundamentação teórica do trabalho focado em tópicos de experimentação de *software*, qualidade de *software* e técnicas que permitam análise do desempenho de produtos de *software*;
- Realizar um mapeamento das ferramentas que serão utilizadas para coleta e análise dos dados referentes ao desempenho do produto de *software*;
- Planejar o estudo observacional;
- Coletar dados referente ao desempenho do produto de *software* da organização selecionada;
- Analisar os dados coletados, e
- Documentar os resultados obtidos.

1.5 Organização do Trabalho

A organização deste trabalho está estrutura da seguinte maneira:

- **Capítulo 1 - Introdução:** Nesta introdução, apresentou-se o trabalho compreendendo tópicos de contextualização, problemática envolvida, questão de pesquisa e os objetivos que se pretende alcançar com a realização do trabalho;

- **Capítulo 2 - Referencial Teórico:** Nesse capítulo, é apresentado o referencial teórico utilizado como fundamentação do presente trabalho. Nele, constam tópicos como: Experimentação de Software com foco em Estudo Observacional, Qualidade de Software e Comparação de desempenho;
- **Capítulo 3 - Revisão Estruturada da Literatura**: Aborda a metodológica investigativa empregada para coleta do material bibliográfico utilizado neste trabalho, descrevendo os processos de seleção de artigos e das técnicas que serão utilizadas durante a execução do estudo observacional;
- **Capítulo 4 - Estudo Observacional:** Descrição da estratégia de pesquisa adotada, que tem como objetivo principal estabelecer um protocolo para o estudo observacional realizado e a documentação dos resultados obtidos. Esse protocolo compreende elementos essenciais da pesquisa, tais como a identificação e explicação do problema a ser resolvido, dos objetivos a serem alcançados no estudo observacional, bem como dos métodos de coleta e análise dos dados, e
- **Capítulo 5 - Conclusão:** Oferece uma visão geral sobre os resultados obtidos pela execução da pesquisa, além de limitações encontradas e sugestões de seguimentos para pesquisas futuras.

1.6 Cronograma e Fluxo de atividades

1.6.1 Fluxo de atividades da primeira etapa do trabalho

Descrição dos principais processos requeridos para elaboração da primeira etapa do TCC, de acordo com a Figura 2.

- **Contextualização sobre experimentação de software**: Etapa inicial da monografia, onde é necessário contextualizar o conhecimento no âmbito de experimentação de software mediante livros, artigos e vídeos. Esse tópico é necessário para o entendimento do processo da abordagem científica que será utilizada na monografia;
- **Contextualização sobre qualidade de software**: Consumir materiais recomendados pelo orientador, relacionados ao modelo QRapids (MARTíNEZ-FERNáNDEZ et al., 2018; MARTíNEZ-FERNáNDEZ et al., 2019) de qualidade estática de software, bem como um mapeamento sistemático sobre métricas de qualidade de *software* (LóPEZ et al., 2022). Por meio dessas leituras, foi possível adquirir uma visão abrangente do estado da arte dos estudos relacionados ao tema; identificar as barreiras que seriam encontradas na busca por materiais relacionados a métricas dinâmicas, e aprimorar a diversidade de termos que seriam posteriormente utilizados para compor a *string* de busca;

- **Definição do GQM**: O GQM é um acrônimo para *Goal* (Objetivo), *Question* (Questão) e *Metrics* (Métricas). O presente método estabelece um arcabouço hierárquico que culmina na formulação de um modelo de medição voltado para um conjunto específico de questões, bem como na definição de diretrizes para a interpretação dos dados resultantes dessas medições (BASILI; CALDIERA; ROMBACH, 1994). Nesse sentido, é necessário definir o GQM para a elaboração da questão de pesquisa do trabalho, bem como os objetivos específicos e as métricas utilizadas para alcançá-los;

- **Construção da *String* de Busca**: Realizar levantamento e refinamento dos termos a serem empregados na busca de artigos acadêmicos, a fim de utilizá-los como fundamentação teórica no desenvolvimento da presente monografia. O propósito desse procedimento é identificar modelos, ferramentas e recursos que podem auxiliar na análise de métricas dinâmicas de software;

- **Seleção de artigos**: Realizar a seleção de quais artigos serão utilizados como referência bibliográfica, selecionando-os a partir do título ou resumo e, posteriormente, a partir da introdução ou conclusão com critérios de inclusão e exclusão bem definidos;

- **Leitura completa do material selecionado**: Realizar o exame minucioso do conteúdo integral dos artigos selecionados, com o intuito de compreender de forma abrangente os modelos empregados, os contextos nos quais foram utilizados e suas respectivas nuances. Quando necessário, foi efetuado o rastreamento das referências citadas, a fim de obter pleno domínio do conhecimento essencial para a aplicação dos modelos identificados;

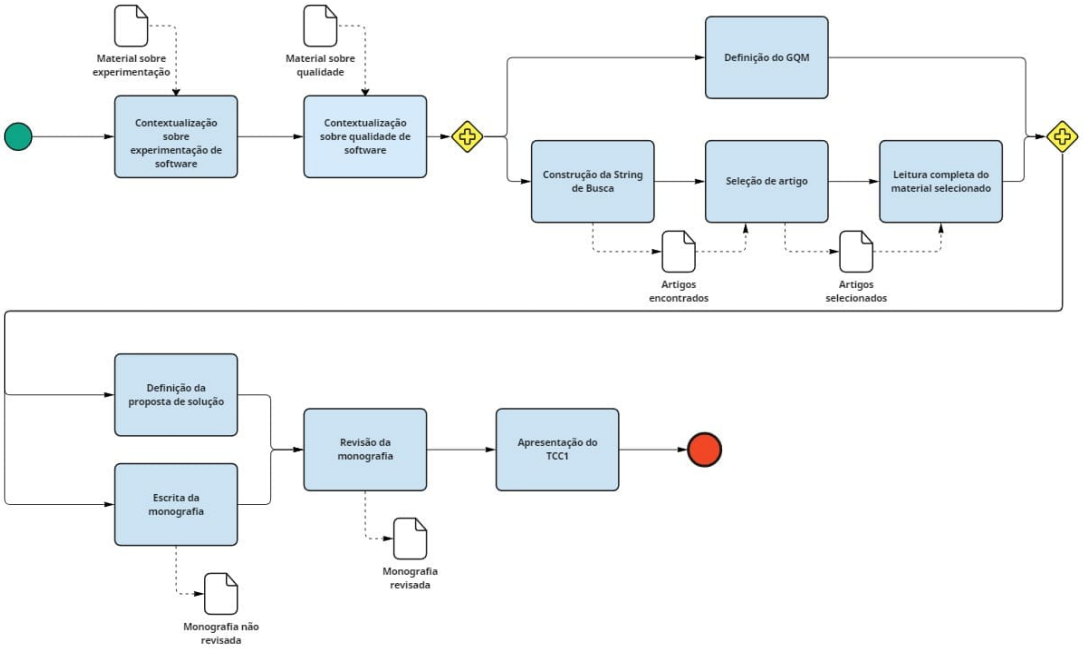
- **Definição da proposta de solução**: Definir os modelos a serem aplicados, o produto de software a ser utilizado como objeto de estudo, as ferramentas a serem empregadas, os limites estabelecidos para a extensão do estudo, bem como a estratégia empírica utilizada para assegurar a viabilidade do estudo;

- **Escrita da monografia**: Iniciar a escrita a partir dos materiais levantados, contemplando tópicos de: introdução, referencial teórico, revisão estruturada da literatura, proposta de estudo observacional e condição do trabalho;

- **Revisão da monografia**: Realizar uma revisão minuciosa do material escrito, com o objetivo de identificar e atualizar pontos que necessitam de atualização, levando em consideração os novos conhecimentos adquiridos durante o progresso do trabalho, e corrigindo quaisquer informações imprecisas que podem estar presentes, e

- **Apresentação do TCC 1**: Apresentar a primeira parte do trabalho de conclusão de curso à banca avaliadora.

Figura 2 – Fluxo de atividades referente ao TCC 1



Fonte: Autores

1.6.2 Fluxo de atividades da segunda etapa do trabalho

Descrição dos principais processos requeridos para elaboração da segunda etapa do TCC, de acordo com a Figura 3.

- **Evolução do TCC a partir das considerações da banca:** essa etapa diz respeito às correções e evoluções identificadas no trabalho a partir das considerações da banca acerca da primeira etapa do estudo.

- **Construção do ambiente de testes com as ferramentas necessárias:** É relevante a configuração dos ambientes de teste com as especificações necessárias do produto a ser observado.

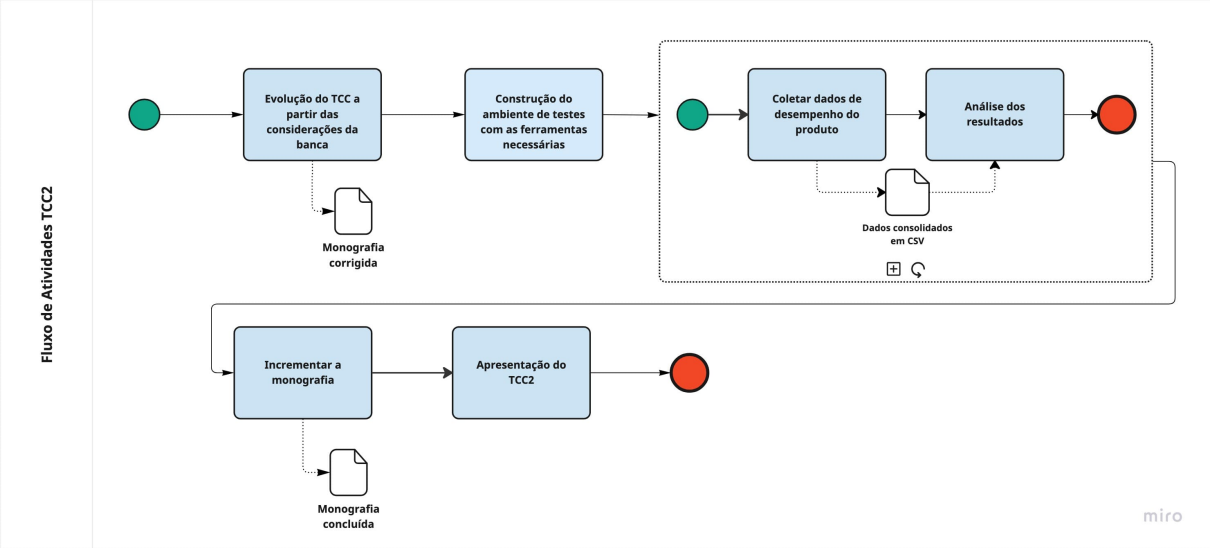
- **Coleta de dados sobre o desempenho do produto:** São coletados dados de desempenho do produto a partir dos testes de cargas realizados.

- **Análise dos resultados:** Esse subprocesso trata da análise dos resultados a partir da coleta de dados das etapas anteriores. É de interesse do estudo que essa análise seja capaz de responder às perguntas levantadas no GQM. Para assim, responder à questão de pesquisa.

- **Incrementar a monografia:** A partir do processo de desenvolvimento da solução realizado, é necessária a documentação das etapas e dos resultados obtidos.

- **Apresentação do TCC 2:** Apresentar o TCC à banca avaliadora com o estudo em sua plenitude.

Figura 3 – Fluxo de atividades referente ao TCC 2



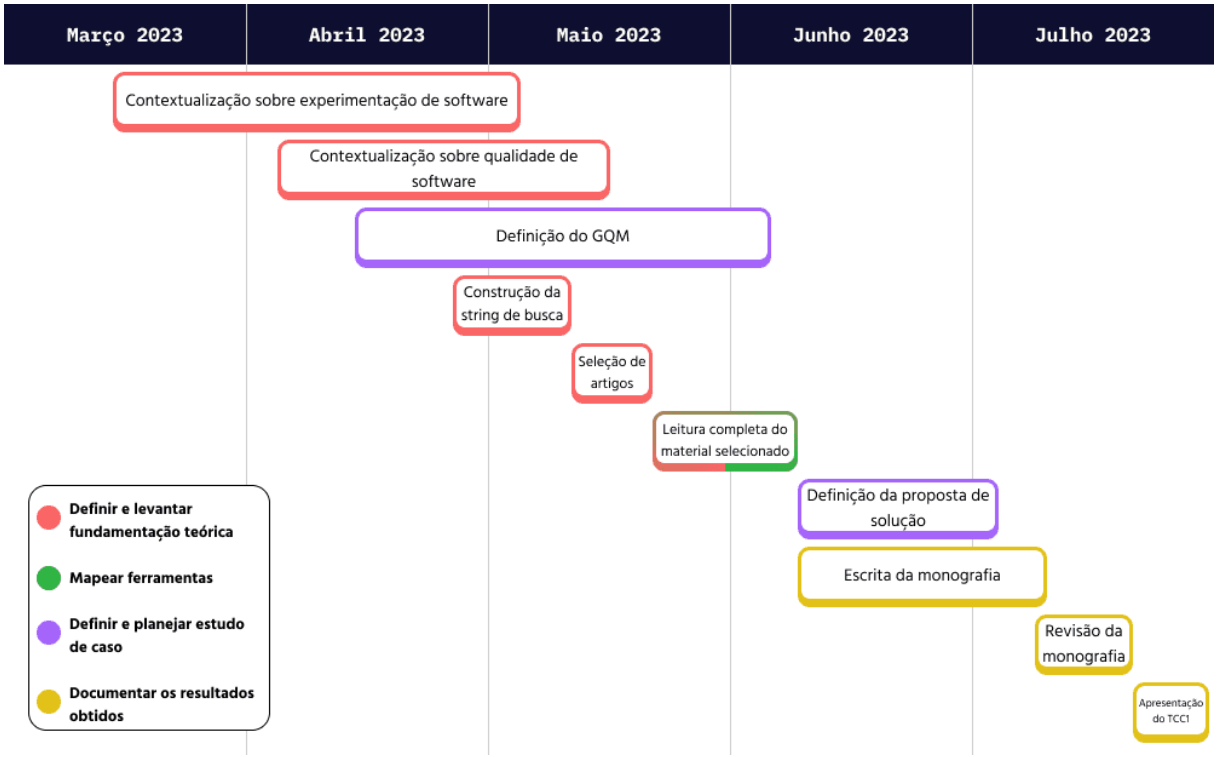
Fonte: Autores

1.6.3 Cronograma das etapas

A Figura 4 representa o cronograma relativo às atividades definidas para a primeira etapa do TCC. Os cronogramas estão representados em uma ordem de grandeza mensal, e categorizados em cores conforme os objetivos específicos do trabalho.

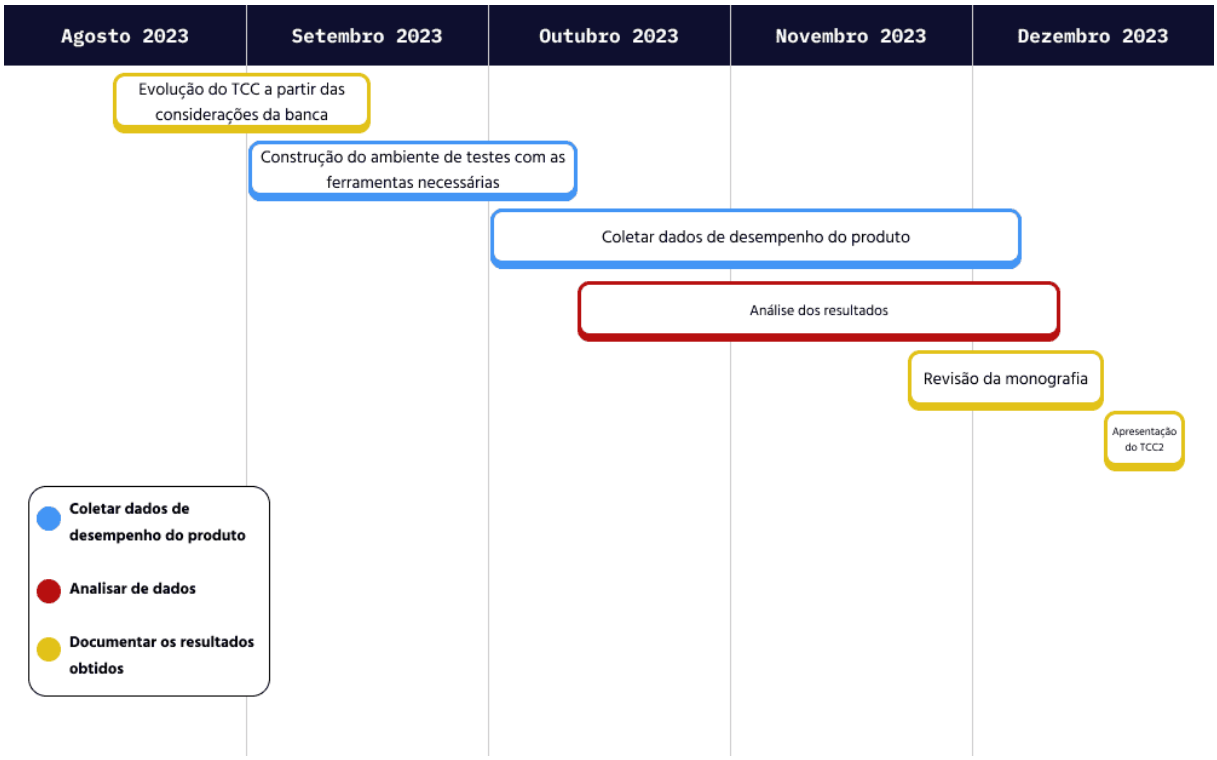
Já a Figura 5 representa o cronograma relativo às atividades definidas para a segunda etapa do TCC.

Figura 4 – Cronograma referente ao TCC 1



Fonte: Autores

Figura 5 – Cronograma referente ao TCC 2



Fonte: Autores

2 Referencial Teórico

Neste capítulo, serão apresentados os conceitos teóricos fundamentais que embasam esta monografia, com o intuito de proporcionar uma compreensão das decisões tomadas que levaram à definição da proposta do estudo.

2.1 Experimentação em Engenharia Software

De acordo com Wohlin et al. (2012), a experimentação oferece uma abordagem sistemática, disciplinada, quantificável e controlada para investigar um fenômeno de interesse. O principal motivo para utilizar experimentação em engenharia de *software* é poder identificar e entender a relação entre diferentes fatores ou variáveis, validando ideias pré-concebidas e evoluindo o entendimento sobre a área (WOHLIN et al., 2012).

Existem diferentes tipos de estudo que seguem a metodologia de experimentação, e que podem ser aplicados conforme o contexto em que está sendo realizado, dentre eles estão:

- **Questionário:** Geralmente ocorre retroativamente por meio da aplicação de questionários de opinião ou entrevistas a uma amostra da população em estudo. O objetivo dessas abordagens é obter o ponto de vista das pessoas envolvidas sobre uma determinada ferramenta, processo ou técnica. Essas informações são coletadas com o intuito de compreender e analisar as percepções e experiências dos indivíduos, auxiliando no desenvolvimento e aprimoramento dos elementos estudados;

- **Estudo de caso:** Trata-se de um método observacional, no qual dados de interesse sobre o fenômeno são coletados em seu contexto real, por exemplo, a partir do uso de um produto de software. Não há intervenção do pesquisador durante a observação. Assim, observa-se e monitora-se um atributo específico ou a relação entre múltiplos atributos que quantificam ou qualificam o fenômeno observado. Devido à sua natureza observacional, essa técnica apresenta pouca possibilidade de controle sobre as variáveis do estudo, e

- **Revisão sistemática da literatura:** É uma abordagem metodológica que tem como objetivo coletar e sintetizar evidências empíricas provenientes de diversas fontes. Essa revisão é conduzida de maneira sistemática e objetiva, aplicando-se palavras-chave pertinentes ao escopo da pesquisa em portais de periódicos, bancos de dados acadêmicos e até mesmo na literatura cinzenta. Além disso, a técnica de bola de neve (*snowballing*) pode ser utilizada para expandir a busca por referên-

cias relevantes, permitindo a navegação entre as citações e referências dos trabalhos selecionados anteriormente. Dessa forma, é possível encontrar outros estudos que possam contribuir para a análise abrangente do tema em questão.

2.1.1 Estudo Observacional

A estratégia empírica adotada neste trabalho consistiu em uma adaptação do estudo de caso, onde o fenômeno foi isolado de seu contexto real para ser observado em um ambiente simulado, sendo, portanto, um estudo observacional.

A seguir, serão detalhados os cinco principais processos envolvidos na abordagem de estudo de caso, de acordo com Wohlin et al. (2012): definição, preparação, coleta de dados, análise de dados e relato dos resultados. Vale ressaltar que a condução do estudo de caso é flexível, podendo haver iterações repetidas dos processos, conforme a necessidade de coletar e analisar uma quantidade maior de dados (WOHLIN et al., 2012).

2.1.1.1 Definição

Para executar um estudo de caso de maneira sistemática e efetiva, é fundamental estabelecer alguns elementos de planejamento. Os elementos essenciais a serem considerados são:

- **Objetivo**: A formulação do objetivo é caracterizada por uma abordagem mais abrangente e menos precisa em comparação a outros métodos de pesquisa mais estruturados. Inicialmente, assemelha-se a um foco principal, que se desenvolverá ao longo do estudo, e estabelece os objetivos a serem alcançados por meio da pesquisa;

- **Caso**: Define o objeto de estudo da pesquisa, o qual pode ser um projeto de software, um indivíduo, um grupo, um processo, um produto, dentre outros. É importante ressaltar que no estudo de caso o objeto selecionado deve possuir um contexto real passível de observação e análise, porém o estudo observacional realizado extraiu o caso de seu contexto real e o analisou de forma isolada.

- **Trabalhos Relacionados**: É necessário entender um contexto geral sobre o estado atual de conhecimento sobre a área em que o estudo será realizado para que seja possível utilizar o conhecimento adquirido em outros trabalhos como ponto de partida. É uma prática comum utilizar outros estudos como referência para permitir que o leitor tenha uma compreensão do ponto de vista considerado na pesquisa;

- **Questões de Pesquisa**: É essencial estabelecer as perguntas que orientarão o estudo e direcionarão a investigação. Ao longo da pesquisa, os pesquisadores devem constantemente questionar se as decisões tomadas estão contribuindo para responder

às perguntas previamente definidas. Esse processo contínuo de reflexão garante a coerência e a relevância dos passos adotados na busca dos "achados";

- **Métodos**: A fim de responder às questões de pesquisa, é imprescindível desenvolver estratégias para a coleta de dados pertinentes ao estudo. É necessário definir quais dados serão coletados, como serão obtidos e com qual frequência, e

- **Seleção**: Deve-se definir também qual será a fonte dos dados a serem coletados, levando em consideração os elementos previamente estabelecidos.

2.1.1.2 Preparação e Coleta de Dados

Ao realizar um estudo de caso, é recomendável utilizar diferentes fontes de dados, a fim de aumentar a validade e evitar viés nos resultados. Dessa forma, é possível empregar a técnica de triangulação, que consiste em obter conclusões a partir de diferentes fontes de dados, bem como comparar os resultados provenientes de diferentes fontes. Ao realizar a triangulação, o estudo pode se beneficiar da convergência das evidências, fortalecendo a confiabilidade das descobertas e proporcionando uma visão mais abrangente e consistente do fenômeno em análise.

De acordo com Lethbridge, Sim e Singer (2005), existem três categorias de técnicas de coleta de dados, classificadas segundo o grau de contato humano necessário:

- **Primeiro grau**: Requer acesso direto à população em estudo. Algumas técnicas disponíveis nesse grau incluem entrevistas, questionários e debates;

- **Segundo grau**: Requer acesso ao ambiente em funcionamento, sem necessariamente envolver contato direto com os participantes. Nesse grau, as técnicas são observacionais, como observar a equipe durante o desenvolvimento ou coletar métricas do sistema em execução, e

- **Terceiro grau**: Utiliza apenas artefatos e registros previamente desenvolvidos como fonte de dados, como documentação do sistema e código-fonte.

Cada grau apresenta suas próprias limitações, custos e tipos de dados que podem ser coletados. Técnicas que exigem maior contato interpessoal exigem o desenvolvimento de uma relação mais estreita entre o pesquisador e a população estudada, além de tenderem a ter um custo mais elevado. No estudo observacional conduzido neste trabalho foram utilizadas apenas fontes de dados de segundo grau, impedindo assim a triangulação dos dados.

2.1.1.3 Análise de Dados

Para a análise de dados quantitativos, é comum aplicar técnicas estatísticas descritivas, análise de correlação, desenvolvimento de modelos preditivos e testes de hipóteses. Essas técnicas são empregadas para análise dos dados coletados, descrever as relações entre as métricas obtidas em diferentes processos, e determinar se existe uma relação de causa e efeito entre as variáveis coletadas. É fundamental ressaltar que o processo de coleta de dados quantitativos deve ser mantido inalterado, a fim de possibilitar a interpretação conjunta dos dados e garantir a confiabilidade dos resultados observados (WOHLIN et al., 2012).

No caso de dados qualitativos, o objetivo principal é desenvolver hipóteses a partir dos dados coletados, e então utilizar técnicas para confirmar se a hipótese é verdadeira. Para isso, é necessário apresentar todas as informações relevantes em cada etapa do estudo, a fim de que o leitor possa obter as mesmas conclusões a partir dos mesmos dados. É importante ressaltar que a coleta e a análise dos dados devem ocorrer de maneira simultânea, permitindo que as novas descobertas possam aprimorar o processo. Dessa forma, as descobertas obtidas durante a análise dos dados podem influenciar a coleta de dados adicionais ou ajustes na metodologia, resultando em um ciclo iterativo que aprimora continuamente a pesquisa (WOHLIN et al., 2012).

2.1.1.4 Relato

O relato dos achados do estudo de caso é a forma pela qual as descobertas são comunicadas, e é a principal fonte de informação para avaliar a qualidade do estudo. O modelo de relato a ser seguido pode depender do público-alvo para o qual o relato é direcionado. No entanto, independentemente disso, é essencial fornecer detalhes sobre o objeto de estudo, os objetivos do estudo e os métodos e processos utilizados para obter os resultados apresentados.

No contexto do estudo observacional desenvolvido neste trabalho, as discussões apresentadas no Capítulo 4 serão tidas como relato.

2.2 Qualidade de Software

Segundo a ISO/IEC 25000 (2014), o conceito de qualidade de *software* é definido como: “A capacidade do produto de software de satisfazer necessidades declaradas e implícitas quando usado sob condições especificadas”. Esta definição, dada sua natureza abstrata, apresenta limitações para sua operacionalização direta, o que tem conduzido ao desenvolvimento de diferentes modelos para garantia de qualidade de software ao longo dos últimos anos. Alguns modelos como ISO/IEC 9126-1 (2001) ou seu sucessor ISO/IEC

25010 (2011) têm o propósito de reunir definições conceituais, taxonômicas, a respeito da qualidade de software.

Estes modelos seguem uma estrutura hierárquica composta de características e suas respectivas subcaracterísticas que descrevem as visões de qualidade interna, externa e em uso. Entretanto, essa estrutura composta pelas características e subcaracterísticas continua apresentando limitações quanto a como quantificá-las, qualificá-las e interpretá-las. Existem alguns modelos de qualidade amplamente difundidos que buscam solucionar essa limitação, como, por exemplo, o Squale (MORDAL-MANET et al., 2009), Quamoco (WAGNER et al., 2016) ou Q-rapids (MARTíNEZ-FERNáNDEZ et al., 2019). Porém, nenhum desses modelos analisaram medidas de eficiência de desempenho.

O modelo da ISO/IEC 25010 (2011) define as características de qualidade do produto e a qualidade em uso do software. Esse modelo de qualidade engloba treze características principais, sendo que oito tratam sobre qualidade do produto e cinco sobre qualidade em uso do *software*. As características de qualidade em uso dizem respeito ao comportamento percebido com a operação do produto de software em um determinado contexto de uso. Já as características da qualidade de produto auxiliam quanto à observação tanto das propriedades estáticas do *software* quanto de algumas propriedades dinâmicas de um sistema computacional ou produto de *software*.

As definições detalhadas dessas características e subcaracterísticas estão documentadas no Quadro 2. Segundo Ouhbi et al. (2014), trata-se do modelo de qualidade de *software* mais estudado ao longo do tempo. A concepção desta norma foi fortemente influenciada pelos seminais e precursores modelos de McCall (MCCALL; RICHARDS; WALTERS, 1977) e Boehm (BOEHM, 1978). É importante salientar que essa norma traz conceitos importantes para a definição da qualidade a partir da perspectiva do sistema em execução, porém, ainda é necessário um processo de medição para definição das métricas a serem coletadas e como elas se relacionam com os conceitos definidos na ISO.

Para observar a característica de qualidade de eficiência de desempenho, é necessário que o sistema esteja em execução. A norma ISO/IEC 25010 (2011) define a característica relevante relacionada às propriedades dinâmicas do produto, acompanhada de suas respectivas subcaracterísticas. Conforme a norma, a característica de eficiência de desempenho refere-se ao desempenho relativo à quantidade de recursos utilizados sob condições especificadas. Esses recursos podem incluir outros produtos de software, a configuração de software e hardware do sistema, bem como materiais relacionados. Já as subcaracterísticas de eficiência de desempenho podem ser definidas como:

- Tempo de resposta: avalia o grau em que os tempos de resposta e processamento, juntamente com as taxas de produtividade, atendem aos requisitos estabelecidos ao executar as funções do produto ou sistema.

- Utilização de recursos: analisa o grau em que as quantidades e os tipos de recursos empregados por um produto ou sistema, ao realizar suas funções, atendem aos requisitos estabelecidos.

- Capacidade: avalia o grau em que os limites máximos de um parâmetro específico de um produto ou sistema atendem aos requisitos estabelecidos. Esses parâmetros podem incluir o número de itens que podem ser armazenados, o número de usuários simultâneos suportados, a largura de banda de comunicação, o rendimento das transações e o tamanho do banco de dados.

A medição do *software* desempenha um papel fundamental na viabilização da análise do mesmo. Conforme mencionado por Martínez-Fernández et al. (2019), a análise de *software* tem como objetivo principal o desenvolvimento do produto orientado a dados, utilizando informações coletadas durante o processo de desenvolvimento e do uso do produto. Essa atividade é de extrema importância, uma vez que proporciona subsídios essenciais para os engenheiros e gestores tomarem decisões embasadas na análise dos dados obtidos.

Segundo Wagner et al. (2016), a ISO IEC 25010 não dispõe de mecanismos de operacionalização entre as métricas, medidas e suas definições conceituais. Porém, essa operacionalização é indicada em alguns modelos mais recentes como Squale (MORDALMANET et al., 2009), o próprio Quamoco (WAGNER et al., 2016), ou também o Q-rapids (MARTíNEZ-FERNáNDEZ et al., 2019). Para que isso ocorra, é necessário compreender às fórmulas de cálculo, mecanismos de agregação, valores de referência para que essas informações possam auxiliar as análises do time de desenvolvimento. Com esses estudos, foram definidos valores de referência para medidas e características de manutenibilidade, confiabilidade, entre outros. Entretanto, no caso de eficiência de desempenho, depende da especificidade de cada caso, da infraestrutura computacional e de requisitos não funcionais específicos ao contexto.

2.3 Análise de Desempenho

Problemas de desempenho podem acarretar uma série de desafios para produtos de *software*, tanto em pequena como em larga escala. Essas questões vão desde a perda de confiança por parte dos usuários e investidores até prejuízos significativos, podendo chegar a cifras bilionárias, dependendo da gravidade da falha e do tamanho do *software* em questão (LIAO et al., 2020).

Para identificar tais problemas, uma abordagem comumente utilizada é a realização de testes de carga, nos quais são simulados os comportamentos reais dos usuários na aplicação em análise. São criados cenários de teste baseados no comportamento esperado

Quadro 2 – Modelo de qualidade de *software* de acordo com ISO IEC 25010

Característica	**Subcaracterísticas**
Eficiência do Desempenho (Qualidade do produto)	Tempo de resposta Utilização de recursos Capacidade
Compatibilidade (Qualidade do produto)	Co-existência Interoperabilidade
Usabilidade (Qualidade do produto)	Adequação Facilidade de uso Operabilidade Proteção ao erro do usuário Estética da Interface de usuário Acessibilidade
Confiabilidade (Qualidade do produto)	Maturidade Disponibilidade Tolerância a falhas Recuperabilidade
Segurança (Qualidade do produto)	Confidencialidade Integridade Não repudiação Responsabilidade Autenticidade
Manutenibilidade (Qualidade do produto)	Modularidade Reusabilidade Analisabilidade Modificabilidade Testabilidade
Portabilidade (Qualidade do produto)	Adaptabilidade Capacidade de Instalação Capacidade de substituição

Fonte: Adaptado de ISO/IEC 25010 (2011)

dos usuários do *software* em produção. Esses testes são aplicados em diferentes versões do produto, coletando-se métricas relevantes. Em seguida, é feita uma comparação dessas métricas visando determinar se houve melhora ou piora nos resultados entre as versões testadas (JIANG et al., 2009).

2.3.1 Modelos de Análise de Desempenho

Devido ao estudo ter como base um produto de *software* proprietário, existem limitações quanto ao acesso às estruturas internas, regras de negócio e código-fonte. Portanto, existem limitações quanto aos modelos de análise de desempenho aplicáveis.

De acordo com Gao et al. (2016), os modelos de análise de desempenho podem ser divididos em três categorias distintas:

- **Modelos baseados em fila:** Neste tipo de modelo, os recursos do sistema são tratados como filas, e busca-se prever o tempo que cada cenário de teste levará em cada uma dessas filas. Por exemplo, ao receber uma requisição do usuário, o modelo estimará quanto tempo essa requisição irá aguardar até ser recebida pelo processador, e quanto tempo levará para ser processada completamente. Esse tipo de modelo requer um conhecimento detalhado sobre a estrutura interna do *software*, sendo denominado como modelo caixa-branca;

- **Modelos baseados em mineração de dados:** São aplicadas técnicas estatísticas ou de inteligência artificial para relacionar uma ou mais métricas coletadas durante a execução dos cenários de teste. Por exemplo, pode-se estabelecer uma relação entre a quantidade de requisições em cada fluxo da aplicação e o uso de um recurso computacional específico, para prever como o sistema iria se comportar em outros cenários. Ao contrário dos modelos de caixa-branca, esses modelos não requerem conhecimento detalhado da estrutura interna do *software*, sendo assim denominados modelos de caixa-preta, e

- **Modelos baseados em regras:** Diferentemente das duas categorias anteriores, esse tipo de modelo não visa prever o comportamento do sistema com base nos cenários de teste. Em vez disso, o objetivo é aplicar regras pré-definidas às métricas coletadas durante a execução dos testes e analisá-las para compreender o estado atual do sistema. Existem diversos modelos nessa categoria, sendo o mais simples deles o cálculo da média e mediana das métricas coletadas.

Devido às limitações apresentadas, foram selecionados, para desenvolvimento do estudo observacional, os modelos **baseados em mineração de dados** e os **baseados em regras**. Para o modelo baseado em mineração de dados foi escolhido o método de regressão numérica **Floresta Aleatória** (Seção 2.3.2), e baseado em regras o método de **Gráficos de Controle** (Seção 2.3.3).

2.3.2 Floresta Aleatória

De acordo com Breiman (2001), a técnica estatística de Floresta Aleatória, utilizada para realizar regressão numérica, consiste na combinação de árvores de decisão formadas por vetores de amostragens independentes, mas com a mesma distribuição. Ou seja, em vez de gerar uma única árvore de decisão que engloba toda a amostragem coletada, vários conjuntos de dados são formados por meio de combinações aleatórias a partir do conjunto inicial, resultando na criação de uma árvore de decisão para cada conjunto. A previsão é feita aplicando-se um novo conjunto de dados em cada uma das árvores de decisão geradas, e o resultado indicado pela maioria das árvores é considerado correto.

2.3.2.1 Regressão Numérica

Conforme Maia (2019), a regressão numérica estatística é o estudo da relação entre uma variável dependente e uma ou mais variáveis explanatórias, ou independentes. Seu objetivo é prever o valor da variável dependente com base na variação das variáveis independentes. A regressão pode ser simples, quando envolve apenas uma variável dependente e uma variável independente, ou múltipla, quando há mais de uma variável independente.

2.3.2.2 Árvore de Decisão

Conforme Cutler, Cutler e Stevens (2012), as árvores de decisão são estruturas recursivas que dividem os dados de entrada de forma binária, com base em variáveis individuais. O primeiro nó da árvore, conhecido como "nó raiz", contém todos os dados de entrada. Os nós que não possuem divisão são chamados de "nós terminais", e cada um deles apresenta uma opção de valor de saída. Cada nó que não é terminal, divide-se em dois, com base em uma condição imposta. Valores que atendem a condição são direcionados para a ramificação da esquerda, enquanto valores que não atendem a condição são direcionados para a ramificação da direita. Esse processo de divisão e ramificação é repetido até que sejam alcançados os nós terminais, que fornecem as saídas desejadas.

2.3.2.3 Exemplo de Floresta Aleatória

Utilizando dados fictícios referentes a alguns meses do ano (variáveis independentes) e sua respectiva incidência de chuva (variáveis dependentes), é possível definir uma relação capaz de estimar a incidência de chuva em outros períodos.

Na Figura 6, é possível observar um exemplo simplificado de árvore de decisão, onde são utilizados todos os dados do conjunto para a construção de uma única árvore.

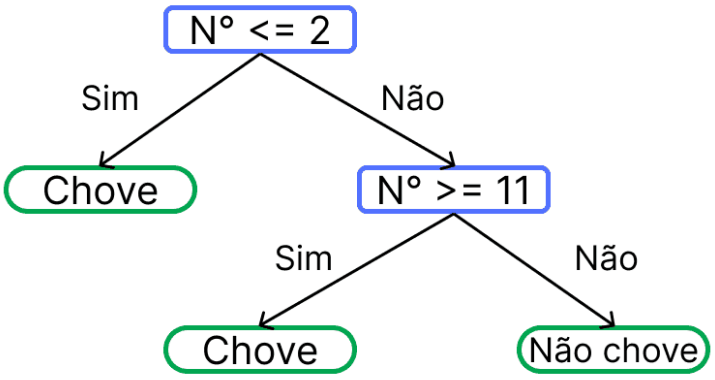
Na Figura 7, é possível observar um exemplo de floresta aleatória. A partir dos dados iniciais são gerados dois novos conjuntos de dados e consequentemente duas árvores de decisão distintas. Portanto, para definir a incidência de chuva em algum mês, é necessário aplicar a variável independente a ambas as árvores de decisão.

2.3.2.4 Aplicação de Floresta Aleatória

Para o contexto do estudo, foram realizados testes de carga para simular o comportamento de usuários reais fazendo requisições ao sistema. As informações coletadas a partir de uma versão considerada como **base** foram utilizadas para gerar o algoritmo, sendo as variáveis independentes os dados referentes à quantidade de requisições feitas para cada um dos *endpoints* do fluxo escolhido em um determinado período de coleta, e as variáveis dependentes sendo os estados de uso dos recursos computacionais ao final deste período.

Figura 6 – Exemplo de árvore de decisão

Mês	Número do Mês	Chove?
Fevereiro	2	Chove
Maio	5	Não chove
Agosto	8	Não chove
Novembro	11	Chove



Fonte: Autores

Figura 7 – Exemplo de floresta aleatória

Mês	Número do Mês	Chove?
Fevereiro	2	Chove
Fevereiro	2	Chove
Maio	5	Não chove
Agosto	8	Não chove

Mês	Número do Mês	Chove?
Maio	5	Não chove
Agosto	8	Não chove
Agosto	8	Não chove
Novembro	11	Chove

Fonte: Autores

O mesmo teste de carga é executado na versão de produto que se deseja comparar, a chamada versão **alvo**, mantendo o intervalo de coleta e as métricas coletadas. Ao aplicar no algoritmo a contagem de requisições feitas no teste da versão alvo, é possível obter como aquelas mesmas requisições iriam afetar o sistema na versão base.

Portanto, é necessário comparar as métricas de uso de recursos computacionais coletadas no teste da versão alvo com as métricas retornadas do algoritmo de floresta

aleatória desenvolvido a partir da versão base. Para isso, foram utilizadas duas técnicas que haviam sido aplicadas em um contexto similar por Gao et al. (2016):

- **Erro Percentual Absoluto Médio:** Este método de avaliação foi empregado no estudo mencionado para determinar o erro absoluto médio para cada conjunto de dados usado no treinamento do algoritmo de floresta aleatória e as previsões geradas por ele. Contudo, no contexto deste estudo, o método foi utilizado para calcular a porcentagem média absoluta de variação para cada amostra utilizada na comparação entre a versão alvo e as previsões das chamadas da versão alvo, aplicadas ao algoritmo de floresta aleatória treinado com a versão base.
- **Cliff Delta:** No contexto do estudo mencionado, essa técnica foi utilizada para quantificar o grau de diferença quando analisados todos os valores reais em relação aos valores previstos pelo algoritmo de floresta aleatória, utilizando limites estabelecidos pela própria técnica, conforme apresentado na Tabela 1, onde x representa o valor do Cliff Delta calculado, cujo valor absoluto pode variar entre 0 e 1. Gao et al. (2016) utilizou para avaliar a capacidade do algoritmo em prever os resultados do mesmo teste usado para o treinamento. No entanto, neste estudo específico, a técnica foi empregada para avaliar o grau de diferença entre os dados de testes realizados em versões distintas, ou seja, quando comparando todas as amostras de ambos conjuntos de dados.

Tabela 1 – Limites para interpretação dos valores de Cliff Delta

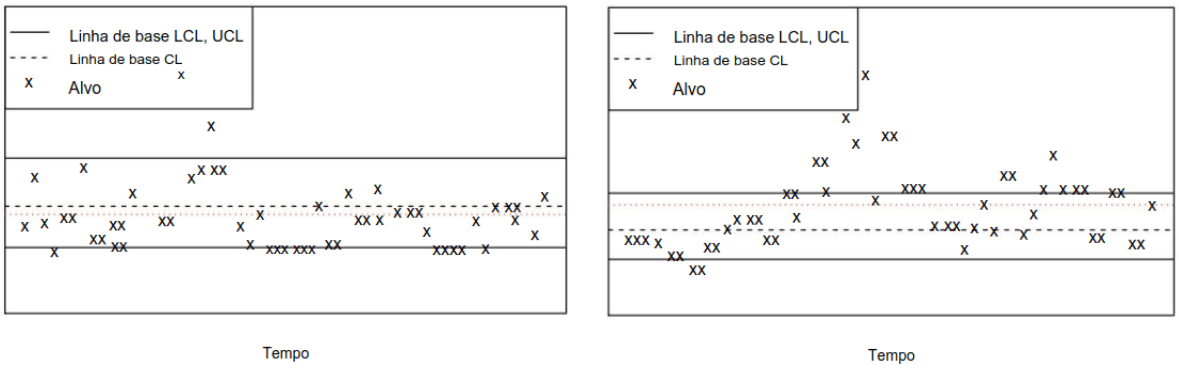
Valor	Grau de diferença
$x < 0,147$	Trivial
$0,147 \leq x \leq 0,33$	Pequena
$0,33 \leq x \leq 0,474$	Média
$0,474 \leq x$	Grande

Fonte: Adaptado de Gao et al. (2016)

2.3.3 Gráficos de Controle

Segundo Nguyen et al. (2012), os gráficos de controle são amplamente empregados na detecção de problemas em processos de fabricação, nos quais as matérias-primas são consideradas entradas e os produtos acabados como saídas. O objetivo dos gráficos de controle consiste em fornecer uma avaliação automática para determinar se um desvio em um processo é decorrente de causas comuns, tais como flutuações nas entradas do processo, ou se é resultado de causas especiais, como a presença de defeitos. Por meio da análise dos dados coletados e da interpretação visual, os gráficos de controle permitem que sejam tomadas decisões adequadas com base na natureza dos desvios identificados.

Figura 8 – Exemplos de gráficos de controle



(a) Gráfico de controle normal (b) Gráfico de controle fora de controle

Fonte: Adaptado de (NGUYEN et al., 2012)

Dois exemplos visuais de gráficos de controle podem ser encontrados na Figura 8. Onde o item 8a representa um gráfico normal, onde o conjunto de teste alvo se assemelha bastante ao conjunto de teste base. Já o item 8b representa um gráfico fora de controle, onde o conjunto de teste alvo se diferencia consideravelmente do conjunto de teste base.

2.3.3.1 Construção do gráfico de controle

Segundo Nguyen et al. (2012), um gráfico de controle pode ser construído utilizando dois diferentes conjuntos de dados, que seria um conjunto de base e um conjunto alvo, que, no caso desse estudo representariam a versão livre de problemas (base) e a versão com problemas de desempenho inseridos (alvo). A partir do conjunto de dados base, é possível definir os limites de controle onde a linha central se configura como a média das amostras do teste base em um determinado período de tempo, podendo ser de alguma métrica específica como uso de *CPU* ou uso de memória. A partir disso, é possível inferir os limites superiores e inferiores de diferentes maneiras.

Para a definição dos limites inferiores e superiores, existem diferentes técnicas que podem ser utilizadas, dentre elas, estão:

- Escolha de três desvios padrão a partir média encontrada, e
- 1^{o}, 5^{o} ou 10^{o} percentis do limite de controle inferior, e 90^{o}, 95^{o} ou 99^{o} percentis do limite de controle superior, por exemplo, se uma leitura de uso de *CPU* retorna resultados percentuais de uso de 12, 15, 18, 21, 25, 27, 30, 33, 37, 40, 43, 47, isso significa que o décimo percentil corresponde a 15 e o nonagésimo percentil corresponde a 43, e a média desse conjunto seria 29.

Cabe salientar que, apesar da denominação, os gráficos de controle vão além de uma mera técnica de visualização, configurando-se como uma abordagem estatística que

viabiliza a geração de um índice de medição denominado taxa de violação. A taxa de violação, derivada da aplicação dos gráficos de controle, representa um indicador estatístico que quantifica a proporção dos dados alvo que se encontram fora dos limites de controle.

A definição do limite da taxa de violação pode ser determinada pelo operador, no entanto, um limite apropriado deve ser superior ao valor esperado com base nos limites de controle. Por exemplo, se os limites de controle são estabelecidos pelo décimo e nonagésimo percentil do conjunto de dados de teste base, espera-se uma taxa de violação de 20%. Portanto, é recomendável escolher um valor para a taxa de violação igual ou superior a 25%.

2.3.3.2 Premissas do gráfico de controle

Segundo Nguyen et al. (2012), existem duas premissas básicas para os gráficos de controle, que são também um desafio quanto ao seu cumprimento, porém, de extrema importância para validação do teste de comparação de desempenho, são elas:

- **Entrada de processo invariável:** Observa-se que a saída do processo tende a variar em conformidade com as flutuações da entrada do mesmo. Caso ocorra um aumento na taxa de entrada do processo, consequentemente, a taxa de violação aumentará e um alarme será ativado, independentemente da reação do sistema frente às variações da entrada. É importante ressaltar que tais alarmes podem resultar em falsos positivos, já que não indicam a ocorrência de um problema real. Assim sendo, torna-se imprescindível estabelecer como primeira condição para a aplicação dos gráficos de controle a estabilidade da entrada do processo, e

- **Normalidade da saída do processo:** É observado que a relação entre a saída do processo e sua entrada costuma ser linear. Essa linearidade resulta em uma distribuição normal da saída do processo, que constitui a base estatística fundamental subjacente aos gráficos de controle. No entanto, em alguns processos de fabricação, são utilizados diversos tipos de insumos, cada um dos quais produz, individualmente, uma distribuição normal. Entretanto, a combinação dessas entradas resultará em uma distribuição multimodal, a qual não pode ser adequadamente analisada por meio de gráficos de controle.

2.3.3.3 Abordagem utilizando gráfico de controle

Segundo Nguyen et al. (2012), existem diferentes etapas que compõe a construção do gráfico de controle, que se estende de tratamento de dados até chegar na taxa de violação e ter uma resposta para a comparação de desempenho. Essas etapas estão representadas na Figura 10, e são constituídas por:

- **Teste Base e Teste Alvo:** Nesta etapa, é estabelecida a aplicação da carga necessária para realizar os testes de desempenho tanto na versão base quanto na nova versão (alvo). É de extrema importância que ambas as versões sejam submetidas à mesma carga de entrada, frequentemente referida como perfil de carga. Esse perfil de carga descreve o comportamento esperado do sistema uma vez que esteja operacional em produção (AVRITZER; WEYUKER, 1995). O perfil de carga é configurado com base nos cenários de casos de uso e suas respectivas taxas de execução.

  Durante a realização dos testes, é crucial monitorar a aplicação com o intuito de capturar os registros de execução, assim como as métricas de desempenho. No caso específico de gráficos de controle, as métricas coletadas são suficientes para a construção do referido gráfico. Um teste de carga usualmente coleta quatro tipos principais de métricas de desempenho. A saber:

  - Utilização de CPU: Porcentagem de utilização de CPU por máquina,
  - Utilização de memória RAM: Porcentagem de utilização de memória RAM por máquina,
  - Tempo de resposta: Média em segundos do tempo entre o início e fim das requisições HTTP, e
  - Leitura/Escrita no banco de dados: Quantidade de linhas lidas e linhas escritas no banco de dados.

- **Normalização dos dados:** A fim de utilizar os gráficos de controle de maneira adequada, é fundamental atender aos dois pressupostos mencionados na subseção 2.3.3.2, a saber: a entrada de processo invariável e a normalidade da saída do processo. Para cumprir esses pressupostos, é imprescindível realizar uma etapa de pré-processamento dos dados, pois seria ineficiente utilizar diretamente os dados brutos das métricas de desempenho (NGUYEN et al., 2012). Portanto, a seguir são apresentadas as duas etapas necessárias para a normalização dos dados, visando satisfazer os pressupostos para a construção do gráfico de controle e reduzir as incertezas na análise comparativa de desempenho.

  - **Processamento em escala:** Para a condução adequada dos testes de comparação de desempenho, é necessário aplicar a mesma carga à versão base e à versão alvo. No entanto, devido a interferências externas, como variação na conexão de rede, ou fatores internos, como variação no tempo de resposta devido a problemas de desempenho, os dados referente a quantidade de requisições realizadas tendem a variar, tornando inviável a comparação direta entre ambas as versões. Em outras palavras, as discrepâncias observadas nas métricas de desempenho podem ser atribuídas às diferenças na carga de entrada, ao in-

vés de representarem efetivamente alterações relacionadas ao desempenho do código.

A abordagem proposta por Nguyen et al. (2012) consiste em processar os dados de desempenho em escala, levando em consideração as cargas de teste. Em um teste de desempenho, é esperado que as métricas coletadas sejam linearmente proporcionais à intensidade da carga de teste. Em outras palavras, à medida que a carga aumenta, o software é submetido a uma demanda maior, resultando em métricas de desempenho mais elevadas. Portanto, a relação entre os valores das métricas de desempenho e a carga de entrada pode ser descrita por meio da Equação 2.1:

$$c = \alpha * l + \beta \tag{2.1}$$

onde, c é o valor médio das métricas de desempenho nas amostras em um determinado período de uma execução. l é a real carga de entrada naquele período. α e β são os coeficientes do modelo linear.

Após obter essa relação linear entre as métricas de desempenho e a carga de entrada do conjunto de dados do teste base, é gerado um novo conjunto de dados que consiste do teste alvo na mesma escala do teste base, para satisfazer a premissa de entrada de processo invariável na hora da comparação entre os dois testes. A Equação 2.2 é utilizada para trazer os valores do teste alvo para a mesma escala de carga do teste base.

$$C_t = C_b * \frac{\alpha * L_t + \beta}{\alpha * L_b + \beta} \tag{2.2}$$

Onde C_t será o valor da métrica do teste alvo em escala, C_b é o valor da métrica no teste base, os coeficientes α e β são os coeficientes da regressão linear do teste base derivados de 2.1, L_t e L_t são as cargas do teste alvo e teste base, respectivamente. Caso a carga total seja composta por cargas distintas que possuam diferentes impactos nos recursos computacionais, faz-se necessário a utilização de uma regressão linear múltipla, utilizando, então, o somatório de cada α com o valor de cada carga individual.

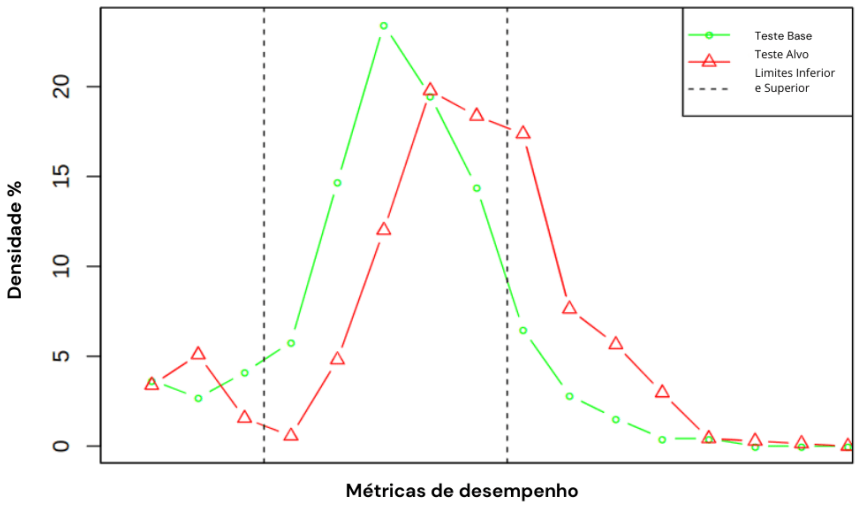
– **Filtro:** A solução proposta por Nguyen et al. (2012) visa apresentar uma abordagem destinada à filtragem das amostras coletadas pelos contadores associados à tarefa secundária em consideração. Essa abordagem demonstra eficácia ao considerar o teste de comparação de desempenho, cujo principal foco é avaliar o desempenho do sistema em situações de carga elevada. Nesse contexto, os contadores de desempenho registram valores significativamente mais altos. Desvios de menor magnitude não são relevantes para a análise em questão.

Para realizar a filtragem, é necessário utilizar um algoritmo simples para detectar os mínimos locais. Os mínimos locais representam o ponto mais baixo entre os dois picos de uma distribuição bimodal. Nesse caso, basta remover os pontos de dados à esquerda dos mínimos locais. Um exemplo desses mínimos locais pode ser visto na Figura 9, sendo representados pelos 3 primeiros triângulos à esquerda do gráfico.

Uma solução alternativa consiste em aumentar a carga à medida que o *hardware* do servidor se torna mais potente. O aumento da carga garantirá que o sistema gaste menos tempo ocioso e mais tempo processando a carga, eliminando assim o pico à esquerda. No entanto, aumentar artificialmente a carga em prol da normalidade compromete o objetivo do teste de comparação de desempenho, e compromete a capacidade de comparar novos testes com testes anteriores.

- **Criação do gráfico de controle e taxa de violação:** Uma explicação mais detalhada sobre esses tópicos pode ser encontrada na subseção 2.3.3.1.

Figura 9 – Gráfico de densidade do teste de 2 versões distintas



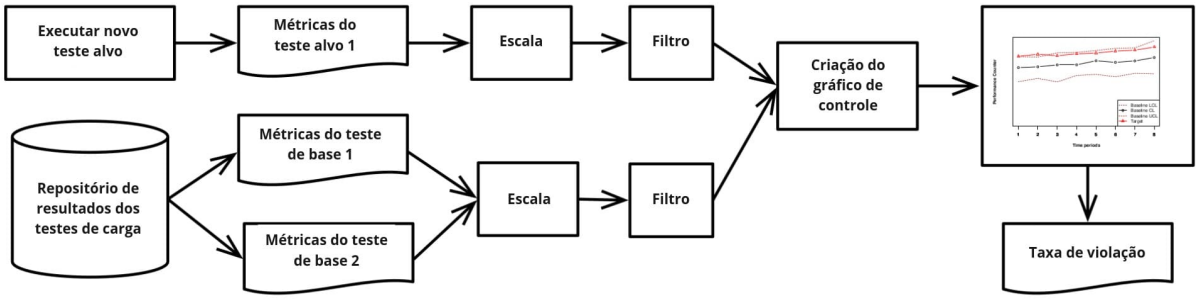
Fonte: Adaptado de (NGUYEN et al., 2012)

2.4 Resumo do capítulo

Este capítulo abordou os conceitos relevantes para o desenvolvimento do estudo, incluindo metodologias de experimentação, qualidade de software e comparação de desempenho de software.

No que diz respeito às metodologias de experimentação, o foco principal foi dado ao estudo de caso, pois a estrutura do estudo observacional utilizada neste trabalho foi

Figura 10 – Abordagem para comparação de desempenho com gráficos de controle



Fonte: Adaptado de (NGUYEN et al., 2012)

baseada nele. Foram apresentados os conceitos e as características do estudo de caso, destacando sua aplicabilidade, seus processos essenciais e as adaptações realizadas no contexto deste trabalho.

Em relação à qualidade de software, foram explorados conceitos gerais do tema e os específicos à característica de eficiência de desempenho. Esses conceitos foram embasados em normas técnicas, e visam fornecer critérios e diretrizes para avaliar a qualidade do desempenho de um software.

Na comparação de desempenho de software, foram apresentadas as categorias dos modelos existentes, destacando-se a técnica de Floresta Aleatória e os Gráficos de Controle, em conjunto com os processos necessários para aplicá-las. Essas técnicas serão utilizadas no estudo observacional para analisar e comparar o desempenho de diferentes versões de um software.

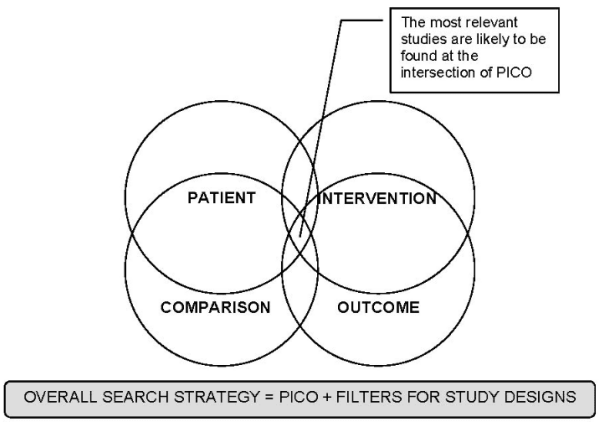
3 Revisão Estruturada da Literatura

Este capítulo aborda o passo a passo para a realização da coleta do material bibliográfico necessário para a realização do trabalho. Foram selecionados materiais de diferentes categorias como livros, artigos, normas técnicas, vídeos, entre outros. Para a coleta dos materiais, foram utilizadas, inicialmente, a plataforma indexadora do Periódico Capes (Capes, 2022) e posteriormente, deu-se continuidade com a base de dados Scopus (Elsevier, 2022). Essas plataformas possuem a capacidade de indexar a maioria dos jornais relevantes que possuem material com maior confiabilidade devido à revisão por pares.

3.1 Estruturação da *string* segundo protocolo PICO

A estrutura da string de busca foi baseada no protocolo PICO (PAI et al., 2004). Esse protocolo foi concebido para identificar os estudos mais pertinentes relacionados a temas abordados na área médica, conforme ilustrado na Figura 11. Esse protocolo tem sido utilizado em revisões sistemáticas da literatura em engenharia de software. Seguiu-se esse protocolo na presente monografia. PICO é um acrônimo para *patient* (população de pacientes), *intervention* (intervenção), *comparison* (comparação) e *outcome* (resultado).

Figura 11 – Representação da interseção do PICO



Fonte: (PAI et al., 2004)

A população de pacientes foi ajustada para abranger as áreas específicas da engenharia de software que se destacam pela sua preocupação com questões de desempenho e restrições na utilização de recursos computacionais. Adicionalmente, foram utilizados termos mais abrangentes como "*embedded*" e "*real-time*", com o intuito de garantir que artigos de outras áreas da engenharia de software também fossem incluídos na busca.

No contexto da medicina, a intervenção refere-se ao tratamento que está sendo aplicado e analisado. Entretanto, neste trabalho, o termo "intervenção" foi empregado para englobar os recursos computacionais, juntamente com termos relacionados à análise e gerenciamento dos mesmos, como por exemplo "*CPU analysis*" ou "*CPU management*".

O resultado representa a saída desejada dos estudos levantados. Nesse caso, o interesse era encontrar trabalhos que alcançassem conclusões relacionadas a medidas de qualidade, desempenho e/ou eficiência. Com base nessas necessidades, foram concebidas palavras compostas que abordassem tais resultados, combinando-as com termos como modelo, métrica, indicador ou medida.

O papel da comparação, no contexto do estudo, consistiria em identificar alternativas para a intervenção proposta. Entretanto, devido à dificuldade de encontrar estudos relacionados a modelos de qualidade de software que utilizem métricas dinâmicas (LóPEZ et al., 2022), entende-se que a área ainda não possui um corpo de conhecimento estruturado que proveja os requisitos necessários para permitir comparações entre diferentes tratamentos.

Para estruturar a *string* de busca seguindo o modelo PICO, utilizamos o operador lógico "*AND*" para delimitar a população, intervenção e saída, respectivamente. Adicionalmente, dentro de cada uma dessas camadas, empregou-se o operador lógico "*OR*" para separar as palavras-chave contidas em cada uma delas. O resultado alcançado foi:

("Embedded Software" OR "Embedded Application" OR "Embedded System" OR "Software System" OR "Software Application" OR "real-time operating system" OR "Software Engineering" OR "Software Quality") AND ("Performance Efficiency" OR "Performance Parameterization" OR "Performance Analysis" OR "Performance management" OR "Performance Evaluation" OR "CPU Efficiency" OR "CPU Parameterization" OR "CPU Analysis" OR "CPU Utilization" OR "CPU management" OR "Memory Efficiency" OR "Memory Parameterization" OR "Memory Analysis" OR "Memory Utilization" OR "Memory management" OR "Resources Efficiency" OR "Resources Parameterization" OR "Resources Analysis" OR "Resources Utilization" OR "Resources management" OR "runtime resources Efficiency" OR "runtime resources Parameterization" OR "runtime resources Analysis" OR "runtime resources management" OR "runtime resources Evaluation" OR "Software Performance") AND ("Quality Measurement" OR "Quality Indicator" OR "Quality Metric" OR "Quality Measure" OR "Quality Calculation" OR "Quality Model" OR "Efficiency Metric" OR "Efficiency model" OR "Performance Indicator" OR "Performance Model" OR "Performance Metric")

3.2 Execução da *string* de busca

A base escolhida para executar a *string* de busca foi a Scopus. Scopus é um banco de dados de resumos e citações de literatura revisada por pares, incluindo periódicos científicos, livros e anais de conferências. Essa plataforma oferece uma visão abrangente

da produção mundial de pesquisa nos campos da ciência, tecnologia, medicina, ciências sociais, artes e humanidades (Elsevier, 2022).

A evolução dos testes e construção da *string* de busca podem ser encontrados no Apêndice A.

Os resultados obtidos por meio da primeira *string* de busca resultaram em uma quantidade significativa de informações, porém, não se apresentaram muito relacionados ao contexto do trabalho. Diante disso, evidenciou-se a necessidade clara de aprimorar os termos empregados.

A estratégia adotada visando otimizar os resultados envolveu a substituição do uso excessivo de parênteses e a adoção de termos conjuntos entre aspas. Outro enfoque foi o aperfeiçoamento da seção de intervenção, com termos mais alinhados às necessidades do trabalho. No entanto, constatou-se que essa *string* resultou em apenas 75 registros, um número relativamente baixo para uma potencial revisão. Diante disso, identificaram-se falhas em nosso conjunto de população e resultado do PICO.

Por fim, procedeu-se à remoção de termos em excesso que resultavam em informações não diretamente relacionadas ao escopo desejado, além de realizar aprimoramentos na definição da população e resultado. Esses ajustes resultaram em um aumento na quantidade de resultados encontrados, totalizando 288, um número considerado viável para análise dentro do intervalo de tempo disponível.

3.3 Critérios de Inclusão e Exclusão

Por meio da elaboração da estratégia de busca, obtivemos resultados favoráveis que possibilitaram o início do processo de seleção de literatura relevante ao contexto desta pesquisa. Nessa etapa, foram identificados vários critérios com o intuito de aprimorar a seleção dos materiais estudados. Esses critérios foram estabelecidos com o propósito de guiar a escolha dos estudos. Importante salientar que tais critérios não possuem um caráter de dependência, ou seja, os critérios de inclusão e exclusão são tratados de forma independente. Os critérios empregados foram os seguintes:

- Leitura do título e/ou resumo
  - Quantidade de trabalhos excluídos: 145 (49,65%)
  - Quantidade de trabalhos ao final: 143
  - Critérios de Inclusão (conter as seguintes palavras compostas):
    - *Performance model; Performance analysis; Performance evaluation; Performance modeling; Performance Monitoring;*
    - *Efficiency analysis; Efficiency model;*

    - *Software Analysis; Software quality; Quality model;*
  - Critérios de Exclusão:
    - Tratar sobre desempenho de processo (pessoas)
    - Trabalhar exclusivamente com métricas de desempenho não-computacional (Ex: desempenho de eficiência de consumo energético)

- Leitura da introdução e/ou conclusão
  - Quantidade de trabalhos excluídos: 130 (90,9%)
  - Quantidade de trabalhos ao final: 13
  - Critérios de Inclusão:
    - Aplicar um modelo de análise de desempenho e citar a utilização de métricas de desempenho de *software*
    - Possuir um exemplo prático de um caso de uso de aplicação de um modelo de análise de desempenho
    - Revisão ou mapeamento da literatura que citem trabalhos relacionados a qualidade de software do ponto de vista de eficiência de desempenho
    - Adequação ao contexto do trabalho (ambiente de testes envolvendo produtos *web*)
  - Critérios de Exclusão:
    - Usar exclusivamente técnicas de *deep learning*, devido à alta complexidade intrínseca ao uso de redes neurais
    - Enfoque em arquitetura de *hardware*

Após a aplicação desses critérios, obtivemos um conjunto final de 13 trabalhos que abrangem diversos domínios, tais como sistemas embarcados, computação em nuvem e aplicações baseadas em *web* ou *mobile*. O resumo sobre a seleção de trabalhos é apresentado a seguir:

- Quantidade total de trabalhos levantados: 288
- Filtro por data: 2014 - Atual (2023)
- Idioma: Inglês
- Quantidade de trabalhos selecionados: 13

Quadro 3 – Trabalhos selecionados após aplicação dos critérios

N.º	**Título do trabalho**	**Domínio**	**Referência**
1	*Empirical study on the discrepancy between performance testing results from virtual and physical environments*	*Desktop* e *Web*	(ARIF; SHANG; SHIHAB, 2017)
2	*AIM: Adaptable Instrumentation and Monitoring for Automated Software Performance Analysis*	*Desktop* e *Web*	(WERT; SCHULZ; HEGER, 2015)
3	*Early performance prediction in bioinformatics systems using palladio component modeling*	*Desktop* e *Web*	(DORGHAM; BELAL; ABDELMOEZ, 2021)
4	*Runtime Performance Management for Cloud Applications with Adaptive Controllers*	Nuvem	(BARNA et al., 2018)
5	*Using black-box performance models to detect performance regressions under varying workloads: an empirical study*	*Desktop*	(LIAO et al., 2020)
6	*A Generic Platform for Transforming Monitoring Data into Performance Models*	*Web*	(KUNZ; HEGER; HEINRICH, 2017)
7	*Performance evaluation metrics for multi-objective evolutionary algorithms in search-based software engineering: Systematic literature review*	Não possui domínio específico (por se tratar de uma revisão sistemática)	(NUH et al., 2021)
8	*Towards a DevOps Approach for Software Quality Engineering*	*Web* e Nuvem	(PEREZ; WANG; CASALE, 2015)
9	*A quality model for mobile thick client that utilizes web API*	*Mobile* e *Web*	(FAUZIA; LAKSMIWATI; HENDRADJAYA, 2014)
10	*Log4Perf: Suggesting Logging Locations for Web-based Systems' Performance Monitoring*	*Web*	(YAO et al., 2018)
11	*Efficiency Analysis of Provisioning Microservices*	Nuvem	(KHAZAEI et al., 2016)
12	*Cloud Function Performance: A Component Modeling Approach*	Nuvem	(FLORES-GONZáLEZ; TREJOS-ZELAYA, 2020)
13	*Hierarchical performance modeling of embedded systems*	Embarcados	(ALSHEIKHY; HAN; AMMAR, 2015)

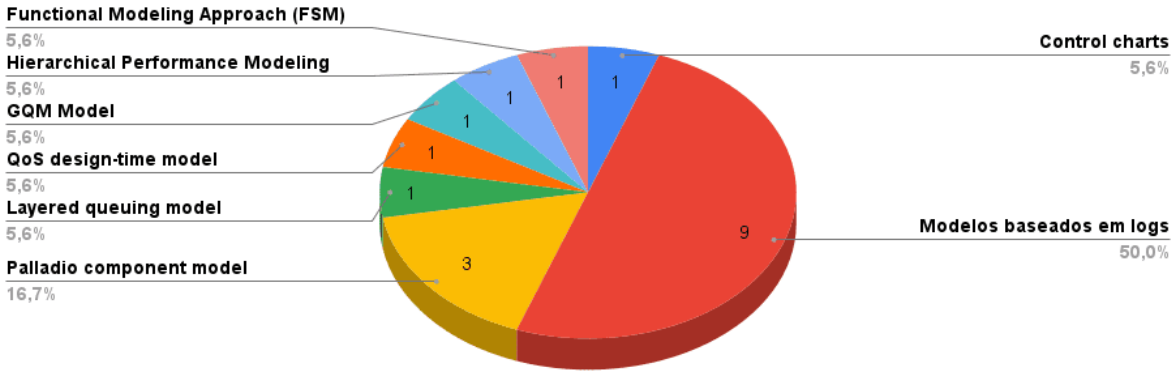
Fonte: Autores

3.4 Análise dos Artigos Selecionados

A partir da seleção dos artigos procedeu-se à leitura integral do seu conteúdo, a fim de identificar e listar os modelos de análise de desempenho de software aplicados, compreender o contexto em que eram aplicáveis e compreender o funcionamento de cada um desses modelos. Para isso, em alguns casos, foi necessário navegar entre as referências bibliográficas citadas a fim de localizar outros estudos relevantes. Além disso, também foram listadas as ferramentas utilizadas nos estudos analisados e o domínio em que os modelos poderiam ser aplicados.

Foram identificados oito modelos distintos de análise de eficiência de desempenho de software, conforme ilustrado na Figura 12. Os modelos que se baseiam na análise dos *logs* dos sistemas foram agrupados em uma mesma categoria devido às suas semelhanças significativas em relação à captura de métricas, contexto de aplicação e forma de execução. Esses modelos geralmente utilizam alguma forma de regressão numérica para inferir a relação entre os *logs* capturados e o recurso computacional em análise.

Figura 12 – Gráfico de modelos encontrados nos artigos selecionados

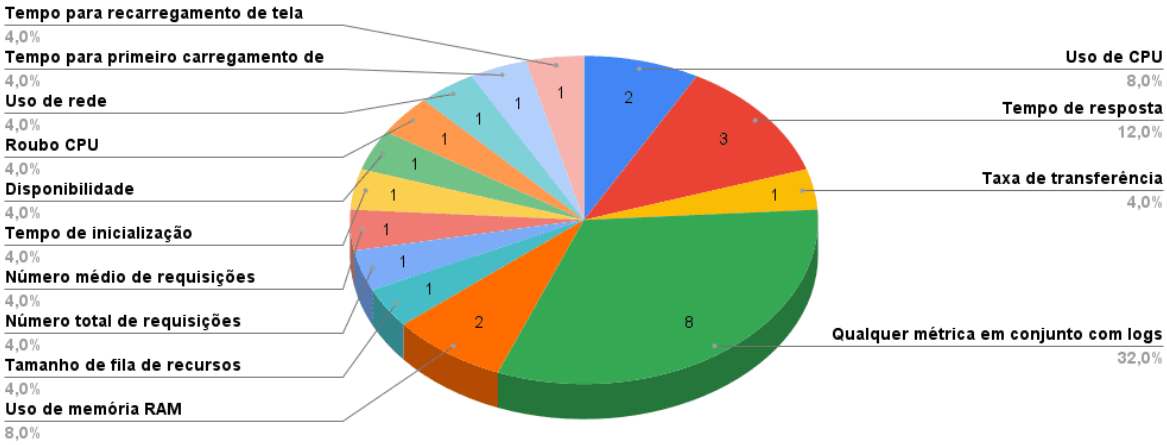


Fonte: Autores

A proporção de utilização de cada métrica nos artigos selecionados pode ser visualizada na Figura 13. É importante ressaltar que o uso dos *logs* do sistema, nos modelos observados, almeja realizar a contagem de requisições realizadas ao sistema para definir uma relação entre essas requisições e alguma outra métrica dinâmica utilizando regressão numérica (GAO et al., 2016).

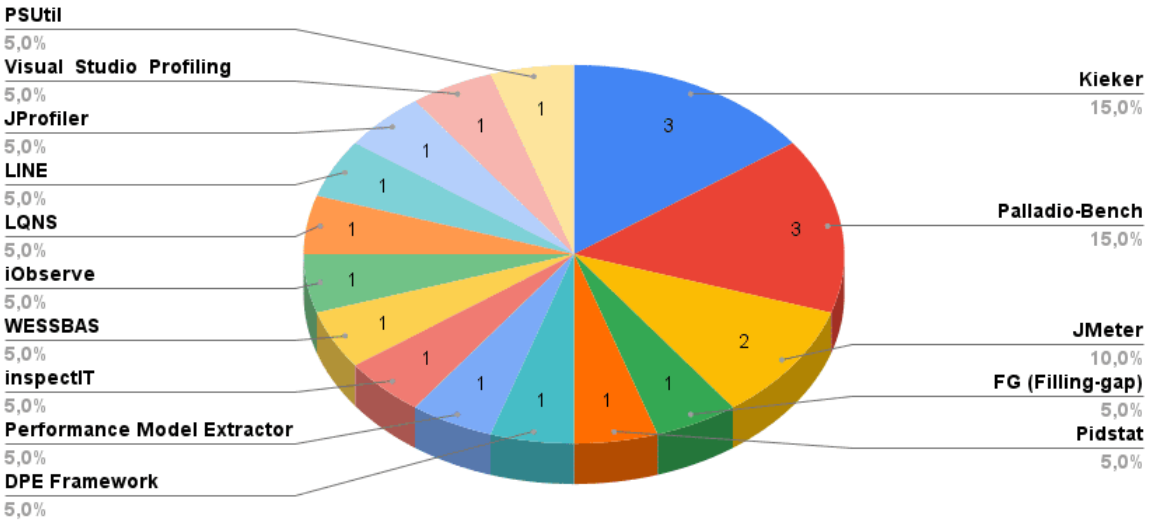
Além disso, alguns dos artigos selecionados também mencionaram as ferramentas utilizadas para coletar métricas, gerar carga de testes ou aplicar o modelo de forma completa. A distribuição de citações das diferentes ferramentas pode ser observada na Figura 14. É importante salientar que grande parte das ferramentas citadas é de suporte da linguagem Java quando se trata de domínios *web* ou *mobile*.

Figura 13 – Gráfico de métricas utilizadas nos modelos dos artigos selecionados



Fonte: Autores

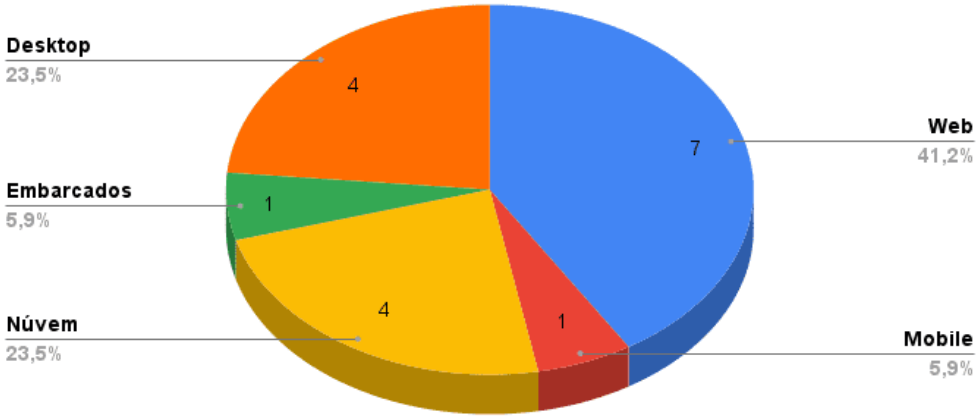
Figura 14 – Gráfico de ferramentas utilizadas nos artigos selecionados



Fonte: Autores

Por fim, a proporção de modelos aplicados por domínio pode ser observada na Figura 15. A contabilização foi realizada com base nos relatos dos autores de cada um dos artigos. Quando não houve uma declaração explícita sobre quais domínios o modelo poderia ser aplicado, considerou-se o domínio das aplicações utilizadas como objeto dos casos de uso.

Figura 15 – Gráfico de domínios dos objetos de estudo dos artigos selecionados



Fonte: Autores

3.5 Seleção de métodos

No processo de seleção dos métodos a serem utilizados neste estudo observacional, foram considerados os seguintes critérios:

- Domínio do produto analisado: Dado que pretendemos observar um produto *web*, optamos por escolher modelos que tivessem sido aplicados nesse contexto nos artigos selecionados, de modo a obter resultados mais assertivos acerca do estudo estabelecido;
- Conhecimento sobre a lógica interna do produto: Dado que o presente estudo é de contexto rápido, não se pretende chegar a um nível de conhecimento e maturidade alto sobre a lógica interna do produto. Logo, são interessantes modelos que não necessitem desse conhecimento, e
- Versatilidade na implementação: É importante a escolha de modelos que sejam versáteis no sentido de quais são as métricas necessárias para análise de desempenho do produto. Nesse caso, modelos baseados em *logs* são ideais porque podem ser utilizados em conjunto com qualquer outra métrica de interesse para extrair informações sobre a variação da mesma.

Utilizando a técnica de bola de neve (*snowballing*) descrita por Wohlin et al. (2012), foram explorados os artigos que apresentavam modelos que se encaixam nos critérios estabelecidos. Através da bola de neve, foram encontradas outros cinco trabalhos, detalhados

no Quadro 4, que trazem referências a diferentes técnicas e conceitos. Alguns desses trabalhos foram utilizados como fonte de referência para os métodos escolhidos, que estão melhores descritos na Seção 2.3.

Quadro 4 – Trabalhos selecionados por bola de neve

Artigo de origem	Título do trabalho	Referência
Artigo 1	*Automated performance analysis of load tests*	(JIANG et al., 2009)
Artigo 5	*Random Forests*	(BREIMAN, 2001)
Artigo 10	*Detecting Performance Anti-patterns for Applications Developed using Object-Relational Mapping*	(CHEN, 2014)
Artigo 10	*A Framework to Evaluate the Effectiveness of Different Load Testing Analysis Techniques*	(GAO et al., 2016)
Artigo 10	*Automated Detection of Performance Regressions Using Statistical Process Control Techniques*	(NGUYEN et al., 2012)

Fonte: Autores

Para a caracterização do presente trabalho, foram selecionados os modelos baseados em mineração de dados e em regras, visto que somente essas categorias apresentam métodos que se encaixam nos critérios definidos anteriormente. Quanto aos métodos estatísticos escolhidos para cada modelo, tem-se respectivamente:

Floresta aleatória: Este método, discutido na Seção 2.3.2, será utilizado para estabelecer uma relação entre as requisições feitas ao sistema (variáveis independentes) e os registros de uso de recursos computacionais (variáveis dependentes). Esse método foi escolhido devido aos resultados positivos que obteve em comparação a outros modelos de regressão numérica, conforme mencionado por Liao et al. (2020).

Gráficos de controle: A utilização dos gráficos de controle como um método estatístico para o controle de processos é abordada de forma detalhada na Seção 2.3.3. Esse método foi selecionado devido à sua facilidade de compreensão e comunicação, especialmente pelo fato de ser apresentado visualmente em forma de gráfico, e também sua abordagem bem definida que contribui para a aplicação precisa do método. Além disso, esse método atingiu cerca de 75% de precisão e 100% de recuperação em comparação com a avaliação real do objeto de estudo em questão (NGUYEN et al., 2012).

3.6 Resumo do capítulo

Neste capítulo, é explicado como ocorreu a etapa de revisão estruturada da literatura do trabalho, descrevendo desde a construção da *string* de busca até a seleção dos

artigos principais que serviram como fundamentação teórica para a elaboração do presente estudo.

4 Estudo Observacional

Neste capítulo, apresentamos o planejamento e a execução do estudo observacional, que permitiu endereçarmos as respostas questão de pesquisa principal deste Trabalho de Conclusão de Curso.

4.1 Planejamento do Estudo Observacional

4.1.1 Definição

Para a definição da estrutura do estudo observacional, utilizamos o protocolo de pesquisa de estudo de caso. Estudo de caso é um método cujo objetivo é investigar fenômenos contemporâneos em seu contexto situacional. Yin (2009) interpreta a abordagem de estudo de caso como uma investigação que reconhece que a fronteira entre o fenômeno de interesse e seu contexto pode não ser claramente definida.

No presente estudo, devido ao curto prazo de realização, foi necessária a adaptação para conseguirmos observar o fenômeno em seu contexto situacional, sem, contudo, capturar sua contemporaneidade, ou seja, sua manifestação em seu ambiente natural, real.

Um estudo observacional pode incorporar elementos de outros métodos de pesquisa. Por exemplo, uma pesquisa pode ser conduzida por meio de um estudo observacional, precedido de uma revisão sistemática da literatura. Entrevistas e observações, são comumente utilizados para coletar dados em estudos observacionais (WOHLIN et al., 2012).

Segundo Yin (2009), é característica de um estudo de caso lidar com situações tecnicamente distintas, no qual há uma quantidade significativamente maior de variáveis do que pontos de dados disponíveis. Como consequência, é recomendado contar com múltiplas fontes de evidências, nas quais os dados precisam convergir de maneira triangular. Além disso, outra vantagem desse enfoque é o benefício proveniente do desenvolvimento prévio de proposições teóricas para orientar tanto a coleta quanto a análise dos dados. Neste estudo, apenas fontes de evidência de segundo grau foram analisadas.

No decorrer do estudo observacional, os dados são coletados com um propósito determinado, permitindo a realização de análises que visam obter percepções e inferências relevantes. O objetivo principal de um estudo observacional é acompanhar um atributo específico ou estabelecer relações entre diferentes atributos.

A abordagem de estudo de caso é particularmente aplicada na área da engenharia de *software*, onde a compreensão aprofundada do fenômeno em seu contexto real é

relevante (WOHLIN et al., 2012).

4.1.1.1 Objetivo

A partir do mapeamento sistemático conduzido por López et al. (2022), observa-se que a quantidade de estudos encontrados que empregam métricas extraídas do sistema em uso, como o uso de recursos computacionais, como fonte de dados para a definição de indicadores de qualidade, é substancialmente menor em comparação com aqueles que utilizam outras fontes de informação, como o código-fonte. Diante desse cenário, o presente estudo observacional caracteriza a análise do desempenho de uso de recursos computacionais em produtos de software, do ponto de vista de pesquisadores.

4.1.1.2 Caso

O escopo de análise do presente estudo recai sobre um produto de *software* desenvolvido por uma empresa especializada em soluções de crédito para diversas modalidades (como crédito consignado e crédito pessoal). Esta empresa disponibiliza tal *software* como um produto *white label* [1] para múltiplos clientes. O produto conta atualmente com mais de 200 mil linhas de código e, em decorrência do contínuo crescimento da base de dados pertencente aos clientes, foi identificado o interesse em monitorar a variação de desempenho do produto.

O objeto de estudo abordado nesta pesquisa é uma aplicação desenvolvida em Python, utilizando especificamente o framework Django para criar o serviço *web* na camada de *Backend*. Essa aplicação adota o padrão ORM (Mapeamento Objeto-Relacional), que possibilita o mapeamento de objetos para bancos de dados relacionais.

Esse projeto foi selecionado por possibilitar a execução do estudo em um projeto que está em uso no mercado. Além disso, existe a facilidade de contato com os desenvolvedores responsáveis, visto que a empresa está disposta a colaborar com a pesquisa, o que permite que, sob o ponto de vista deles, sejam coletados dados reais, que servirão para conseguirmos definir quais fluxos do produto devem ser priorizados na execução dos testes.

4.1.1.3 Trabalhos Relacionados

Os trabalhos relacionados foram derivados da revisão estruturada, descrita no Capítulo 3. Os trabalhos apresentados nessa subseção possuem uma temática semelhante ao estudo proposto.

O trabalho de Gao et al. (2016) foi utilizado como base para o desenvolvimento desta monografia. Nesse artigo, foi conduzido um estudo observacional que envolveu a

[1] *white label*: Produtos construídos para serem adaptados/vendidos a diferentes tipos de negócio, partindo de um mesmo modelo.

aplicação de diferentes técnicas de análise automatizada de testes de carga, em produtos de *software* de código aberto visando comparar a efetividade de cada uma dessas técnicas. Dentre os métodos de análise, destacou-se o uso de Gráficos de Controle (discutidos na Seção 2.3.3) e Árvores de Decisão (detalhadas na Seção 2.3.2.2).

Dentre as conclusões apresentadas nesse trabalho, destaca-se que, ao utilizar mais de duas versões anteriores para aplicar os modelos de análise, há aumento consideravelmente do tempo necessário para a análise, sem proporcionar melhorias significativas nos resultados.

O trabalho do Yao et al. (2018) apresenta uma abordagem automatizada, cujo propósito é fornecer sugestões acerca dos pontos apropriados para a inserção de declarações de registro ou *logs*. O objetivo principal dessa técnica é monitorar o desempenho de software em sistemas *web*, de forma a subsidiar a construção de um modelo estatístico de desempenho. Tal modelo visa identificar os locais específicos no código-fonte que possuem uma influência estatisticamente significativa sobre o desempenho do software.

O estudo conduzido por Shang et al. (2015) propõe uma abordagem automatizada voltada à detecção de queda de desempenho, adotando uma análise abrangente, onde todas as métricas coletadas são analisadas, ao invés de utilizar um número limitado de métricas específicas, as quais, normalmente, são determinadas com base na experiência ou intuição. Inicialmente, neste estudo, as métricas de desempenho são agrupadas em *clusters*, com o intuito de determinar a quantidade necessária de métricas para uma representação precisa do desempenho do sistema. Em seguida, são aplicados testes estatísticos para selecionar as métricas de desempenho relevantes, que servirão de base para a construção de modelos de regressão numérica. A partir disso, as versões são comparadas com a utilização da regressão numérica, e os dados estimados são analisados para entender o quanto é possível explicar, a partir deles, a variação nos dados observados

4.1.1.4 Questão de Pesquisa

A questão de pesquisa, definida na seção 1.3, pode ser descrita da seguinte forma:

"Como analisar dinamicamente a característica de qualidade de eficiência de desempenho, para apoiar a tomada de decisão sobre a implantação de versões de produto de software, em ambiente *web*?"

Para respondermos essa questão principal de pesquisa, foi então derivada a questão específica e as métricas que serão apresentadas a seguir.

Objetivo: Identificar a variação de desempenho entre diferentes versões de um produto de *software*.

- **Questão Específica 1 (QE1):** Ao aplicar problemas de desempenho no código-

fonte observado, como, por exemplo, o excesso de dados retornados em uma consulta, foi possível identificar variação no desempenho?

- **Métrica 1.1**: Métricas de desempenho computacional coletadas.
  - Porcentagem média de uso de CPU
  - Porcentagem média de uso de memória RAM
  - Tempo de resposta médio
  - Transações concluídas no banco de dados
  - Linhas buscadas no banco de dados
  - Linhas retornadas pelo banco de dados
- **Métrica 1.2**: Limites estabelecidos pelas técnicas aplicadas para comparação de desempenho:
  - Valor de limites superior (Gráfico de Controle)
  - Valor de limites inferior (Gráfico de Controle)
  - Classificações de grau de diferença da técnica Cliff Delta (Floresta Aleatória)
- **Métrica 1.3**: Quantidade de variação de desempenho detectada nas versões com problemas inseridos.
  - Taxa de violação (Gráfico de Controle)
  - Valor calculado de Cliff Delta (Floresta Aleatória)

4.1.1.5 Fonte de Dados

A fonte de dados da análise, será uma versão do código-fonte do produto selecionado como objeto de estudo. Uma nova versão será produzida, a partir da injeção de anomalias de desempenho na versão do produto analisada. Portanto, é esperado que essas alterações produzam um efeito de degradação de desempenho dos recursos computacionais, durante a execução do *software*. A análise de diferentes versões, com ou sem problemas inseridos, permitirá a observação de possível ocorrência de variação de desempenho entre as versões comparadas.

4.1.2 Procedimentos

Para aumentar a confiança das conclusões obtidas em um estudo de caso, a literatura indica a necessidade de se utilizar diferentes fontes de informação (YIN, 2009), (WOHLIN et al., 2012). Para tanto, é necessária a realização de coleta de dados de diferentes ordens. Considerando o escopo de um trabalho de conclusão de curso de graduação e o exíguo período para a instrumentação, execução e análise de um estudo observacional em seu ambiente real, a coleta de dados de primeira ordem se tornou inviável. Por outro

lado, foi imprescindível a obtenção de dados de segunda ordem, por meio da coleta e posterior análise das métricas de desempenho do produto de software selecionado.

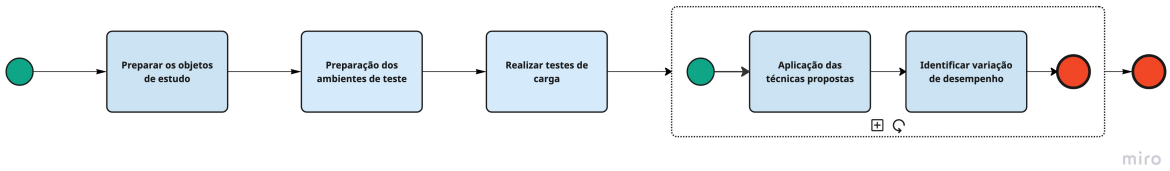
4.1.2.1 Fluxo de Atividades

A condução do estudo observacional requer a adoção de uma sequência de atividades metodológicas que englobam a coleta, o tratamento e a análise dos dados. Os procedimentos envolvidos nesse processo são descritos a seguir:

- Preparar o objeto de estudo: Entender o contexto da aplicação, realizando ajustes para ser possível extrair todas as métricas necessárias;
- Preparação do ambiente de teste: Configurar ferramentas e máquinas que realizarão a coleta dos dados da aplicação;
- Realizar testes de carga: Desenvolver e aplicar os testes de carga para cada uma das diferentes versões do produto;
- Aplicação das técnicas propostas: Aplicar tratamento e análise dos dados mediante a técnica escolhida, construção de *scripts* visando automação desse processo para ser realizado em ciclos iterativos, e
- Identificar variações de desempenho: Uma vez obtidos os resultados dos testes, é necessário realizar uma comparação do desempenho entre as diversas versões disponíveis, visando identificar variações entre as versões.

A condução do estudo foi feita de maneira iterativa, para viabilizar diferentes coletas e análises ao longo do desenvolvimento do produto. O fluxo de atividades seguindo o modelo BPMN pode ser observado na Figura 16

Figura 16 – Fluxo de atividades referente ao processo de experimentação



Fonte: Autores

4.1.2.2 Métricas De Desempenho Coletadas

Para este estudo os recursos computacionais a serem monitorados e analisados serão: porcentagem de uso de CPU da aplicação e do banco de dados; porcentagem do

uso de memória RAM da aplicação e do banco de dados; leitura e escrita em disco, adaptadas para linhas buscadas, linhas retornadas e transações concluídas com sucesso no banco de dados, e tempo de resposta. As técnicas escolhidas não especificam o uso de métricas. Essas métricas foram escolhidas devido ao alto grau de impacto das mesmas na aferição do desempenho computacional, além de serem amplamente utilizadas como indicadores de desempenho, conforme (LóPEZ et al., 2022) e (NGUYEN et al., 2012).

4.1.2.3 Inserção de Problemas de Desempenho

O critério de validação adotado para as técnicas utilizadas consiste na inserção arbitrária de problemas de desempenho no código-fonte, com o propósito de impactar negativamente o desempenho da aplicação. Desse modo, espera-se obter resultados que demonstrem uma variação negativa ou piora no desempenho da aplicação. A maioria dos problemas a serem inseridos requer um conhecimento aprofundado do código-fonte e das tecnologias utilizadas. Nesse sentido, é necessária uma avaliação com o time de desenvolvedores para entender quais dos problemas mencionados a seguir são factíveis para o presente estudo.

Alguns problemas de aplicações *Backend* mais genéricos, citados em Nguyen et al. (2012), podem ser encontrados no Quadro 5. Além desses, existem outros problemas, inerentes a aplicações que utilizam ORM, que podem impactar significativamente o desempenho da aplicação. Dentre eles, destaca-se o problema de dados excessivos e o processamento individual de cada registro (CHEN, 2014).

A questão dos dados excessivos está relacionada ao processo de recuperar informações, do banco de dados, que não são necessárias em determinada requisição. Essa prática resulta no tráfego e no processamento desnecessário de dados adicionais, afetando negativamente o desempenho da aplicação. Por sua vez, o processamento individual de cada registro refere-se à execução de operações similares em bancos de dados, por meio de laços de repetição, o que pode acarretar uma sobrecarga significativa no sistema e afetar o desempenho.

Devido à natureza complexa do sistema observado e após discussões com os desenvolvedores do produto, foi determinado que apenas dois cenários de inserção de problemas seriam considerados: o cenário relacionado à limitação de consulta (*query limit*) e o problema de excesso de retorno de dados (dados excessivos).

Diversos motivos foram identificados como impedimentos para a adesão aos cenários propensos a causar problemas. Dentre esses motivos, exemplo: a complexidade das saídas de logs para o problema de registro do sistema; e a prática padrão do Django de administrar uma nova conexão ao banco de dados para cada requisição, dificultando o cenário de conexão com o banco de dados. Adicionalmente, observa-se a atual subutilização de chaves de indexação na aplicação, tornando oneroso o mapeamento adicional de chaves

Quadro 5 – Cenários causadores de problemas de desempenho

Cenário	**Caso bom**	**Caso ruim**
Limite de consulta (*Query limit*)	O número de linhas retornadas pelo banco de dados é limitado ao que será exibido na interface gráfica do sistema	Recuperar todos os dados salvos em banco, relacionados a uma requisição, para então, filtrar aqueles necessários, na camada de apresentação, interface gráfica, do sistema.
Registro do sistema (*System print*)	O sistema não faz registros indevidos de *logs* de depuração	O sistema possui registros indevidos de *logs* de depuração
Conexão do Banco de Dados (*DB Connection*)	O sistema reutiliza a conexão com o banco de dados sempre que possível	São criadas novas conexões para cada requisição
Chave de indexação (*Key index*)	O sistema possui chaves de indexação em requisições frequentemente utilizadas	Não são criadas chaves de indexação para requisições frequentemente utilizadas
Índice de texto (*Text index*)	Criação de índice de texto completo para colunas de texto	Inexistência de índice de texto completo para colunas de texto

Fonte: Adaptado de Nguyen et al. (2012)

e sua subsequente exclusão para gerar diferença nas versões, especialmente no contexto do problema relacionado à chave de indexação. Outros motivos que contribuem para a não adesão a certos cenários problemáticos incluem a limitada utilização de índices de texto, uma vez que algumas requisições não fazem uso da pesquisa por texto. Além disso, a complexidade do sistema em termos das requisições utilizadas dificulta o manuseio do processamento individual de cada registro.

4.1.2.4 Testes de Carga

A determinação da quantidade de usuários a serem adicionados ao grupo foi conduzida mediante a observação do comportamento do sistema durante a execução do fluxo. Métricas relevantes, como a utilização da CPU e a ocupação da memória RAM, foram monitoradas, evitando que atingissem a marca de 100%. Isso se justifica pelo fato de que, ao alcançar o limite desses recursos, o próprio sistema operacional das máquinas adotaria medidas reativas, como a limitação do processamento e a interrupção de tarefas, visando reduzir a carga sobre os recursos. Esse fenômeno poderia afetar e, por conseguinte, enviesar as análises comparativas entre as versões do produto. Com isso, caracterizando uma ameaça à validade do estudo.

Consequentemente, caso uma dessas métricas se aproximasse significativamente do seu limite, o número de usuários no teste seria ajustado, diminuído. Desta maneira, a quantidade final de usuários no teste é determinada, segundo a capacidade da máquina específica onde o sistema observado foi executado. Garantindo uma execução estável e representativa do comportamento do sistema.

Os testes foram realizados em três cenários distintos. O primeiro cenário reproduziu o fluxo conforme implementado na aplicação real, seguindo as diretrizes estabelecidas pela empresa responsável pelo produto. Nos segundo e terceiro cenários, foram introduzidos problemas de desempenho, nomeados "limite de consulta" e "dados excessivos", como variações do primeiro cenário.

Durante a execução dos testes, é fundamental coletar e armazenar todos os registros de requisições feitas ao sistema, juntamente com as informações precisas de data e hora em que foram registrados. Além disso, é importante realizar a coleta e o armazenamento do estado de uso de recursos computacionais das máquinas durante toda a execução dos testes.

4.1.3 Técnicas para Análise de Dados

A análise quantitativa dos dados foi conduzida utilizando duas técnicas distintas: Floresta Aleatória (descrita na Seção 2.3.2) e Gráficos de Controle (descrita na Seção 2.3.3). A aplicação dessas técnicas nos possibilitou estabelecer os de valores de referência utilizados para o julgamento e interpretação da ocorrência de variação de desempenho.

Antes de empregar as técnicas de análise escolhidas, foi necessário adotar a técnica de corte interquartil para processar os conjuntos de dados coletados. Essa abordagem envolveu a remoção de valores atípicos e, quando necessário, a normalização da distribuição das métricas.

Utilizando como base para o treinamento do algoritmo Floresta Aleatória os dados relacionadas à contagem de requisições realizadas para cada *endpoint*, em conjunto com as métricas de desempenho coletadas durante o teste de carga realizado na versão do sistema sem problemas inseridos, é possível aplicar novos dados de entrada e obter uma previsão de como o sistema se comportaria nesse novo contexto. Portanto, a contagem de requisições realizadas no segundo teste de carga (alvo da comparação) será aplicada ao algoritmo de Floresta Aleatória para obter uma estimativa de como aquelas chamadas se comportariam na versão do teste tido como base.

A utilização de métodos estatísticos, como o Erro Percentual Absoluto Médio (MAPE) e o Cliff Delta, permite quantificar a disparidade no comportamento das métricas entre ambas as versões, adaptando o contexto utilizado por Gao et al. (2016), que utiliza os métodos estatísticos como forma de validação da assertividade das previsões realizadas

pelo algoritmo. Considerando que as métricas devem manter níveis similares caso não haja alteração nas versões testadas, valores altos no Cliff Delta indicam que existe a tendência de valores maiores em um dos conjuntos de dados comparados. Já o MAPE indica qual é a porcentagem de diferença média para cada amostra do conjunto de dados.

Já em relação à técnica de Gráfico de Controle, que usa medidas de tendências centrais oriundas da estatistifica descritiva, calcula-se a taxa de violação. Essa taxa de violação é determinada ao verificar a porcentagem de dados que ultrapassam os limites superiores e inferiores predefinidos.

A partir do limite estabelecido, é conduzida uma análise da taxa de violação identificada. Se essa taxa exceder o limite, isso indicará que houve diferença de desempenho entre as versões comparadas. Essa diferença pode ser interpretada como uma melhoria ou uma queda no desempenho, dependendo da interpretação dos dados fora de controle e da natureza da métrica de qualidade em questão. Por exemplo, considerando-se que um menor tempo de resposta é indicativo de um bom desempenho, a presença de dados predominantemente abaixo do limite inferior do gráfico indicará uma versão que influencia positivamente o desempenho.

Para analisar os dados, ambas técnicas são aplicadas nas versões, alvo, quer sejam, àquelas onde foram inseridos problemas de desempenho.

Com essa abordagem, será analisada a adequação de ambas as técnicas, em quantificar a percepção da característica de qualidade de Eficiência e Desempenho. Espera-se que, os resultados das análises realizadas nas versões alvo, apresentem um pior aproveitamento dos recursos computacionais, no caso, uma piora de desempenho.

4.1.4 Instrumentação

Na presente seção, serão apresentadas as ferramentas essenciais para o desenvolvimento dos estudos abordados nesta monografia. Considerando que, durante a execução deste estudo foram gerados arquivos que estão em constante atualização, faz-se necessário utilizar ferramentas de versionamento de código-fonte, bem como ferramentas para coleta de métricas e simulação de carga de usuários. Além disso, a seção descreve as máquinas que serão utilizadas para a execução do objeto de estudo, bem como a coleta das métricas de uso computacional.

4.1.4.1 Git e github

O Git é uma ferramenta de gerenciamento do controle de versões de um código-fonte (em inglês, *Source Code Management*) (Git, 2023). Trata-se do sistema de controle de versão mais utilizado na atualidade, desenvolvido em 2005, pelo mesmo criador do

kernel do sistema operacional Linux, Linus Torvalds [2]. O Git realiza o controle de versão de *software*, visando registrar as alterações existentes em um arquivo ou conjunto de arquivos ao longo do tempo, permitindo a retomada ou incorporação dessas versões.

O Github é uma plataforma de desenvolvimento baseada em Git, sendo a mais popular no mundo. Atualmente, conta com mais de 330 milhões de repositórios, 4 milhões de organizações e 100 milhões de desenvolvedores (Github, 2023).

O Github possui recursos colaborativos como *wikis*, gerenciamento de tarefas para qualquer projeto e, também, algumas funcionalidades semelhantes às redes sociais como gerenciamento de perfil, possibilidade de compartilhar e/ou favoritar um repositório (BOUQUIN, 2015). Isso corrobora com o fato da popularidade dessa ferramenta para hospedagem de projetos de código-fonte aberto, ou *open-source* (OSS).

4.1.4.2 Zabbix

O Zabbix é uma ferramenta de código-fonte aberto, projetada para monitorar parâmetros relacionados à saúde e integridade de diversos elementos, como redes, servidores, máquinas virtuais, aplicativos, serviços, bancos de dados, sites e ambientes em nuvem, entre outros (Zabbix, 2023b). Uma característica distintiva do Zabbix é o seu mecanismo de notificação altamente flexível, que permite aos usuários configurar notificações por e-mail para praticamente qualquer tipo de evento. Essa capacidade de notificação imediata possibilita uma resposta ágil aos problemas identificados nos servidores monitorados.

O Zabbix foi uma das ferramentas citadas por López et al. (2022) para coleta de dados do sistema em tempo de execução. Outra ferramenta citada neste estudo foi o Nagios, que se configurou inacessível devido à limitação de acesso a recursos disponibilizados de maneira gratuita pela ferramenta.

Além disso, o Zabbix oferece recursos avançados de relatórios e visualização de dados, utilizando as informações armazenadas para fornecer informações valiosas aos usuários. Essa funcionalidade torna o Zabbix uma solução ideal para o gerenciamento de capacidade, permitindo um acompanhamento eficaz e uma análise abrangente do desempenho dos sistemas monitorados. O Zabbix possui diversas funcionalidades (Zabbix, 2023a), dentre as quais, destacam-se as que serão mais bem aproveitadas na pesquisa:

- **Coleta de dados**
  - Verificações de disponibilidade e desempenho: São realizadas verificações regulares para determinar a disponibilidade e o desempenho dos componentes monitorados. Isso permite identificar qualquer falha ou degradação no sistema em tempo hábil, e

[2] https://pt.wikipedia.org/wiki/Linus_Torvalds

  - Coleta de dados em intervalos pré-definidos e agendados: É possível configurar intervalos de coleta de dados customizados para cada elemento monitorado. Isso garante que as informações sejam atualizadas conforme a frequência desejada, possibilitando um monitoramento em tempo real.

- **Armazenamento de dados históricos**

  - Dados armazenados em um banco de dados: os dados históricos são armazenados em um banco de dados seguro e confiável. Esse ambiente de armazenamento permite uma organização estruturada dos dados e facilita o acesso e a análise posterior;
  - Histórico configurável: O sistema permite a configuração do período para o armazenamento dos dados históricos. Essa configuração é ajustável às necessidades específicas do ambiente monitorado, permitindo um controle preciso sobre a quantidade de dados armazenados, e
  - Procedimento de limpeza (*housekeeping*) nativo: O sistema possui um procedimento de limpeza nativo, conhecido como *housekeeping*, que gerencia a exclusão automática de dados históricos obsoletos. Esse procedimento garante a otimização do espaço de armazenamento e a remoção de informações desnecessárias, mantendo o banco de dados limpo e eficiente.

- **Facilidade na configuração**

  - Diferentes maneiras de instalação: É possível instalar o sistema de diferentes maneiras (código-fonte, pacotes, contêineres), facilitando o seu *setup* e reprodução dessa instalação em diferentes ambientes, e
  - Adiciona dispositivos monitorados como hosts: Os dispositivos a serem monitorados são facilmente adicionados ao sistema como hosts. Essa funcionalidade permite que novos dispositivos sejam rapidamente integrados ao ambiente de monitoramento.

4.1.4.3 Apache JMeter

O Apache JMeter é uma ferramenta desenvolvida em Java e amplamente utilizada para realizar testes de carga, principalmente em sistemas web. Sua principal funcionalidade é simular o comportamento de múltiplos usuários, permitindo a realização de requisições HTTP simultâneas para fluxos específicos dos sistemas em teste. Dessa forma, é possível avaliar o uso dos recursos computacionais, como se estivessem em uma situação real, simulando o comportamento de usuários para avaliar o desempenho do sistema em condições de uso intensivo (Apache Software Foundation, 2022).

É importante destacar que o JMeter não simula o comportamento de um navegador web completo, uma vez que não executa JavaScript nem renderiza HTML. Seu foco principal está em simular as chamadas HTTP, permitindo testar o desempenho, escalabilidade e a estabilidade do sistema alvo (Apache Software Foundation, 2022).

4.1.4.4 Scikit-learn

O Scikit-learn é uma biblioteca de código-fonte aberto desenvolvida para a linguagem de programação Python, amplamente utilizada para criar algoritmos baseados em aprendizado de máquina. Essa biblioteca oferece uma ampla gama de ferramentas para processamento de dados, seleção de modelos de aprendizado de máquina, avaliação de modelos, entre outras funcionalidades (PEDREGOSA et al., 2011).

No contexto desta monografia, o Scikit-learn é utilizado devido à sua vasta coleção de algoritmos prontos para uso em problemas de regressão numérica. Um desses algoritmos é o de Floresta Aleatória, mencionado na Seção 2.3.2. Esse algoritmo será empregado para estabelecer a relação entre os dados coletados durante os testes de carga do estudo observacional.

4.1.4.5 Jupyter Notebook

O Jupyter Notebook é uma aplicação de código-fonte aberto que permite a criação e compartilhamento de documentos interativos que contêm código, texto explicativo, equações matemáticas e diferentes visualizações, como figuras e modelos 3D. Esses documentos, ou *notebooks*, como são popularmente chamados, fornecem uma plataforma flexível e colaborativa para análise de dados, modelagem computacional, visualização de resultados e documentação de projetos de pesquisa e desenvolvimento (Jupyter, 2023).

O Notebook Jupyter processa instruções de uma variedade de linguagens de programação, tornando-o uma ferramenta versátil para pesquisadores e estudantes em várias disciplinas acadêmicas. No contexto desta pesquisa, ele é utilizado para automação do processo de tratamento e análise dos dados, aproveitando suas funcionalidades para programar um algoritmo que seja capaz de, a partir de uma entrada de dados brutos, gerar os resultados da análise de comparação de desempenho conforme a técnica utilizada.

4.1.4.6 Especificação das Máquinas

Devido à demanda computacional necessária para simular o comportamento de usuários durante os testes, a geração de carga na mesma máquina em que o produto é executado pode ocasionar interferências nos resultados das medições. Para garantir maior precisão nos dados, optamos por utilizar três máquinas distintas.

A primeira máquina foi utilizada para gerar a carga de usuários com JMeter, além de hospedar o servidor Zabbix que forneceu os dados de uso dos recursos computacionais disponíveis nas outras máquinas, coletados pelo agente Zabbix. Além desta, foram utilizadas duas máquinas distintas para o pleno funcionamento do *software* estudado, a primeira máquina possui maior capacidade computacional para executar o servidor, e a segunda foi utilizada para o banco de dados. Os detalhes de cada máquina podem ser encontrados abaixo:

- **Máquina 1 (Zabbix server e JMeter):**
  - **Processador:** Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
  - **CPUs:** 2
  - **Memória RAM:** 4 GB
  - **Sistema operacional:** Debian 11 (bullseye)
- **Máquina 2 (Servidor):**
  - **Processador:** Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
  - **CPUs:** 4
  - **Memória RAM:** 16 GB
  - **Sistema operacional:** Debian 11 (bullseye)
- **Máquina 3 (Banco de dados):**
  - **Processador:** Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
  - **CPUs:** 2
  - **Memória RAM:** 2 GB
  - **Sistema operacional:** Debian 11 (bullseye)

A criação e instância desta infraestrutura de observação foi possível graças ao LAPP1S - Laboratório Avançado de Pesquisa, Produção e Inovação em Software[3], coordenado por professores do curso de graduação em Engenharia de Software, na Universidade de Brasília.

4.2 Execução e Análises do Estudo Observacional

4.2.1 Configuração do Ambiente

O procedimento de configuração dos ambientes revela-se indispensável para a efetiva realização da coleta precisa e subsequente análise dos dados no âmbito do estudo

[3] https://github.com/lappis-unb

observacional. Nesse contexto, a ferramenta JMeter é empregada com o propósito de gerar as cargas de teste da aplicação, enquanto o Zabbix é utilizado para efetuar a coleta das métricas relacionadas à utilização computacional da aplicação em estudo.

Para operacionalizar o ambiente de observação deste estudo, foram criadas as máquinas virtuais, descritas na Subseção 4.1.4.6. Na **Máquina 1**, foram instalados o servidor Zabbix, citado na Subseção 4.1.4.2 e o Apache JMeter, citado na Subseção 4.1.4.3.

Na **Máquina 2**, foi implantada a versão 2.1.0, em essência, um sistema *WEB*, escrito em Python-Django, versão 4.2.1. Já na **Máquina 3**, foi instalado o sistema de gerenciamento de banco de dados Postgres, na versão 14, onde foi implantando o banco de dados relacionado a versão do produto observado. Essa implantação foi viabilizada mediante execução de *scripts* , que incorporam os contêineres de cada serviço, disponibilizados pela própria empresa.

4.2.1.1 Configuração do Teste de Carga

A configuração do teste de carga demandou uma análise aprofundada, com o intuito de identificar qual o fluxo do produto possuiria maior relevância para organização proprietária do produto, e também que fosse mais adequando ao contexto deste estudo. Dada a complexidade inerente ao sistema em questão, tornou-se necessário simplificar o cenário de uso, restringindo-o a um único fluxo observável, a fim de possibilitar a adequada configuração do ambiente de teste por meio do JMeter.

Para identificar o fluxo do produto mais relevante, do ponto de vista da empresa, foi realizada uma entrevista, informal, a dois profissionais sêniores desta organização. Um dos profissionais é responsável pela área de qualidade (QA), e outro, um desenvolvedor. Ambos possuem mais de quatro anos de empresa, além de amplo conhecimento sobre o produto. As perguntas dirigidas aos profissionais foram as seguintes:

- Qual é o fluxo de transação mais utilizado, do produto em uso?
- Qual fluxo agrega mais valor de negócio ao produto?
- Qual fluxo está sendo mais impactado com atualizações de código-fonte?
- Atualmente, existe algum fluxo que sofrerá modificações visando a melhoria de desempenho? Se sim, qual?
- Qual fluxo mais necessita de monitoramento de desempenho?

As respostas obtidas convergiram para um fluxo em comum. Um conjunto composto por 23 rotas foi fornecido, juntamente com a sequência específica em que essas rotas deveriam ser acionadas, os dados requeridos para autenticação, o método HTTP, o corpo

da requisição e os parâmetros associados a cada uma dessas rotas. Subsequentemente, o cenário de teste foi definido, utilizando como base os dados fornecidos pela empresa.

Um grupo de usuários para realizar as requisições foi então criado na ferramenta JMeter. Para definir a quantidade final de usuários presentes no grupo, foram realizados testes variando essa quantidade e observando o comportamento das métricas de uso de CPU e uso de memória RAM. Caso essas métricas alcançassem o limite de 100% de utilização, a quantidade de usuários no grupo era reduzida, visto que ao atingir essa marca, o próprio sistema operacional das máquinas utilizadas iria intervir para reduzir a utilização dos recursos. Dessa forma, foi definida a quantidade final de 10 usuários simultâneos para o cenário do teste de carga.

No início da execução do teste de carga, percebeu-se que havia, uma sobrecarga substancial dos recursos da infraestrutura computacional onde o sistema estava implantado. Isso ocorria devido ao início simultâneo das atividades de todos os usuários. Essa sincronização resultava em um "gargalo" temporário. Para mitigar esse impacto, foi implementado um intervalo de 10 segundos entre a inicialização de cada usuário dentro do grupo. Sendo que, cada usuário foi configurado para reiniciar o fluxo desde o início, caso o concluísse ou caso ocorresse algum erro na requisição, mantendo esse comportamento até o término da duração planejada do teste.

A estratégia de introduzir um intervalo temporal entre a inicialização dos disparos de requisições por parte dos usuários, distribuiu de maneira mais uniforme a carga sobre os recursos do sistema, evitando picos de demanda simultâneos. Essa abordagem visou simular, de forma mais fidedigna, o ambiente de uso do sistema observado, onde os usuários não iniciam suas atividades de maneira simultânea.

Nguyen et al. (2012) informam que milhares de amostras são suficientes para ajustar o algoritmo de regressão linear utilizado para realizar a normalização dos dados para técnica de Gráficos de Controle. Como o período de coleta de métricas deste estudo foi de um minuto, para alcançar a quantia de mil amostras, seria necessário executar o teste de carga por aproximadamente 17 horas consecutivas. Devido às limitações de tempo do estudo; instabilidade de acesso às máquinas utilizadas de maneira remota, e a necessidade de realizar a comparação do teste executado em diferentes versões do produto, optou-se por utilizar o período de cinco horas de execução contínua para cada versão analisada, produzindo assim 300 amostras por teste.

Com o propósito de simplificar a execução dos testes de carga, foi desenvolvido um contêiner Docker utilizando a versão 5.5 do JMeter, utilizando como base a imagem disponível no Dockerhub sob o nome "justb4/jmeter" [4]. Este contêiner possibilita a utilização dos mesmos parâmetros disponíveis na ferramenta JMeter. Adicionalmente, foi necessário

[4] https://hub.docker.com/r/justb4/jmeter/

transferir o arquivo contendo o roteiro do teste de carga para o interior do contêiner, para os mesmos poderem ser acessados pelo ambiente Docker. O Dockerfile que encapsula essa configuração específica pode ser visualizado no Código-Fonte 1.

Código 1 – Dockerfile para criação do contêiner do JMeter

```
FROM justb4/jmeter:5.5

COPY ./testes .
```

Fonte: Autores

O *script* apresentado no Código-Fonte 2 foi elaborado para configurar os parâmetros relevantes, incluindo a porta na qual o contêiner seria executado; o fuso-horário utilizado na geração de relatórios, e o gerenciamento de volumes para extrair os resultados dos testes do contêiner. Adicionalmente, foram incluídas instruções para gerar relatórios de execução contendo registros de todas as chamadas feitas ao sistema durante o período do teste. Esses relatórios incluem informações relevantes, como a duração, código HTTP obtido como resposta, data e horário de início de cada chamada, proporcionando uma visão abrangente do desempenho do sistema, durante o teste.

Código 2 – Script docker compose para execução do teste de carga

```
services:
  jmeter:
    container_name: docker_jmeter
    build: .
    environment:
      TZ: "America/Sao_Paulo"
    volumes:
      - ./results:/opt/apache-jmeter-5.5/logs
    ports:
        - 1099:1099
    command: "-Dlog_level.jmeter=DEBUG -n -t flowCEP.jmx \
    -l /opt/apache-jmeter-5.5/logs/test-plan.csv \
    -j /opt/apache-jmeter-5.5/logs/jmeter.log \
    -e -o /opt/apache-jmeter-5.5/logs/reports/"
```

Fonte: Autores

Finalmente, no Código-Fonte 3, é possível visualizar o arquivo Makefile elaborado, que descreve uma sequência de comandos que configura o ambiente para a execução dos testes. Esses comandos englobam a remoção de vestígios de testes anteriores, seguida pela reinicialização do teste utilizando o ambiente Docker. Este Makefile simplifica e automatiza o processo de preparação e execução dos testes de carga, promovendo eficiência e consistência nas repetidas execuções dos testes.

Código 3 – Makefile para execução do teste de carga

```
JMETER_FOLDER = ./jmeter
DOCKER_COMPOSE_JMETER = docker-compose -f ${JMETER_FOLDER}/docker-compose.yml

load_test:
        @rm -rf ${JMETER_FOLDER}/results
        @${DOCKER_COMPOSE_JMETER} down
        @${DOCKER_COMPOSE_JMETER} rm
        @mkdir ${JMETER_FOLDER}/results
        ${DOCKER_COMPOSE_JMETER} up --build
```

Fonte: Autores

4.2.1.2 Configuração da Coleta de Métricas

Conforme explicado na Subseção 4.1.4.2, o Zabbix representa a fonte de informação para a coleta das métricas relacionadas ao uso computacional da aplicação, que por sua vez, representam as variáveis dependentes neste estudo. Sua instalação mostrou-se uma tarefa relativamente simples, dada a disponibilidade de diferentes abordagens documentadas para este propósito. Optou-se, de forma deliberada, pela implementação mediante imagens de contêineres, fundamentada na conveniência de replicação e na familiaridade com essa tecnologia.

O agente Zabbix foi implementado na **Máquina 2** (4.1.4.6) e na **Máquina 3** (4.1.4.6), possibilitando a captura de métricas convencionais e específicas do sistema operacional, a exemplo de CPU e memória. Adicionalmente, o agente Zabbix possibilita a coleta de métricas personalizadas, como aquelas relacionadas a bancos de dados, mediante a utilização de macros de configuração ou *scripts*. Embora o agente possa ser instalado em variados sistemas operacionais, destaca-se que o servidor deve ser implantado em ambientes Unix ou Linux.

O servidor Zabbix pode recuperar dados para o monitoramento dos serviços usando como fonte de dados os agentes instalados em máquinas distintas, ou a própria máquina onde o servidor se encontra instalado. Este servidor preserva um histórico dos dados coletados em um banco de dados, no presente contexto, o PostgreSQL, do qual são gerados gráficos, painéis de acompanhamento e apresentações alternadas que exibem informações de forma sequencial.

Os *scripts* utilizados para funcionamento do servidor Zabbix estão listados nos Códigos-Fonte 4 e 5. O Código-Fonte 4 representa a criação do servidor zabbix que utiliza a imagem zabbix/zabbix-server-pgsql [5].

O Código-Fonte 5 representa os contêineres do banco de dados postgreSQL, onde foram persistidos os dados das métricas de uso computacional coletados, além, do servidor

[5] https://hub.docker.com/r/zabbix/zabbix-server-pgsql/

Código 4 – *Script docker compose* reduzido do servidor zabbix

```
  zabbix-server:
    hostname: zabbix-server
    image: zabbix/zabbix-server-pgsql
    ports:
      - "10051:10051"
    restart: on-failure
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - ./zbx_env/usr/lib/zabbix/alertscripts:/usr/lib/zabbix/alertscripts:ro
      - ./zbx_env/usr/lib/zabbix/externalscripts:/usr/lib/zabbix/externalscripts:ro
      - ./zbx_env/var/lib/zabbix/dbscripts:/var/lib/zabbix/dbscripts:ro
      - ./zbx_env/var/lib/zabbix/export:/var/lib/zabbix/export:rw
      - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
      - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
      - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
      - ./zbx_env/var/lib/zabbix/mibs:/var/lib/zabbix/mibs:ro
      - snmptraps:/var/lib/zabbix/snmptraps:rw
    env_file:
      - .postgres.env
      - .server.env
    depends_on:
      - postgres-server+
```

Fonte: Autores

web Zabbix configurado para visualizar e recuperar os dados coletados.

O Código-Fonte 6 representa a criação do contêiner responsável pelo agente Zabbix, que reportou os dados de uso de recursos computacionais coletados ao servidor, tanto ao nível de Sistema Operacional, quanto Banco de Dados. Limitou-se os recursos do agente, para que ele não afetasse as máquinas hospedeiras negativamente, pois estas, foram aquelas utilizadas para a coleta das métricas de desempenho. Isso poderia causar fatores de confusão e gerar ameaças à validade do estudo. As métricas do sistema operacional são coletadas automaticamente, porém, para o banco de dados, é necessário configurar *macros* para fazê-lo coletar dados especificamente de banco de dados. Essa configuração pode ser visualizada na documentação.

A configuração do processo de coleta de métricas por meio do Zabbix é estendida ao servidor web. Nessa instância, incumbe ao servidor a responsabilidade de determinar quais métricas serão objeto de coleta, a partir dos modelos ou *templates* disponíveis para essa finalidade, ou mediante a configuração manual dessas métricas. Adicionalmente, o servidor web assume a tarefa de especificar o intervalo de coleta para cada uma das métricas selecionadas. Caso seja estabelecido um intervalo de um minuto para a coleta das métricas, o resultado consistirá na média obtida durante esse período de um minuto, uma vez que, frequentemente, a coleta é realizada em intervalos de segundos.

No âmbito do presente estudo observacional, empregou-se duas máquinas hospedeiras para o envio de métricas de uso computacional ao servidor Zabbix. Estas máquinas desempenham as funções de alocação da aplicação e do banco de dados utilizado por ela. Optamos por utilizar dois *templates* distintos, a saber, *"Linux by Zabbix agent"* e

Código 5 – *Script docker compose* reduzido da criação do banco e servidor *web* zabbix

```
  postgres-server:
    container_name: postgres-server
    image: postgres:15-alpine
    ports:
      - "5432:5432"
    volumes:
      - ./zbx_env/var/lib/postgresql/data:/var/lib/postgresql/data:rw
      - ./env_vars/.ZBX_DB_CA_FILE:/run/secrets/root-ca.pem:ro
      - ./env_vars/.ZBX_DB_CERT_FILE:/run/secrets/server-cert.pem:ro
      - ./env_vars/.ZBX_DB_KEY_FILE:/run/secrets/server-key.pem:ro
    env_file:
      - .postgres.env
    restart: unless-stopped

  zabbix-web-nginx-pgsql:
    container_name: zabbix-web-nginx-pgsql
    image: zabbix/zabbix-web-nginx-pgsql
    ports:
      - "443:8443"
      - "80:8080"
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /etc/timezone:/etc/timezone:ro
      - ./zbx_env/etc/ssl/nginx:/etc/ssl/nginx:ro
      - ./zbx_env/usr/share/zabbix/modules/:/usr/share/zabbix/modules/:ro
    restart: on-failure
    env_file:
      - .postgres.env
    environment:
      - ZBX_SERVER_HOST=zabbix-server
      - PHP_TZ=America/Sao_Paulo
    depends_on:
      - zabbix-server
```

Fonte: Autores

"PostgreSQL by Zabbix agent 2" para a máquina responsável pelo banco de dados. Adicionalmente, para a máquina que hospeda a aplicação, estamos utilizando os *templates* *"Linux by Zabbix agent"* e *"Server Health by Linux"*.

Cada um desses *templates* oferece uma gama de gráficos e configurações predefinidas para a coleta de diversos itens, alinhados às exigências de nosso levantamento métrico. Os itens coletados em cada máquina hospedeira, conforme configurados pelos *templates* adotados, podem ser acessados por meio da aba *"Data Collection -> Items"* em formato textual. Nesse contexto, é possível realizar filtros com base em data e hora de coleta, além da capacidade de converter manualmente os resultados para o formato CSV em etapa subsequente.

4.2.2 Consolidação dos dados

Utilizou-se o relatório gerado após a execução do teste de carga, fornecido pelo JMeter, além dos arquivos com as métricas de desempenho coletadas pela ferramenta Zabbix, e gerou-se um arquivo(.csv) consolidado. Essa consolidação representa a junção dos arquivos gerados pelas diferentes ferramentas e consequente integração de dados. Esse processo foi automatizado por meio de um *script*, que agrupou essas informações em

Código 6 – *Script docker compose* do agente zabbix

```
version: "3.8"
services:

  zabbix-agent:
    image: zabbix/zabbix-agent
    deploy:
      resources:
        limits:
          cpus: '0.2'
          memory: 128M
      mode: global
    privileged: true
    pid: "host"
    restart: unless-stopped
    volumes:
      - /var/run:/var/run
      - /etc/localtime:/etc/localtime:ro
      - /etc/timezone:/etc/timezone:ro
      - ./zbx_env/etc/zabbix/zabbix_agentd.d:/etc/zabbix/zabbix_agentd.d:ro
      - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
      - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
      - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
    ports:
      - '10050:10050'
    environment:
      - ZBX_SERVER_HOST=IP_ADDR
```

Fonte: Autores

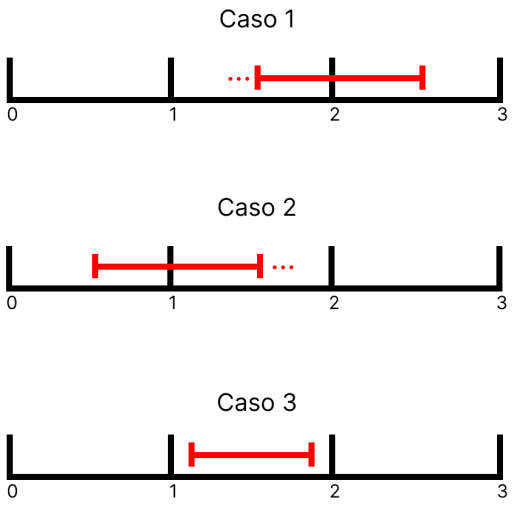
intervalos fixos de um minuto. O arquivo consolidado registra a data e hora de início e fim de cada intervalo, a quantidade de requisições ao sistema, para cada um dos endpoints, bem como, as métricas de desempenho computacional coletadas nesses intervalos. Essa consolidação também contribuiu para a análise integrada das métricas provenientes de ambas as fontes, contribuindo para a organização e padronização dos dados, facilitando a análise subsequente.

Para determinar quais requisições aos *endpoints* seriam contabilizadas em cada intervalo, definiu-se que seriam consideradas as requisições que estavam sendo processadas em qualquer momento dentro do intervalo. Três cenários possíveis foram definidos, conforme exemplificado na Figura 17. Os intervalos são representados em preto e as requisições em vermelho. Nesse exemplo, a requisição será contabilizada no intervalo entre os minutos 1 e 2 em todos os casos apresentados.

O primeiro caso abrange requisições que iniciaram antes do fim do intervalo e terminaram após o mesmo; o segundo inclui requisições que começaram antes do início do intervalo e terminaram após o mesmo; por fim, o terceiro caso engloba requisições que começaram e terminaram dentro do intervalo. Essa abordagem visa garantir uma contagem mais abrangente e variada das atividades de processamento de requisições, proporcionando uma visão completa do volume de trabalho durante cada intervalo de tempo determinado.

A função criada para realizar essa contagem a partir do arquivo consolidado, pode

Figura 17 – Possíveis casos de inclusão na contagem de *endpoints*



Fonte: Autores

ser visualizada no Código-Fonte 7. A data e hora de início e término de cada requisição são coletadas, e essas informações são comparadas com a data e hora de início e término do intervalo analisado, conforme os casos definidos. Se a requisição se enquadrar em algum dos casos, o tempo de resposta em segundos é adicionado a uma variável temporária de soma. Simultaneamente, tanto o contador geral de chamadas quanto o contador específico do endpoint chamado são incrementados. O processo é repetido enquanto houver registros de requisições ou até que as requisições analisadas comecem a ocorrer após o fim do intervalo.

Ao final, a variável contendo a soma dos tempos de resposta é dividida pela quantidade total de requisições realizadas no intervalo, resultando no tempo de resposta médio para as requisições feitas naquele intervalo específico.

Algumas métricas, como utilização de CPU e memória RAM, foram coletadas pela ferramenta Zabbix com um intervalo fixo de um segundo, resultando em sessenta pontos de dados coletados para o intervalo de um minuto. Para serem integradas ao arquivo CSV consolidado, o valor médio da métrica durante o intervalo foi calculado. Esse processo visa fornecer uma visão mais representativa do comportamento dessas métricas ao longo do tempo, considerando a granularidade da coleta original.

4.2.3 Análise dos dados

4.2.3.1 Remoção de valores atípicos

Em virtude da natureza dos dados originados pelo uso de recursos computacionais, tornou-se necessário um tratamento inicial no conjunto de dados, processo de filtragem, vi-

Código 7 – Função responsável pela contagem de requisições no intervalo

```
while (linha_jmeter < len(jmeter_csv.index)
        and timestamp_inicio_jmeter < timestamp_fim_intervalo):
    timestamp_inicio_jmeter = jmeter_timestamp(jmeter_csv['timeStamp'].iloc[linha_jmeter])
    timestamp_final_jmeter = jmeter_timestamp(jmeter_csv['timeStamp_end'].iloc[linha_jmeter])
    caso_1 = (timestamp_inicio_jmeter < timestamp_fim_intervalo
                and timestamp_final_jmeter > timestamp_fim_intervalo)
    caso_2 = (timestamp_inicio_jmeter < timestamp_inicio_intervalo
                and timestamp_final_jmeter > timestamp_inicio_intervalo)
    caso_3 = (timestamp_inicio_jmeter > timestamp_inicio_intervalo
                and timestamp_final_jmeter < timestamp_fim_intervalo)
    if caso_1 or caso_2 or caso_3:
        timestamp_tempo_de_resposta = timestamp_final_jmeter - timestamp_inicio_jmeter
        tempo_de_resposta += timestamp_tempo_de_resposta.total_seconds()
        contador[INDICE_TOTAL_DE_CHAMADAS] += 1
        contador_index = 0
        while ENDPOINTS[contador_index] not in jmeter_csv['label'].iloc[linha_jmeter]:
            contador_index += 1
        contador[contador_index] += 1
    linha_jmeter += 1
media_tempo_de_resposta = tempo_de_resposta/contador[INDICE_TOTAL_DE_CHAMADAS]
```

Fonte: Autores

sando extrair apenas os dados que melhor representem o fenômeno, na maior parte. Dessa maneira, eliminamos variáveis espúrias, tipicamente, fatores de confusão. Para removê-las, realizamos a análise inter-quartil.

Para a utilização da técnica de floresta aleatória, foram eliminados pontos de observação baseando-se nos valores extremos do total de requisições realizadas. Devido ao intervalo inicial definido para cada usuário começar a realizar requisições, os primeiros minutos do teste apresentam uma quantidade crescente de chamadas ao sistema, afetando as métricas de forma diferente de quando todos os usuários estão ativos. Ao final do teste de carga, uma quantidade decrescente de requisições totais é observada, portanto, também é identificada pela técnica.

A medida pertinente à remoção foi um pouco diferente na técnica do gráfico de controle. Nesse contexto, foram eliminadas os valores das métricas computacionais que geravam fatores de confusão extremos, selecionando apenas os dados das métricas que estavam mais próximos da mediana. Desse modo, evitou-se a comparação de fatores de confusão entre os conjuntos de dados. Esses fatores de confusão, geralmente, geravam uma distribuição não normal dos valores das métricas, desrespeitando a premissa de normalidade na saída do processo da própria técnica.

Os quartis dividem os dados de uma distribuição em quartas partes iguais, sendo que cada parte representa 25% dos dados. São definidos como Q1 = Primeiro Quartil, Q2 =

Mediana e Q3 = Terceiro Quartil. O intervalo interquartil (IQR) é definido pela diferença entre Q3 e Q1, sendo que quaisquer dados situados além dos limites $Q3 + 1.5 * IQR$ ou $Q1 - 1.5 * IQR$ são identificados como valores atípicos (*outliers*). A função que realiza a identificação dos valores atípicos, pode ser visualizada no Código-Fonte 8.

Código 8 – Função responsável por recuperar os *outliers* da amostra

```
def recupera_outliers(df, target):
  outliers_indexes = []
  q1 = df[target].quantile(0.25)
  q3 = df[target].quantile(0.75)
  iqr = q3-q1
  maximum = q3 + (1.5 * iqr)
  minimum = q1 - (1.5 * iqr)
  outlier_samples = df[(df[target] < minimum) | (df[target] > maximum)]
  outliers_indexes.extend(outlier_samples.index.tolist())
  outliers_indexes = list(set(outliers_indexes))
  return outliers_indexes
```

Fonte: Autores

4.2.3.2 Análise com Floresta Aleatória

Ao comparar os dados coletados das versões sob análise, é importante ressaltar que dois testes nunca apresentarão comportamento idêntico. Falhas em requisições, utilização de recursos computacionais em tarefas secundárias, variações na rede, entre outros fatores, são situações que impactam a forma como o sistema lida com as requisições, podendo interferir no processamento e no tempo de resposta. Portanto, mesmo ao comparar testes distintos realizados na mesma versão do produto, é essencial pré-processar os dados, garantindo que estejam sempre normalizados na mesma escala de carga aplicada.

O propósito de empregar o algoritmo de floresta aleatória é precisamente realizar essa normalização, conforme descrito na Seção 2.3.2.4, considerando as variações resultantes das diferenças entre as versões e também as variações inerentes a fatores externos.

Assim, elaborou-se uma função que recebe os dados do arquivo consolidado, no formato CSV contendo o resultado do teste de carga. Essa função extrai as métricas de desempenho (variáveis dependentes) da quantidade de requisições de cada *endpoint* (variáveis independentes) para cada intervalo da amostra. Além de treinar um algoritmo de floresta aleatória para cada uma das métricas utilizando o *RandomForestRegressor* da biblioteca Python *scikit learn*. A implementação dessa função é apresentada no Código-Fonte 9.

Dessa maneira, as versões distintas do teste de carga foram comparadas, sempre utilizando como base uma versão do fluxo conforme especificado pela empresa responsável pelo produto. Para avaliar a precisão das predições estimadas pelo algoritmo de

Código 9 – Função responsável pelos algoritmos de Floresta Aleatória

```
def extrai_dados(dados_csv):
    arrays = extrai_arrays(dados_csv)
    requisicoes = arrays[0]
    metricas = arrays[1]

    florestas_aleatorias = []
    metricas= []

    for nome_da_metrica in range(len(nomes_das_metricas)):
      metrica = []
      for linha in metricas_conjuntas:
        metrica.append(linha[nome_da_metrica])

      metricas.append(metrica)
      floresta_aleatoria = RandomForestRegressor(random_state=0)
      floresta_aleatoria = floresta_aleatoria.fit(requisicoes, metrica)
      florestas_aleatorias.append(floresta_aleatoria)

    return [florestas_aleatorias, requisicoes, metricas]
```

Fonte: Autores

floresta aleatória, treinado com a versão base, analisou-se o coeficiente de determinação R^2-ajustado e o Erro Percentual Absoluto Médio (MAPE).

Calculou-se essas medidas a partir do conjunto de dados representado pelas métricas de desempenho, coletadas durante a execução dos testes de carga (base), e do conjunto de dados representado pelas métricas de desempenho estimadas pelo algoritmo, considerando as mesmas chamadas. Os resultados dos cálculos podem ser visualizados na Tabela 3.

Caso a regressão numérica fosse perfeita, não haveria disparidade entre os valores reais e as estimativas feitas pelo algoritmo Floresta Aleatória. Porém, para todas as métricas analisadas, o Erro Percentual Absoluto Médio (MAPE) foi maior que zero. Portanto, ao comparar versões diferentes com a versão base, a interpretação do MAPE deve sempre levar em consideração que podem ser identificadas variações inerentes à assertividade da técnica de regressão utilizada.

Para identificar se houve ou não variação entre as métricas de desempenho nas versões, utilizou-se a medida Cliff Delta, apresentada na Subseção 2.3.2.4, como a abstração estatística, quantitativa, para interpretar a diferença observada nos resultados da execução dos testes. Assim, comparou-se o conjunto de dados estimados pelo algoritmo (base), com aqueles extraídos da versão do sistema em teste (alvo), visto que o algoritmo estima como as chamadas se comportam no cenário utilizado para treinamento.

Graus de diferenças classificados como “Trivial” ou “Pequeno” indicam que houve pouca ou nenhuma variação ao comparar as versões. Já graus classificados como “Médio”, indicam que houve uma diferença moderada. Portanto, cabe ao observador definir se a diferença será considerada, de fato, significativa. Por fim, graus classificados como

Tabela 3 – Precisão nas previsões do algoritmo floresta aleatória da versão base

Métrica	**R² ajustado**	**MAPE**
Uso de CPU da aplicação	0,889	2,528%
Uso de Memória RAM da aplicação	0,846	0,183%
Uso de CPU do banco de dados	0,932	2,658%
Uso de Memória RAM do banco de dados	0,871	0,495%
Linhas buscadas no banco de dados	0,915	4,945%
Linhas retornadas pelo banco de dados	0,916	3,898%
Linhas escritas no banco de dados	0,871	1,697%
Tempo de resposta do sistema	0,938	2,4%

Fonte: Autores

"Grande", indicam que houve uma substancial diferença, ou seja, a métrica se comportou de maneira claramente diferente nas versões analisadas, portanto, o sistema deve ser analisado para identificar o que culminou nessa diferença.

Na Tabela 4 é exibido o resultado da comparação entre a versão base com a versão alvo, no cenário "versão boa x versão boa". Quer seja, a mesma versão do sistema em teste, sem inserção de problemas de desempenho. Ou seja, comparando ele com ele mesmo estimado.

É notável que, ainda que tenham sido utilizados dados da mesma versão do sistema como parâmetro para o treinamento do modelo, e para realizar as estimativas, houve uma diferença significativa nas métricas de uso de memória RAM. Isso ocorreu tanto na Máquina 2, onde a aplicação está implantada, quanto na Máquina 3, que hospeda o banco de dados.

Quando a diferença é positiva, isso indica que as métricas de desempenho assumem valores maiores na versão base, do que na versão alvo. Isso sugere a interpretação que a versão alvo possui melhor desempenho. Já, quando a diferença é negativa, indica que as métricas de desempenho assumem valores menores na versão base, do que na versão alvo. Isso sugere a interpretação que a versão alvo possui pior desempenho.

Portanto, os resultados apontam a métrica de uso de memória RAM na Máquina 2 foram melhores na versão base, e a mesma métrica relacionada à Máquina 3 obteve

Tabela 4 – Comparação com floresta aleatória: versão boa x versão boa

Métrica	Valor Cliff Delta	Grau de Diferença Cliff Delta	MAPE
Uso de CPU da aplicação	0,048	Trivial	11,818%
Uso de Memória RAM da aplicação	- 1	Grande	53,245%
Uso de CPU do banco de dados	- 0,127	Trivial	6,761%
Uso de Memória RAM do banco de dados	0,897	Grande	4,052%
Linhas buscadas no banco de dados	- 0,132	Trivial	13,639%
Linhas retornadas pelo banco de dados	- 0,151	Pequeno	13,066%
Linhas escritas no banco de dados	0,014	Trivial	4,722%
Tempo de resposta do sistema	0,01	Trivial	4,67%

Fonte: Autores

melhores na versão alvo.

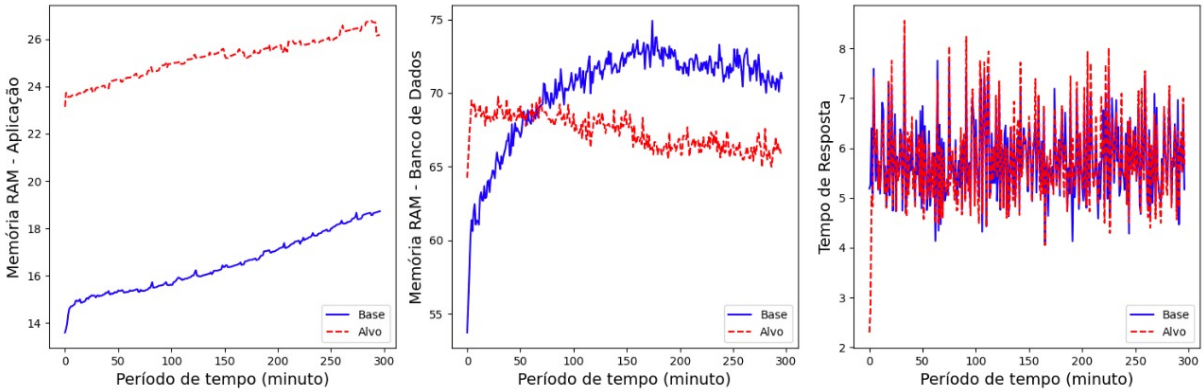
A disparidade pode ser elucidada ao examinar a variação nos valores dessas métricas durante a execução de ambos os testes. A Figura 18 exibe gráficos para as métricas de uso de memória RAM em ambas as máquinas, e a métrica de tempo de resposta, contendo amostras coletadas para o teste base e o teste alvo. É evidente que o comportamento das métricas de uso de memória RAM é mais constante em comparação com a métrica de tempo de resposta. Isso sugere que, no contexto deste estudo, as métricas de uso de memória RAM não indicam ter correlação com a variação da carga aplicada.

Devido às discrepâncias de comportamento identificadas ao comparar testes realizados na mesma versão utilizando floresta aleatória, as métricas de uso de memória RAM não serão incluídas nas análises entre versões distintas.

Como o Erro Percentual Absoluto Médio desconsidera o sinal, é possível que a técnica apresente um valor alto, mas o grau de diferença pelo Cliff Delta seja baixo. Isso ocorre quando os valores variam em módulo, de forma proporcional, ou seja, parte das amostras apresentam valores maiores no teste base quando comparado ao teste alvo, e outra parte das amostras apresentam valores maiores no teste alvo quando comparado ao teste base.

Para todas as métricas coletadas, exceto as de uso de memória RAM, a comparação

Figura 18 – Gráfico de métricas com comportamento distintos: versão boa x versão boa



Fonte: Autores

entre os testes apresentou índices de diferença baixos. Isso permite identificar que entre um teste e outro não houve grandes diferenças de desempenho, visto que nenhum problema foi inserido.

Gráficos exibindo a comparação entre duas versões estáveis podem ser observados nas Figuras 23 e 24, presentes no Apêndice B. São comparados os valores sem tratamento de dados em ambos os testes. Também são apresentados os valores do teste base sem tratamento comparados com os valores do teste base normalizados pelo algoritmo Floresta Aleatória. E, por fim, os valores do teste alvo sem tratamento comparados com os valores do teste base normalizados pelo algoritmo Floresta Aleatória, estimando como as chamadas do alvo impactariam as métricas na versão base.

Os resultados da análise de variação entre a versão base e a versão com o problema de aumento no limite de busca inseridos, podem ser observados na Tabela 5. Onde é possível notar que, todas as métricas apresentaram um grau de diferença “Grande”, segundo a interpretação do valor calculado com o Cliff Delta. Logo, a técnica aponta que houve variação no desempenho do sistema quando as duas versões são comparadas.

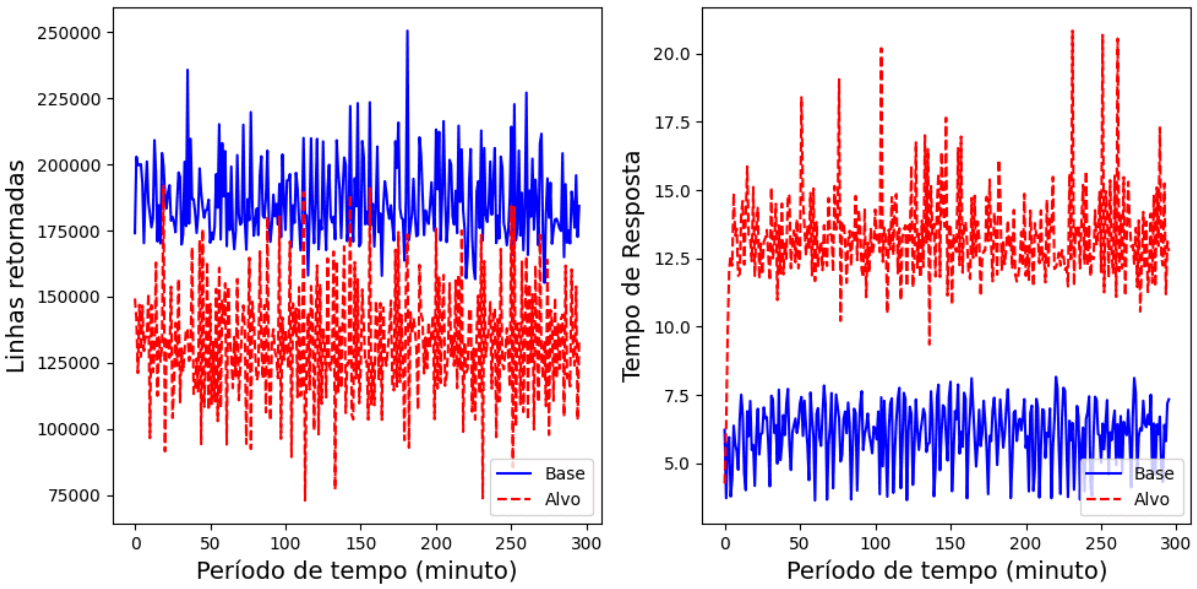
Como todas as métricas apresentaram grau de diferença “Grande”, é indicada a tendência de valores maiores em apenas uma das versões comparadas. Portanto, a taxa de erro calculada pelo MAPE pode ser utilizada para identificar qual seria a porcentagem média nessa diferença. Por exemplo, na Figura 19, é possível observar os valores das métricas de tempo de resposta e linhas retornadas pelo banco de dados quando as mesmas chamadas são feitas na versão base e na versão alvo com problemas de limite de busca. A métrica de linhas retornadas apresentou Cliff Delta positivo, indicando que os valores da versão base são maiores que na versão alvo, com uma diferença percentual média de 28,926%, conforme o valor MAPE calculado. Por outro lado, a métrica de tempo de resposta, com o Cliff Delta negativo, mostra o teste alvo com valores maiores, e a porcentagem média de diferença é de 124,04%.

Tabela 5 – Comparação com floresta aleatória: versão boa x versão limite de busca

Métrica	**Valor Cliff Delta**	**Grau de Diferença Cliff Delta**	**MAPE**
Uso de CPU da aplicação	0,972	Grande	27,449%
Uso de CPU do banco de dados	0,915	Grande	23,454%
Linhas buscadas no banco de dados	0,939	Grande	31,449%
Linhas retornadas pelo banco de dados	0,951	Grande	28,926%
Linhas escritas no banco de dados	- 1	Grande	68,765%
Tempo de resposta do sistema	- 0,991	Grande	124,041%

Fonte: Autores

Figura 19 – Gráfico de comparação para linhas retornadas e tempo de resposta



Fonte: Autores

É importante destacar que, ao introduzir problemas de desempenho que afetam o tempo de resposta nos testes de carga, a quantidade de requisições realizadas durante os intervalos de coleta pode variar sensivelmente. Esse comportamento pode resultar na coleta de valores menores para alguns recursos, devido ao maior consumo de outro, não significando necessariamente que o desempenho desses recursos melhorou.

Gráficos exibindo a comparação entre os conjuntos de dados da versão boa e a versão com problemas de limite de consulta podem ser observados nas Figuras 25 e 26,

presentes no Apêndice B. São comparados os valores sem tratamento de dados em ambos os testes. Também são apresentados os valores do teste base sem tratamento comparados com os valores do teste base normalizados pelo algoritmo Floresta Aleatória. E, por fim, os valores do teste alvo sem tratamento comparados com os valores do teste base normalizados pelo algoritmo Floresta Aleatória, estimando como as chamadas do alvo impactariam as métricas na versão base.

Os resultados da análise da variação observada, entre a versão base e a versão alvo, com o problema de dados excessivos podem ser observados na Tabela 6. Assim como, na versão alvo, com problemas de limites de busca, é possível notar que todas as métricas apresentaram um grau de diferença "Grande" segundo o valor do Cliff Delta, porém, a versão com o problema de dados excessivos apresenta valores de Erro Percentual Absoluto Médio bem mais extremos. Isso indica que a diferença de desempenho se intensificou, principalmente para a métrica de tempo de resposta.

Tabela 6 – Comparação com floresta aleatória: versão normal x versão dados excessivos

Métrica	**Valor Cliff Delta**	**Grau de Diferença Cliff Delta**	**MAPE**
Uso de CPU da aplicação	1	Grande	39,825%
Uso de CPU do banco de dados	1	Grande	48,738%
Linhas buscadas no banco de dados	0,957	Grande	50,038%
Linhas retornadas pelo banco de dados	0,992	Grande	51,374%
Linhas escritas no banco de dados	- 1	Grande	60,133%
Tempo de resposta do sistema	- 1	Grande	1198,405%

Fonte: Autores

Gráficos exibindo a comparação entre os conjuntos de dados da versão boa e a versão com problemas de dados excessivos podem ser observados nas Figuras 27 e 28, presentes no Apêndice B. São comparados os valores sem tratamento de dados em ambos os testes. Também são apresentados os valores do teste base sem tratamento comparados com os valores do teste base normalizados pelo algoritmo Floresta Aleatória. E, por fim, os valores do teste alvo sem tratamento comparados com os valores do teste base normalizados pelo algoritmo Floresta Aleatória, estimando como as chamadas do alvo impactariam as métricas na versão base.

4.2.3.3 Análise com Gráficos de Controle

Conforme mencionado na Subseção 2.3.3.2, para realizar a análise utilizando gráficos de controle é necessário satisfazer duas premissas básicas. A primeira premissa é referente a entrada de processo invariável, na qual a saída do processo tende a variar conforme as flutuações da entrada do mesmo. A segunda, o atendimento aos pressupostos teóricos da distribuição normal (normalidade na saída do processo), onde a relação linear que ocorre entre a entrada e saída do processo devem possuir comportamento semelhante ao da distribuição normal.

O procedimento de normalização dos dados revela-se pertinente apenas para métricas de desempenho que possuam relação linear positiva, com quantidade de chamadas a cada *endpoint*, após a carga aplicada, caso contrário, os dados resultantes da normalização podem estar enviesados pelos coeficientes do modelo de regressão linear. Para assegurar que essa relação linear proporcional seja efetivamente estabelecida, torna-se necessário selecionar métricas que sejam substancialmente impactadas pela carga do teste. De acordo com Chin (1998), são definidos intervalos de valores para o coeficiente de determinação ajustado (R^2), onde o limite de 0,67 a 1 representa um modelo de regressão linear considerado substancial.

Considerando, que cargas distintas são aplicadas a *endpoints* distintos, e que, cada *endpoint* exerce impacto diferente nos valores das métricas computacionais. Para interpretarmos o grau de relação entre as variáveis, realizou-se uma regressão linear múltipla. As variáveis independentes (*features*) correspondem a quantidade de requisições a cada *endpoint*.

Ao realizar a regressão linear múltipla, utilizou-se o algoritmo que computa os mínimos quadrados ordinários, conhecido como *Ordinary Least Squares* (OLS). A análise dos resultados da aplicação desse algoritmo, proveniente da biblioteca statsmodel [6], proporcionou uma compreensão aprimorada dos dados e do modelo treinado, viabilizando uma análise subsequente sobre a qualidade do modelo gerado. Para recuperar os valores dos coeficientes, ou parâmetros e o intercepto, ou constante do modelo, utilizou-se a função expressa no Código-Fonte 10.

Dessa forma, na versão base, realizou-se a seleção das métricas de desempenho e quantidade de chamadas aos endpoints, cujo R^2 ajustado, fosse maior do que 0,67. A Tabela 7 apresenta o coeficiente de determinação ajustado associado ao conjunto de dados da versão base. A amostra analisada foi 300 pontos de dados, N=300. Essa amostra foi gerada durante o período de duração da execução do teste de carga, cinco horas. Após a execução dos testes de carga na versão base, notou-se que, o R^2 ajustado apresentou uma tendência de melhora. Ou seja, tendeu a ficar mais bem calibrado, se aproximando

[6] https://www.statsmodels.org/stable/generated/statsmodels.formula.api.ols.html

Código 10 – Função responsável por recuperar coeficientes do modelo de regressão linear dos conjuntos de dados

```
def recupera_coeficientes(df, target):
  X = sm.add_constant(df.drop(possible_metrics, axis=1))
  results = sm.OLS(df[target], X).fit()
  # Termo independente
  intercept = float(results.summary().tables[1].data[1][1].strip())
  coef_independente = []
  for i in range(23):
    coef_independente.append(float(results.summary().tables[1].data[i+2][1].strip()))
  return intercept, coef_independente
```

Fonte: Autores

de um, à medida que a quantidade de amostras do teste aumentou.

Na mesma tabela, pode-se perceber, que das sete métricas observadas, três foram excluídas das análises em função da interpretação do R^2. Essa restrição configura-se como uma questão significativa, pois, torna-se inviável a avaliação de três das sete métricas de desempenho computacional sob análise. Porém, essa restrição pode ser mitigada por meio da extensão do período de teste da versão base, desde que haja uma boa relação linear entre as métricas de desempenho e a carga aplicada.

Tabela 7 – R^2 ajustado para métricas da versão base analisada

Métrica	R^2 ajustado
Utilização de CPU % (Aplicação)	0,528
Utilização de Memória % (Aplicação)	0,156
Utilização de CPU % (Banco de dados)	0,678
Utilização de Memória % (Banco de dados)	0,300
Linhas buscadas	0,741
Linhas retornadas	0,806
Transações concluídas com sucesso	0,684
Tempo de resposta	0,714

Para satisfazer a premissa de entrada de processo invariável, foi necessária a normalização dos dados do conjunto de teste da versão alvo. Essa normalização visa trazer o teste da versão base para a mesma escala de carga aplicada no teste da versão alvo. Para isso, foi utilizada a seguinte equação, adaptada de (NGUYEN et al., 2012):

$$Mt = Mb * \frac{(\sum_{i=1}^{n} \alpha_i * Ct_i) + \beta}{(\sum_{i=1}^{n} \alpha_i * Cb_i) + \beta} \tag{4.1}$$

Onde: Mt é o valor da métrica no teste na versão alvo, ou *target*; Mb é o valor da métrica no teste na versão base; α são os coeficientes das variáveis independentes

(chamadas aos endpoints), e β é o termo independente (valor constante). Após a seleção das métricas que possuem uma relação linear substancial entre o valor da métrica de desempenho e a carga aplicada, procede para o processo da normalização nos conjuntos de dados das métricas selecionadas. O processo de normalização conduz à geração de conjuntos de dados distintos para cada métrica sob análise. A função que gera os conjuntos de dados normalizados é exemplificada no Código-Fonte 11.

Código 11 – Função responsável por gerar novos conjuntos de dados normalizados

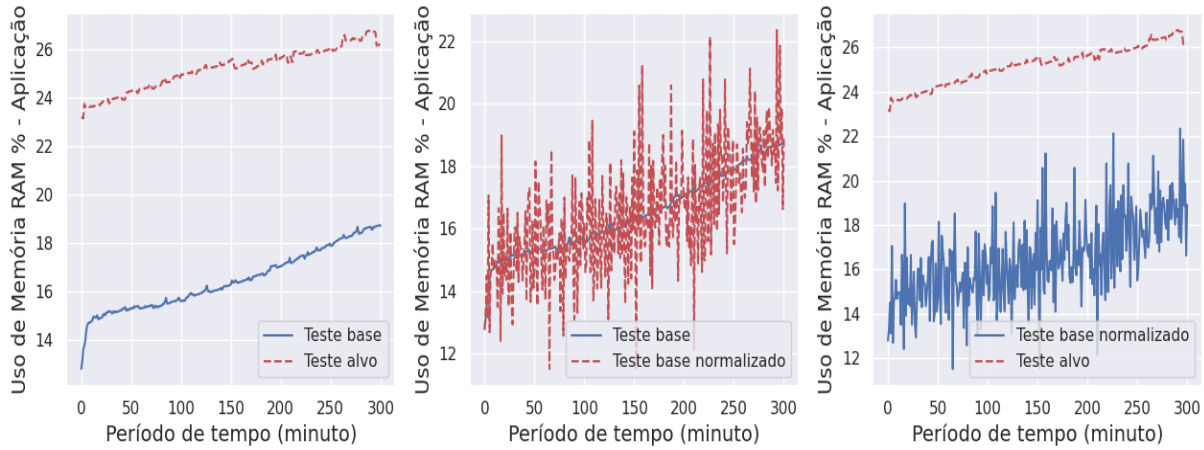
```
def recupera_soma_coeficientes(df, intercept, coef_independente, index_line):
  return sum(
  [value*df.iloc[index_line][index] for index, value in enumerate(coef_independente)]
  )+intercept

df_target_scaled = df_target.copy()
for metric in analyzable_metrics:
  intercept_base, coef_independente_base = recupera_coeficientes(df_base, metric)
  for index, Mb in enumerate(df_base[metric]):
    df_target_scaled[metric][index] = (
      Mb * (
        recupera_soma_coeficientes(df_target, intercept_base, coef_independente_base, index)/
        recupera_soma_coeficientes(df_base, intercept_base, coef_independente_base, index)
      )
    )
```

Fonte: Autores

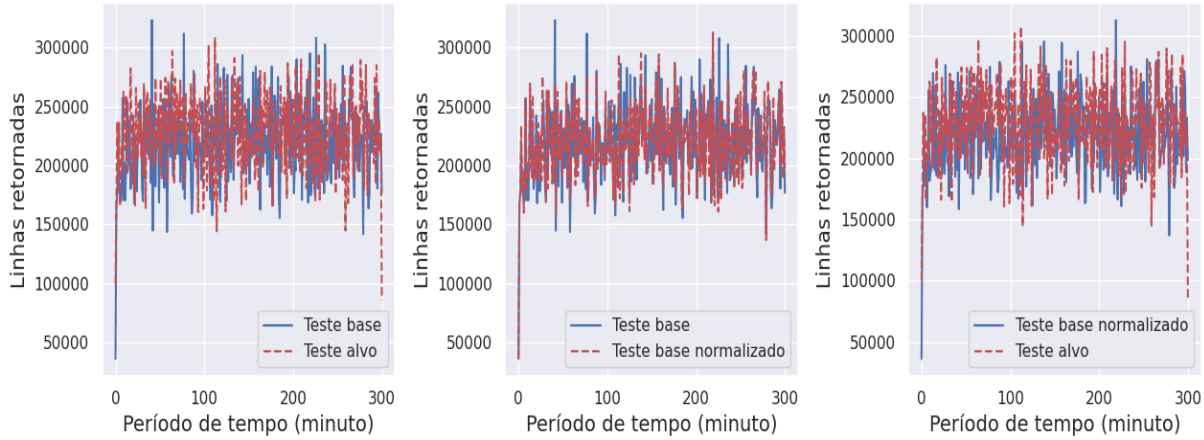
A Figura 20 apresenta a comparação de gráficos dos dados, brutos e normalizados, no teste base e teste alvo. Nessa comparação, em específico, os testes alvo e base estão na mesma versão, e a carga média aplicada apresenta uma diferença mínima de uma unidade. O teste base apresenta uma média 116 *endpoints* chamados por minuto, enquanto o teste alvo exibe uma média de 117 *endpoints* chamados por minuto. Logo, espera-se que não haja discrepâncias significativas entre o teste base e o teste base normalizado, considerando a quantidade de requisições realizadas em cada teste, bem como a substancial relação linear entre a métrica de desempenho e a carga aplicada.

A Figura 20a ilustra a métrica de uso de memória RAM da aplicação, a qual não exibe uma boa relação linear com a carga aplicada. Consequentemente, o processo de normalização revela-se ineficaz para essa circunstância específica, justificando, assim, a exclusão dessa métrica na análise por meio do gráfico de controle. O comportamento esperado na figura central (Teste base x Teste base normalizado) consiste na convergência dos gráficos, dado que ambos os testes foram conduzidos na mesma versão da aplicação. Entretanto, esta métrica não apresenta uma boa relação linear com a carga aplicada, como evidenciado pelo coeficiente de determinação ajustado no conjunto base, que atinge o valor de 0,22.

Na Figura 20b, observa-se uma métrica de desempenho que exibe uma boa relação



(a) Gráfico de normalização em métrica com má relação linear com a carga aplicada



(b) Gráfico de normalização em métrica com boa relação linear com a carga aplicada

Figura 20 – Normalização sendo feita em métricas com diferentes valores de coeficiente de determinação ajustado para testes na mesma versão (Bom_1 x Bom_2

linear com a carga aplicada. Nesse cenário de observação, é possível notar a semelhança de comportamento entre as versões de teste idênticas, bem como a aproximação dos dados do teste base normalizado em relação ao teste base, dada a similaridade na carga. Neste caso, a premissa de entrada de processo invariável pôde ser satisfeita, uma vez que o teste alvo é submetido à mesma carga do teste base. Essa constatação é confirmada pela relação de linearidade existente entre a métrica em questão e a carga aplicada, uma vez que o R^2 ajustado foi de 0,82. Pode-se perceber esse comportamento, visualmente, nos gráficos comparativos.

No Apêndice B, as Figuras 31 e 32 retratam a aplicação da técnica de normalização nos conjuntos de dados das versões alvo e base, correspondentes a duas versões distintas, sendo a versão do teste base considerada normal e a versão do teste alvo com a injeção do problema de limite de consulta. A carga média aplicada, em geral, apresenta uma diferença significativamente maior em comparação aos testes anteriormente mencionados em versões idênticas.

No teste com a versão base, a média é de 116 *endpoints* chamados por minuto, enquanto no teste com a versão alvo essa média é de 68 *endpoints* chamados por minuto. Nesse caso, espera-se que haja uma diferença no conjunto de dados quando aplicada a normalização. Portanto, o conjunto de dados normalizados dificilmente deve apresentar valores superiores ao teste na versão base, devido à comparação de carga, uma vez que o teste da versão alvo representa apenas 58% da média de carga total aplicada no teste base.

O comportamento esperado pode ser observado nos gráficos relacionados ao processo de normalização dos conjuntos de dados. No entanto, a métrica de tempo de resposta se comporta de maneira um tanto divergente, uma vez que, no teste da versão alvo, foram observados valores consideravelmente superiores em relação ao teste da versão base, mesmo com uma carga média geral menor. Isso é, ocorre porque aumenta, o tempo de resposta. Após a normalização, os resultados dos testes, na versão base e base normalizada, se aproximam da convergência. Isso indica que, provavelmente, as chamadas individuais aos endpoints que mais afetaram a métrica de tempo de resposta, se mantenham em quantidade elevada, em ambos conjuntos de dados.

Para aplicar a técnica de gráfico de controle, ainda é necessário avaliar a premissa de normalidade na saída do processo. Contudo, ao analisar os dados dispostos nos gráficos apresentados anteriormente, é possível identificar o impacto da métrica de tempo de resposta na análise do desempenho do sistema. A maioria das métricas no teste alvo, sem qualquer normalização, permanece abaixo das métricas do teste na versão base, mesmo o teste alvo sendo realizados em uma versão onde foram inseridos problemas de desempenho.

O impacto percebido na métrica de tempo de resposta reverbera nas demais métricas. À medida que a aplicação aguarda mais tempo pela resposta do banco de dados, que, por sua vez, processará mais transações, a tendência é que seus recursos se tornem mais ociosos. Ao contrário do que acontece no teste com a versão base, que utiliza intensivamente os recursos de ambas as máquinas o tempo todo. Esse fenômeno ocorre devido ao *design* dos fluxos de teste, nos quais as requisições de um usuário são síncronas, dependendo da conclusão da requisição atual para prosseguir com a próxima.

Quando o tempo de resposta de uma requisição síncrona é elevado, isso tende a tornar ociosos os recursos computacionais do banco de dados ou da aplicação, dependendo de quem está majoritariamente processando a requisição demorada. Neste caso, o banco de dados.

Portanto, o tempo de resposta desempenha um papel relevante na análise do impacto no desempenho computacional, uma vez que a compreensão do comportamento de algumas métricas depende dessa análise inicial de tempo de resposta. Tal análise é essencial para determinar se a métrica está efetivamente consumindo menos recursos devido a uma melhora de desempenho; se está mais ociosa, ou se está sujeita a uma carga de teste

relativamente menor.

A avaliação comparativa dos gráficos de dados entre as versões de teste "boa" e "com injeção de retorno de dados excessivos" assemelha-se à análise realizada entre a versão "boa" e a "com injeção de maior número de consultas". Entretanto, destaca-se que os problemas foram introduzidos nos mesmos *endpoints*, mais precisamente nos *endpoints* de listagem, pois são eles que exercem maior impacto no fluxo do sistema. A distinção reside, principalmente, na magnitude dos efeitos causados por cada inserção de problema.

Para efeitos comparativos, nota-se que a carga média aplicada apresenta uma diferença ainda mais acentuada quando analisadas as versões "boa" e "com limite de consulta aumentado". Enquanto no teste da versão base foram feitos uma média de 116 *endpoints* chamados por minuto, no teste da versão alvo, nessa instância, ocorreu uma média significativamente inferior, totalizando apenas 38 *endpoints* chamados por minuto.

No Apêndice B, a análise da Figura 34 permite observar o impacto direto da métrica de tempo de resposta na carga média aplicada. Constata-se que, conforme o tempo de resposta aumenta, a quantidade de chamadas diminui, devido à sincronicidade do fluxo de teste. Nesses testes específicos, a discrepância no tempo de resposta torna-se ainda mais evidente.

Ao considerar, também no Apêndice B, a Figura 33, verifica-se que o comportamento dos recursos computacionais apresentou semelhanças notáveis em relação ao teste com limite de consulta, influenciado pela ociosidade dos recursos computacionais durante o processamento de uma requisição.

Para testar o comportamento de normalidade das distribuições analisadas, e com isso, verificar o atendimento a segunda premissa de uso de gráficos de controle, eliminaram-se os valores atípicos. Realizou-se esse tratamento no conjunto de dados de cada métrica de desempenho a serem analisadas, visando eliminar os fatores de confusão que não interessam na comparação do gráfico de controle. Foram eliminados dados tanto no conjunto de dados da versão base, quanto na versão alvo. Trata-se de uma prática necessária, que, porém, pode ocasionar o descarte de dados relevantes.

Após a depuração dos valores atípicos, realizou-se o teste de Shapiro-Wilk para verificar a condição de normalidade das distribuições. Definiu-se o $\alpha = 0{,}05$. A hipótese nula(H_0) do teste Shapiro-Wilk assume que a distribuição da amostra testada se comporta de forma semelhante aos preceitos teóricos da distribuição normal. Portanto, caso o valor_p $> 0{,}05$, significa dizer que não há evidência suficiente para refutar a hipótese nula. Logo, a amostra testada se comporta como a distribuição normal. Além disso, elaboraram-se os gráficos de densidade, e gráficos Q-Q[7], visando auxiliar a análise da normalidade.

[7] Um gráfico quantil-quantil (Q-Q *plot*) possibilita a comparação entre a distribuição de um conjunto de dados e a de outro conjunto de dados.

Na Tabela 8, verifica-se a comparação do valor_p dos conjuntos de dados de testes, base e alvo, antes e depois do corte interquartil. A Tabela possui algumas nomenclaturas, onde: Bom_1xBom_2: Versão base estável 1 (Bom_1) em comparação com versão alvo estável 2 (Bom_2); Bom_1xDE: Versão base estável 1 (Bom_1) em comparação com versão alvo de dados excessivos (DE), e Bom_1xDE: Versão base estável 1 (Bom_1) em comparação com versão alvo com limite de consulta alterado (LC).

Tabela 8 – Comparação valor_p do Shapiro-Wilk dos conjuntos de dados antes e depois do corte

		Base		Alvo	
Métrica	Conjunto de testes	Antes	Depois	Antes	Depois
Uso de CPU aplicação	Bom_1xBom_2	-	-	-	-
	Bom_1xDE	-	-	-	-
	Bom_1xLC	-	-	-	-
Memória aplicação	Bom_1xBom_2	-	-	-	-
	Bom_1xDE	-	-	-	-
	Bom_1xLC	-	-	-	-
Uso de CPU BD	Bom_1xBom_2	$2,339^{-7}$	$0,064$	$2,069^{-9}$	$0,670$
	Bom_1xDE	$2,235^{-7}$	$0,058$	$0,050$	$0,001$
	Bom_1xLC	$2,006^{-5}$	$0,144$	$3,512^{-9}$	$0,106$
Memória BD	Bom_1xBom_2	-	-	-	-
	Bom_1xDE	-	-	-	-
	Bom_1xLC	-	-	-	-
Linhas buscadas	Bom_1xBom_2	$3,803^{-5}$	$0,513$	$0,040$	$0,313$
	Bom_1xDE	$5,845^{-9}$	$0,057$	$0,001$	$0,001$
	Bom_1xLC	$2,099^{-5}$	$0,141$	$0,226$	$0,559$
Linhas retornadas	Bom_1xBom_2	$1,029^{-5}$	$0,062$	$4,613^{-3}$	$0,207$
	Bom_1xDE	$1,825^{-13}$	$1,498^{-7}$	$0,001$	$0,003$
	Bom_1xLC	$5,393^{-5}$	$0,061$	$0,001$	$0,206$
Transações BD	Bom_1xBom_2	$5,998^{-15}$	$0,190$	$3,499^{-24}$	$1,065^{-9}$
	Bom_1xDE	$1,349^{-9}$	$0,001$	$9,629^{-25}$	$0,036$
	Bom_1xLC	$1,445^{-5}$	$0,080$	$1,227^{-22}$	$0,570$
Tempo de resposta	Bom_1xBom_2	$2,367^{-6}$	$0,003$	$8,185^{-11}$	$8,112^{-6}$
	Bom_1xDE	$6,669^{-5}$	$0,037$	$4,317^{-12}$	$3,678^{-7}$
	Bom_1xLC	$3,262^{-7}$	$5,443^{-5}$	$2,706^{-17}$	$0,005$

Fonte: Autores

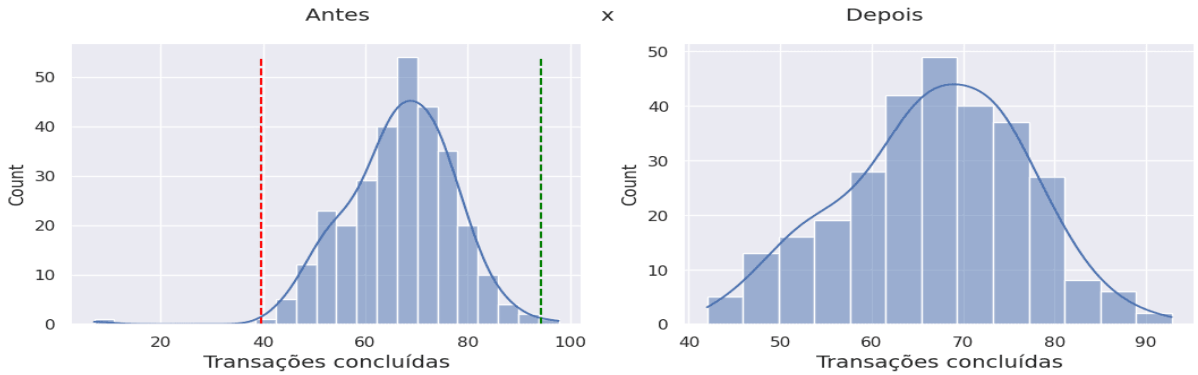
A Tabela 8 evidencia que o processo de corte foi eficaz em transformar 57,14% dos conjuntos de dados, em distribuições normais, conforme corroborado pelo p-value obtido no teste do Shapiro-Wilk.

O principal risco desse método de filtragem, ao utilizar o corte interquartil, se deve, à necessidade de aplicação do corte em ambos os conjuntos de dados utilizados na construção do gráfico de controle, tanto no conjunto de dados da versão base normalizada, quanto na da versão alvo. Apesar de ser uma prática indispensável para preservar a

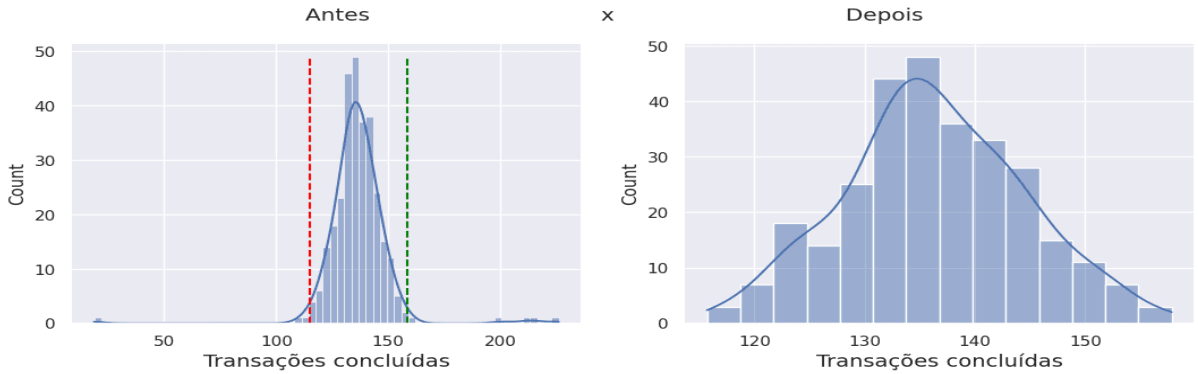
comparabilidade dos dados, pode resultar na exclusão de dados atípicos em um conjunto, mas não em outro. De qualquer modo, essa remoção não impacta significativamente a análise, pois os valores atípicos de cada conjunto estão, majoritariamente, localizados entre os primeiros e últimos pontos de dados.

Na Figura 21, é possível observar, por meio do gráfico de histograma, que o processo de corte eliminou as caudas alongadas das distribuições. Esse comportamento de caudas alongadas é explicado devido às tarefas secundárias ou aos períodos de início e término de teste, quando a carga aplicada pode ser substancialmente inferior. Além disso, cenários de sobrecarga, que indicam estresse do sistema, não são de interesse para a pesquisa, uma vez que, nesse contexto, não seria possível comparar duas versões distintas. Esse cenário também é excluído devido aos valores gerados, que tendem a ser superiores ao padrão, quando o sistema está sob estresse ou com grande acumulação de requisições.

No caso específico do corte aplicado à métrica de transações concluídas, conforme apresentado na Figura 21, ambos os conjuntos de dados demonstraram uma distribuição normal após, a remoção dos valores atípicos. Além do histograma, os gráficos Q-Q presentes na Figura 22, corroboram a distribuição mais próxima da normalidade após o processo de corte, evidenciando que os dados da amostra estão concentrados e próximos, ao centro da distribuição.

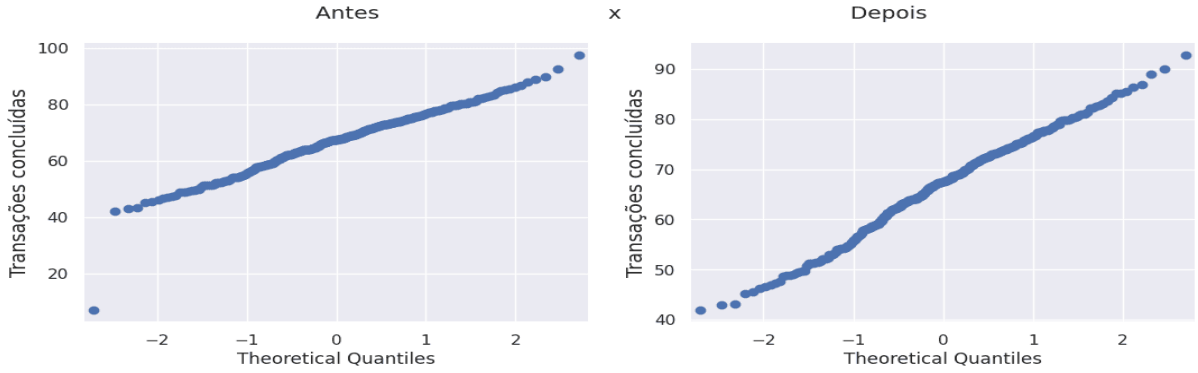


(a) Histograma transações concluídas no conjunto base normalizado

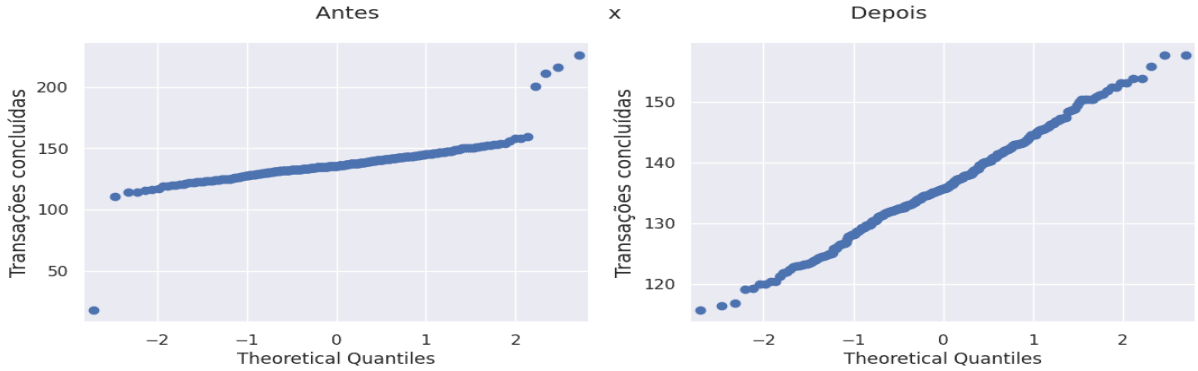


(b) Histograma transações concluídas no conjunto do alvo

Figura 21 – Histogramas da métrica de transações concluídas



(a) Gráfico Q-Q de transações concluídas no conjunto base normalizado



(b) Gráfico Q-Q de transações concluídas no conjunto do alvo

Figura 22 – Gráficos Q-Q da métrica de transações concluídas

Foram analisados, no total, quatro conjuntos de dados, consistindo em: "Bom_1": primeiro conjunto referente a versão estável da aplicação; "Bom_2": segundo conjunto referente a versão estável da aplicação; "LC": conjunto referente a versão com problema de limite de consulta inserido, e "DE": conjunto com problema de dados excessivos inseridos. O conjunto referente a versão "Bom_1" foi selecionado como o conjunto de dados base para a comparação com as demais versões.

A taxa de violação foi estabelecida com base nos percentis 5 e 95 do conjunto de dados normalizados, conforme estabelecido em (NGUYEN et al., 2012). Consequentemente, restou estabelecido que, uma taxa de violação de 10% neste estudo é esperada para qualquer métrica analisada. Para considerar uma discrepância significativa, estabelecemos o limite da taxa de violação em 20%, representando um acréscimo de 10% em relação ao valor esperado para as métricas.

Alternativamente, a taxa de violação poderia ser definida de outra maneira, realizando testes com a versão estável em várias instâncias, e a maior discrepância na taxa de violação identificada durante a comparação dos testes seria considerada como o limite da taxa de violação. Contudo, essa abordagem é dispendiosa, podendo aprimorar os resultados obtidos, mas demanda mais tempo para a execução dos testes na versão estável.

Os resultados obtidos com a aplicação da técnica do gráfico de controle são apresentados na Tabela 9. Observa-se que os testes entre as versões estáveis, (Bom_1 *e* Bom_2), não indicaram gráficos fora de controle para as três métricas passíveis de análise, o que era esperado. Por outro lado, o teste com a inserção do problema de limite de consulta revelou gráficos fora de controle para três das quatro métricas analisadas. Além disso, evidenciou uma taxa de violação significativamente alta. Os gráficos de controle gerados podem ser encontrados no Apêndice C.

Tabela 9 – Resultado da análise do gráfico de controle em diferentes versões

Teste alvo	Teste base	Nº de métricas analisadas	Nº de métricas fora de controle	Média de taxa de violação
Bom 2	Bom 1	3	0	12,68%
LC	Bom 1	4	3	54,18%
DE	Bom 1	0	-	-

Fonte: Autores

4.3 Ameaças à Validade do Estudo

Segundo Wohlin et al. (2012), a validade de um estudo denota a confiabilidade dos resultados e em que medida esses resultados são imparciais, reproduzíveis e com a menor interferência possível, em função do ponto de vista do pesquisador. Neste sentido, Yin (2009) descreve quatro aspectos a serem observados, em relação às discussões sobre a validade de um estudo de caso:

- **Validade de Construção:** Esse elemento de validade analisa em que grau as métricas coletadas para o estudo refletem a intenção do pesquisador em responder à questão de pesquisa (WOHLIN et al., 2012). Como esforço para atender esse aspecto, as métricas coletadas foram definidas utilizando a estrutura hierárquica do GQM, apresentada na Subseção 4.1.1.4. Dessa maneira, garantiu-se que as métricas estão relacionadas à questão de pesquisa do estudo.

- **Validade Interna:** Esse aspecto da validade torna-se uma preocupação quando se investigam as relações causais (WOHLIN et al., 2012). Para o presente estudo, esse aspecto de validade não é aplicável, pois trata-se de um estudo observacional cujo objetivo é descritivo e exploratório, não há inferências ou conjecturas sobre as relações de causalidade. Nesse caso, a validade interna deixa de ser uma preocupação, conforme mencionado em Yin (2009).

- **Validade Externa:** Este aspecto da validade refere-se à extensão em que os resultados podem ser generalizados e até que ponto esses resultados são de interesse

para indivíduos além do contexto específico da investigação (WOHLIN et al., 2012). Para alcançar tal objetivo, Yin (2009) sugere a replicação do estudo em vários casos. Entretanto, em virtude da natureza desta pesquisa na análise de um caso único, e da impossibilidade de correlacionar de forma efetiva os resultados com estudos relacionados, em razão das especificidades e características particulares ao contexto. As conclusões são válidas para o caso observado, não podendo ser generalizada para outros casos. Portanto, a validade externa não pôde ser alcançada.

- **Confiabilidade:** Esse aspecto aborda em que medida os dados e a análise são influenciados pelos pesquisadores. Em teoria, se outro pesquisador conduzisse o mesmo estudo mais tarde, os resultados deveriam ser equivalentes (WOHLIN et al., 2012). Neste estudo, o protocolo de execução do estudo observacional, descrito na Seção 4.1.1, a documentação de configuração do teste de carga, descrito na Subseção 4.2.1.1 e os conjuntos de dados e *scripts* de tratamento e análise, disponibilizados para consulta em um repositório público no GitHub[8], garantem a reprodutibilidade do estudo, de forma a observar outros casos, ou mesmo, reproduzi-lo em outros contextos.

4.4 Resumo do capítulo

Neste capítulo foram apresentados o planejamento, execução e análise de dados desta estudo observacional. Além disso, procedeu-se às discussões sobre os "achados". Na Subseção 4.1.1, foi apresentado o planejamento e a caracterização e protocolo do estudo. São detalhados os planejamentos essenciais, ferramentas utilizadas, métricas coletadas, fluxo de atividades e outras informações relevantes para determinar todo o processo executado no estudo observacional.

Na Subseção 4.2, foram apresentados o processo de configuração do ambiente de observação, sua automação e execução. Subsequentemente, foram descritas as etapas de tratamento de dados, a aplicação das técnicas de Floresta Aleatória e Gráficos de Controle. Por fim, foram apresentadas os resultados observados e provenientes das a análise dos dados e “achados” da pesquisa.

Todos os dados coletados, descaracterizados de quaisquer informações que levem a identificação do produto de *software* ou da empresa responsável por mantê-lo, e os algoritmos utilizados para tratá-los e analisá-los, estão disponíveis, na íntegra, para consulta em um repositório público no GitHub[8].

[8] https://github.com/TCC-EDRAHI/tcc-scripts

5 Conclusão

Neste capítulo, os resultados observados serão apresentados de maneira sintética e coesa, com o propósito de retomarmos a questão de pesquisa principal. Assim, são apresentados os objetivos alcançados e as atividades concluídas, bem como, aquelas que não foram ou puderam ser executadas. Além disso, serão discutidas as limitações identificadas durante a pesquisa, e apontamentos para trabalhos futuros.

5.1 Resultados Obtidos

A questão de pesquisa que motivou e norteou todos os processos realizados nesta pesquisa foi: **Como analisar dinamicamente a característica de qualidade de eficiência de desempenho para apoiar a tomada de decisão sobre a implantação de versões de produto de software em ambiente *web*?**

Para responder à questão de pesquisa principal, executou-se um estudo observacional, estruturado no protocolo de um estudo de caso. Neste estudo, como parte do endereçamento à resposta da questão de pequisa principal, derivou-se uma questão específica, de forma que, investiga a aplicação de métodos e técnicas propostos na literatura, para o caso observado. A questão específica foi: **Ao aplicar problemas de desempenho no código-fonte observado, foi possível identificar variação no desempenho?**

Utilizou-se a técnica GQM e estrutou-se um protocolo, definindo três tipos de métricas, variáveis dependentes e independentes, que foram observadas e analisadas, de forma quantitativa, conforme descrito na Subseção 4.1.1.4.

Nessa investigação, e a partir de evidências da literatura técnica da área, utilizaram-se as técnicas, Floresta Aleatória e Gráfico de Controle, como tratamentos, para comparar o desempenho de versões distintas de um mesmo produto de software, em uso na indústria. As conclusões a seguir, endereçam a respostas às questões de pesquisa específica, e por consequência, à principal.

As métricas de desempenho computacional, variáveis dependentes neste estudo, e a quantidade de chamadas aos *endpoints*, variáveis independentes, foram coletadas no ambiente de observação instrumentado para esta pesquisa. Nesse ambiente, foram implantadas as versões do produto analisadas no caso de observação. Após a coleta e tratamento inicial nos dados, elas foram utilizadas como parâmetros para aplicação de ambas as técnicas.

As mesmas métricas de desempenho computacional coletadas foram definidas na Subseção 4.1.1.4, sendo elas: porcentagem de uso de CPU, porcentagem de uso de memória RAM, linhas buscadas no banco de dados, linhas retornadas pelo banco de dados, número

de transações concluídas no banco de dados e tempo de resposta das requisições.

A técnica de Floresta Aleatória foi utilizada para estimar, dado um determinado conjunto de requisições, como as métricas de desempenho computacional se comportariam na versão cujos dados foram utilizados para treinar o algoritmo. A medida Cliff Delta foi utilizada para quantificar o grau de diferença entre as versões comparadas. Consideramos como variação de desempenho entre as versões, somente aquelas cujo valor assumido pelo Cliff Delta foi classificado como graus de diferença "Média" ou "Grande". Já o Erro Percentual Absoluto Médio (MAPE) foi calculado para quantificar e interpretar a diferença entre os pares comparados, nas diferentes amostras.

A técnica de Gráficos de Controle já possui mecanismos próprios para definir limites de referência, que indicam variação entre as versões. Neste estudo, foi utilizado um limite de 20% de taxa de violação, devido ao parâmetro de corte realizado no conjunto de dados, da versão base. Utilizamos os dados dispersos entre os percentis 5% e 95%, no conjunto de dados da versão base. Entretanto, para realizar a normalização dos dados entre diferentes versões, foi necessário aplicar uma técnica baseada em regressão linear, disposta na Equação 4.1.

Para utilizar o gráfico de controle, ainda realizou-se o tratamento de descarte de dados atípicos, extremos. Para tanto, testou-se a condição de normalidade das distribuições das métricas de desempenho, e aquelas que não atendiam essa condição, foram descartadas. Das cinco métricas passíveis de análise quanto a normalidade das suas distribuições, são elas: uso de *cpu* do banco de dados, linhas buscadas, linhas retornadas, transações concluídas e tempo de resposta, apenas quatro delas atestaram normalidade na distribuição após o descarte.

A métrica de tempo de resposta foi a única que não conseguiu normalidade na distribuição após o corte, considerando todos os cenários de teste, que são: comparação entre versões "boas"; comparação entre versão "boa" e com "limite de consulta aumentado", e comparação entre versão "boa" e "retorno de dados excessivos". Portanto, para analisar métricas que quantificam a percepção de desempenho e eficiência, por meio do uso de da técnica de Gráficos de Controle, é preciso observar a premissa de normalidade na saída do processo.

Na comparação entre dois testes realizados na mesma versão do produto, a técnica de Floresta Aleatória identificou variação "Trivial" ou "Pequena" para seis das oito métricas de desempenho observadas. Apenas a métrica de porcentagem de uso de memória RAM foi identificada com uma variação "Alta". Observou-se isso tanto na máquina onde a aplicação estava implantada, quanto na máquina que hospedava o banco de dados dessa aplicação.

Contudo, essa identificação de variação como "Alta", é um falso positivo. Isso é

explicado pelo fato dessa métrica ter apresentado correlação positiva fraca, com a quantidade de chamadas aos endpoints, requisições realizadas na simulação de carga. Para esse mesmo teste, a técnica de gráfico de controle foi capaz de analisar apenas três das oito métricas, são elas: uso de *cpu* da máquina do banco de dados, linhas buscadas e linhas retornadas. Nenhuma das métricas analisadas, foi apontada como fora de controle. Isso significa, que não foi identificada variação entre as versões.

Comparando a versão base com uma versão contendo o problema de limite de busca, a Floresta Aleatória identificou variação "Grande" para as seis métricas analisadas, visto que as métricas de porcentagem de uso de memória RAM foram removidas da análise. Nesse cenário de teste, foi possível analisar apenas 4 métricas, por meio do Gráfico de controle, sendo que três dessas métricas identificadas corretamente, como fora de controle, são elas: linhas buscadas, linhas retornadas e transações concluídas no banco de dados.

Na comparação entre a versão base e uma versão com o problema de dados excessivos, a técnica da Floresta Aleatória, novamente identificou diferenças "Grandes" para as seis métricas. Houve aumento na maioria dos Erros Percentuais Absolutos Médios em comparação à análise anterior, indicando que a diferença nos valores foi maior nesta comparação. No entanto, nesse cenário de teste, nenhuma métrica atendeu aos requisitos necessários para a aplicação da técnica de Gráfico de Controle.

É importante destacar que identificar a variação de desempenho em uma métrica específica, não é suficiente para concluir se esta, foi afetada de maneira positiva ou negativa. A diminuição dos valores assumidos por algumas métricas pode também ser causada pela ociosidade de um recurso, devida à sobrecarga de outro. Além disso, a métrica de tempo de resposta foi determinante no resultado geral, devido à sincronicidade do fluxo de testes utilizado. Isso aconteceu, pois, uma requisição demorada priva a execução das requisições posteriores, diminuindo de maneira significativa o uso de alguns recursos computacionais, devido à ociosidade.

Portanto, apesar das limitações, as técnicas utilizadas viabilizaram a realização de análise dinâmica, de alguns aspectos da característica de qualidade de eficiência de desempenho. Contudo, para determinar se os resultados indicam que uma versão é superior ou inferior à outra, é necessária uma análise mais abrangente, considerando a observação de mais métricas, em múltiplos casos, triangulando essas evidências quantitativas com a opinião dos desenvolvedores dos produtos. Enfim, as técnicas mostraram-se promissoras, mas necessita-se de mais estudos para generalizar essas conclusões.

5.2 Atividades concluídas

Na seção 1.6, foram definidas e detalhadas as atividades a serem realizadas neste trabalho, visando alcançar os objetivos propostos e guiar o desenvolvimento e escrita da

monografia. o Quadro 6 apresenta novamente essas atividades, juntamente com a situação atual de cada uma, e a referência para a documentação que comprova sua conclusão, quando há.

Quadro 6 – Condição das Atividades do Trabalho

Atividade	**Condição**	**Referência**
Contextualização sobre experimentação de software	Concluído	Seção 2.1
Contextualização sobre qualidade de software	Concluído	Seção 2.2
Definição do GQM	Concluído	Seção 4.1.1.4
Construção da *String* de Busca	Concluído	Seção 3.1
Seleção de artigos	Concluído	Seção 3.3
Leitura completa do material selecionado	Concluído	Seção 3.4
Definição da proposta de solução	Concluído	Seção 4.1.1
Escrita da monografia	Concluído	-
Apresentação do TCC 1	Concluído	-
Evolução do TCC a partir das considerações da banca	Concluído	-
Construção do ambiente de testes	Concluído	Seção 4.2.1
Coleta de dados sobre o desempenho do produto	Concluído	Seção 4.2.2
Análise dos resultados	Concluído	Seção 4.2.3
Incrementar a monografia	Concluído	-
Apresentação do TCC 2	Agendado	-

Fonte: Autores

5.3 Objetivos específicos

Os objetivos específicos delineados na seção 1.4 foram imprescindíveis para o desenvolvimento da pesquisa. No Quadro 7, estão documentados novamente cada um dos objetivos específicos, bem como suas condições e referências para documentação.

Devido ao prazo final para entrega do trabalho, não foi possível realizar o objetivo específico que visava coletar o ponto de vista dos desenvolvedores do projeto a respeito das técnicas aplicadas e dos resultados obtidos. Por conta disso, o objetivo específico foi marcado como não concluído.

5.4 Limitações

Foi necessário adequar e limitar o escopo do trabalho, ao limite do prazo de entrega. Isso inviabilizou que fosse realizada uma entrevista com os desenvolvedores do produto e fosse feita uma análise qualitativa. Inicialmente, com essa análise, era esperado triangular as fontes de observação. Isso aumentaria a confiança das conclusões.

Quadro 7 – Condição dos objetivos específicos

Objetivos	Condição	Referência
Definir e levantar fundamentação teórica do trabalho focado em tópicos de experimentação de software, qualidade de software e técnicas que permitam análise do desempenho de produtos de *software*	Concluído	Capítulos 2 e 3
Definir, projetar e caracterizar o estudo de caso	Concluído	Seção 4.1.1
Realizar um mapeamento das ferramentas que serão utilizadas para coleta e análise dos dados referentes ao desempenho do produto de software	Concluído	Seção 4.1.4
Coletar dados referente ao desempenho do produto de software da organização selecionada	Concluído	Seção 4.2.2
Coletar dados sobre os resultados das técnicas aplicadas a partir do ponto de vista dos desenvolvedores do projeto	Não concluído	-
Analisar os dados coletados	Concluído	Seção 4.2.3
Documentar os resultados obtidos	Concluído	-

Fonte: Autores

A técnica de Floresta Aleatória possui uma limitação na interpretação dos dados. Por não impor restrições aos dados utilizados, é possível gerar comparações para métricas que são independentes das chamadas realizadas ao sistema, o que pode levar a interpretações incorretas dos resultados. É relevante realizar uma análise cuidadosa do comportamento dos dados coletados, verificando se a variação nas requisições está efetivamente afetando a métrica observada.

Por outro lado, a técnica de Gráficos de Controle apresenta mais restrições aos dados utilizados. Em nenhum dos cenários testados, foi possível analisar todas as métricas coletadas. Os pré-requisitos para utilização da técnica de gráficos de controle para sistemas complexos, como o do estudo, pode limitar bastante a efetividade do uso da técnica. Sistemas mais simples, ou a avaliação de apenas uma camada da aplicação, pode gerar resultados melhores na utilização da técnica de gráficos de controle, devido ao provável melhor ajuste dos dados em uma regressão linear simples.

5.5 Trabalhos Futuros

Devido à limitação do escopo do trabalho por consequência do prazo de entrega, não foi possível comparar versões do produto com alterações desenvolvidas pela própria

equipe responsável pela manutenção da aplicação e posteriormente a validação das técnicas pelos próprios desenvolvedores.

Considerando que o algoritmo de Floresta Aleatória é empregado para a normalização dos dados nos testes, mas não inclui nativamente um método de comparação, enquanto o algoritmo de Gráficos de Controle é utilizado para comparação, mas não possui nativamente uma técnica para a normalização dos dados nos testes, uma hipótese levantada durante a conclusão do estudo foi a viabilidade de empregar o algoritmo de Floresta Aleatória para realizar a escala dos testes e o algoritmo de Gráficos de Controle para efetuar a análise. A Floresta Aleatória demonstrou valores de R^2 ajustado significativamente altos em comparação com os apresentados pelo algoritmo de regressão linear utilizado para a normalização dos Gráficos de Controle, sugerindo que a combinação dessas técnicas poderia resultar em análises mais aprimoradas.

Referências

ALSHEIKHY, A.; HAN, S.; AMMAR, R. Hierarchical performance modeling of embedded systems. In: *2015 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.: s.n.], 2015. p. 936–942. Citado 2 vezes nas páginas 29 e 61.

Apache Software Foundation. *Apache JMeterTM*. 2022. Disponível em: <https://jmeter.apache.org/>. Acesso em: Jun. 2023. Citado 2 vezes nas páginas 77 e 78.

ARIF, M. M.; SHANG, W.; SHIHAB, E. Empirical study on the discrepancy between performance testing results from virtual and physical environments. *Empirical Software Engineering*, Springer Science and Business Media LLC, v. 23, n. 3, p. 1490–1518, out. 2017. Disponível em: <https://doi.org/10.1007/s10664-017-9553-x>. Citado na página 61.

AVRITZER, A.; WEYUKER, E. The automatic generation of load test suites and the assessment of the resulting software. *IEEE Transactions on Software Engineering*, v. 21, n. 9, p. 705–716, 1995. Citado na página 52.

BARNA, C. et al. Runtime performance management for cloud applications with adaptive controllers. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ICPE '18), p. 176–183. ISBN 9781450350952. Disponível em: <https://doi.org/10.1145/3184407.3184438>. Citado na página 61.

BASILI, V. R.; CALDIERA, G.; ROMBACH, D. H. The goal question metric approach. In: _____. [S.l.]: John Wiley & Sons, 1994. I. Citado 3 vezes nas páginas 31, 32 e 34.

BOEHM, B. W. *Characteristics of Software Quality*. North-Holland, 1978. (TRW Series of Software Technology). Disponível em: <https://books.google.com.br/books?id=jLFXswEACAAJ>. Citado na página 43.

BOUQUIN, D. Github. *Journal of the Medical Library Association*, v. 103, n. 3, p. 166+, July 2015. Citado na página 76.

BREIMAN, L. Random forests. *Machine Learning*, Springer Science and Business Media LLC, v. 45, n. 1, p. 5–32, 2001. Disponível em: <https://doi.org/10.1023/a:1010933404324>. Citado 2 vezes nas páginas 46 e 65.

Capes. *Periódico Capes*. 2022. Disponível em: <https://www.periodicos.capes.gov.br/>. Acesso em: May. 2023. Citado na página 57.

CHEN, T.-H. P. Detecting performance anti-patterns for applications developed using object-relational mapping. In: . [S.l.: s.n.], 2014. Citado 2 vezes nas páginas 65 e 72.

CHIN, W. W. The partial least squares approach to structural equation modeling. *Modern methods for business research*, Mahwah, NJ, v. 295, n. 2, p. 295–336, 1998. Citado na página 96.

CUTLER, A.; CUTLER, D. R.; STEVENS, J. R. Random forests. In: *Ensemble Machine Learning*. Springer New York, 2012. p. 157–175. Disponível em: <https://doi.org/10.1007/978-1-4419-9326-7_5>. Citado na página 47.

DORGHAM, D. M. T.; BELAL, N. A.; ABDELMOEZ, W. Early performance prediction in bioinformatics systems using palladio component modeling. *Applied Sciences (Switzerland)*, v. 11, n. 12, 2021. Cited by: 0; All Open Access, Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85108317957&doi=10.3390%2fapp11125426&partnerID=40&md5=22dda4bcc80a0f31624ccdd649ecb463>. Citado na página 61.

Elsevier. *Banco de dados de resumos e citações organizado por especialistas*. 2022. Disponível em: <https://www.periodicos.capes.gov.br/>. Acesso em: May. 2023. Citado 2 vezes nas páginas 57 e 59.

FAUZIA, H.; LAKSMIWATI, I. H.; HENDRADJAYA, B. A quality model for mobile thick client that utilizes web api. In: . [s.n.], 2014. Cited by: 3. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84946691536&doi=10.1109%2fICODSE.2014.7062681&partnerID=40&md5=323015abadfb19273dc20869fec75018>. Citado na página 61.

FLORES-GONZáLEZ, M.; TREJOS-ZELAYA, I. Cloud function performance: A component modeling approach. In: *2020 XLVI Latin American Computing Conference (CLEI)*. [S.l.: s.n.], 2020. p. 193–202. Citado na página 61.

GAO, R. et al. A framework to evaluate the effectiveness of different load testing analysis techniques. In: *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. [S.l.: s.n.], 2016. p. 22–32. Citado 7 vezes nas páginas 30, 45, 49, 62, 65, 68 e 74.

Git. *Git*. 2023. Disponível em: <https://git-scm.com/>. Acesso em: Jun. 2023. Citado na página 75.

Github. *About github*. 2023. Disponível em: <https://github.com/about>. Acesso em: Jun. 2023. Citado na página 76.

ISO 9000. *Quality management systems — Fundamentals and vocabulary*. [S.l.], 2015. Disponível em: <https://www.iso.org/standard/45481.html>. Citado na página 29.

ISO/IEC 25000. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE*. [S.l.], 2014. Disponível em: <https://www.iso.org/standard/64764.html>. Citado na página 42.

ISO/IEC 25010. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. [S.l.], 2011. Disponível em: <https://www.iso.org/standard/35733.html>. Citado 3 vezes nas páginas 29, 43 e 45.

ISO/IEC 9126-1. *Software engineering — Product quality — Part 1: Quality model*. [S.l.], 2001. Disponível em: <https://www.iso.org/standard/15054.html>. Citado na página 42.

ISO/IEC/IEEE 15939. *Systems and software engineering — Measurement process*. [S.l.], 2017. Disponível em: <https://www.iso.org/standard/71197.html>. Citado na página 29.

JIANG, Z. M. et al. Automated performance analysis of load tests. In: *2009 IEEE International Conference on Software Maintenance*. IEEE, 2009. Disponível em: <https://doi.org/10.1109/icsm.2009.5306331>. Citado 4 vezes nas páginas 29, 30, 45 e 65.

Jupyter. *Project Jupyter Documentation*. 2023. Disponível em: <https://docs.jupyter.org/en/latest/>. Acesso em: Jun. 2023. Citado na página 78.

KHAZAEI, H. et al. Efficiency analysis of provisioning microservices. In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.: s.n.], 2016. p. 261–268. Citado na página 61.

KUNZ, J.; HEGER, C.; HEINRICH, R. A generic platform for transforming monitoring data into performance models. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. New York, NY, USA: Association for Computing Machinery, 2017. (ICPE '17 Companion), p. 151–156. ISBN 9781450348997. Disponível em: <https://doi.org/10.1145/3053600.3053635>. Citado na página 61.

LETHBRIDGE, T. C.; SIM, S. E.; SINGER, J. Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, Springer Science and Business Media LLC, v. 10, n. 3, p. 311–341, jul. 2005. Disponível em: <https://doi.org/10.1007/s10664-005-1290-x>. Citado na página 41.

LIAO, L. et al. Using black-box performance models to detect performance regressions under varying workloads: an empirical study. *Empirical Software Engineering*, Springer Science and Business Media LLC, v. 25, n. 5, p. 4130–4160, ago. 2020. Disponível em: <https://doi.org/10.1007/s10664-020-09866-z>. Citado 4 vezes nas páginas 30, 44, 61 e 65.

LóPEZ, L. et al. Quality measurement in agile and rapid software development: A systematic mapping. *Journal of Systems and Software*, v. 186, p. 111187, 2022. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121221002661>. Citado 5 vezes nas páginas 33, 58, 68, 72 e 76.

MAIA, A. *Econometria: Conceitos e Aplicações*. Saint Paul Editora, 2019. ISBN 9788580041385. Disponível em: <https://books.google.com.br/books?id=qTKDDwAAQBAJ>. Citado na página 47.

MARTíNEZ-FERNáNDEZ, S. et al. A quality model for actionable analytics in rapid software development. In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. [S.l.: s.n.], 2018. p. 370–377. Citado na página 33.

MARTíNEZ-FERNáNDEZ, S. et al. Continuously assessing and improving software quality with software analytics tools: A case study. *IEEE Access*, v. 7, p. 68219–68239, 2019. Citado 3 vezes nas páginas 33, 43 e 44.

MCCALL, J.; RICHARDS, P. A.; WALTERS, G. F. Factors in software quality: concept and definitions of software quality. In: . [S.l.: s.n.], 1977. Citado na página 43.

MORDAL-MANET, K. et al. The squale model — a practice-based industrial quality model. In: *2009 IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2009. p. 531–534. Citado 2 vezes nas páginas 43 e 44.

NGUYEN, T. H. et al. Automated detection of performance regressions using statistical process control techniques. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2012. (ICPE '12), p. 299–310. ISBN 9781450312028. Disponível em: <https://doi.org/10.1145/2188286.2188344>. Citado 14 vezes nas páginas 30, 49, 50, 51, 52, 53, 54, 55, 65, 72, 73, 81, 97 e 104.

NUH, J. A. et al. Performance evaluation metrics for multi-objective evolutionary algorithms in search-based software engineering: Systematic literature review. *Applied Sciences (Switzerland)*, v. 11, n. 7, 2021. Cited by: 4; All Open Access, Gold Open Access. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85103552991&doi=10.3390%2fapp11073117&partnerID=40&md5=c2baaf374612ba5acbd852f76199471e>. Citado na página 61.

OUHBI, S. et al. Evaluating Software Product Quality: A Systematic Mapping Study. In: *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*. [S.l.: s.n.], 2014. p. 141–151. Citado na página 43.

PAI, M. et al. Systematic reviews and meta-analyses: an illustrated, step-by-step guide. *The National medical journal of India*, v. 17(2), p. 86–95, 2004. Citado na página 57.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 78.

PEREZ, J. F.; WANG, W.; CASALE, G. Towards a devops approach for software quality engineering. In: *Proceedings of the 2015 Workshop on Challenges in Performance Methods for Software Development*. New York, NY, USA: Association for Computing Machinery, 2015. (WOSP '15), p. 5–10. ISBN 9781450333405. Disponível em: <https://doi.org/10.1145/2693561.2693564>. Citado na página 61.

SHANG, W. et al. Automated detection of performance regressions using regression models on clustered performance counters. In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2015. (ICPE '15), p. 15–26. ISBN 9781450332484. Disponível em: <https://doi.org/10.1145/2668930.2688052>. Citado na página 69.

WAGNER, S. et al. Operationalised product quality models and assessment: The quamoco approach. *CoRR*, abs/1611.09230, 2016. Disponível em: <http://arxiv.org/abs/1611.09230>. Citado 2 vezes nas páginas 43 e 44.

WERT, A.; SCHULZ, H.; HEGER, C. Aim: Adaptable instrumentation and monitoring for automated software performance analysis. In: *2015 IEEE/ACM 10th International Workshop on Automation of Software Test*. [S.l.: s.n.], 2015. p. 38–42. Citado na página 61.

WOHLIN, C. et al. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, 2012. Disponível em: <https://doi.org/10.1007/978-3-642-29044-2>. Citado 9 vezes nas páginas 39, 40, 42, 64, 67, 68, 70, 105 e 106.

YAO, K. et al. Log4perf: Suggesting logging locations for web-based systems' performance monitoring. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: Association for Computing Machinery, 2018. (ICPE '18), p. 127–138. ISBN 9781450350952. Disponível em: <https://doi.org/10.1145/3184407.3184416>. Citado 2 vezes nas páginas 61 e 69.

YIN, R. *Case Study Research: Design and Methods*. [S.l.]: SAGE Publications, 2009. (Applied Social Research Methods). ISBN 9781412960991. Citado 4 vezes nas páginas 67, 70, 105 e 106.

Zabbix. *Funcionalidades do Zabbix*. 2023. Disponível em: <https://www.zabbix.com/documentation/current/pt/manual/introduction/features>. Acesso em: Jun. 2023. Citado na página 76.

Zabbix. *O que é o Zabbix*. 2023. Disponível em: <https://www.zabbix.com/documentation/current/pt/manual/introduction/about>. Acesso em: Jun. 2023. Citado na página 76.

Apêndices

APÊNDICE A – Evolução da *string* de busca

Este apêndice tem como propósito esclarecer a progressão da *string* de busca adotada. A Tabela 10 contém três colunas relacionadas à seguinte informação: a *string* de busca empregada, a data na qual a pesquisa foi realizada utilizando a referida *string* e a quantidade de artigos encontrados após a aplicação da *string* de busca na data específica. Para aprimorar os resultados obtidos, a maioria dos termos foi expressa por meio de palavras compostas.

Tabela 10 – Evolução string de busca

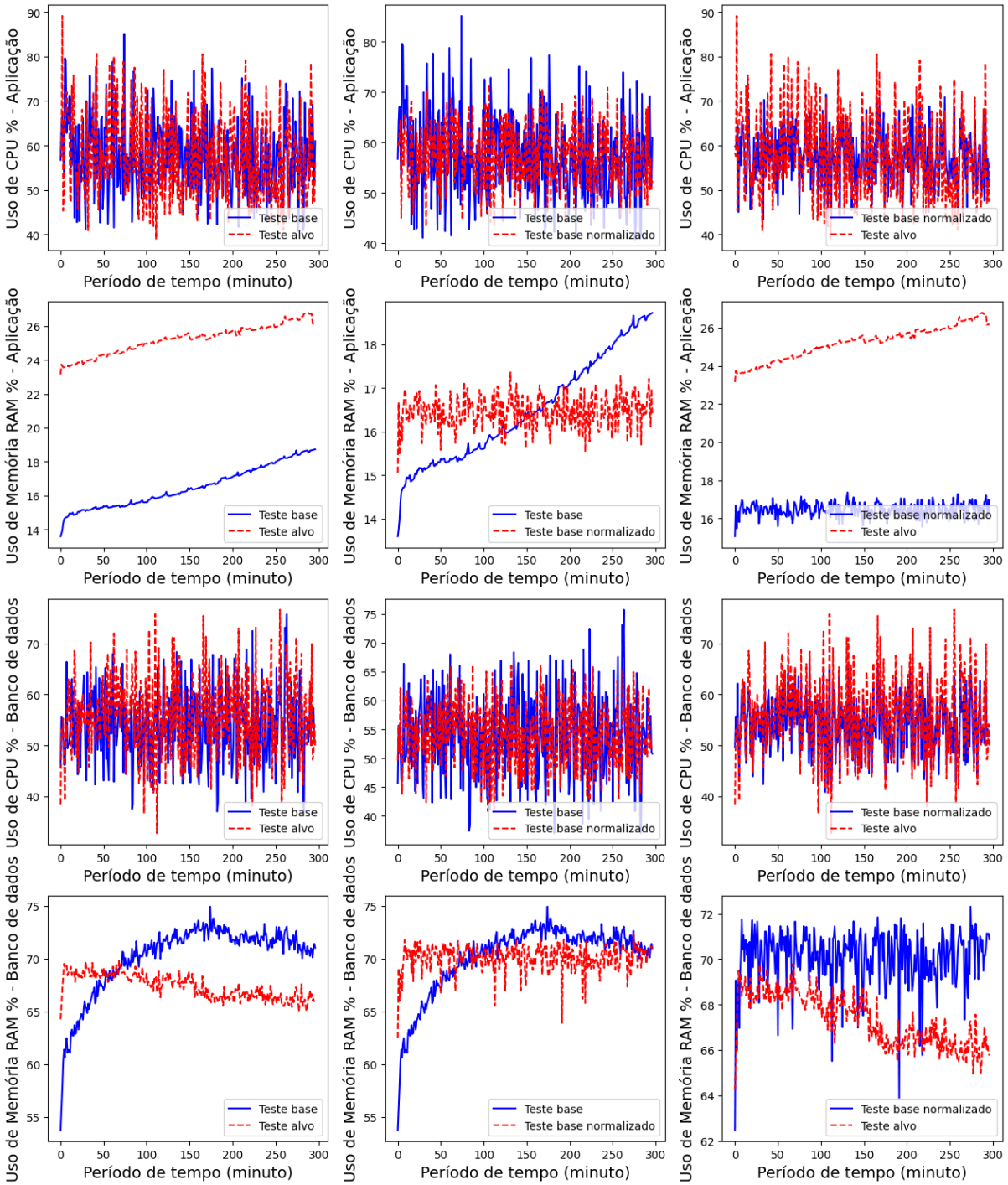
String de Busca	Data da string	Resultados
("Software Experimentation" OR (Embedded AND (Software OR Application OR System))) AND (((Performance OR CPU OR Memory) AND (Efficiency OR Quality OR Requirement OR Parameterization OR Evaluate))) AND (Quality AND (Measurement OR Indicator OR Metric* OR Measure OR Calculation OR Model*))*	10/05/2023	2274
(Software OR "Embedded Software"OR "Embedded Application"OR "Embedded System"OR "Software System" OR "Software Application" OR "real-time operating system") AND ("Performance Efficiency"OR "Performance Parameterization"OR "Performance Analysis"OR "Performance management"OR "CPU Efficiency"OR "CPU Parameterization"OR "CPU Analysis"OR "CPU Utilization"OR "CPU management"OR "Memory Efficiency"OR "Memory Parameterization"OR "Memory Analysis"OR "Memory Utilization"OR "Memory management"OR "Resources Efficiency"OR "Resources Parameterization"OR "Resources Analysis"OR "Resources Utilization"OR "Resources management"OR "runtime resources Efficiency"OR "runtime resources Parameterization"OR "runtime resources Analysis"OR "runtime resources management") AND ("Quality Measurement"OR "Quality Indicator"OR "Quality Metric"OR "Quality Measure"OR "Quality Calculation"OR "Quality Model")	11/05/2023	75
("Embedded Software"OR "Embedded Application"OR "Embedded System"OR "Software System" OR "Software Application" OR "real-time operating system"OR "Software Engineering" OR "Software Quality") AND ("Performance Efficiency"OR "Performance Parameterization"OR "Performance Analysis"OR "Performance management"OR "Performance Evaluation" OR "CPU Efficiency"OR "CPU Parameterization"OR "CPU Analysis"OR "CPU Utilization"OR "CPU management"OR "Memory Efficiency"OR "Memory Parameterization"OR "Memory Analysis"OR "Memory Utilization"OR "Memory management"OR "Resources Efficiency"OR "Resources Parameterization"OR "Resources Analysis"OR "Resources Utilization"OR "Resources management"OR "runtime resources Efficiency"OR "runtime resources Parameterization"OR "runtime resources Analysis"OR "runtime resources management"OR "runtime resources Evaluation" OR "Software Performance") AND ("Quality Measurement"OR "Quality Indicator"OR "Quality Metric"OR "Quality Measure"OR "Quality Calculation"OR "Quality Model"OR "Efficiency Metric" OR "Efficiency model" OR "Performance Indicator" OR "Performance Model" OR "Performance Metric")	12/05/2023	288

Fonte: Autores

APÊNDICE B – Gráficos de normalização dos conjuntos de dados

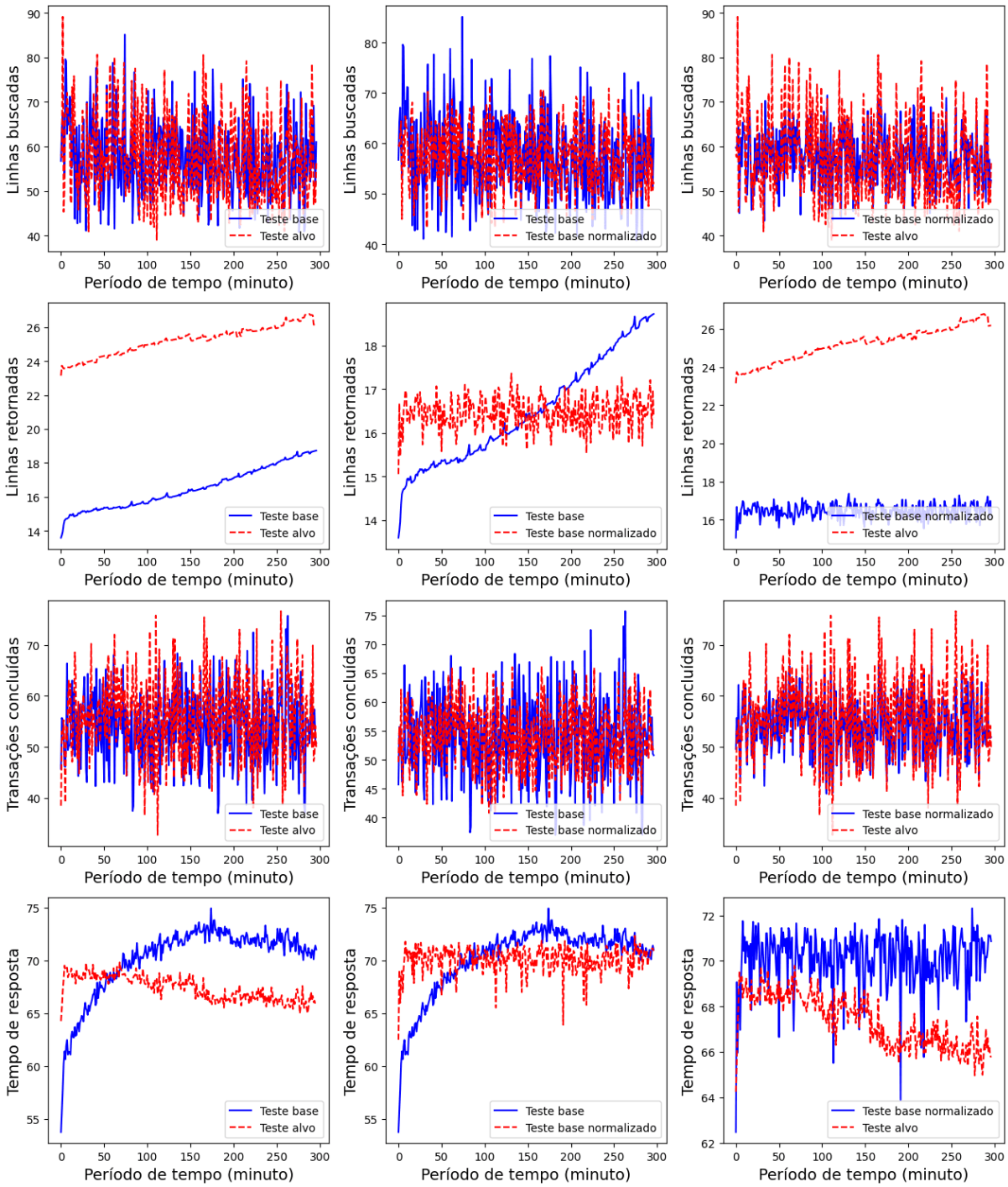
Este apêndice visa apresentar os gráficos resultantes da normalização dos dados por meio de diferentes técnicas, como o gráfico de controle e a floresta aleatória. No total, um gráfico foi gerado para cada comparação conduzida. Os testes foram realizados utilizando três versões distintas da aplicação. A primeira versão é considerada estável ou boa, e para esta versão, foram conduzidos dois testes distintos, a fim de possibilitar a comparação entre eles. As segunda e terceira versões referem-se, respectivamente, às versões com a inserção de um limite de consulta maior e a introdução de dados excessivos.

Figura 23 – Gráficos de normalização versão Boa x Boa para métricas de uso de *cpu* e memória da aplicação e do banco de dados - Floresta Aleatória



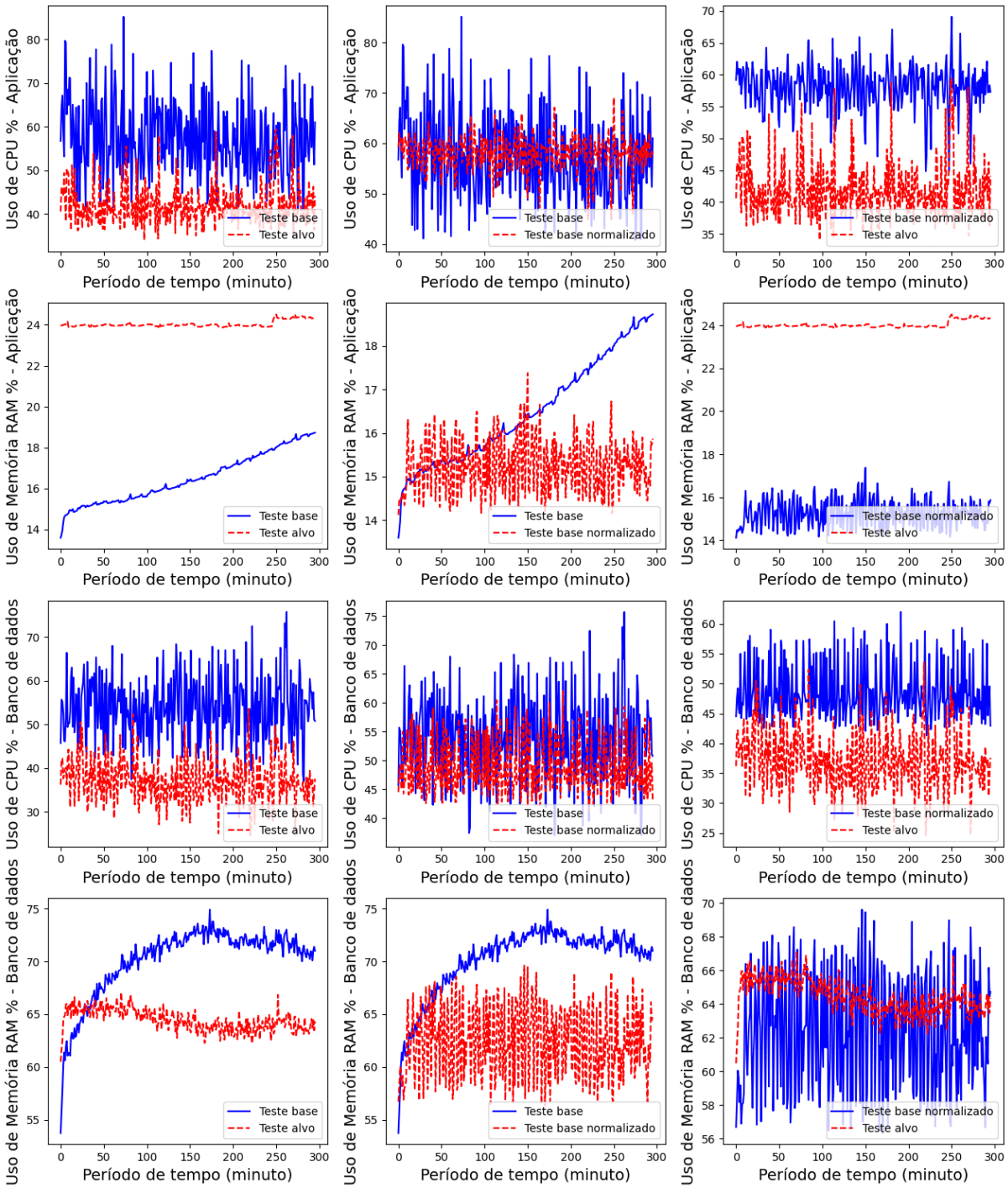
Fonte: Autores

Figura 24 – Gráficos de normalização versão Boa x Boa para métricas de linhas recebidas, linhas retornadas, transações concluídas e tempo de resposta - Floresta Aleatória



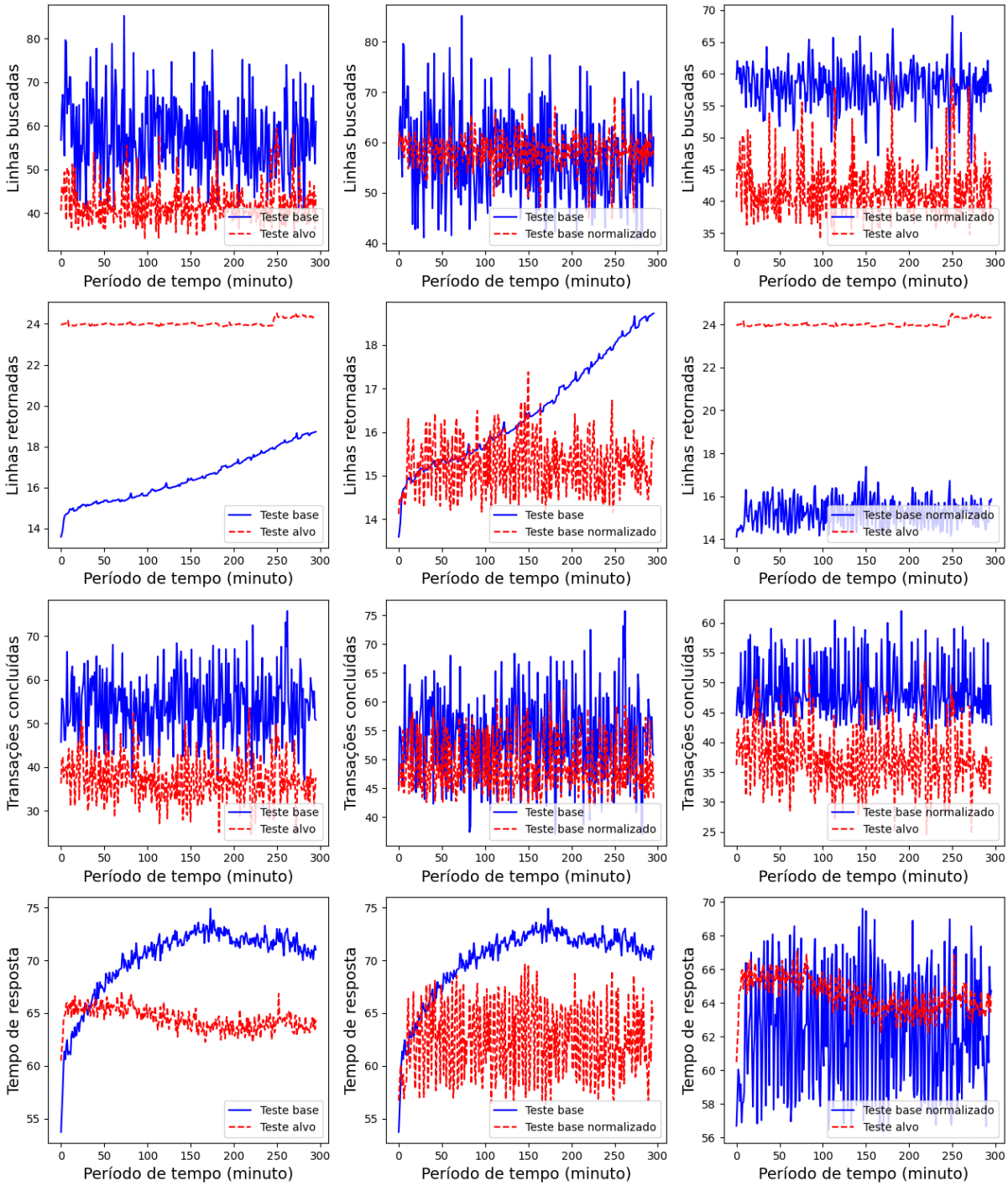
Fonte: Autores

Figura 25 – Gráficos de normalização versão Boa x LC para métricas de uso de *cpu* e memória da aplicação e do banco de dados - Floresta Aleatória



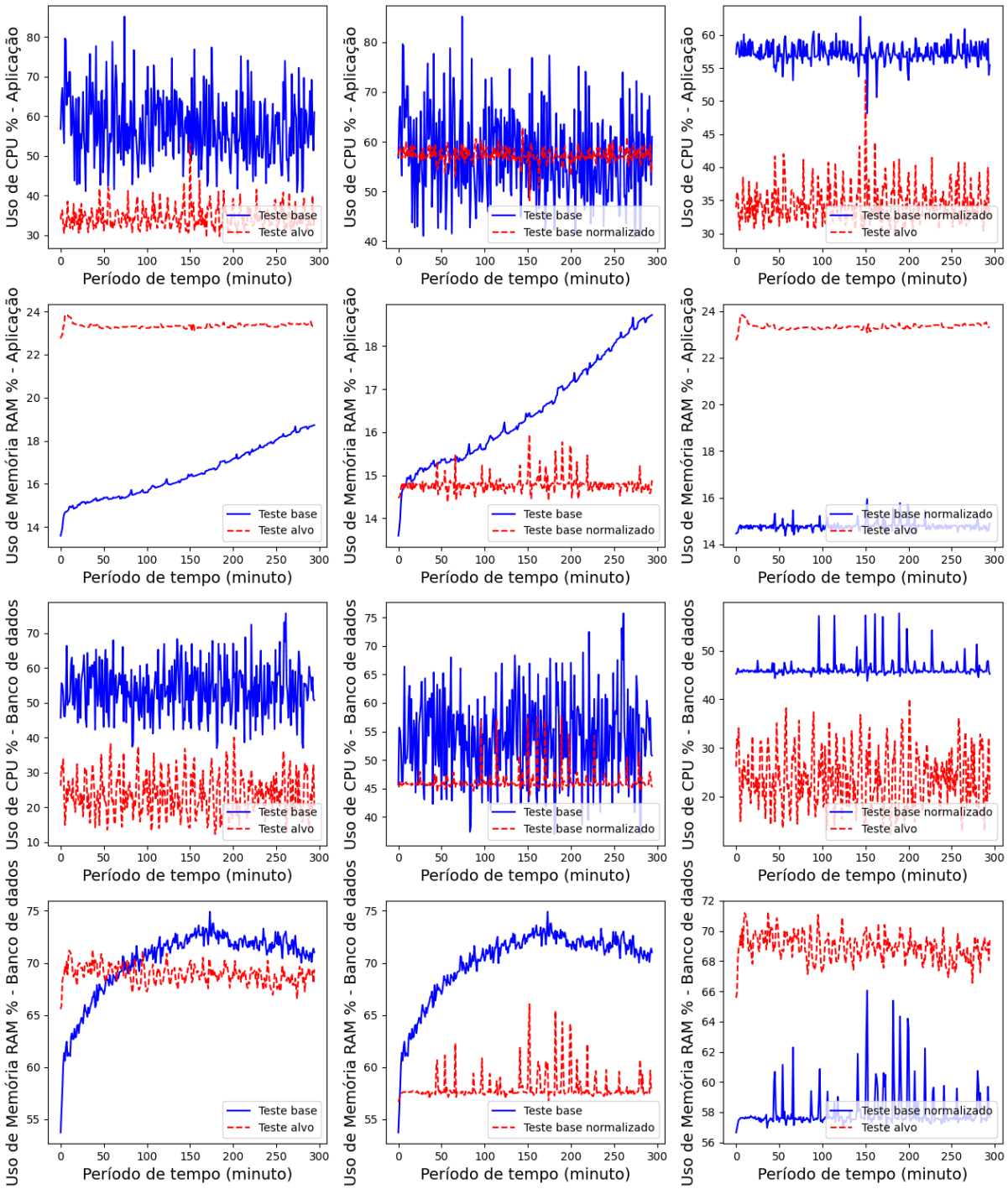
Fonte: Autores

Figura 26 – Gráficos de normalização versão Boa x LC para métricas de linhas recebidas, linhas retornadas, transações concluídas e tempo de resposta - Floresta Aleatória



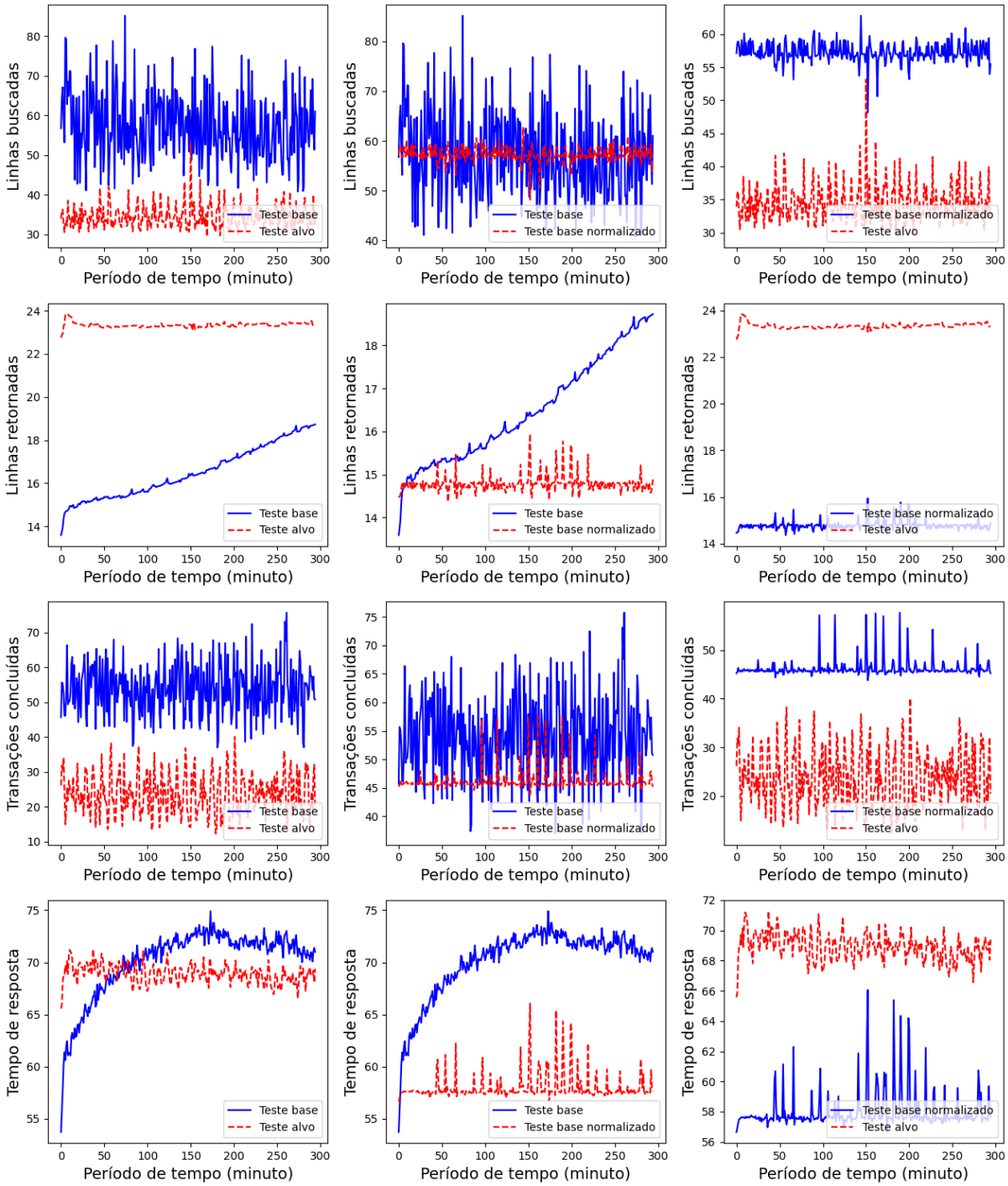
Fonte: Autores

Figura 27 – Gráficos de normalização versão Boa x DE para métricas de uso de *cpu* e memória da aplicação e do banco de dados - Floresta Aleatória



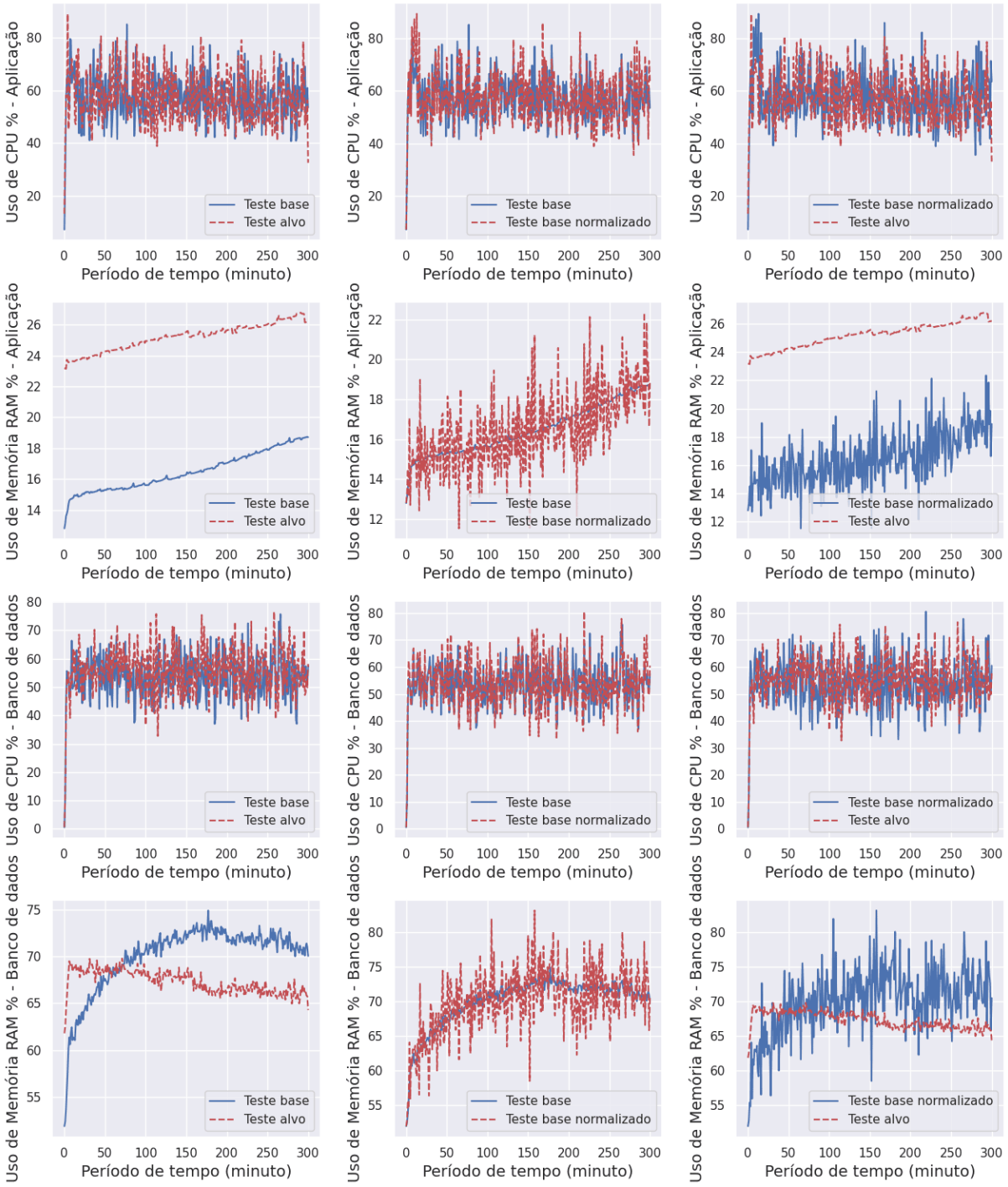
Fonte: Autores

Figura 28 – Gráficos de normalização versão Boa x DE para métricas de linhas recebidas, linhas retornadas, transações concluídas e tempo de resposta - Floresta Aleatória



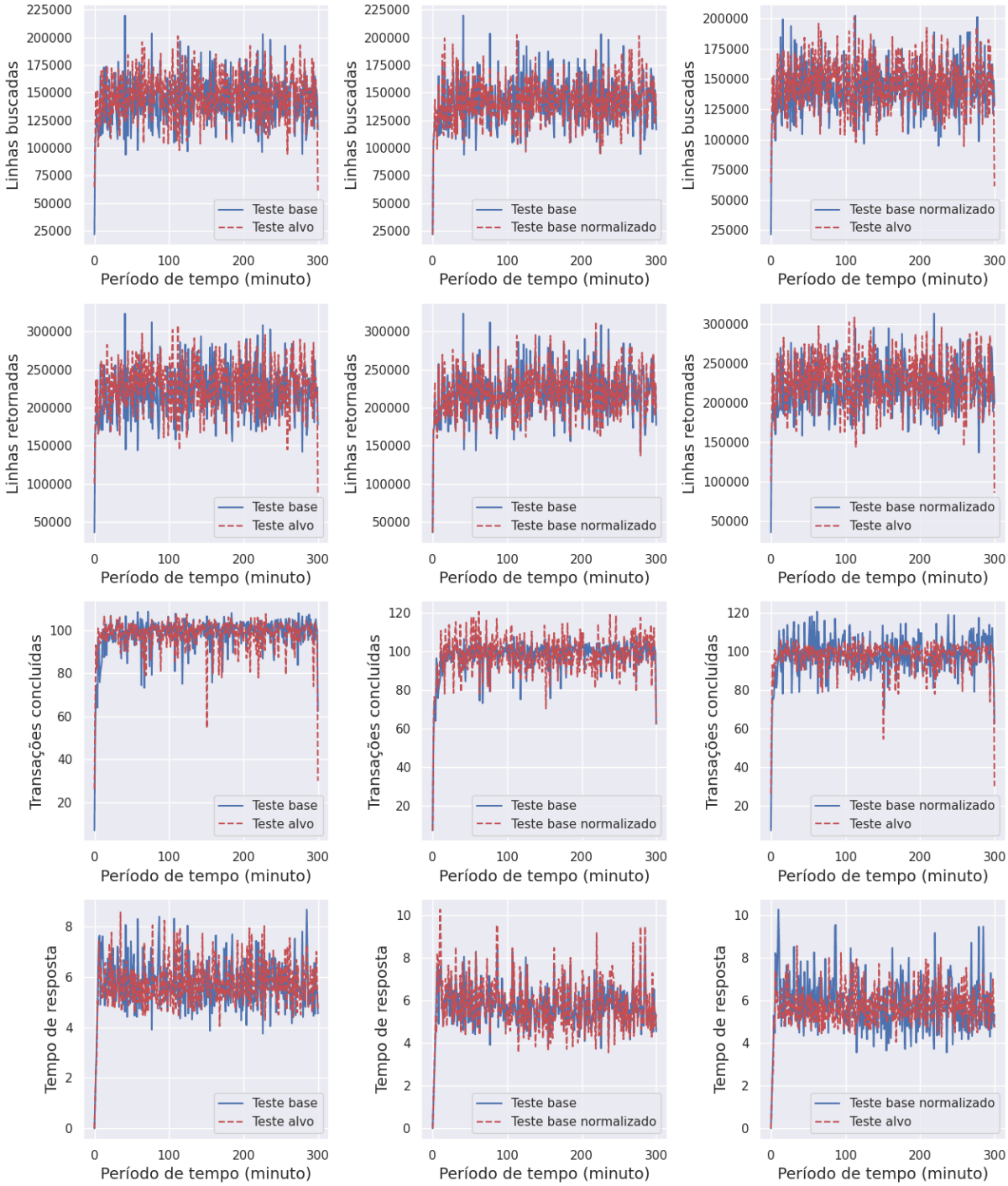
Fonte: Autores

Figura 29 – Gráficos de normalização versão Boa x Boa para métricas de uso de *cpu* e memória da aplicação e do banco de dados - Gráfico de controle



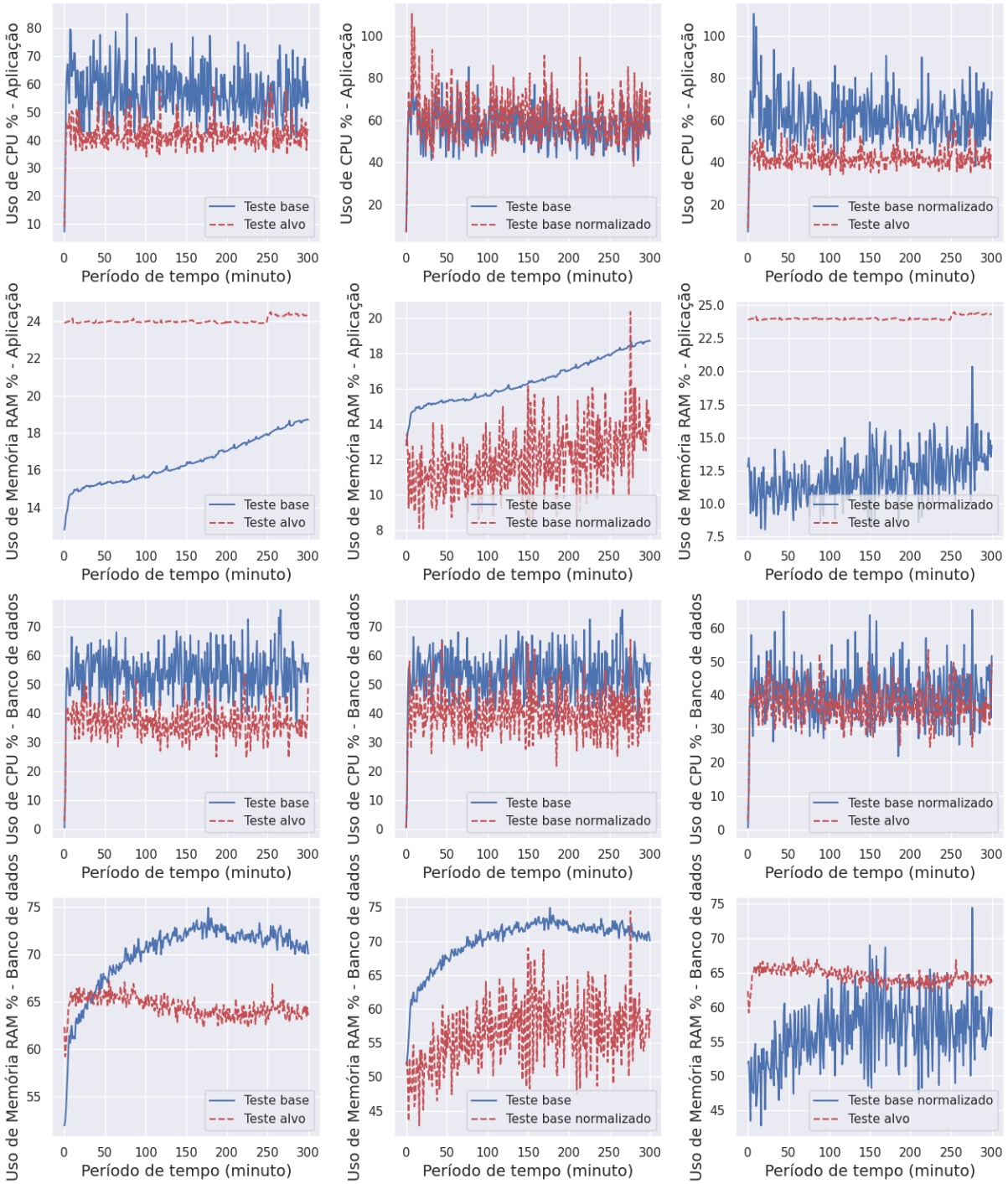
Fonte: Autores

Figura 30 – Gráficos de normalização versão Boa x Boa para métricas de linhas recebidas, linhas retornadas, transações concluídas e tempo de resposta - Gráfico de controle



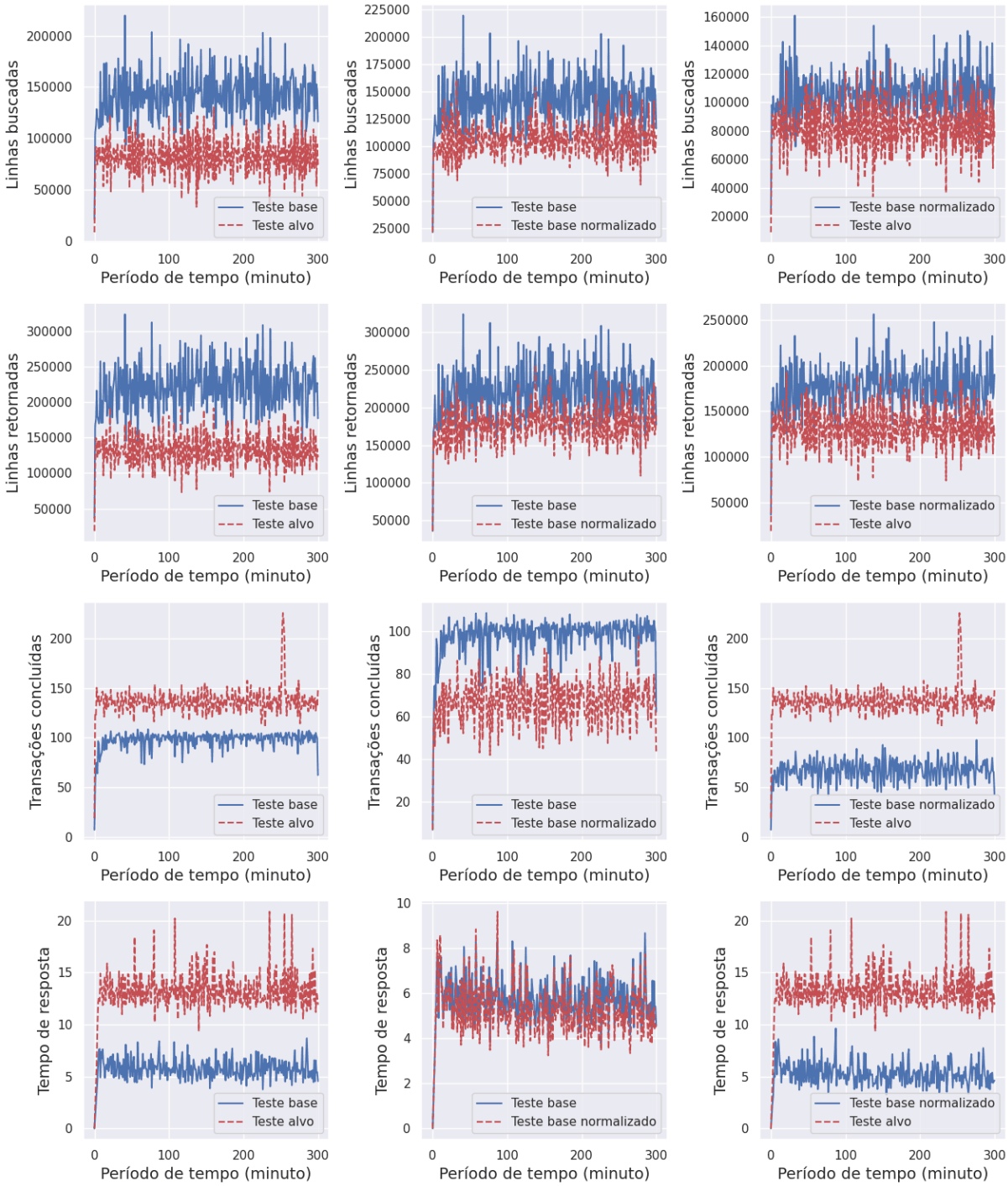
Fonte: Autores

Figura 31 – Gráficos de normalização versão Boa x LC para métricas de uso de *cpu* e memória da aplicação e do banco de dados - Gráfico de controle



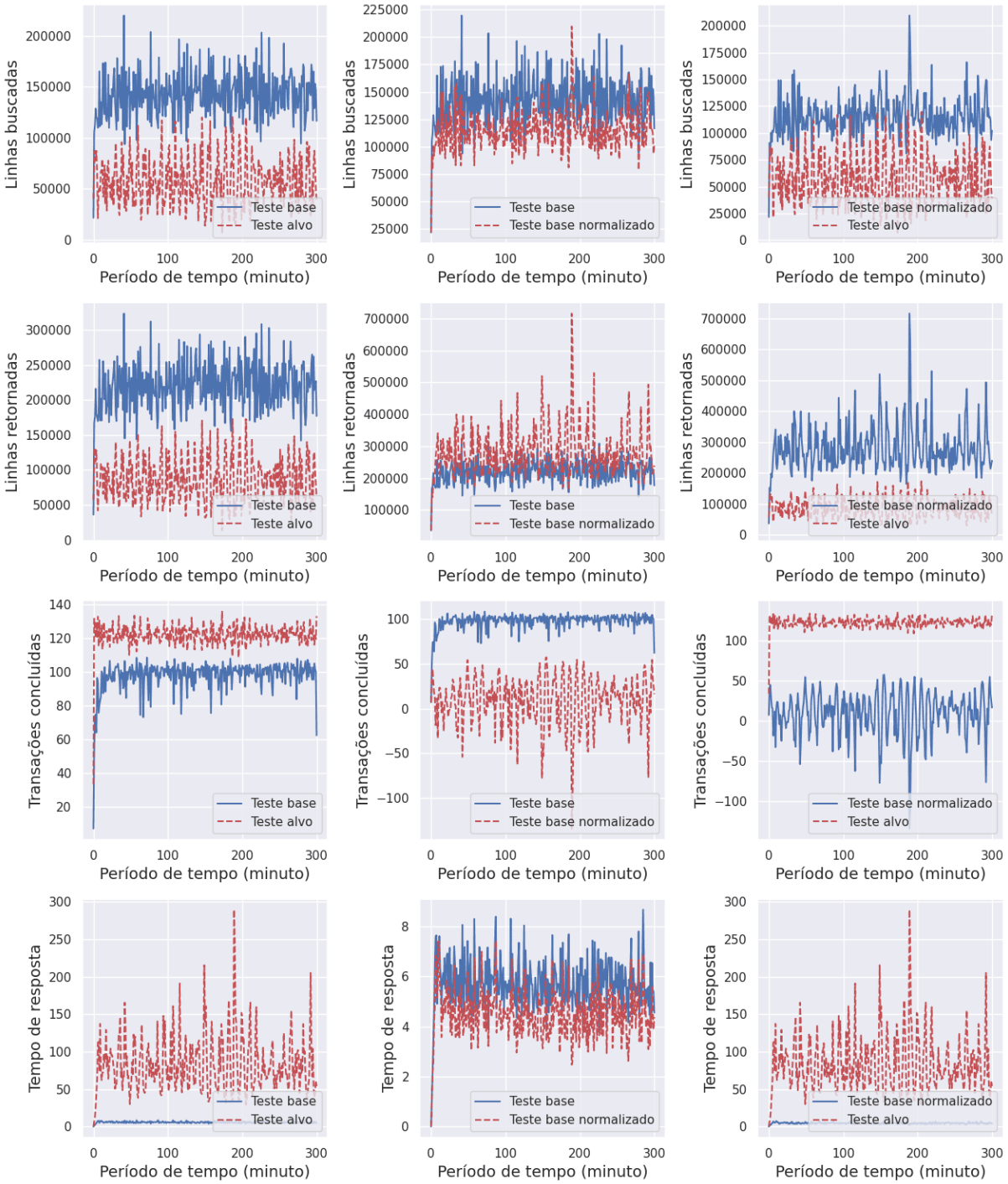
Fonte: Autores

Figura 32 – Gráficos de normalização versão Boa x LC para métricas de linhas recebidas, linhas retornadas, transações concluídas e tempo de resposta - Gráfico de controle



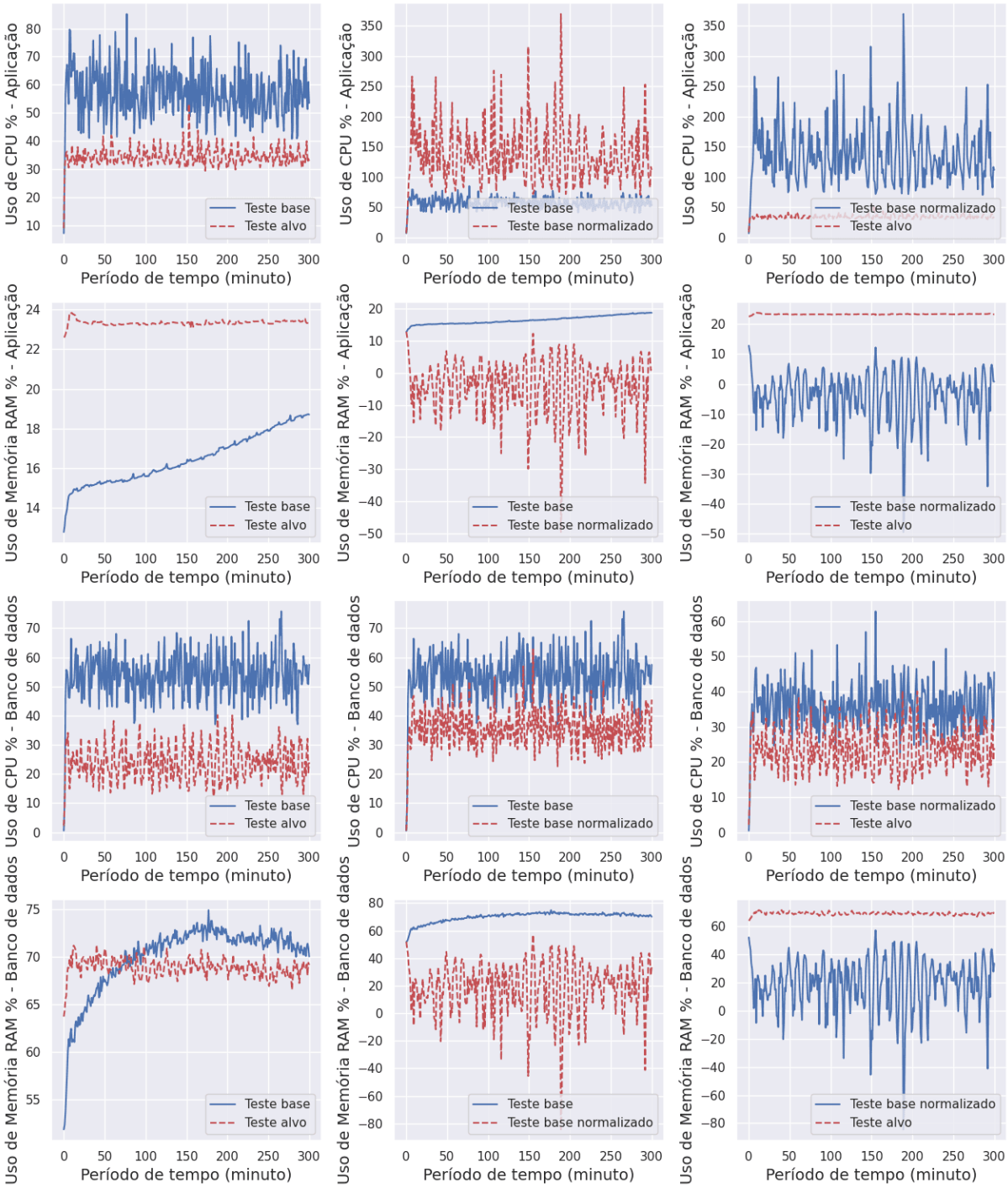
Fonte: Autores

Figura 33 – Gráficos de normalização versão Boa x DE para métricas de linhas recebidas, linhas retornadas, transações concluídas e tempo de resposta - Gráfico de controle



Fonte: Autores

Figura 34 – Gráficos de normalização versão Boa x DE para métricas de uso de *cpu* e memória da aplicação e do banco de dados - Gráfico de controle

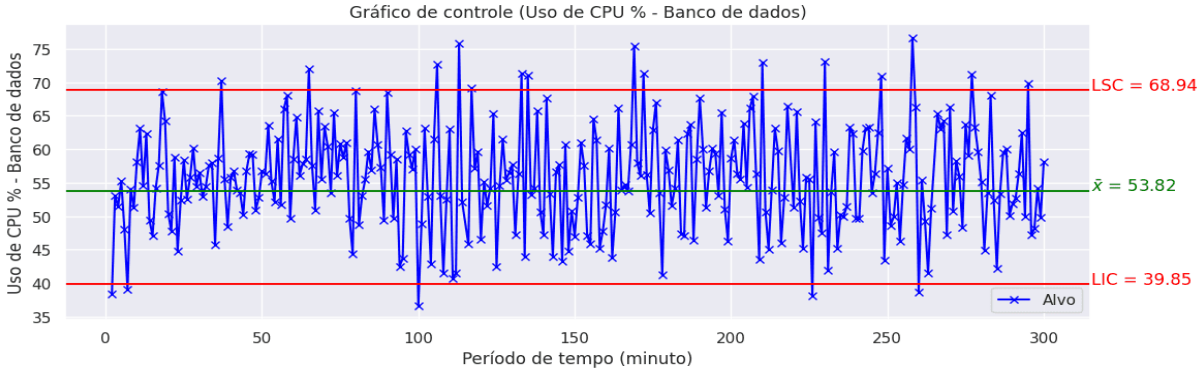


Fonte: Autores

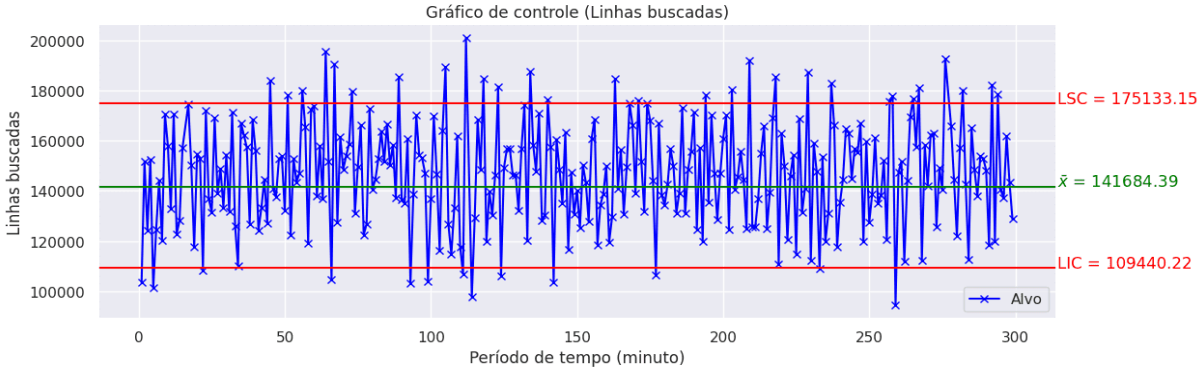
APÊNDICE C – Gráficos de controle gerados das análises

Este apêndice apresenta os gráficos de controle derivados da comparação entre a versão estável e outra versão estável, bem como entre a versão estável e a versão com um limite de consulta maior. Não foram obtidos resultados para a comparação entre a versão estável e a versão com retorno de dados excessivos, uma vez que, na ocasião, a versão com retorno de dados excessivos não atendeu às premissas fundamentais para a construção do gráfico de controle.

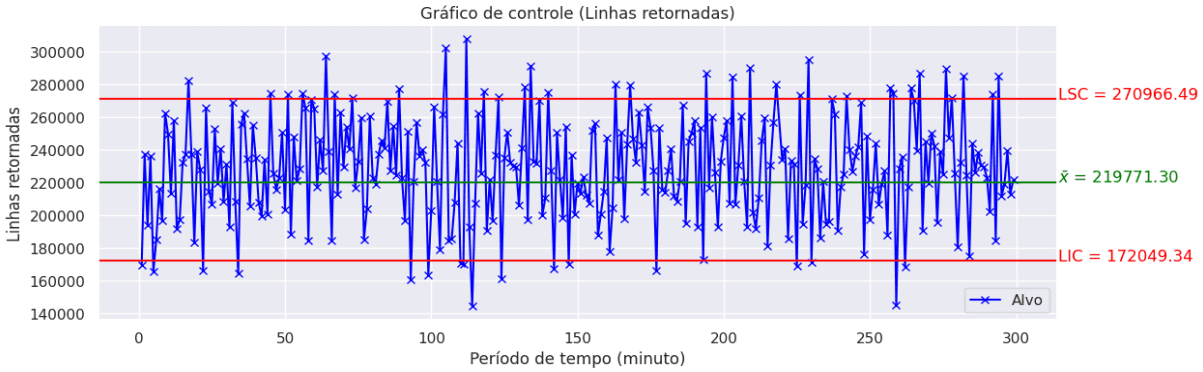
Figura 35 – Gráficos de controle da versão Boa x versão Boa



(a) Gráfico de controle métrica de porcentagem do uso de CPU banco de dados



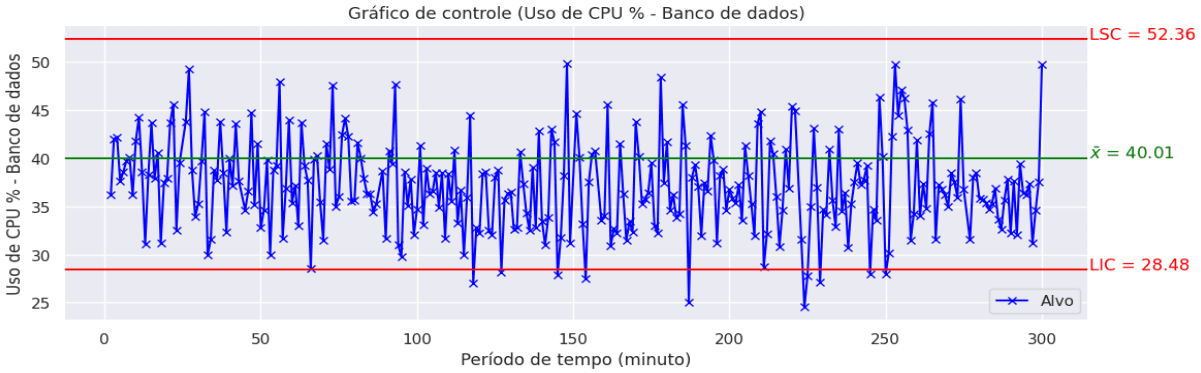
(b) Gráfico de controle métrica de linhas buscadas



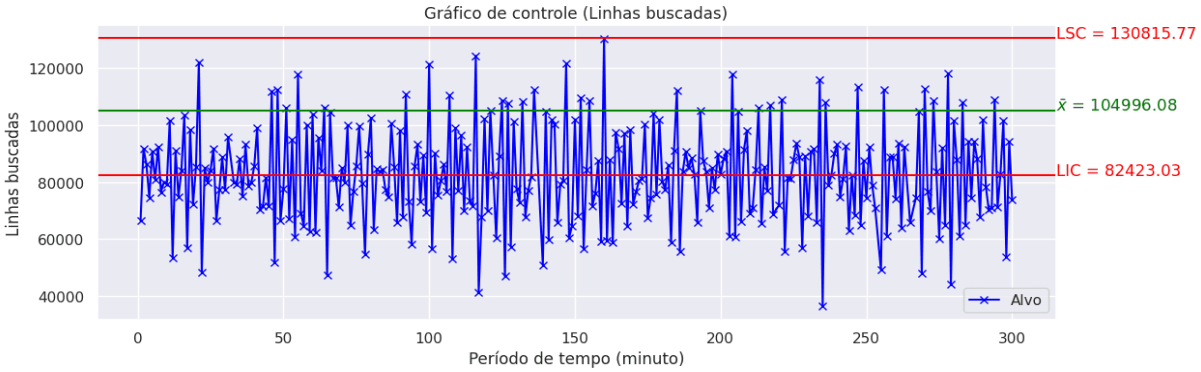
(c) Gráfico de controle métrica de linhas retornadas

Fonte: Autores

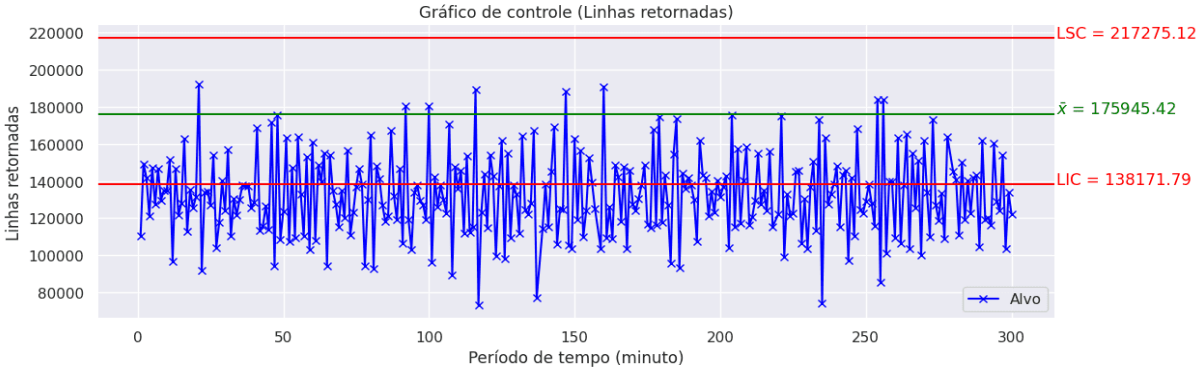
Figura 36 – Gráficos de controle da versão Boa x versão Boa



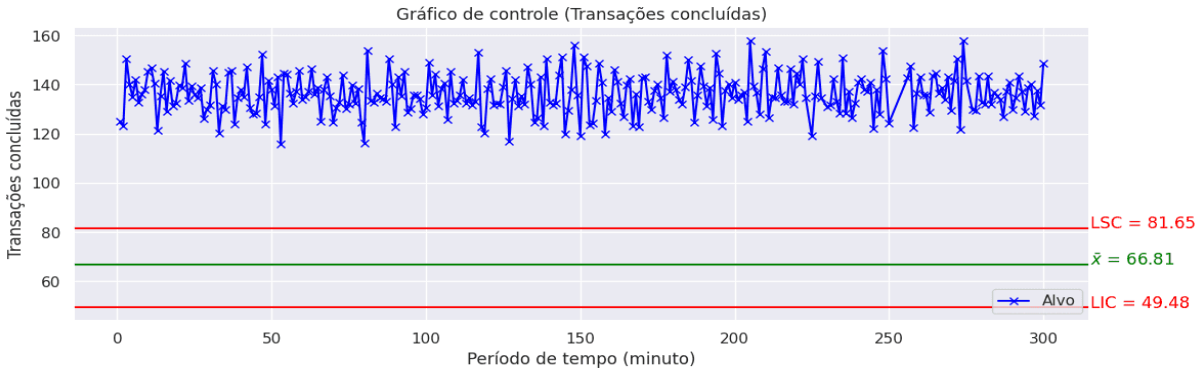
(a) Gráfico de controle métrica de porcentagem do uso de CPU banco de dados



(b) Gráfico de controle métrica de linhas buscadas



(c) Gráfico de controle métrica de linhas retornadas



(d) Gráfico de controle métrica de transações concluídas

Fonte: Autores