IDENTIFICAÇÃO DE VALORES DE REFERÊNCIA DE MÉTRICAS EM SOFTWARE PYTHON[1]

Marcus Vinícius Diniz Ribeiro[2]
Kecia Aline Marques Ferreira[3]

Submetido em: 17/06/2026 | Aprovado em: 17/06/2026

RESUMO

A qualidade de software pode ser avaliada por meio de métricas quantitativas capazes de indicar características estruturais como complexidade, acoplamento, herança e coesão. Embora a linguagem Python tenha ampla adoção em diferentes domínios, há carência de valores de referência que auxiliem a interpretação objetiva dessas métricas em sistemas orientados a objetos. Diante desse cenário, este trabalho tem como objetivo identificar valores de referência para métricas de software aplicadas a software Python. Para isso, foi utilizada uma ferramenta responsável pela extração automática de 11 métricas orientadas a objetos, em uma amostra de 45 repositórios de código aberto selecionados no GitHub. Os dados obtidos foram consolidados e analisados estatisticamente com base em percentis, seguindo uma abordagem já aplicada em estudos anteriores com sistemas Java. A partir dessa abordagem, definimos intervalos de classificação para as métricas analisadas, categorizando seus valores como Bom/Frequente, Regular/Ocasional e Ruim/Raro. Os valores de referência identificados foram avaliados por meio de um estudo de caso. Os resultados sugerem que os limiares propostos mostram-se promissores como instrumentos de triagem para apontar componentes complexos com potencial necessidade de refatoração.

Palavras-chave: Métricas de software. Qualidade de software. Python. Programação orientada a objetos. Valores de referência.

ABSTRACT

Software quality can be assessed using quantitative metrics that represent structural characteristics of source code, such as complexity, coupling, inheritance, cohesion, and class size. Although Python is widely used in several domains, there is still a lack of thresholds to support the objective interpretation of object-oriented software metrics in this language. In this context, this work aims to identify thresholds for

[1] Artigo científico elaborado para o trabalho de conclusão do curso de Engenharia de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG.)

[2] Graduando em Engenharia de Computação no campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). marcus.ribeiro@aluno.cefetmg.br

[3] Orientadora do trabalho. Docente do Departamento de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). kecia@cefetmg.br

software metrics applied to Python systems. To achieve this goal, 45 open-source repositories hosted on GitHub were selected and analyzed using a tool proposed in the literature to automatically extract 11 object-oriented metrics from Python software systems. The collected data were consolidated and statistically analyzed following an approach previously applied in studies with Java systems. We then defined classification ranges for the analyzed metrics, categorizing their values as Good/Frequent, Regular/Occasional, and Bad/Rare. The identified thresholds were evaluated through a case study. The results indicate that the proposed thresholds are applicable to identifying classes that require refactoring.

Keywords: Software metrics. Software quality. Python. Object-oriented programming. Thresholds.

1 INTRODUÇÃO

Softwares podem ser resumidos como um conjunto de instruções e dados que controlam o funcionamento de dispositivos computacionais, abrangendo desde sistemas operacionais e compiladores a aplicações comerciais, científicas e industriais (Sommerville, 2018). Com a presença crescente desses sistemas em diferentes áreas, também aumenta a necessidade de produzir códigos compreensíveis, organizados e capazes de evoluir ao longo do tempo. Na década de 1960, surgem as primeiras bases da programação Orientada a Objetos. Este paradigma organiza o software em torno de entidades chamadas objetos, os quais encapsulam atributos (responsáveis por armazenar dados) e métodos (responsáveis por representar comportamentos ou operações). Esse paradigma promove princípios como encapsulamento, herança e reutilização, contribuindo para a construção de sistemas mais modulares e extensíveis (Booch *et al.*, 2011).

O paradigma orientado a objetos, então, possui características que favorecem a criação de softwares de alta qualidade. Todavia, é importante que a qualidade dos softwares seja mensurável. A medição de software possibilita, entre outros aspectos, que Engenheiros de Software identifiquem necessidades de melhoria na estrutura dos softwares. Nesse contexto, métricas de software passam a ser utilizadas como instrumentos quantitativos para avaliar características internas dos sistemas. Em softwares orientados a objetos, essas métricas permitem observar propriedades relacionadas a classes, métodos, atributos, acoplamento, coesão e herança. Assim, torna-se possível analisar aspectos estruturais do código de forma menos subjetiva, o que apoia decisões de manutenção, revisão e refatoração.

Entre os conjuntos de métricas mais conhecidos para sistemas orientados a objetos está o conjunto CK, proposto por Chidamber e Kemerer (Chidamber; Kemerer, 1994). Esse conjunto reúne métricas voltadas à avaliação de aspectos como complexidade, acoplamento, herança e coesão em classes. Por esse motivo, tornou-se uma referência importante na literatura sobre qualidade de software, sendo utilizado em diferentes estudos para apoiar a análise estrutural de sistemas orientados a objetos.

Entretanto, o uso de métricas, por si só, não é suficiente para indicar o grau de qualidade estrutural e se há necessidade de melhoria na estrutura do sistema. Para que uma métrica seja útil na prática, é necessário compreender quais valores podem ser considerados bons, regulares ou ruins em determinado contexto. Por exemplo, um valor considerado elevado para a quantidade de linhas de código em uma determinada linguagem de programação pode representar um padrão comum em outra. Essa variação ocorre em

função das características próprias de cada linguagem, do nível de abstração fornecido pelos *frameworks* utilizados e das práticas de desenvolvimento predominantes na comunidade associada à tecnologia.

Essa necessidade motivou a realização de trabalhos voltados à definição de valores de referência para métricas de software. No contexto de sistemas Java, Gonçalves (2022) propôs uma abordagem estatística baseada na análise de percentis de valores de métricas observados na prática de desenvolvimento de software. Aplicando essa abordagem, o autor definiu intervalos capazes de indicar valores considerados bons, regulares ou ruins, reduzindo a dependência de interpretações subjetivas. Ele aplicou essa abordagem para propor valores de referência para 46 métricas de software em Java. Posteriormente, Mendes (2024) aplicou esses valores de referência para avaliar a qualidade estrutural de softwares Java voltados à Inteligência Artificial. O estudo reforça que os valores de referência podem variar conforme o domínio analisado, indicando que diferentes contextos de desenvolvimento podem demandar parâmetros específicos de avaliação. Apesar desses avanços em outros contextos, especialmente em sistemas Java, o cenário ainda é menos consolidado quando se observa a linguagem Python.

Atualmente, o Python destaca-se como a linguagem de programação mais popular,[4] refletindo sua ampla adoção em áreas como Ciência de Dados, Automação, Desenvolvimento Web e Inteligência Artificial. Apesar de sua popularidade, observa-se uma carência de ferramentas e estudos acadêmicos focados na avaliação objetiva da qualidade de código em Python, uma vez que grande parte das métricas e dos instrumentos disponíveis foi desenvolvida para linguagens como Java e C++. Alguns estudos realizados sobre medição de software Python aplicam métricas que não avaliam aspectos específicos da orientação a objetos. O trabalho de Moldovan *et al.* (2024), por exemplo, apresenta um conjunto de métricas coletadas de 80 projetos em Python. Entretanto, as métricas foram coletadas utilizando ferramentas como o SonarQube, que não coletam métricas específicas da orientação a objetos.

Visando preencher parte dessa lacuna, Sabioni (2025) desenvolveu a ferramenta PyCKTool, que coleta um conjunto de métricas orientadas a objetos aplicáveis a sistemas em Python. Essa ferramenta permite extrair automaticamente os indicadores a partir do código-fonte, promovendo análises quantitativas que podem apoiar decisões de refatoração, avaliação de legibilidade e manutenção de código Python. Entretanto, os valores de referência das métricas de software especificamente para o contexto do Python não foram definidos. O presente trabalho visa contribuir para contornar esse problema.

1.1 Objetivo

Este trabalho tem como objetivo principal estabelecer valores de referência para métricas de software orientado a objetos aplicadas a sistemas desenvolvidos em Python.

A pesquisa foi conduzida a partir da aplicação da PyCKTool (Sabioni, 2025), que coleta 11 métricas de software orientado a objetos em software Python. A ferramenta foi utilizada para coletar métricas de uma amostra selecionada de 45 projetos *open source* escritos em Python. Os dados coletados foram organizados e analisados estatisticamente com base no método proposto por Gonçalves (2022), que utiliza percentis para identificar valores típicos e atípicos nas métricas avaliadas. Com base nisso, são propostos intervalos

[4] De acordo com o índice TIOBE, disponível em: <https://www.tiobe.com/tiobe-index/>. Acesso em 10 de junho de 2026.

de valores de referência que representem padrões comuns entre os projetos analisados e que possam ser utilizados como apoio na avaliação de qualidade, manutenção e refatoração de sistemas.

1.2 Relevância

A importância deste estudo está relacionada à ausência de diretrizes quantitativas consolidadas para avaliação da qualidade de código em sistemas desenvolvidos com a linguagem Python. Enquanto linguagens como Java já contam com ferramentas amplamente utilizadas, como a CK Tool, e conjuntos de métricas bem estabelecidos, o ecossistema Python ainda apresenta lacunas nesse aspecto.

A aplicação da PyCKTool em projetos reais tem potencial para contribuir com a consolidação de parâmetros objetivos que auxiliem na avaliação de sistemas Python. A derivação de valores de referência, a partir da análise estatística de métricas extraídas dos projetos selecionados, pode fornecer suporte para atividades como revisão de código, identificação de anomalias estruturais, priorização de refatorações e formação de desenvolvedores.

Além disso, o estudo busca fortalecer a utilidade prática da ferramenta PyCKTool, validando sua aplicação em cenários reais e ampliando seu alcance como instrumento gratuito e acessível para análise de qualidade de software orientado a objetos na linguagem Python.

O restante deste trabalho está organizado da seguinte forma: a Seção 2 descreve os conceitos fundamentais aplicados neste trabalho; a Seção 3 discute os trabalhos relacionados à avaliação de qualidade de software, à definição de valores de referência para métricas e às ferramentas de coleta de métricas em Python; a Seção 4 apresenta os procedimentos adotados para seleção dos projetos, coleta, processamento e análise dos dados; a Seção 5 apresenta os valores de referência obtidos e discute sua interpretação no contexto da linguagem Python; a Seção 6 avalia a aplicabilidade dos valores propostos por meio de um estudo de caso; a Seção 7 descreve as principais limitações da pesquisa; e, por fim, a Seção 8 apresenta as considerações finais e indica possibilidades de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os conceitos fundamentais que sustentam esta pesquisa, com foco na qualidade de software e no uso de métricas orientadas a objetos como instrumentos de avaliação estrutural. São discutidos os princípios de modularidade, coesão e acoplamento, bem como descreve as métricas de software orientado a objetos aplicadas neste trabalho.

2.1 Modularidade de Software

A norma ISO 25010 (ISO, 2023) define as características de qualidade de produto de software. Dentre essas características está a manutenibilidade, que é o grau de facilidade para realizar modificações no software. A manutenibilidade é diretamente afetada pela forma como o software é construído, em especial pelo grau de modularidade do software. A modularidade é a característica que reflete o grau em que o sistema é constituído por módulos o mais independentes possíveis entre si. Segundo Meyer (1997), sistemas modulares favorecem arquiteturas flexíveis compostas por elementos autônomos, organizados em estruturas simples e coesas, o que contribui diretamente para os objetivos de reutilização e

extensibilidade. A efetividade da modularidade, nesse contexto, pode ser avaliada com base em dois atributos fundamentais: coesão e acoplamento.

O acoplamento refere-se ao grau de dependência entre diferentes módulos do sistema, sendo desejável que essa dependência seja minimizada. Segundo Booch *et al.* (2011) sistemas orientados a objetos bem estruturados apresentam baixo acoplamento e alta coesão, características essenciais para a modularidade e a qualidade do software. Um baixo acoplamento reduz os impactos de alterações em módulos individuais sobre os demais, promovendo maior facilidade na manutenção, na evolução e na reutilização do software. Por sua vez, a coesão refere-se à intensidade do relacionamento entre os elementos internos de um módulo. Módulos altamente coesos concentram-se em uma única responsabilidade, o que melhora a clareza, facilita a testabilidade e contribui para uma manutenibilidade.

As características da orientação a objetos favorecem a construção de software modular: classes, herança e ocultação de informações. Entretanto, avaliar manualmente se um sistema orientado a objetos aplica adequadamente essas características é uma tarefa complexa, especialmente em softwares de grande porte. A medição de software auxilia nessa avaliação.

2.2 Métricas de Software

Para avaliar a presença ou ausência de atributos como coesão e acoplamento, uma solução é realizar medidas específicas. Nesse contexto, métricas de software podem ser entendidas como instrumentos quantitativos utilizados para medir e monitorar características internas do software (Meyer, 1997). Entre os diversos conjuntos de métricas disponíveis, destaca-se o conjunto CK, proposto por Chidamber e Kemerer (1994), que visa quantificar, de forma objetiva, propriedades fundamentais do design orientado a objetos. Essas métricas foram concebidas para suprir as limitações das abordagens tradicionais e possibilitam a mensuração de fatores como complexidade, acoplamento, herança e coesão em classes, atributos fundamentais para o controle e evolução da qualidade estrutural em sistemas orientados a objetos. O conjunto CK é composto por seis métricas principais:

- WMC (Weighted Methods per Class): quantifica a complexidade de uma classe a partir da soma ponderada das complexidades de seus métodos. Os cálculos de complexidade não são específicos e podem variar de implementação para implementação, geralmente baseados no número de declarações de cada método. Classes com WMC elevado tendem a ser mais complexas, o que dificulta o entendimento, os testes e a manutenção.
- DIT (Depth of Inheritance Tree): representa a profundidade de uma classe na hierarquia de herança do sistema, ou seja, a distância entre a classe e a raiz da árvore. Valores altos indicam maior especialização e podem aumentar a reutilização, mas também tornam a predição de comportamento mais complexa.
- NOC (Number of Children): indica o número de subclasses diretas de uma determinada classe. Um NOC elevado pode sinalizar maior reutilização e importância estrutural da classe, mas também aumenta o impacto de eventuais alterações em sua implementação.
- CBO (Coupling between Object Classes): mede o número de outras classes com as quais uma classe está acoplada, ou seja, com as quais mantém dependências por meio

da utilização de métodos, atributos ou outro mecanismos de interação. Quanto maior o CBO, maior a dependência externa, o que pode comprometer a modularidade e dificultar a manutenção do sistema.

- RFC (Response for a Class): calcula o total de métodos próprios de uma classe, somado aos métodos externos que cada um deles invoca. Um RFC alto indica maior complexidade na resposta a chamadas, sugerindo um fluxo de execução mais difícil de prever e controlar.
- LCOM (Lack of Cohesion in Methods): avalia a coesão interna de uma classe com base no compartilhamento de variáveis de instância entre seus métodos. Um LCOM elevado indica baixa coesão, sugerindo que a classe pode estar assumindo responsabilidades excessivas e ser candidata à refatoração.

Além das métricas originalmente propostas por Chidamber e Kemerer (1994), este trabalho também considera métricas complementares extraídas pela ferramenta PyCKTool (Sabioni, 2025), uma vez que contribuem para caracterizar aspectos de tamanho, dependência e estrutura interna das classes em sistemas Python.

- FIN (Fan-in):representa o número de dependências de entrada associadas a uma classe, isto é, a quantidade de outras classes que utilizam, referenciam ou dependem da classe analisada. Valores elevados de FIN podem indicar que uma classe exerce papel central no sistema, o que aumenta sua relevância estrutural e também o impacto potencial de alterações em sua implementação.
- FOUT (Fan-out): indica o número de dependências de saída de uma classe, ou seja, a quantidade de outras classes das quais a classe analisada depende. Valores elevados de FOUT sugerem maior dependência externa, o que pode indicar maior acoplamento e maior sensibilidade a mudanças em outros componentes do sistema.
- NOA (Number of Attributes): corresponde ao número de atributos associados a uma classe. Classes com muitos atributos podem concentrar uma quantidade elevada de estado interno, o que pode dificultar a compreensão, a manutenção e a evolução da implementação, especialmente quando esses atributos estão relacionados a responsabilidades distintas.
- NOM (Number of Methods): representa o número de métodos definidos em uma classe. Valores elevados de NOM podem indicar classes com muitas operações ou responsabilidades, sendo úteis para identificar possíveis violações do princípio de responsabilidade única.
- LLOC (Logical Lines of Code): contabiliza as linhas lógicas de código associadas à classe, desconsiderando elementos puramente textuais, como comentários e linhas em branco, conforme a implementação da ferramenta utilizada. Essa métrica fornece uma estimativa do tamanho funcional da classe e auxilia na identificação de componentes extensos ou potencialmente difíceis de manter.

Dessa forma, enquanto as métricas CK fornecem a base clássica para avaliação de complexidade, herança, acoplamento e coesão, as métricas complementares FIN, FOUT, NOA, NOM e LLOC ampliam a análise estrutural realizada neste trabalho.

3 TRABALHOS RELACIONADOS

a revisão da literatura foram identificados um extensa produção relacionada a métricas e limiares de qualidade de código em Java, por outro lado foram encontrados menos estudos com o mesmo enfoque aplicados especificamente a projetos Python. Este trabalho se concentra em atender a essa necessidade: diferentemente de pesquisas anteriores, que se concentraram na criação de novas ferramentas ou na análise de outras linguagens, este estudo utiliza uma solução existente para adaptar um método estatístico já conhecido, ao contexto do Python. O objetivo final é estabelecer valores de referência que permitam avaliar a qualidade do código nesta linguagem.

Atualmente, ferramentas como o SonarQube, Understand e Radon são muito utilizadas na análise de sistemas Python, mas priorizam indicadores gerais de estilo e complexidade estrutural. Com isso, há uma carência de ferramentas que extraiam, de forma unificada, as métricas clássicas de orientação a objetos (conjunto CK) especificamente para essa linguagem. Além disso, algumas dessas soluções dependem de licenciamento pago ou exigem configurações avançadas para uso em projetos de menor complexidade. Frente a esse cenário, Sabioni (2025) desenvolveu a ferramenta PyCKTool, uma solução gratuita e de código aberto, escrita em Python, que viabiliza a coleta de métricas orientadas a objetos de forma semelhante à da ferramenta CKTool, que coleta métricas para Java.[5] Baseada na biblioteca `astroid` para o *parsing* do código, PyCKTool extrai métricas CK e métricas adicionais como FIN, FOUT, NOA, NOM e LLOC, descritas na Seção 2.2. Embora apresente limitações, como a análise isolada por arquivo, o que pode dificultar a identificação de acoplamentos entre classes distribuídas, e a ausência de interface gráfica, PyCKTool possui recursos suficientes para possibilitar a extração de dados necessária para este estudo.

Do ponto de vista metodológico, este trabalho fundamenta-se na abordagem proposta por Gonçalves (2022) para a derivação de valores de referência. Em seu estudo sobre sistemas Java, Gonçalves (2022) definiu limites quantitativos para classificar métricas em três faixas: Bom/Frequente, Regular/Ocasional e Ruim/Raro. A abordagem fundamenta-se na análise estatística por percentis dos valores das métricas coletadas em um conjunto de softwares abertos desenvolvidos em Java. A metodologia consistiu na consolidação de métricas de 46 sistemas, seguida da aplicação dos percentis 70 e 90 para a definição dos intervalos, com tratamento específico para a remoção de valores nulos quando necessário. Com isso, para cada métrica de software, os limiares dos valores de referência são definidos da seguinte forma: *Bom/Frequente* corresponde aos valores abaixo do percentil 70, *Regular/Ocasional* engloba os valores entre o percentil 70 e o 90, e *Ruim/Raro* corresponde aos valores acima do percentil 90. A avaliação desses valores de referência evidenciou que as classes com métricas na faixa Ruim apresentam maior incidência de falhas, corroborando a eficácia do uso de percentis na avaliação da qualidade. O método de Gonçalves (2022) serve, portanto, como o principal referencial metodológico adotado e adaptado neste trabalho para o contexto da linguagem Python.

A aplicabilidade e a robustez dessa metodologia de percentis têm sido exploradas em outros trabalhos, como demonstra a pesquisa de Mendes (2024), que utilizou a mesma abordagem proposta por Gonçalves (2022) para caracterizar a qualidade estrutural de softwares Java aplicados especificamente à Inteligência Artificial. Ao analisar 42 métricas em 50 repositórios, foram estabelecidos valores de referência que permitiram concluir que softwares de IA tendem a apresentar qualidade estrutural inferior à média de sistemas

[5] <https://github.com/mauricioaniche/ck>

de domínios gerais, especialmente nos aspectos de acoplamento e complexidade. Esse estudo reforça a importância de estabelecer valores de referência específicos para diferentes contextos e linguagens, validando o esforço desta pesquisa ao traduzir e adaptar tais métricas à realidade do desenvolvimento em Python.

4 METODOLOGIA

Esta seção descreve a metodologia empregada neste estudo.

4.1 Seleção de Softwares

A seleção dos sistemas Python analisados nesta pesquisa foi realizada por meio da API pública de busca do GitHub, com o objetivo de identificar repositórios abertos relevantes e amplamente utilizados pela comunidade. Buscou-se compor uma amostra contendo aproximadamente 45 a 50 projetos, de modo a manter uma quantidade de sistemas analisados próxima à utilizada por Gonçalves (2022). Inicialmente, os repositórios foram obtidos a partir da ordenação pelo número de estrelas, restringindo os resultados àqueles que definiam Python como linguagem principal. Nessa etapa, aproximadamente 100 projetos foram coletados. Posteriormente, realizou-se uma curadoria manual com o objetivo de remover repositórios de caráter estritamente educacional, como tutoriais e materiais de cursos, bem como projetos que agregavam múltiplas aplicações em um único repositório, situação que poderia comprometer a consistência da análise das métricas.

Inicialmente, considerou-se restringir a análise a projetos com mais de cinco anos de existência, a fim de garantir maturidade e estabilidade. No entanto, com o crescimento de iniciativas relacionadas à inteligência artificial nos últimos anos, impulsionadas pela popularização de modelos de linguagem (LLMs), optou-se por incluir projetos mais recentes que apresentassem forte representatividade atual. Um exemplo notável é o repositório do modelo DeepSeek, identificado pelo GitHub com Python como linguagem principal, o que reflete o uso contemporâneo da linguagem. Essa escolha tem o intuito de capturar o estado atual do ecossistema Python, considerando tanto projetos consolidados quanto aqueles em ascensão.

A versão completa desta tabela, incluindo os respectivos *links* para os repositórios no GitHub, encontra-se disponível no Apêndice A deste trabalho, possibilitando a replicação dos experimentos e futuras análises comparativas.

4.2 Coleta e Processamento dos Dados

A coleta de dados foi realizada a partir dos 45 repositórios Python selecionados, executados individualmente na ferramenta PyCKTool (Sabioni, 2025), responsável pela extração automática de métricas orientadas a objetos. Os resultados foram exportados no formato CSV, contendo os valores das métricas WMC, DIT, NOC, CBO, RFC, LCOM, NOA, NOM, LLOC, Fan-in e Fan-out.

Em seguida, os arquivos gerados foram consolidados e tratados por meio de *scripts* em Python, desenvolvidos especificamente para este estudo e disponibilizados em um repositório no GitHub. Esses *scripts* realizaram a leitura, a limpeza e a normalização dos dados, removendo registros inválidos e garantindo a consistência entre os diferentes projetos. O processamento resultou em uma base unificada de métricas, utilizada para a derivação estatística dos valores de referência.

4.3 Derivação dos Valores de Referência

Este trabalho aplica uma adaptação do método proposto por (Gonçalves, 2022) para a derivação de valores de referência de métricas de software. Conforme descrito na Seção 3, o método baseia-se na análise estatística de métricas extraídas de um conjunto de softwares abertos. Basicamente, o método consiste em definir três faixas de valores de referência para uma métrica:

- *Bom/Frequente*: corresponde aos valores mais frequentes na prática. Esses valores são aqueles que estão abaixo do percentil 70;
- *Regular/Ocasional*: engloba os valores com frequência intermediária na prática, correspondendo aos valores entre o percentil 70 e o 90;
- *Ruim/Raro*: corresponde aos valores com baixa frequência na prática, situados acima do percentil 90.

A adoção específica dos percentis de ordem 70 e 90 como limiares para a divisão das faixas de qualidade fundamenta-se estritamente nos critérios metodológicos estabelecidos por Gonçalves (2022). Em sua abordagem, o autor baseou-se no estudo de Filó (2014) para estipular que, para a maioria das métricas de software orientadas a objetos, esses dois índices delimitam os intervalos de forma adequada para refletir o comportamento dos sistemas na prática. Conforme descrito por Gonçalves (2022), o percentil de ordem K atua como uma medida de posição que ordena o conjunto de dados de forma crescente, separando K das observações menores ou iguais a ele. Dessa forma, o percentil 70 estabelece o teto para o comportamento mais frequente e ideal (faixa Bom), enquanto o percentil 90 isola os 10% de valores extremos da cauda superior da distribuição (faixa Ruim). O autor aponta ainda que o emprego dos percentis 70 e 90 para a classificação de faixas de controle encontra paralelo em critérios consolidados de validação de qualidade em outras áreas do conhecimento científico, a exemplo da análise de dados epidemiológicos. A manutenção desses mesmos parâmetros neste estudo viabiliza a consistência metodológica necessária para a comparação dos perfis estruturais entre os ecossistemas Java e Python.

Na análise dos dados das métricas em softwares Java, (Gonçalves, 2022) observou a presença de uma grande quantidade de valores nulos (iguais a zero) em algumas métricas, o que enviesaria a definição das faixas de referência. Nesses casos, os valores nulos foram desconsiderados no cálculo dos percentis. Neste trabalho, antes da derivação via percentis, foi realizada uma inspeção preliminar dos conjuntos de métricas obtidos, com o objetivo de identificar a ocorrência de valores nulos, conforme a metodologia proposta por Gonçalves (2022). Essa verificação é importante porque, em alguns contextos, métricas que assumem valor zero recorrentemente podem indicar a ausência de comportamento mensurável (por exemplo, classes sem herança, sem acoplamento ou sem métodos).

No entanto, ao analisar os resultados dos 45 projetos em Python, observou-se que a presença de valores nulos não se repetiu de forma generalizada. Em outras palavras, não houve concentração significativa de métricas cujo percentil 70 fosse igual a zero, o que indicaria predominância de classes estruturalmente simples ou pouco representativas. Esse comportamento reforça que, no conjunto analisado, não foi observada concentração suficiente de valores zero para justificar a remoção desses registros antes do cálculo dos percentis.

Essa diferença em relação ao Java decorre de características estruturais das próprias linguagens. No Java, as instruções, constantes e funções utilitárias são comumente encapsuladas em classes, o que frequentemente gera componentes puramente organizacionais e sem lógica interna. Já o Python permite declarar funções e variáveis diretamente no escopo dos módulos (arquivos `.py`). Com isso, a criação de classes em Python costuma se restringir a cenários que realmente exigem a modelagem de dados e métodos, reduzindo a ocorrência de métricas zeradas na amostra.

Além do fator da linguagem, a seleção dos projetos também ajuda a entender esse resultado. Como a amostra é composta por 45 repositórios populares e consolidados no GitHub (como `Django`, `FastAPI` e `PyTorch`), o conjunto de dados reflete softwares maduros e estáveis. Nesses sistemas, a ocorrência de classes vazias, códigos temporários ou testes inacabados é potencialmente menor.

Durante a análise, também foi levantada uma questão relevante sobre a influência do número de classes de cada projeto na derivação dos valores de referência. Projetos de grande porte, com centenas de classes, potencialmente produzem um volume de amostras maior, o que poderia impactar a distribuição geral das métricas quando comparado a projetos menores. Essa diferença poderia, em tese, atribuir maior peso estatístico aos projetos mais extensos, deslocando ligeiramente os percentis globais.

Para investigar esse possível efeito, os projetos foram segmentados em quatro grupos de acordo com sua quantidade total de classes. A divisão foi realizada a partir dos quartis da distribuição do número de classes dos softwares analisados, permitindo separar os projetos em faixas representativas de porte. O primeiro quartil (Q1) correspondeu a 100 classes, a mediana (Q2) a 513 classes e o terceiro quartil (Q3) a 1580 classes. Com base nesses limites, os projetos foram classificados nos grupos Pequeno (até 100 classes), Médio-Pequeno (101 a 513 classes), Médio-Grande (514 a 1580 classes) e Grande (mais de 1580 classes).

A análise comparativa, apresentada na Tabela 1, demonstra que os percentis 70 e 90 apresentaram variações reduzidas entre os diferentes grupos e o conjunto completo de projetos (“Todos”).

Essa baixa variabilidade sinaliza uma homogeneidade na distribuição das métricas na amostra avaliada, independentemente do tamanho do projeto, o que justifica o uso de dados consolidados para definir valores de referência globais. Tal fato corrobora o objetivo da pesquisa de identificar tendências gerais das métricas no ecossistema Python, sugerindo que a agregação de todas as classes representa, de forma ampla, o comportamento estrutural da fatia de projetos de código aberto mais expressivos do ecossistema atual.

Por fim, uma vez validada a estratégia de consolidação dos dados, procedeu-se ao cálculo efetivo dos valores de referência. Utilizando a base unificada, que contém todas as classes processadas, foram computados os percentis 70 (P70) e 90 (P90) para cada uma das métricas analisadas. Esses limiares estatísticos foram obtidos por meio de *scripts* em Python utilizando a biblioteca Pandas, servindo posteriormente como divisores para as faixas de classificação de qualidade: “Bom/Frequente” (valores abaixo do P70), “Regular/Ocasional” (entre P70 e P90) e “Ruim/Raro” (acima de P90).

4.4 Avaliação dos Valores de Referência

Um dos objetivos da definição de valores de referência para métricas de qualidade de código é auxiliar o programador na identificação de classes que merecem maior atenção

Tabela 1 – Comparação dos Percentis (P70 e P90) por Grupo de Tamanho do Projeto

	Percentil 70					Percentil 90				
Métrica	**Gr**	**M-Gr**	**M-Pq**	**Pq**	**Todos**	**Gr**	**M-Gr**	**M-Pq**	**Pq**	**Todos**
CBO	3	2	2	2	3	6	6	5	4.70	6
DIT	2	2	2	1	2	3	3	3	2	3
FIN	1	1	1	1	1	3	3	3	2	3
FOUT	2	2	1	1	2	4	3	3	3	4
LCOM	2	2	2	2	2	5	5	3	4	5
LLOC	23	25	25	23	23	70	74	84	62,70	71
NOA	2	1	3	2	2	6	6	9	7	6
NOC	3	3	2	2	3	7	8	9	4,70	7
NOM	3	4	3	3	3	9	9	9	8	9
RFC	11	11	11	10	11	27	26	29,50	26,70	27
WMC	19	21	20	18	19	63	65	75,50	56	63

Fonte: Elaborado pelo autor (2026).

Legenda: Gr (Grande), M-Gr (Médio-Grande), M-Pq (Médio-Pequeno), Pq (Pequeno).

durante atividades de revisão, manutenção e refatoração. Diante disso, foi realizado um estudo de caso com o intuito de avaliar a aplicabilidade prática dos valores de referência derivados neste trabalho em um sistema Python real.

Para a realização do estudo de caso, foi selecionado um dos sistemas pertencentes ao conjunto de projetos analisados nesta pesquisa. O sistema escolhido foi o *Requests*, uma biblioteca Python amplamente utilizada para realização de requisições HTTP. A escolha desse sistema se justifica por dois fatores principais. Primeiro, trata-se de um projeto relacionado a um domínio mais próximo da experiência prática do autor, o que favorece a interpretação das responsabilidades das classes analisadas. Segundo, o projeto possui um porte relativamente reduzido em comparação com outros sistemas da amostra, contendo aproximadamente 72 classes, o que torna viável uma inspeção manual mais detalhada dos resultados obtidos.

Após a escolha do sistema, o projeto *Requests* foi analisado novamente por meio da ferramenta PyCKTool. Em seguida, esses valores foram comparados com as faixas de referência previamente definidas, classificando cada métrica em uma das três categorias adotadas: Bom/Frequente, Regular/Ocasional e Ruim/Raro.

A análise foi realizada com base na seleção de métricas mais diretamente relacionadas à complexidade, tamanho, acoplamento e quantidade de responsabilidades internas das classes, como WMC, CBO, RFC, LCOM, LLOC, NOA e NOM. Com base nessas métricas, foram identificadas classes que apresentavam valores elevados, especialmente aquelas enquadradas na faixa Ruim/Raro em mais de uma métrica, e também valores baixos. Posteriormente, essas classes foram examinadas manualmente, considerando sua função dentro do sistema e a coerência entre os valores obtidos e sua implementação.

Dessa forma, o estudo de caso não teve como objetivo validar os valores de referência de forma estatística, mas sim verificar sua utilidade prática como instrumento de apoio à análise estrutural de sistemas Python. A avaliação buscou observar se as classes indicadas pelas métricas como pontos de atenção correspondem de fato a componentes mais complexos, extensos ou centrais no sistema, e se as classes classificadas em faixas

melhores apresentavam responsabilidades mais simples e bem delimitadas.

5 RESULTADOS

Esta Seção apresenta a consolidação e a análise dos dados extraídos dos 45 projetos em Python selecionados. O objetivo central é definir os valores de referência para as métricas suportadas pela ferramenta PyCKTool. Os resultados aqui apresentados colocam em prática o método proposto, convertendo os números extraídos em faixas de referência: *Bom/Frequente* (abaixo do percentil 70), *Regular/Ocasional* (entre o percentil 70 e o 90) e *Ruim/Raro* (acima do percentil 90).

5.1 Análise Estatística e Distribuição dos Dados

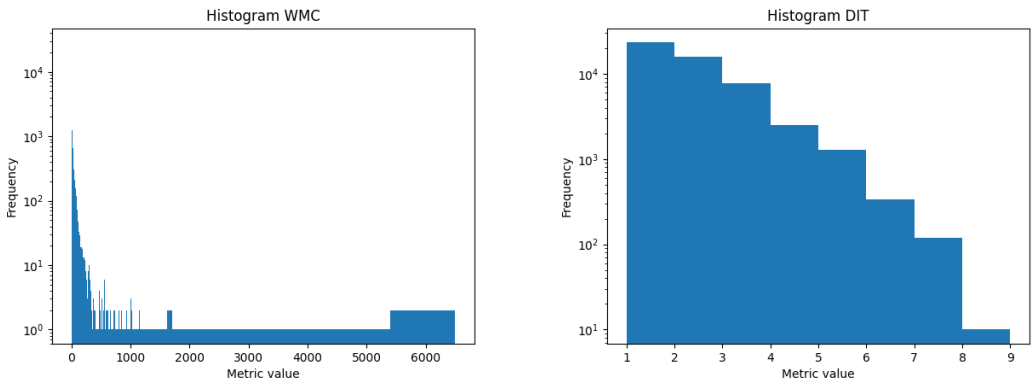
Inicialmente, buscou-se descrever a distribuição estatística dos dados coletados. Esta seção dedica-se a examinar o comportamento real dos dados obtidos, identificando padrões de distribuição que justificam as escolhas estatísticas subsequentes.

A análise dos histogramas gerados para cada métrica, apresentados nas Figuras 1 a 6, revela um padrão consistente a todo o ecossistema analisado: uma distribuição fortemente assimétrica positiva. Esse comportamento é corroborado pelos dados estatísticos descritivos, que, em geral, apresentam médias baixas e desvios-padrão elevados, evidenciando a dispersão causada pelos valores extremos.

Ao observar os gráficos, nota-se que a vasta maioria das classes se concentra no lado esquerdo do eixo X, apresentando valores baixos nas métricas de complexidade (WMC), acoplamento (CBO) e tamanho (LLOC). Em contrapartida, observa-se uma cauda longa à direita, que representa uma pequena quantidade de classes, os *outliers*, que acumulam valores extremamente elevados.

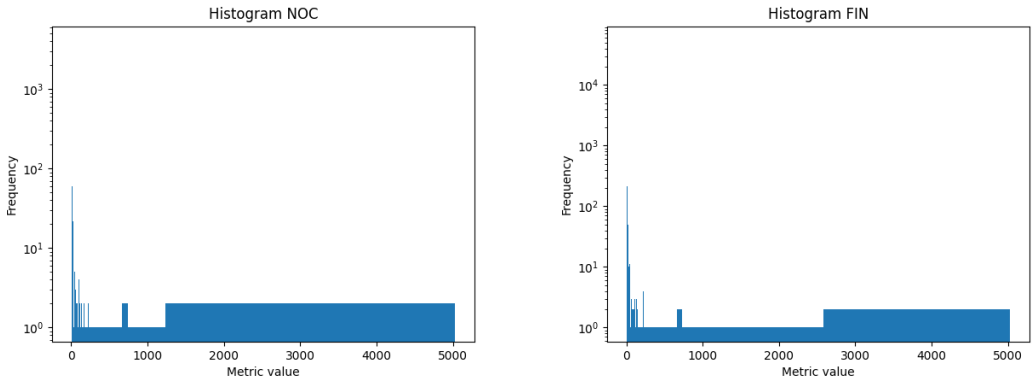
Tomando como exemplo o gráfico da métrica WMC na Figura 1, verifica-se visualmente que a maioria das classes em Python possui complexidade próxima de zero. Esse comportamento se repete tanto para métricas de acoplamento, como CBO (Figura 3), quanto para métricas de tamanho, como LLOC (Figura 5).

Figura 1 – Distribuição de frequência das métricas WMC e DIT.



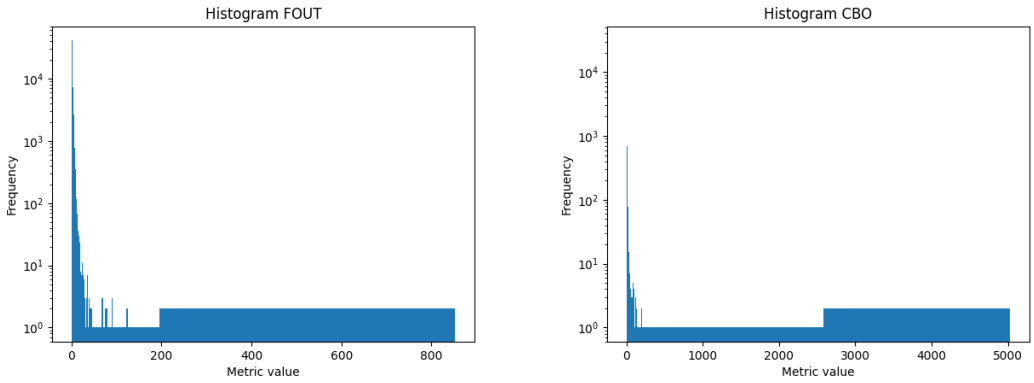
Fonte: Elaborado pelo autor (2026).

Figura 2 – Distribuição de frequência das métricas NOC e FIN.



Fonte: Elaborado pelo autor (2026).

Figura 3 – Distribuição de frequência das métricas FOUT e CBO.



Fonte: Elaborado pelo autor (2026).

Essa constatação reforça a adequação da metodologia de usar percentis (uma medida não paramétrica) para definir os limites de referência, em vez de média e desvio padrão, que seriam fortemente distorcidos pelos valores extremos.

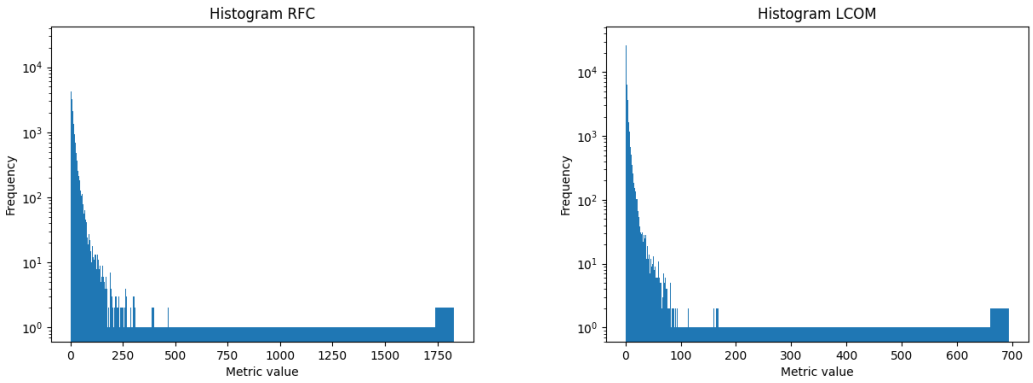
5.2 Derivação dos Percentis

Seguindo a metodologia de Gonçalves (2022), o conjunto consolidado de dados de todas as classes dos 45 projetos foi utilizado para calcular os percentis 70 (P70) e 90 (P90) de cada métrica. A Tabela 2 apresenta os valores obtidos.

A partir dos percentis definidos na Tabela 2, os valores de referência para as métricas de software Python podem ser categorizados em três faixas:

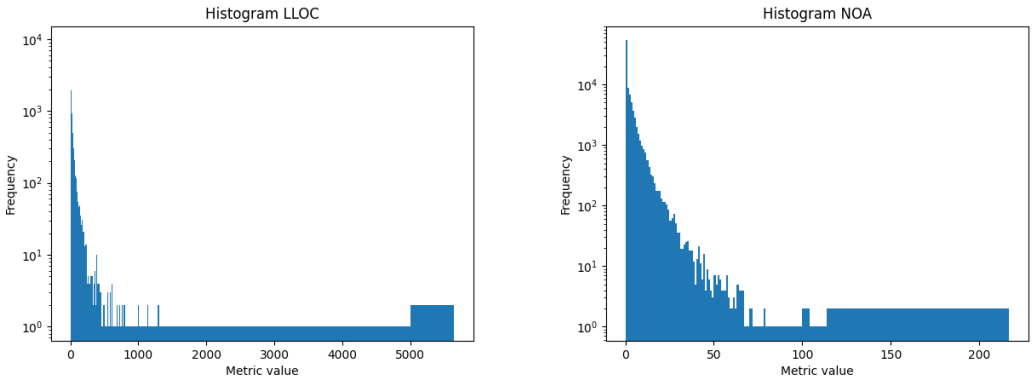
- Bom/Frequente: valores abaixo do percentil 70. Representam a faixa mais comum da distribuição e, em geral, indicam menor complexidade estrutural segundo a métrica analisada.
- Regular/Ocasional: valores entre o percentil 70 e o percentil 90. Indicam classes menos frequentes na distribuição e que podem merecer atenção durante revisões de código.

Figura 4 – Distribuição de frequência das métricas RFC e LCOM.



Fonte: Elaborado pelo autor (2026).

Figura 5 – Distribuição de frequência das métricas LLOC e NOA.



Fonte: Elaborado pelo autor (2026).

- Ruim/Raro: valores acima do percentil 90. Representam *outliers* e são fortes candidatos a classes com problemas de design, complexidade excessiva ou alta instabilidade, constituindo pontos de atenção para refatoração.

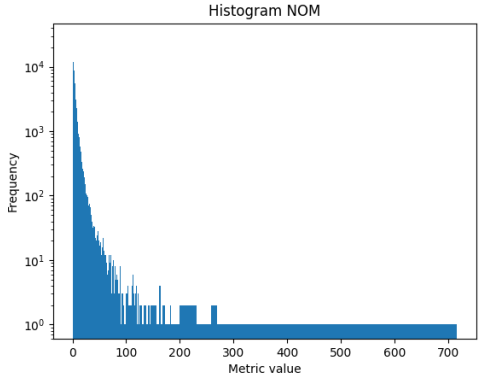
A Tabela 3 consolida esses intervalos para cada métrica, representando a principal contribuição quantitativa deste trabalho.

5.3 Comparação dos valores de referência em Python e Java

Para contextualizar os valores de referência obtidos para sistemas Python, esta seção estabelece uma comparação direta com os limiares previamente derivados para sistemas Java no trabalho de Gonçalves (2022).

É importante destacar que essa comparação possui caráter interpretativo, pois as ferramentas e os contextos analisados não são idênticos. Em especial, a comparação entre LOC, utilizada no estudo Java, e LLOC, utilizada neste trabalho, deve ser feita com cautela, uma vez que as métricas podem diferir quanto ao critério de contagem de linhas. Da mesma forma, a métrica de coesão não é diretamente comparável quando calculada por fórmulas ou ferramentas distintas.

Figura 6 – Distribuição de frequência da métrica NOM.



Fonte: Elaborado pelo autor (2026).

Tabela 2 – Percentis 70 e 90 por métrica.

Métrica	**P70**	**P90**
WMC	19	63
DIT	2	3
NOC	3	7
FIN	1	3
FOUT	2	4
CBO	3	6
RFC	11	27
LCOM	2	5
LLOC	23	71
NOA	2	6
NOM	3	9

Fonte: Elaborado pelo autor (2026).

A partir dos valores consolidados por Gonçalves (2022), foi possível recuperar os intervalos de referência em Java para as métricas também consideradas neste estudo, como WMC, DIT, NOC, FIN, FOUT, CBO, RFC, LOC/LLOC, LCOM e NOM. A Tabela 4 apresenta esses limiares por faixa de qualidade, seguindo a mesma convenção adotada para Python na Tabela 3.

Em primeiro lugar, observa-se um contraste relevante nas métricas de tamanho e complexidade por classe. Em Python, os limiares de referência para WMC (19 e 63) são consideravelmente superiores aos observados em Java (7 e 22). Esse resultado indica que, na amostra analisada, valores mais elevados de complexidade por classe são mais frequentes em projetos Python do que em projetos Java, fazendo com que uma classe Python precise atingir um WMC maior para ser enquadrada nas faixas “Regular” ou “Ruim”. No entanto, ao se observar o número de métodos (NOM) e o tamanho em linhas de código (LLOC/LOC), o comportamento se inverte: em Java, os limiares de NOM (5 e 12) e de LOC (36 e 114) são maiores do que em Python (3 e 9 para NOM; 23 e 71 para LLOC). Em conjunto, esses resultados sugerem uma tendência, para o conjunto de projetos avaliados,: em Java, há indícios de que seja mais comum encontrar classes maiores,

Tabela 3 – Faixas de referência por métrica a partir de P70 e P90

Métrica	**Bom**	**Regular**	**Ruim**
WMC	< 19	$19 \leq WMC < 63$	≥ 63
DIT	< 2	$2 \leq DIT < 3$	≥ 3
NOC	< 3	$3 \leq NOC < 7$	≥ 7
FIN	< 1	$1 \leq FIN < 3$	≥ 3
FOUT	< 2	$2 \leq FOUT < 4$	≥ 4
CBO	< 3	$3 \leq CBO < 6$	≥ 6
RFC	< 11	$11 \leq RFC < 27$	≥ 27
LCOM	< 2	$2 \leq LCOM < 5$	≥ 5
LLOC	< 23	$23 \leq LLOC < 71$	≥ 71
NOA	< 2	$2 \leq NOA < 6$	≥ 6
NOM	< 3	$3 \leq NOM < 9$	≥ 9

Fonte: Elaborado pelo autor (2026).

Tabela 4 – Faixas de referência por métrica em Java a partir de P70 e P90

Métrica	**Bom**	**Regular**	**Ruim**
WMC	< 7	$7 \leq WMC < 22$	≥ 22
DIT	< 2	$2 \leq DIT < 3$	≥ 3
NOC	< 4	$4 \leq NOC < 9$	≥ 9
FIN	< 2	$2 \leq FIN < 7$	≥ 7
FOUT	< 7	$7 \leq FOUT < 15$	≥ 15
CBO	< 10	$10 \leq CBO < 22$	≥ 22
RFC	< 9	$9 \leq RFC < 29$	≥ 29
LOC	< 36	$36 \leq LOC < 114$	≥ 114
LLCOM	$< 0,4$	$0,4 \leq LLCOM < 0,7917$	$\geq 0,7917$
NOM	< 5	$5 \leq NOM < 12$	≥ 12

Fonte: Elaborado por Gonçalves (2022).

com mais métodos e mais linhas de código, porém com métodos individualmente menos complexos; em Python, por outro lado, as classes da amostra apresentaram propensão a serem menores em tamanho absoluto, mas com métodos que concentram mais lógica interna, resultando em WMC mais elevados.

No que diz respeito à herança, os limiares de DIT são muito próximos entre as duas linguagens, indicando hierarquias de profundidade semelhantes para a maioria das classes. Já a métrica NOC apresenta valores ligeiramente superiores em Java (4 e 9) em comparação com Python (3 e 7), o que sugere que, na amostra de Java, é um pouco mais frequente que classes atuem como superclasses de um número maior de subclasses. Esse resultado é coerente com o uso mais intensivo de hierarquias de herança em *frameworks* e bibliotecas Java, enquanto, em Python, é comum adotar outras formas de composição e reutilização.

As métricas de acoplamento evidenciam diferenças ainda mais marcantes. Em Python, os limiares de FIN (1 e 3), FOUT (2 e 4) e CBO (3 e 6) permanecem em faixas

consideravelmente mais baixas do que em Java, em que FIN (2 e 7), FOUT (7 e 15) e CBO (10 e 22) assumem valores substancialmente superiores. Isso indica que, para a maior parte dos sistemas analisados, as classes Java tendem a manter dependências mais numerosas, tanto em termos de chamadas a outras classes (FOUT) quanto de dependências de entrada (FIN) e de acoplamento entre classes (CBO). Em outras palavras, os dados sugerem que as classes presentes nos sistemas Java da amostra tendem a ser mais fortemente conectadas entre si do que as equivalentes no contexto do Python. A métrica RFC mantém magnitudes próximas entre as linguagens (11/27 em Python e 9/29 em Java), o que sugere que, apesar das diferenças de acoplamento e tamanho, o número de métodos potencialmente acionáveis a partir de uma classe permanece em uma faixa semelhante. Por fim, a métrica LCOM não pode ser comparada, pois os cálculos foram realizados de forma diferente nos dois trabalhos.

5.4 Discussão dos Resultados

A definição dos valores de referência apresentada na Tabela 4 contribui para reduzir uma lacuna importante na avaliação da qualidade de software no ecossistema Python. A categorização em faixas Bom/Frequente, Regular/Ocasional e Ruim/Raro oferece aos desenvolvedores e gerentes de projeto um instrumento quantitativo de apoio à análise estrutural. No entanto, essas faixas devem ser interpretadas como referências empíricas baseadas na distribuição observada dos dados, e não como diagnósticos absolutos de qualidade. Valores elevados indicam que a classe apresenta características estatisticamente menos frequentes e potencialmente candidatas à inspeção, mas sua interpretação depende do contexto arquitetural e funcional do sistema.

Uma das contribuições mais relevantes é a constatação de que as heurísticas de qualidade importadas de outras linguagens, como Java, não são diretamente aplicáveis ao Python. Conforme observado na comparação entre as Tabelas 3 e 4, um desenvolvedor que utilizasse os limiares de Java para avaliar a complexidade (WMC) de um código Python poderia classificar erroneamente uma classe como “Ruim” (WMC > 22), quando, na realidade do ecossistema Python, ela estaria apenas no início da faixa “Regular”, na qual o limite inferior é 19.

Por outro lado, os baixos valores de acoplamento (CBO, FAN-OUT) observados no Python sugerem uma arquitetura em que as dependências são gerenciadas de forma distinta. Enquanto em Java é comum o uso extensivo de injeção de dependências e padrões que aumentam o acoplamento entre tipos explícitos, em Python, o baixo acoplamento entre classes pode indicar um uso mais frequente de módulos, funções soltas ou tipagem dinâmica. Portanto, valores de CBO que seriam considerados normais em Java (> 10) devem ser vistos como um sinal de alerta (Ruim) em Python, em que a referência é consideravelmente mais baixa (< 3).

6 AVALIAÇÃO DOS RESULTADOS

A definição de valores de referência para métricas de software tem como principal finalidade apoiar o programador na identificação de possíveis problemas estruturais no código. Nesse sentido, os valores apresentados neste trabalho não devem ser interpretados como uma classificação absoluta de qualidade, mas como um instrumento de apoio à análise. Classes enquadradas na faixa “Ruim/Raro” não necessariamente apresentam defeitos, mas

indicam pontos que merecem maior atenção durante atividades de revisão, manutenção e refatoração.

Para avaliar a aplicação prática dos valores obtidos, foi realizada uma inspeção em classes do projeto Requests, um dos sistemas Python analisados neste trabalho. A análise considerou principalmente métricas relacionadas à complexidade, tamanho, acoplamento e quantidade de responsabilidades internas, como WMC, CBO, RFC, LCOM, LLOC, NOA e NOM. Essas métricas foram escolhidas por estarem diretamente relacionadas à dificuldade de compreensão e manutenção de uma classe.

Inicialmente, observa-se que algumas classes apresentaram valores elevados em várias métricas ao mesmo tempo. Esse comportamento é importante porque, isoladamente, uma métrica alta pode representar apenas uma característica específica da implementação. No entanto, quando uma mesma classe apresenta simultaneamente altos valores de WMC, RFC, LLOC, NOA e NOM, há um indício mais forte de concentração de responsabilidades e maior custo de manutenção.

No projeto analisado, a classe `HTTPAdapter` apresentou um dos perfis mais críticos. Seus valores foram WMC igual a 130, RFC igual a 65, LLOC igual a 146, NOA igual a 24 e NOM igual a 15, além de CBO igual a 6 e FOUT igual a 4. Esses valores a colocam na faixa "Ruim/Raro" em diversas métricas. Ao observar sua implementação, percebe-se que a classe concentra responsabilidades relacionadas ao transporte HTTP, gerenciamento de conexões, configuração de *proxies*, verificação de certificados, construção de respostas e envio de requisições. Assim, os valores obtidos são coerentes com o código, pois a classe realmente exerce um papel central e acumula diferentes comportamentos. Nesse caso, a métrica indica que a classe é uma forte candidata à inspeção em processos de manutenção.

Situação semelhante ocorre com a classe `PreparedRequest`. Essa classe apresentou WMC igual a 128, RFC igual a 54, LLOC igual a 141, NOA igual a 13 e NOM igual a 13. Esses valores também a colocam em faixas críticas para complexidade, tamanho, número de atributos e número de métodos. Ao analisar sua função no sistema, percebe-se que ela é responsável por preparar diferentes partes de uma requisição HTTP, como método, URL, cabeçalhos, *cookies*, corpo da mensagem, autenticação e *hooks*. Portanto, os valores elevados condizem com a implementação, uma vez que a classe participa de várias etapas do ciclo de vida de uma requisição.

A classe `Session` também apresenta um perfil de atenção. Foram observados valores como WMC igual a 104, RFC igual a 56, LLOC igual a 124, NOA igual a 15, NOM igual a 19 e FOUT igual a 5. Esses resultados indicam uma classe extensa, com muitos métodos, muitos atributos e interação com diferentes componentes. No código, essa característica é coerente com seu papel: a classe `Session` gerencia configurações persistentes de requisições, como *cookies*, autenticação, *proxies*, cabeçalhos, redirecionamentos, adaptadores e envio de chamadas HTTP. Assim, os valores de referência evidenciam que essa classe possui grande importância estrutural e, por isso, alterações nela tendem a exigir maior cuidado.

A classe `Response` também apresentou valores elevados em métricas importantes, como WMC igual a 116, RFC igual a 46, LCOM igual a 5, LLOC igual a 139, NOA igual a 15 e NOM igual a 22. Esses valores sugerem que a classe concentra uma quantidade significativa de estado interno e comportamento. Isso é compatível com seu papel no sistema, pois uma resposta HTTP precisa armazenar informações como código de status, cabeçalhos, conteúdo, *cookies*, URL, histórico, codificação e outros dados associados ao retorno de uma requisição. Nesse caso, a classificação como "Ruim/Raro" deve ser interpretada com cautela: a classe é de fato grande e complexa, mas essa complexidade está relacionada à

relevância funcional que ela possui dentro da biblioteca.

Por outro lado, algumas classes apresentaram valores baixos e coerentes com uma boa separação de responsabilidades. A classe `AuthBase`, por exemplo, possui uma estrutura mínima, funcionando como uma base para classes de autenticação. Seus valores baixos de WMC, RFC, LLOC, NOA e NOM indicam que ela possui responsabilidade bem delimitada. De forma semelhante, a classe `BaseAdapter` atua como uma estrutura base para adaptadores de transporte, definindo poucos métodos e mantendo baixa complexidade. Esses casos mostram como os valores de referência conseguem representar classes simples, pequenas e com comportamento bem controlado.

Também foram observadas várias classes de exceção com valores de métricas muito baixos, como `HTTPError`, `URLRequired`, `TooManyRedirects`, `MissingSchema`, `InvalidSchema`, `InvalidHeader` e `RetryError`. Essas classes possuem pouca ou nenhuma lógica interna, funcionando principalmente como tipos específicos de erro. Por isso, seus baixos valores de complexidade, tamanho e número de métodos estão condizentes com o código. Nesse caso, a classificação como "Bom/Frequente" é coerente, pois essas classes possuem responsabilidade simples e bem definida.

Entretanto, a análise também mostra que algumas métricas precisam ser interpretadas de acordo com o contexto da classe. A classe `CaseInsensitiveDict`, por exemplo, apresentou valores elevados em métricas como CBO, LCOM e NOM. Em uma análise puramente numérica, isso poderia indicar um ponto crítico. No entanto, ao observar o código, percebe-se que a classe implementa o comportamento de um dicionário, com chaves insensíveis a maiúsculas e minúsculas, o que exige métodos específicos como inserção, busca, remoção, iteração, comparação e cópia. Assim, parte da elevação das métricas está relacionada à necessidade de implementar uma interface semelhante à de um dicionário, e não necessariamente a uma má organização estrutural.

Outro exemplo importante é a classe `RequestException`. Embora apresente baixa complexidade interna, ela possui valores elevados em NOC, FIN e CBO, pois é uma classe base para diversas exceções específicas. Nesse caso, o valor alto não indica excesso de responsabilidade interna, mas sim centralidade na hierarquia de exceções. Isso reforça que métricas de herança e acoplamento devem ser analisadas considerando o papel arquitetural da classe.

Com base nessa avaliação, observa-se que os valores de referência definidos neste trabalho são úteis para indicar classes que merecem atenção durante a manutenção. As classes com maior número de métricas na faixa "Ruim/Raro" geralmente correspondem a componentes centrais, extensos e com maior número de responsabilidades, como `HTTPAdapter`, `PreparedRequest`, `Session` e `Response`. Como indício de consistência dos intervalos propostos, identificou-se que as classes classificadas em faixas melhores tendem a apresentar comportamento mais simples, menor quantidade de métodos e menor estado interno, como `AuthBase`, `BaseAdapter` e várias classes de exceção.

Portanto, os resultados obtidos são condizentes com a proposta deste trabalho. As métricas não substituem a análise do programador, mas ajudam a direcionar seu esforço. Em vez de revisar o sistema inteiro de forma indiscriminada, o desenvolvedor pode utilizar os valores de referência para priorizar as classes estatisticamente atípicas, investigar se elas acumulam responsabilidades e decidir se há necessidade de refatoração. Dessa forma, os valores propostos cumprem seu papel como instrumento prático de apoio à avaliação da qualidade estrutural de sistemas Python.

7 LIMITAÇÕES

A realização deste estudo, embora tenha cumprido seus objetivos principais de estabelecer valores de referência para métricas de software em Python, apresenta limitações tanto nas ferramentas utilizadas quanto na metodologia de seleção e análise dos dados. Essas restrições devem ser consideradas na interpretação e no escopo de aplicação dos resultados obtidos:

- **Uso de Ferramenta Única e Maturidade do Extrator:** A coleta de dados dependeu exclusivamente de uma única ferramenta de extração, o PyCKTool. Embora o PyCKTool cumpra um papel importante por ser a única solução de código aberto voltada especificamente para capturar métricas puramente orientadas a objetos na linguagem Python, a ferramenta possui a limitação de ainda não ter sido amplamente utilizada ou validada em outros trabalhos da literatura. Além disso, a análise estática executada sobre uma linguagem dinâmica impõe desafios operacionais, podendo resultar em imprecisões na identificação de acoplamentos (CBO e RFC) quando os tipos são definidos em tempo de execução.

- **Tamanho e Vieses da Amostra:** O tamanho da amostra limitou-se a 45 projetos. Como a seleção foi baseada estritamente em critérios de popularidade (maior quantidade de estrelas no GitHub), a base de dados apresenta vieses de distribuição: os projetos de Inteligência Artificial dominam a amostra e os frameworks web aparecem em grande quantidade. Por outro lado, sistemas como bibliotecas científicas — que notoriamente possuem padrões arquiteturais e distribuições de classes distintos — não foram proporcionalmente representados, o que impede a generalização irrestrita dos limiares para o ecossistema Python global, softwares proprietários ou sistemas legados.

- **Falta de Análise Estatística de Representatividade:** O estudo carece de uma fundamentação estatística rigorosa para afirmar que 45 projetos são numericamente suficientes para representar a totalidade do desenvolvimento em Python. Os percentis encontrados atuam como uma descrição matemática da amostra selecionada, sem evidência matemática prévia de sua distribuição probabilística ideal no universo da linguagem.

- **Restrição da Avaliação Prática:** A avaliação empírica das faixas propostas restringiu-se ao cenário de um único estudo de caso por meio da inspeção manual das classes do projeto `Requests`. A ausência de uma análise estatística cruzada com múltiplos projetos externos limita a validação da sensibilidade dos limiares para outros contextos funcionais.

- **Ausência de Correlação com Defeitos Reais:** O estudo não realiza uma comparação direta ou mapeamento organizado das faixas de métricas com dados concretos de qualidade histórica, como a densidade de *bugs* ou a abertura de *issues* de falhas nos repositórios analisados. Não há, portanto, evidência estatística imediata neste estudo que correlacione diretamente os limiares numéricos à presença de falhas operacionais.

8 CONSIDERAÇÕES FINAIS

A garantia da qualidade de software é um desafio constante, tornando-se crítica com a expansão da linguagem Python em domínios estratégicos, como a Inteligência Artificial. Diante da inadequação da aplicação direta de parâmetros de linguagens como Java, este trabalho cumpriu o objetivo de preencher essa lacuna ao identificar valores de referência para métricas específicas de orientação a objetos no ecossistema Python. Para tanto, utilizou-se a ferramenta PyCKTool (Sabioni, 2025) para extrair dados de 45 repositórios de código aberto, aplicando-se a metodologia estatística de percentis proposta por Gonçalves (2022) para definir limiares objetivos de qualidade.

Os resultados obtidos permitiram classificar as métricas nas faixas Bom, Regular e Ruim, revelando contrastes significativos em relação aos padrões do Java, como a tolerância natural a uma maior complexidade interna (WMC) e a exigência de níveis de acoplamento (CBO) consideravelmente menores. Essas descobertas formalizam um catálogo de valores de referência que instrumentaliza a tomada de decisões de refatoração, além de validar a robustez da metodologia estatística para a distribuição de dados em Python. A disponibilização dos *scripts* e dados consolidados reforça a contribuição do estudo, fomentando a replicabilidade e a continuidade das pesquisas na área.

Apesar dos avanços, este trabalho aponta caminhos para investigações futuras. O modelo desenvolvido assume implicitamente que métricas excessivamente altas estão associadas a uma pior qualidade estrutural do código. Torna-se necessário, portanto, conduzir estudos empíricos que validem essa premissa por meio da correlação estatística direta entre as classes classificadas na faixa 'Ruim' e o histórico de defeitos reais (bugs) extraídos dos repositórios.

Adicionalmente, sugere-se analisar se há diferenças significativas nos valores de referência obtidos de acordo com o domínio de aplicação e o tipo de sistema (ferramenta, biblioteca, framework, aplicativo, entre outros). No caso da linguagem Java, Filó (2014) investigaram esse aspecto e concluíram que não existem variações significativas nesses fatores. Contudo, faz-se necessária uma verificação empírica para determinar se essa independência de domínio e tipo de software se mantém sob as características do ecossistema Python.

Por fim, recomenda-se a expansão do conjunto de métricas para capturar aspectos funcionais da linguagem e o desenvolvimento de *plugins* para ferramentas de Integração Contínua (CI) utilizando os limites gerados.

REFERÊNCIAS

Booch, G.; Engle, R.; Conallen, J.; Houston, K.; Young, B. **Object-Oriented Analysis and Design with Applications**. 3. ed. Boston: Pearson Education, 2011. Acesso em: 09 mar. 2025. Citado 2 vezes nas páginas 1 e 4.

Chidamber, S.; Kemerer, C. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, IEEE, v. 20, n. 6, p. 476–493, 1994. Citado 3 vezes nas páginas 1, 4 e 5.

FILÓ, T. G. S. Identificação de valores referência para métricas de softwares orientados por objetos. **Universidade Federal de Minas Gerais**, 2014. Citado 2 vezes nas páginas 8 e 20.

Gonçalves, D. **Derivação de Valores de Referência de Métricas de Software**. Monografia. Departamento de Computação, Centro Federal de Educação Tecnológica de Minas Gerais. Belo Horizonte: [s.n.], 2022. Citado 9 vezes nas páginas 2, 6, 7, 8, 12, 13, 14, 15 e 20.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 25010:2023: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model**. Geneva, 2023. Citado na página 3.

Mendes, H. **Caracterização da Qualidade Estrutural de Software Java para Inteligência Artificial**. Monografia. Departamento de Computação, Centro Federal de Educação Tecnológica de Minas Gerais. Belo Horizonte: [s.n.], 2024. Citado 2 vezes nas páginas 2 e 6.

Meyer, B. **Object-Oriented Software Construction**. [S.l.]: Prentice Hall, 1997. Citado 2 vezes nas páginas 3 e 4.

Moldovan, V.; Berciu, L.; Patcas, R. The python software quality dataset. In: **2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**. [S.l.: s.n.], 2024. p. 395–398. Citado na página 2.

Sabioni, A. **PyCKTool: Ferramenta para Coleta de Métricas de Software Orientado a Objetos em Código Python**. Monografia. Departamento de Computação, Centro Federal de Educação Tecnológica de Minas Gerais. Belo Horizonte: [s.n.], 2025. Citado 5 vezes nas páginas 2, 5, 6, 7 e 20.

Sommerville, I. **Engenharia de software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018. Acesso em: 09 mar. 2025. Citado na página 1.

APÊNDICE A – Softwares Analisados no Trabalho

Tabela 5 – Projetos Python no GitHub, ordenados pela quantidade de estrelas.

Nome do Projeto	**Link do repositório**	**Data de Criação**	**Observações**
AutoGPT	<https://github.com/Significant-Gravitas/AutoGPT>	2023-03-16	Plataforma de automação de GPT
Stable Diffusion web UI	<https://github.com/AUTOMATIC1111/stable-diffusion-webui>	2022-08-22	Interface web para geração de imagens por llm
Hugging Face	<https://github.com/huggingface/transformers>	2018-10-29	Biblioteca de aprendizado de máquina
Deepseek V3	<https://github.com/deepseek-ai/DeepSeek-V3>	2024-12-26	Modelo de linguagem grande (LLM) (deepsek)

Continua na próxima página

Nome do Projeto	**Link do repositório**	**Data de Criação**	**Observações**
The Fuck	<https://github.com/nvbn/thefuck>	2015-04-08	App de corrigir erros de console
Pytorch	<https://github.com/pytorch/pytorch>	2016-08-13	Biblioteca de deep learning
FastAPI	<https://github.com/fastapi/fastapi>	2018-12-08	*Framework* web para python
django	<https://github.com/django/django>	2012-04-28	*Framework* web para python
whisper (openAI)	<https://github.com/openai/whisper>	2022-09-16	Modelo de reconhecimento de fala (openAI)
ComfyUI	<https://github.com/comfyanonymous/ComfyUI>	2023-01-17	Interface visual para pipelines de IA
core	<https://github.com/home-assistant/core>	2013-09-18	Plataforma de automação residencial
manim	<https://github.com/3b1b/manim>	2015-03-22	Motor de animação
langflow	<https://github.com/langflow-ai/langflow>	2023-02-08	Ferramenta de construir agentes de IA (LLM)
screenshot-to-code	<https://github.com/abi/screenshot-to-code>	2023-11-14	IA de reproduzir screenshot em interface funcional
flask	<https://github.com/pallets/flask>	2010-04-06	*Framework* web para python
gpt_academic	<https://github.com/binary-husky/gpt_academic>	2023-03-20	Interfaces para chatGPT com foco academico
cpython	<https://github.com/python/cpython>	2017-02-10	Linguagem de programação Python
sherlock	<https://github.com/sherlock-project/sherlock>	2018-12-24	Busca de usuario em redes sociais
ansible	<https://github.com/ansible/ansible>	2012-03-07	Plataforma de automação de código
browser-use	<https://github.com/browser-use/browser-use>	2024-11-30	*Framework* de mapeamento de website
keras	<https://github.com/keras-team/keras>	2015-03-28	Interface para *deep learning*
scikit-learn	<https://github.com/scikit-learn/scikit-learn>	2010-08-17	Biblioteca para aprendizado de máquina
open-interpreter	<https://github.com/OpenInterpreter/open-interpreter>	2023-07-13	Interface de GPT (LLM)
markitdown	<https://github.com/microsoft/markitdown>	2024-12-13	Conversor de documentos

Continua na próxima página

Nome do Projeto	Link do repositório	Data de Criação	Observações
localstack	<https://github.com/localstack/localstack>	2016-10-25	*Framework* de desenvolvimento para emular serviços AWS
OpenHands	<https://github.com/All-Hands-AI/OpenHands>	2024-03-13	Plataforma para agentes de IA
llama	<https://github.com/meta-llama/llama>	2023-02-14	Modelo de linguagem grande (LLM)
ragflow	<https://github.com/infiniflow/ragflow>	2023-12-12	Motor de realidade aumentada com LLM
scrapy	<https://github.com/scrapy/scrapy>	2010-02-22	*Framework* para extração de dados de websites
MetaGPT	<https://github.com/FoundationAgents/MetaGPT>	2023-06-29	Interface multi agente para LLM
private-gpt	<https://github.com/zylon-ai/private-gpt>	2023-05-01	Interface de GPT privada e segura
face_recognition	<https://github.com/ageitgey/face_recognition>	2017-03-03	API python de reconhecimento facial
yolov5	<https://github.com/ultralytics/yolov5>	2020-05-18	Modelo computacional para detecção de imagens
openpilot	<https://github.com/commaai/openpilot>	2016-11-24	Sistema operacional de robótica
requests	<https://github.com/psf/requests>	2011-02-13	Biblioteca Python para requisições HTTP
hackingtool	<https://github.com/Z4nzu/hackingtool>	2020-01-11	Ferramenta tudo em um de testes de cibersegurança
rich	<https://github.com/Textualize/rich>	2019-11-10	Biblioteca python para textos mais customizados
PaddleOCR	<https://github.com/PaddlePaddle/PaddleOCR>	2020-04-26	Ferramenta OCR
pandas	<https://github.com/pandas-dev/pandas>	2010-08-24	Pacote Python para estruturar dados
sentry	<https://github.com/getsentry/sentry>	2010-08-30	Software para ajudar a detectar e corrigir erros de código
diagrams	<https://github.com/mingrammer/diagrams>	2020-02-11	Software para desenhar arquitetura de sistemas em nuvem
TTS	<https://github.com/coqui-ai/TTS>	2020-05-20	Ferramentas de Aprendizado profundo para Text-to-Speech (Texto-para-Fala)
airflow	<https://github.com/apache/airflow>	2015-04-13	Plataforma de automação de fluxos de trabalho

Continua na próxima página

Nome do Projeto	**Link do repositório**	**Data de Criação**	**Observações**
black	<https://github.com/psf/black>	2018-03-14	Formatador de código Python
streamlit	<https://github.com/streamlit/streamlit>	2019-08-24	Ferramenta de desenvolver aplicativos web a partir de *scripts* Python

Agradecimentos

Agradeço à minha orientadora, Prof.ª Kecia Aline Marques Ferreira, pela orientação, disponibilidade e pelas contribuições ao longo do desenvolvimento deste trabalho. Agradeço também à coordenação do curso e aos professores que fizeram parte da minha formação acadêmica, pelos ensinamentos e pelo apoio durante esta trajetória.

Aos meus familiares e amigos, agradeço pelo incentivo, pela paciência e pelo suporte nos momentos de dificuldade. Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO - NG

FOLHA DE APROVAÇÃO DE TCC Nº 22 / 2026 - DECOM (11.56.03)

Nº do Protocolo: 23062.033158/2026-84

Belo Horizonte-MG, 25 de junho de 2026.

MARCUS VINÍCIUS DINIZ RIBEIRO

IDENTIFICAÇÃO DE VALORES DE REFERÊNCIA DE MÉTRICAS DE SOFTWARE PYTHON

Trabalho de Conclusão de Curso apresentado ao Curso de **Engenharia de Computação** do Centro Federal de Educação Tecnológica de Minas Gerais, do Campus **Nova Gameleira**, como parte do requisito para obtenção do grau de Bacharel em **Engenharia de Computação**.

Aprovado em 17 de junho de 2026.

Banca Examinadora:

Profª. Drª. Kecia Aline Marques Ferreira - Orientadora
Centro Federal de Educação Tecnológica de Minas Gerais

Profª. Drª. Daniela Cristina Cascini Kupsch
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Edson Marchetti da Silva
Centro Federal de Educação Tecnológica de Minas Gerais

Belo Horizonte
2026

(Assinado digitalmente em 26/06/2026 14:23)
DANIELA CRISTINA CASCINI KUPSCH
PROFESSOR ENS BASICO TECN TECNOLOGICO
CECOM (11.51.11)
Matrícula: 2092029

(Assinado digitalmente em 26/06/2026 14:45)
EDSON MARCHETTI DA SILVA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1509224

(Assinado digitalmente em 26/06/2026 15:01)
KECIA ALINE MARQUES FERREIRA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1670931

Visualize o documento original em **https://sig.cefetmg.br/public/documentos/index.jsp** informando seu número: **22**, ano: **2026**, tipo: **FOLHA DE APROVAÇÃO DE TCC**, data de emissão: **25/06/2026** e o código de verificação: **ed811b38c4**