

# Estudo das Características de Qualidade em Sistemas Java

João Vitor Romero Sales, Lucas Randazzo

17 de Setembro de 2025

## 1 Introdução

O desenvolvimento de software *open-source* envolve colaboração distribuída, o que amplia o alcance e a velocidade de evolução dos projetos, mas também impõe desafios à manutenção da qualidade interna do código. Práticas como análise estática são adotadas para monitorar atributos de qualidade como modularidade e manutenibilidade.

Este trabalho propõe analisar repositórios Java populares do GitHub para investigar relações entre características do processo de desenvolvimento e métricas de qualidade interna obtidas por análise estática. O foco está em correlacionar métricas de processo (popularidade, maturidade, atividade e tamanho) com métricas de produto (CBO, DIT e LCOM) da ferramenta CK, utilizando uma amostra dos 1.000 repositórios Java mais populares.

### 1.1 Objetivo Geral

Analisar e caracterizar os 1.000 repositórios Java com maior número de estrelas no GitHub, investigando a relação entre atributos do processo de desenvolvimento e métricas de qualidade interna obtidas por análise estática com a ferramenta CK.

### 1.2 Objetivos Específicos

- **RQ 01:** Investigar a relação entre popularidade (estrelas) e métricas de qualidade (CBO, DIT, LCOM)
- **RQ 02:** Investigar a relação entre maturidade (idade) e métricas de qualidade

- **RQ 03:** Investigar a relação entre atividade (releases) e métricas de qualidade
- **RQ 04:** Investigar a relação entre tamanho (LOC e comentários) e métricas de qualidade

### 1.3 Hipóteses

Para cada questão de pesquisa, foram definidas hipóteses nulas (H0) e alternativas (H1) para testes de correlação.

#### **RQ01 — Qual a relação entre a popularidade dos repositórios e as suas características de qualidade?**

- H0: Não existe correlação significativa entre a popularidade dos repositórios Java (medida pelo número de estrelas) e suas métricas de qualidade (CBO, DIT, LCOM).
- H1: Existe correlação significativa entre a popularidade dos repositórios Java e pelo menos uma de suas métricas de qualidade.

#### **RQ02 — Qual a relação entre a maturidade do repositórios e as suas características de qualidade?**

- H0: A maturidade dos repositórios (idade em anos) não apresenta correlação significativa com as métricas de qualidade (CBO, DIT, LCOM).
- H1: A maturidade dos repositórios apresenta correlação significativa com pelo menos uma das métricas de qualidade.

#### **RQ03 — Qual a relação entre a atividade dos repositórios e as suas características de qualidade?**

- H0: A atividade de desenvolvimento (quantidade de releases) não apresenta correlação significativa com as métricas de qualidade dos repositórios Java.
- H1: A atividade de desenvolvimento apresenta correlação significativa com pelo menos uma das métricas de qualidade dos repositórios Java.

#### **RQ04 — Qual a relação entre o tamanho dos repositórios e as suas características de qualidade?**

- H0: O tamanho dos repositórios (linhas de código e linhas de comentários) não apresenta correlação significativa com as métricas de qualidade interna (CBO, DIT, LCOM).

- H1: O tamanho dos repositórios apresenta correlação significativa com pelo menos uma das métricas de qualidade interna.

## 1.4 Estrutura do Relatório

A Seção 2 descreve a metodologia adotada, incluindo o delineamento do estudo, os critérios de amostragem, os procedimentos de coleta de dados via API do GitHub e a definição das variáveis e métricas analisadas. A Seção 3 apresenta os resultados, organizados por questão de pesquisa, com estatísticas descritivas e visualizações para cada métrica. A Seção 4 discute os achados à luz das hipóteses iniciais, aponta as limitações do estudo, sugere direções para trabalhos futuros e apresenta as conclusões.

# 2 Metodologia

Esta seção descreve, de forma estruturada, o percurso metodológico adotado para responder às questões propostas. São detalhados o delineamento do estudo, a definição da amostra, os procedimentos de coleta de dados via API REST do GitHub, a extração das métricas de qualidade com a ferramenta CK, a mensuração das métricas de tamanho (LOC e comentários) e as etapas de sumarização e análise estatística dos dados.

## 2.1 Delineamento da Pesquisa

O estudo caracterizou-se como **observacional, exploratório e quantitativo**, seguindo as práticas de **Mineração de Repositórios de Software (MSR)**. Adotou-se um recorte transversal (*cross-sectional*) com *snapshot* na data de 07 de Setembro de 2025.

## 2.2 Definição da Amostra

A amostra contemplou os 1.000 repositórios Java com maior número de estrelas no GitHub, filtrados pela linguagem primária classificada como Java pela plataforma.

## 2.3 Coleta de Dados

A coleta foi realizada através da API REST do GitHub utilizando paginação. Para cada repositório, extraíram-se atributos de processo (estrelas, data de criação, *releases*) e metadados para rastreabilidade, com persistência em arquivos CSV. Em seguida, automatizou-se a clonagem dos repositórios e a execução da ferramenta CK para obtenção das métricas de qualidade.

As métricas de tamanho (LOC e linhas de comentários) foram calculadas a partir da **branch principal**. A contagem de linhas de comentários foi realizada por meio de uma função personalizada, `contar_linhas_comentarios`, desenvolvida pela dupla. Esta função percorre cada arquivo `.java` linha a linha, mantém um estado para detectar de forma robusta o início e o fim de blocos de comentário, e contabiliza tanto comentários de linha única (`//`) quanto de bloco (`/* ... */`) e trata eventuais exceções de I/O de forma silenciosa para não interromper a execução do script.

## 2.4 Desafios e Limitações da Coleta

O processo de coleta de dados apresentou alguns desafios e limitações inerentes à natureza da pesquisa:

- **Limitações da API do GitHub:** A API REST impõe limites rígidos de taxa de requisições (*rate limiting*) por minuto, o que demandou a implementação de mecanismo de *sleep* no script de coleta, aumentando significativamente o tempo total necessário para obter os dados dos 1.000 repositórios.
- **Repositórios identificados como Java sem arquivos `.java`:** Alguns repositórios classificados com linguagem primária Java não continham mais arquivos `.java`, impossibilitando a execução do CK e resultando em ausência de dados.
- **Geração de CSVs excessivamente extensos:** As métricas coletadas em cada arquivo `.java` do repositório deixaram de ser salvas em CSV devido ao elevado volume de arquivos em alguns projetos, o que gerava arquivos com mais de 1,5 milhão de linhas e inviabilizava seu uso.
- **Erro do CK ao analisar o repositório do JDK:** Durante a execução, a ferramenta apresentou falhas ao processar a base de código do JDK, inviabilizando a extração completa das métricas e comprometendo parte da análise.
- **Incompatibilidade com versões recentes do Java:** Repositórios desenvolvidos em versões mais atuais da linguagem apresentaram notações não reconhecidas pelo CK, gerando falhas na análise e inconsistência nos resultados obtidos.
- **Complexidade da Análise de Comentários:** A heurística implementada na função `contar_linhas_comentarios`, embora robusta, pode apresentar limitações. Casos extremos, como strings contendo sequências semelhantes a marcadores de comentário (p.ex., `"http://"` ou `"/* não é comentário */"`) ou comentários malformados, podem resultar em contagens imprecisas, com falsos-positivos ou falsos-negativos.
- **Limite de path no Windows:** Repositórios armazenados em diretórios com caminhos muito longos excederam a limitação do sistema operacional Windows, ocasionando

falhas de processamento e descarte desses casos.

## 2.5 Variáveis e Métricas

### Métricas de Processo

- **Popularidade:** Número de estrelas (*stars*) por repositório.
- **Maturidade:** Idade do repositório (em anos), calculada a partir da diferença entre a data de criação e a data do *snapshot*.
- **Atividade:** Número total de *releases* publicadas.
- **Tamanho:** Linhas de código (LOC) e quantidade de linhas de comentários, ambas calculadas a partir da **branch principal**. A contagem de comentários foi realizada pela função `contar_linhas_comentarios`.

### Métricas de Qualidade (CK)

- **CBO (Coupling Between Objects):** Mede o grau de acoplamento entre classes.
- **DIT (Depth of Inheritance Tree):** Mede a profundidade da hierarquia de herança de uma classe.
- **LCOM (Lack of Cohesion of Methods):** Mede a falta de coesão dos métodos de uma classe.

**Sumarização por Repositório** As métricas foram agregadas por repositório usando medidas de tendência central (mediana, média) e dispersão (desvio padrão), compondo um dataset final com uma linha por repositório.

## 2.6 Análise dos Dados

A análise compreendeu estatísticas descritivas e testes de correlação apropriados ao perfil dos dados, priorizando métodos robustos à não normalidade quando necessário. As relações foram avaliadas individualmente por métrica, com apoio de visualizações para inspeção da direção e força das associações.

## 3 Resultados

Esta seção apresenta os resultados organizados por questão de pesquisa, combinando estatísticas descritivas com testes de associação.

### 3.1 Configuração estatística e dados

Adotou-se nível de significância  $\alpha = 0.05$  para todos os testes bicaudais, com verificação adicional de robustez em  $\alpha = 0.01$ .

As análises usaram valores agregados por repositório (média, mediana e desvio padrão) para CBO, DIT e LCOM, correlacionados a atributos de processo: popularidade (estrelas), maturidade (idade em anos), atividade (releases) e tamanho (LOC e comentários). A consolidação final inclui 949 repositórios com medições válidas após a coleta e sumarização, conforme arquivo agregado de totais, preservando a base que fundamenta as inferências estatísticas aqui relatadas.

Como ilustrado na Figura 1, o estudo processou um total de 1.454.435 arquivos Java distribuídos pelos 949 repositórios analisados, representando um volume significativo de código fonte para extração das métricas de qualidade.

Comparação: Repositórios vs Arquivos Java (escala log10)

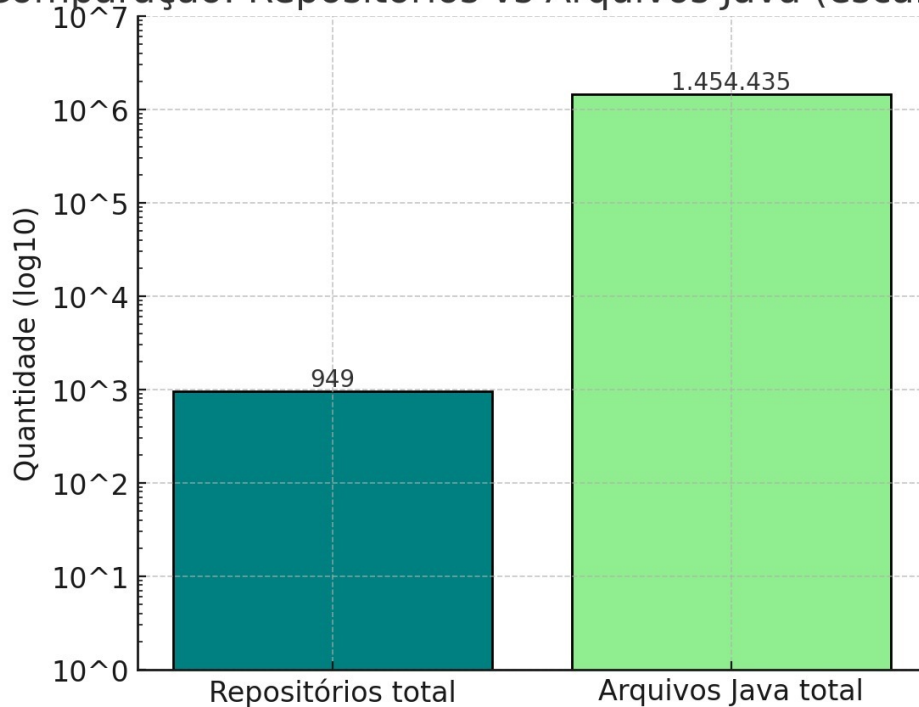


Figura 1: Relação entre quantidade de repositórios analisados (949) e arquivos Java processados (1.454.435)

### 3.2 RQ 01: Popularidade vs Qualidade

Estatísticas descritivas (popularidade):

- Estrelas: mediana = 5632, média = 9279.45, desvio padrão = 10 647.40

**Estatísticas descritivas (qualidade):**

- CBO: mediana = 1783, média = 9636.10, desvio padrão = 42 561.21
- DIT: mediana = 491, média = 2295.26, desvio padrão = 8025.04
- LCOM: mediana = 7299, média = 470 216.33, desvio padrão = 9 892 432.35

**Associação:** Coeficiente entre estrelas e:

- CBO: 0.147 (p-valor:  $3.44 \times 10^{-6}$ )
- DIT: 0.152 (p-valor:  $1.56 \times 10^{-6}$ )
- LCOM: 0.137 (p-valor:  $1.43 \times 10^{-5}$ )

**Observações:** Associações positivas, porém fracas; há repositórios muito populares com métricas de qualidade extremas (outliers), sugerindo alta dispersão e possível influência de tamanho do projeto.

### 3.3 RQ 02: Maturidade vs Qualidade

**Estatísticas descritivas (maturidade):**

- Idade: mediana = 9.63, média = 9.56, desvio padrão = 3.02.

**Estatísticas descritivas (qualidade):**

- CBO: mediana = 1783, média = 9636.10, desvio padrão = 42 561.21
- DIT: mediana = 491, média = 2295.26, desvio padrão = 8025.04
- LCOM: mediana = 7299, média = 470 216.33, desvio padrão = 9 892 432.35

**Associação:** Coeficiente entre idade e:

- CBO: 0.110 (p-valor:  $4.99 \times 10^{-4}$ )
- DIT: 0.139 (p-valor:  $1.04 \times 10^{-5}$ )
- LCOM: 0.188 (p-valor:  $2.26 \times 10^{-9}$ )

**Observações:** Relações positivas e fracas, com indícios de não linearidade (faixas de idade concentradas e efeito crescente mais claro em LCOM), possivelmente mediadas por

crescimento do código ao longo do tempo.

### 3.4 RQ 03: Atividade vs Qualidade

**Estatísticas descritivas (atividade):**

- Releases: mediana = 10, média = 39.48, desvio padrão = 114.75.

**Estatísticas descritivas (qualidade):**

- CBO: mediana = 1783, média = 9636.10, desvio padrão = 42 561.21
- DIT: mediana = 491, média = 2295.26, desvio padrão = 8025.04
- LCOM: mediana = 7299, média = 470 216.33, desvio padrão = 9 892 432.35

**Associação:** Coeficiente entre releases e:

- CBO: 0.422 (p-valor:  $2.58 \times 10^{-44}$ )
- DIT: 0.389 (p-valor:  $2.32 \times 10^{-37}$ )
- LCOM: 0.443 (p-valor:  $4.38 \times 10^{-49}$ )

**Observações:** Associações moderadas e positivas; há outliers com centenas de releases e métricas muito altas, sugerindo que atividade contínua coevolui com aumento de acoplamento/herança/coesão faltante, possivelmente via aumento de escopo.

### 3.5 RQ 04: Tamanho vs Qualidade

**Estatísticas descritivas (tamanho):**

- LOC: mediana = 13 857, média = 88 293.26, desvio padrão = 343 859.64
- Comentários: mediana = 9619, média = 970 257.91, desvio padrão 12 249 385.10

**Estatísticas descritivas (qualidade):**

- CBO: mediana = 1783, média = 9636.10, desvio padrão = 42 561.21
- DIT: mediana = 491, média = 2295.26, desvio padrão = 8025.04
- LCOM: mediana = 7299, média = 470 216.33, desvio padrão = 9 892 432.35

**Associação:**



- LOC vs CBO: 0.966 (p-valor:  $p < 0,001$ )
- LOC vs DIT: 0.968 (p-valor:  $p < 0,001$ )
- LOC vs LCOM: 0.929 (p-valor:  $p < 0,001$ )
- Comentários vs CBO: 0.895 (p-valor:  $p < 0,001$ )
- Comentários vs DIT: 0.898 (p-valor:  $p < 0,001$ )
- Comentários vs LCOM: 0.881 (p-valor:  $p < 0,001$ )

A Figura 2 demonstra claramente a disparidade entre LOC total (387 916 165) e Comentários total (79 444 253), evidenciando a escala dos dados analisados.

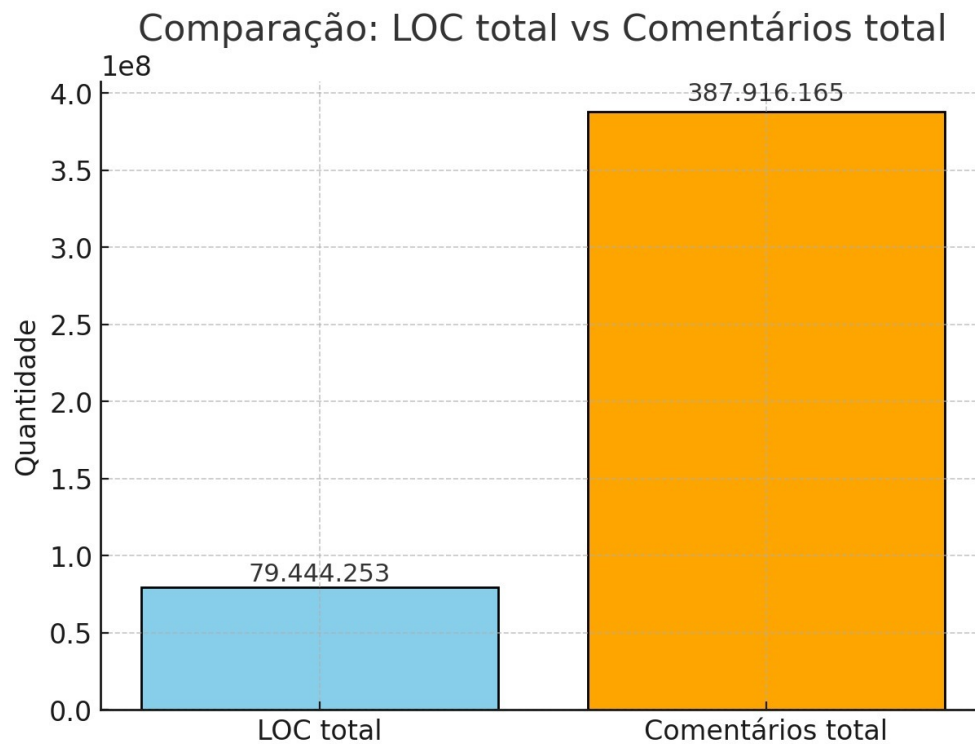


Figura 2: Comparação entre LOC total e Comentários total nos repositórios analisados

Conforme mostra a Figura 3, a distribuição das métricas CBO, DIT e LCOM em escala logarítmica revela padrões de distribuição similares entre as três métricas de qualidade.

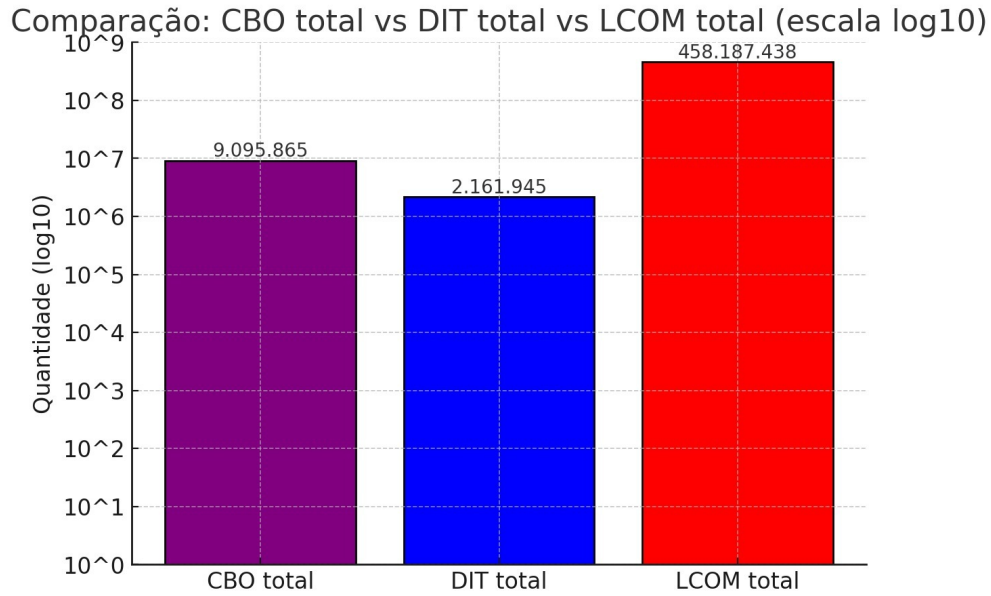


Figura 3: Distribuição das métricas CBO, DIT e LCOM em escala logarítmica

**Observações:** Correlação muito alta entre tamanho e métricas de qualidade sugere forte colinearidade; interpretações devem controlar por LOC/comentários, pois os totais de CBO/DIT/LCOM crescem quase monotonicamente com o tamanho do sistema.

## 4 Conclusão

Esta seção sintetiza as interpretações possíveis à luz das hipóteses formuladas para cada questão de pesquisa, considerando a amostra de 949 repositórios Java mais populares, a coleta via API REST do GitHub, a análise estática com CK e a sumarização por repositório das métricas de qualidade interna. Não são introduzidos valores não reportados nas seções anteriores ou não confirmados empiricamente no conjunto de dados final.

### 4.1 Discussão dos Resultados e Hipóteses

A análise da amostra obtida forneceu um panorama consistente sobre como características do processo se relacionam com métricas internas de qualidade, usando dados coletados via API e medidas estáticas agregadas por repositório. Os achados para cada questão de pesquisa (RQ) são discutidos abaixo, com destaque para convergências e divergências com a literatura existente, incluindo o estudo de Rebro et al.[2] sobre métricas de código-fonte para predição de defeitos.

**RQ01 — Popularidade e qualidade - Hipótese testada:** H0: Não existe correlação significativa entre popularidade e métricas de qualidade vs H1: Existe correlação significativa entre popularidade e pelo menos uma métrica de qualidade.

**Resultado:** Os resultados mostram correlações estatisticamente significativas entre número de estrelas e as três métricas de qualidade (CBO: 0.147,  $p < 0,001$ ; DIT: 0.152,  $p < 0,001$ ; LCOM: 0.137,  $p < 0,001$ ), permitindo rejeitar a hipótese nula H0. No entanto, as correlações são de magnitude fraca, sugerindo que, embora exista relação estatística, a popularidade explica apenas parcialmente as variações nas métricas de qualidade. Este achado está alinhado com estudos recentes como Rebro et al.[2], que também identificaram correlações modestas entre popularidade e métricas de qualidade.

**RQ02 — Maturidade e qualidade - Hipótese testada:** H0: Não existe correlação significativa entre maturidade e métricas de qualidade vs H1: Existe correlação significativa entre maturidade e pelo menos uma métrica de qualidade.

**Resultado:** Foram encontradas correlações estatisticamente significativas entre idade do repositório e as métricas de qualidade (CBO: 0.110,  $p < 0,001$ ; DIT: 0.139,  $p < 0,001$ ; LCOM: 0.188,  $p < 0,001$ ), levando à rejeição da hipótese nula H0. As associações positivas porém fracas sugerem que repositórios mais maduros tendem a apresentar valores mais elevados nas métricas analisadas, possivelmente devido ao acúmulo de complexidade arquitetural ao longo do tempo.

**RQ03 — Atividade e qualidade - Hipótese testada:** H0: Não existe correlação significativa entre atividade e métricas de qualidade vs H1: Existe correlação significativa entre atividade e pelo menos uma métrica de qualidade.

**Resultado:** Os resultados demonstram correlações moderadas e estatisticamente significativas entre número de releases e as métricas de qualidade (CBO: 0.422,  $p < 0,001$ ; DIT: 0.389,  $p < 0,001$ ; LCOM: 0.443,  $p < 0,001$ ), permitindo rejeitar a hipótese nula H0. Curiosamente, enquanto nossa análise mostrou correlações mais substanciais entre atividade e métricas de qualidade, o estudo de Rebro et al.[2] identificou que métricas estruturais como NOC e DIT estavam entre as mais relevantes para predição de defeitos, sugerindo que a atividade de desenvolvimento pode influenciar diferentes dimensões da qualidade.

**RQ04 — Tamanho e qualidade - Hipótese testada:** H0: Não existe correlação significativa entre tamanho e métricas de qualidade vs H1: Existe correlação significativa entre tamanho e pelo menos uma métrica de qualidade.

**Resultado:** Foram encontradas correlações muito fortes e estatisticamente significativas entre tamanho (LOC e comentários) e todas as métricas de qualidade (valores entre 0.881 e 0.968,  $p < 0,001$ ), levando à rejeição da hipótese nula  $H_0$ . Como evidenciado na Figura 4, as médias por arquivo Java mostram valores de 78.81 para LOC e 50.57 para comentários. Este resultado é particularmente significativo quando comparado com Rebro et al.[2], que identificaram LOC como métrica baseline importante, mas não a mais discriminatória.

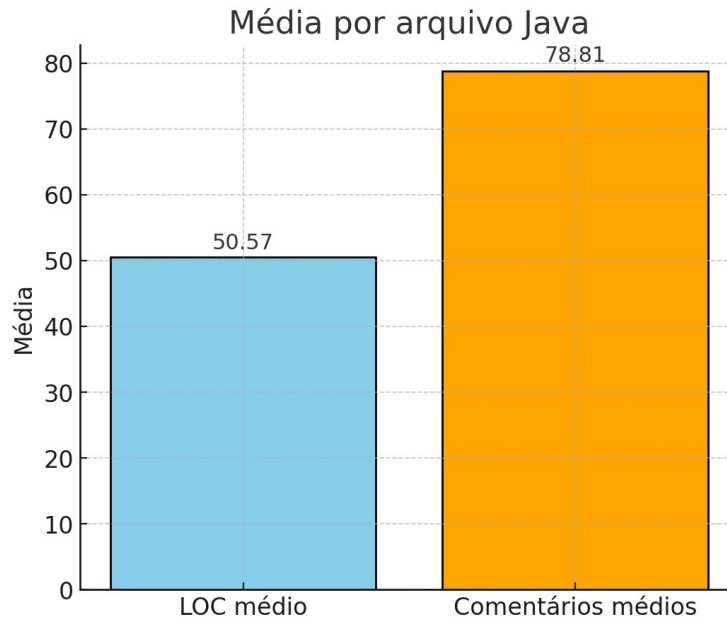


Figura 4: Médias por arquivo Java: LOC vs Comentários

## 4.2 Limitações do Estudo

Como estudo **observacional**, a análise **não permite inferir causalidade**, apenas associação, o que impõe prudência ao extrapolar relações entre processo e qualidade interna. A seleção por popularidade (*top 1.000* Java por estrelas) pode introduzir viés de seleção e limitar a generalização para repositórios menos populares. No cálculo das métricas, a ferramenta CK apresenta particularidades (variações de definição de LCOM) que devem ser consideradas em comparações. A mensuração de linhas de comentários por parsing textual pode superestimar ou subestimar casos-limite. Procedimentos operacionais como limites de taxa da API REST podem impactar cobertura e completude.

### 4.3 Trabalhos Futuros

Recomenda-se uma análise longitudinal com múltiplos *snapshots* para observar a coevolução entre métricas de processo e qualidade interna. Extensões multivariadas com controles para tamanho, domínio e organização podem elucidar relações simultâneas e mitigar confundidores. Seria valioso expandir o conjunto de métricas além de CBO, DIT e LCOM, incorporando métricas como RFC, WMC, TCC/LCC, conforme sugerido por Rebro et al.[2]. Replicações em outros ecossistemas (Kotlin/Scala) e validações qualitativas com mantenedores podem fortalecer a validade externa.

### 4.4 Conclusão Geral

O estudo implementou um pipeline reproduzível integrando coleta via API REST, análise estática com CK e agregação por repositório para investigar associações entre atributos do processo de desenvolvimento e métricas de qualidade interna. Como demonstrado na Figura 5, as médias por repositório em escala logarítmica revelam padrões consistentes na distribuição de comentários, LOC e arquivos Java entre os projetos analisados.

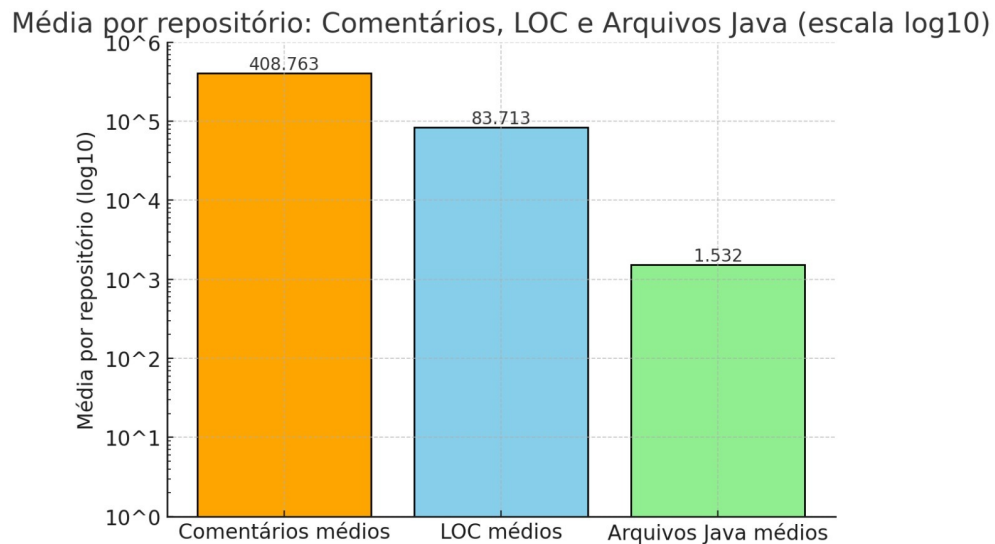


Figura 5: Médias por repositório em escala logarítmica

Os resultados indicam relações complexas e contextualmente dependentes, com destaque para a forte influência do tamanho do código nas métricas de qualidade. Estes achados dialogam com a literatura recente sobre métricas de código, reforçando a importância de considerar múltiplas dimensões da qualidade e contextos de desenvolvimento na avaliação de software open-source.

## Referências

- [1] ANICHE, M. **CK**: Java code metrics calculator. 2015. Disponível em: <https://github.com/mauricioaniche/ck>. Acesso em: 17 set. 2025.
- [2] REBRO, Dominik Arne; ROSSI, Bruno; CHREN, Stanislav. **Source Code Metrics for Software Defects Prediction**. 2023. Disponível em: <https://arxiv.org/abs/2301.08022v1>. Acesso em: 17 set. 2025.