
# Pontifícia Universidade Católica de Minas Gerais

*Campus* Coração Eucarístico
Engenharia de Software

Trabalho apresentado à disciplina de
**Laboratório de Experimentação de Software**
Prof. Danilo de Quadros Maia Filho Danilo de Quadros Maia Filho

Contagem – MG

# Sumário

# 1 Introdução

O desenvolvimento de software de código aberto (*open-source*) transformou a maneira como a tecnologia é criada, promovendo um ambiente colaborativo onde desenvolvedores de todo o mundo podem contribuir para um projeto comum. No entanto, essa natureza distribuída e muitas vezes assíncrona impõe desafios significativos à gestão da qualidade interna do código. Atributos essenciais como manutenibilidade, modularidade e coesão podem se deteriorar ao longo do tempo se não houver práticas de engenharia de software bem definidas, como revisões de código rigorosas e análise estática contínua.

Neste contexto, este estudo busca investigar empiricamente a possível correlação entre as características observáveis do processo de desenvolvimento de um projeto e a qualidade intrínseca de seu código-fonte. O objetivo principal é analisar se métricas de processo — como popularidade, idade, atividade e tamanho de um repositório — podem servir como indicadores para métricas de qualidade de produto, como acoplamento (CBO), profundidade de herança (DIT) e coesão (LCOM). Para isso, realizei uma análise em larga escala sobre os 1.000 repositórios da linguagem Java mais populares na plataforma GitHub.

Para guiar esta investigação, o trabalho se propõe a responder às seguintes quatro questões de pesquisa (RQs):

- **RQ 01:** Qual a relação entre a popularidade dos repositórios e as suas características de qualidade?
- **RQ 02:** Qual a relação entre a maturidade dos repositórios e as suas características de qualidade?
- **RQ 03:** Qual a relação entre a atividade dos repositórios e as suas características de qualidade?
- **RQ 04:** Qual a relação entre o tamanho dos repositórios e as suas características de qualidade?

Com base na experiência prática e em percepções comuns da comunidade de desenvolvimento, formulei as seguintes hipóteses iniciais para cada questão, que serão validadas ou refutadas por meio da análise de dados:

- **Hipótese para RQ01:** Acredito que a **popularidade e a qualidade são diretamente proporcionais**. Repositórios mais populares (com mais estrelas) tendem a atrair uma comunidade maior e mais qualificada, o que levaria a um maior escrutínio e a mais contribuições para a melhoria da qualidade do código.
- **Hipótese para RQ02:** Hipotetizo que **repositórios mais maduros apresentam melhor qualidade**. Projetos com mais tempo de existência teriam passado por mais ciclos de refatoração e aperfeiçoamento, resultando em um código mais estável e bem estruturado em comparação com projetos mais novos.
- **Hipótese para RQ03:** Minha suposição é que **repositórios mais ativos possuem uma qualidade inferior**. A alta frequência de releases e mudanças pode indicar um estado de evolução constante, com a introdução de *hotfixes* e novas funcionalidades que podem degradar temporariamente a qualidade. Repositórios menos ativos, por outro lado, seriam considerados mais consolidados e estáveis.

- **Hipótese para RQ04:** Por fim, acredito que **quanto maior o repositório, menor será sua qualidade**. O aumento da base de código tende a elevar a complexidade e o acúmulo de débito técnico, o que se refletiria negativamente nas métricas de qualidade.

# 2 Metodologia

Para responder às questões de pesquisa propostas, foi executado um processo metodológico em três etapas principais: (1) seleção dos repositórios e coleta de métricas de processo via API do GitHub; (2) extração de métricas de qualidade do código-fonte utilizando a ferramenta CK; e (3) tratamento e análise estatística dos dados coletados. Todo o processo foi automatizado por meio de scripts em Python.

## 2.1 Seleção dos Repositórios e Coleta de Métricas de Processo

O objeto de estudo deste trabalho foram os repositórios de código aberto da linguagem Java. A primeira etapa consistiu na seleção de uma amostra representativa de projetos relevantes da comunidade. Para isso, foi desenvolvido um script (conforme `main.py`) que utiliza a API GraphQL do GitHub para requisitar a lista dos 1.000 repositórios Java públicos com o maior número de estrelas (*stars*).

Para cada repositório da lista obtida, o mesmo script realizou uma segunda consulta à API para extrair as seguintes métricas de processo:

- **Popularidade:** O número total de estrelas (`stargazerCount`).
- **Maturidade:** A idade do repositório em anos, calculada a partir da sua data de criação (`createdAt`).
- **Atividade:** O número total de *releases* (`releases.totalCount`) publicadas no repositório.

Ao final desta etapa, os dados de processo foram salvos em um arquivo CSV (`java_repo_metrics.csv`) para uso posterior.

## 2.2 Coleta de Métricas de Qualidade de Código

Com a lista de repositórios definida, a etapa seguinte focou na extração de métricas de qualidade de produto. Um script de automação foi utilizado para clonar localmente o código-fonte de cada um dos 1.000 repositórios.

Em seguida, a ferramenta de análise estática de linha de comando **CK** foi executada sobre o código-fonte de cada projeto. O CK analisa o código Java e calcula um conjunto de métricas de software orientadas a objetos, gerando um arquivo de saída em formato `.csv` para cada repositório. As métricas de qualidade selecionadas para este estudo, conforme o escopo do laboratório, foram:

- **CBO (Coupling Between Objects):** Mede o nível de acoplamento de uma classe com outras.
- **DIT (Depth of Inheritance Tree):** Indica a profundidade de uma classe na árvore de herança.
- **LCOM (Lack of Cohesion of Methods):** Mede a falta de coesão dos métodos dentro de uma classe.
- **LOC (Lines of Code):** A contagem de linhas de código, utilizada como métrica de tamanho.

## 2.3 Tratamento e Análise dos Dados

A saída da ferramenta CK gera métricas para cada classe individualmente. Para possibilitar a comparação entre repositórios, foi necessário agregar esses dados. Primeiramente, um script (`consolidate_results.py`) consolidou todos os arquivos `.csv` individuais em um único conjunto de dados (`consolidated_metrics.csv`).

Na sequência, o script de análise (`analyze_results.py`) processou este conjunto de dados. Para cada repositório, foi calculada a **mediana** de cada métrica de qualidade (CBO, DIT, LCOM e LOC). A mediana foi escolhida como medida de tendência central por sua robustez estatística, pois ela não é significativamente afetada por valores extremos (*outliers*). Em um repositório, algumas poucas classes muito grandes ou complexas poderiam distorcer a média, enquanto a mediana oferece uma representação mais fiel da "classe típica"do projeto.

Finalmente, o conjunto de dados de processo foi mesclado com o conjunto de dados de qualidade (já agregado pela mediana), criando uma base de dados final onde cada linha representava um único repositório com suas respectivas métricas de processo e qualidade. A análise subsequente, apresentada na Seção 3, foi realizada sobre esta base de dados final, comparando subgrupos de repositórios para investigar as relações propostas nas questões de pesquisa.

# 3 Resultados

Nesta seção, apresentamos os dados coletados e as estatísticas sumarizadas para responder às questões de pesquisa. A análise se baseia na coleta de métricas de processo e de qualidade de 1.000 repositórios Java populares.

## 3.1 Análise Descritiva Geral

A Tabela 1 apresenta um resumo estatístico das principais métricas coletadas. Observa-se uma grande variação nos dados, especialmente em `popularidade` e `atividade`, onde o desvio padrão é consideravelmente alto, sugerindo a presença de outliers e uma distribuição não normal dos dados. Por essa razão, a mediana é utilizada como principal medida de tendência central nas análises subsequentes.

Tabela 1: Estatísticas Descritivas das Métricas Coletadas.

| Métrica | Média | Mediana | Desvio Padrão | Mínimo | Máximo |
|---|---|---|---|---|---|
| popularidade | 9283.72 | 5720.50 | 10571.66 | 3414.00 | 116825.00 |
| maturidade | 9.64 | 9.75 | 3.05 | 0.17 | 16.68 |
| atividade | 38.82 | 10.00 | 85.82 | 0.00 | 1000.00 |
| loc | 19.22 | 17.00 | 13.18 | 2.00 | 209.00 |
| cbo | 3.55 | 3.00 | 1.71 | 0.00 | 30.00 |
| dit | 1.09 | 1.00 | 0.31 | 1.00 | 4.00 |
| lcom | 1.47 | 0.00 | 16.21 | 0.00 | 453.50 |

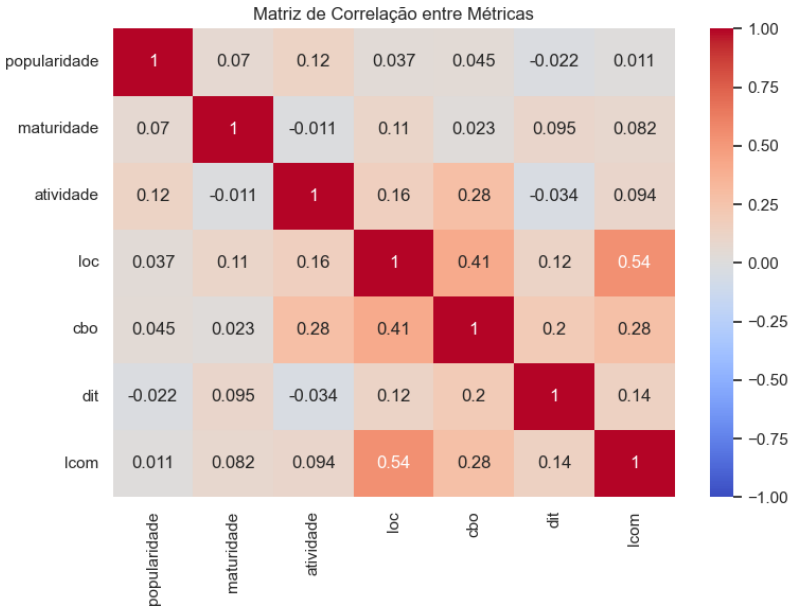


Figura 1: Matriz de correlação entre métricas de processo e qualidade. Valores próximos a 1 indicam forte correlação positiva, enquanto valores próximos a -1 indicam forte correlação negativa.

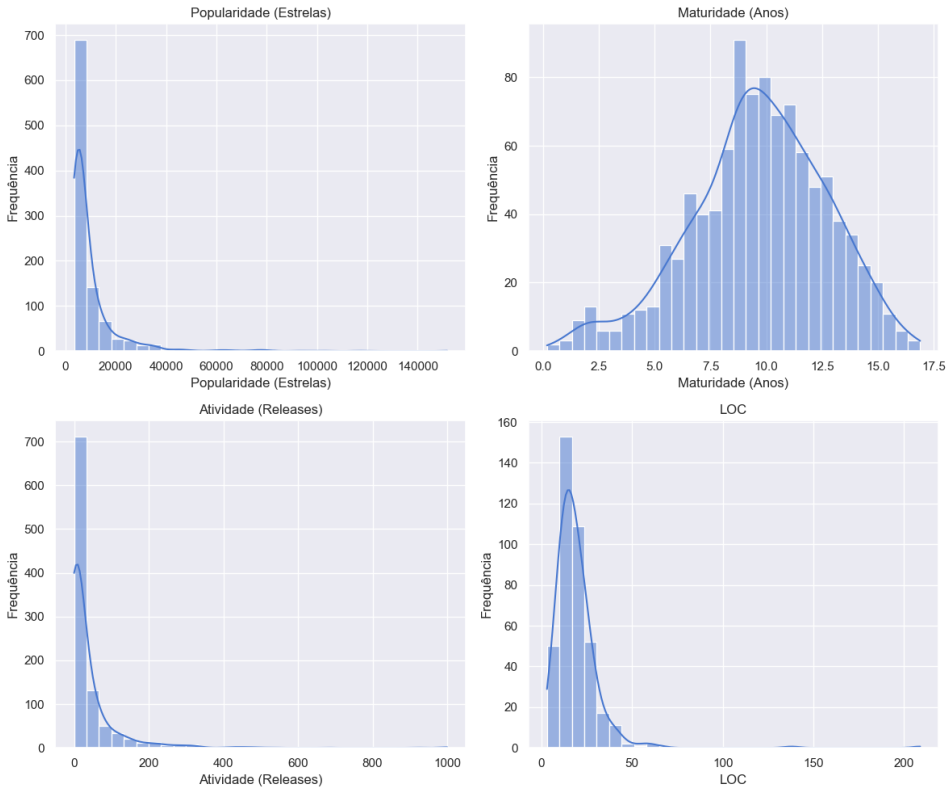


Figura 2: Distribuição das métricas de processo (Popularidade, Maturidade, Atividade) e tamanho (LOC) nos 1.000 repositórios analisados.

## 3.2 RQ01: Relação entre Popularidade e Qualidade

Para analisar a relação entre a popularidade e a qualidade, comparamos a mediana das métricas de qualidade entre os 25% repositórios mais populares e os 25% menos populares. Conforme a Tabela 2, não foram observadas diferenças significativas nos valores de CBO, DIT e LCOM entre os dois grupos.

Tabela 2: Comparação da Mediana das Métricas de Qualidade por Popularidade.

| **Métrica** | **Top 25% Mais Populares** | **Bottom 25% Menos Populares** |
|---|---|---|
| cbo | 3.00 | 3.00 |
| dit | 1.00 | 1.00 |
| lcom | 0.00 | 0.00 |
| loc | 17.00 | 16.00 |

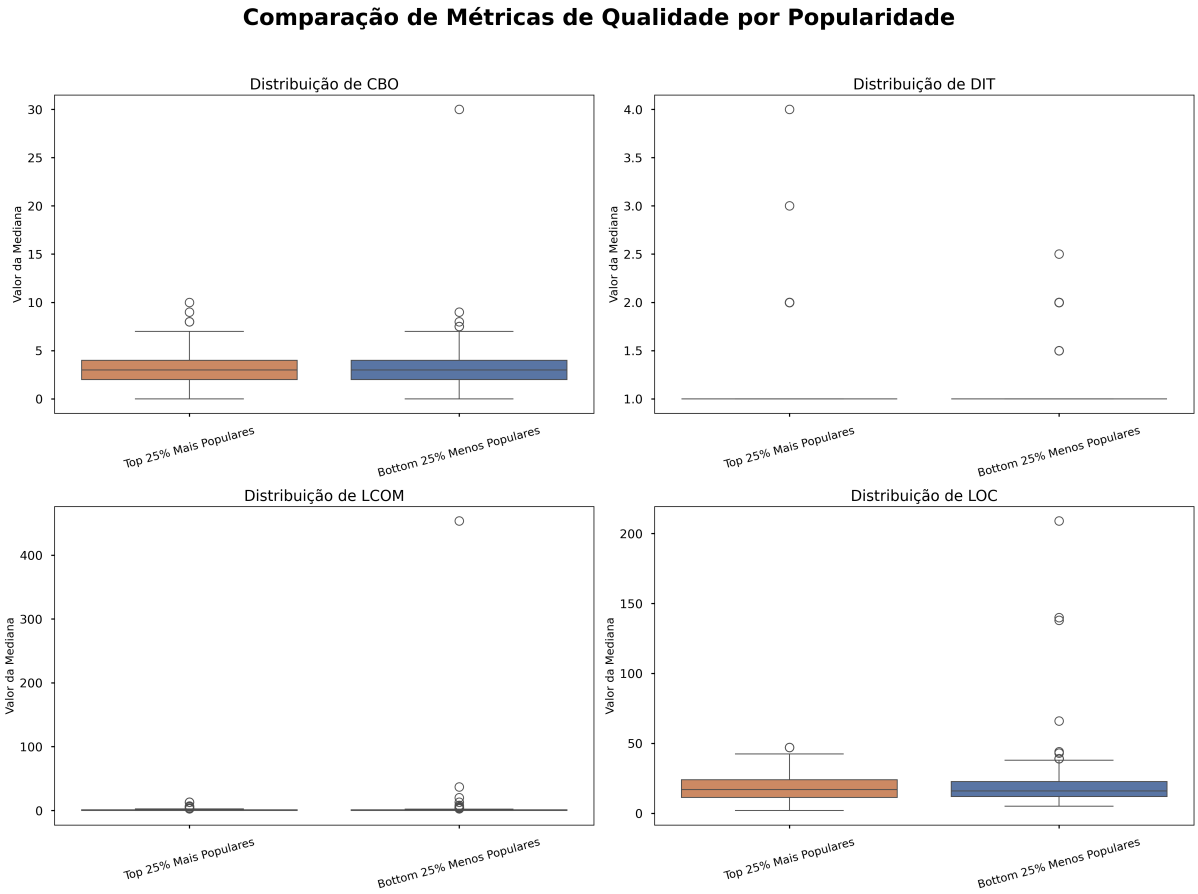


Figura 3: Grid de box plots comparando a distribuição das métricas de qualidade (CBO, DIT, LCOM, LOC) entre os 25% repositórios mais e menos populares. Os gráficos confirmam visualmente que não há diferenças significativas nas medianas ou distribuições entre os dois grupos.

## 3.3 RQ02: Relação entre Maturidade e Qualidade

A análise da maturidade (Tabela 3) comparou repositórios com idade acima e abaixo da mediana (9.75 anos). Semelhante à popularidade, os resultados mostram que não há diferença na mediana das métricas de qualidade (CBO, DIT, LCOM) entre os repositórios mais maduros e os mais jovens.

Tabela 3: Comparação da Mediana das Métricas de Qualidade por Maturidade.

| **Métrica** | **Maduros (Idade >= Mediana)** | **Jovens (Idade < Mediana)** |
|---|---|---|
| cbo | 3.00 | 3.00 |
| dit | 1.00 | 1.00 |
| lcom | 0.00 | 0.00 |
| loc | 18.00 | 16.00 |

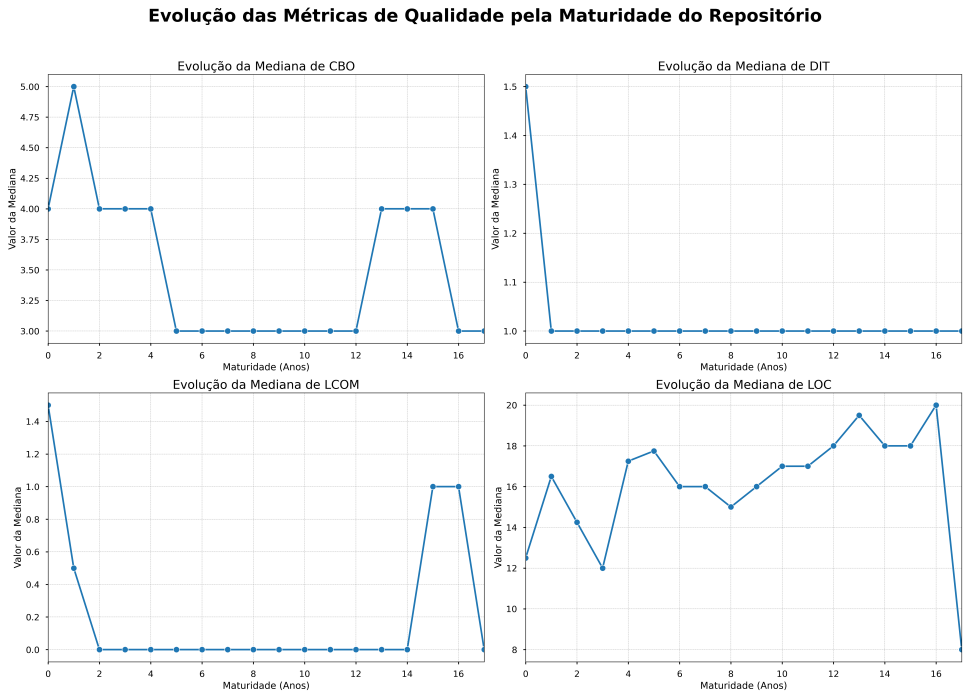


Figura 4: Box plot da distribuição do Acoplamento (CBO) por Categoria de Maturidade do Repositório.

## 3.4 RQ03: Relação entre Atividade e Qualidade

A Tabela 4 compara os 25% repositórios mais ativos (mais releases) com os 25% menos ativos. Observa-se que os repositórios mais ativos apresentam uma mediana de CBO (4.00) ligeiramente superior à dos menos ativos (3.00), sugerindo um acoplamento um pouco maior. As demais métricas permaneceram idênticas.

Tabela 4: Comparação da Mediana das Métricas de Qualidade por Atividade.

| Métrica | Top 25% Mais Ativos | Bottom 25% Menos Ativos |
|---|---|---|
| cbo | 4.00 | 3.00 |
| dit | 1.00 | 1.00 |
| lcom | 0.00 | 0.00 |
| loc | 19.00 | 16.00 |

## 3.5 RQ04: Relação entre Tamanho e Qualidade

Finalmente, ao comparar repositórios maiores (LOC >= mediana) com os menores (Tabela 5), notamos que os projetos maiores apresentam valores de CBO e LCOM medianos superiores aos dos projetos menores. Isso indica que o aumento no tamanho do código está associado a um maior acoplamento e a uma menor coesão.

Tabela 5: Comparação da Mediana das Métricas de Qualidade por Tamanho (LOC).

| **Métrica** | **Grandes (LOC >= Mediana)** | **Pequenos (LOC < Mediana)** |
|---|---|---|
| cbo | 4.00 | 3.00 |
| dit | 1.00 | 1.00 |
| lcom | 1.00 | 0.00 |

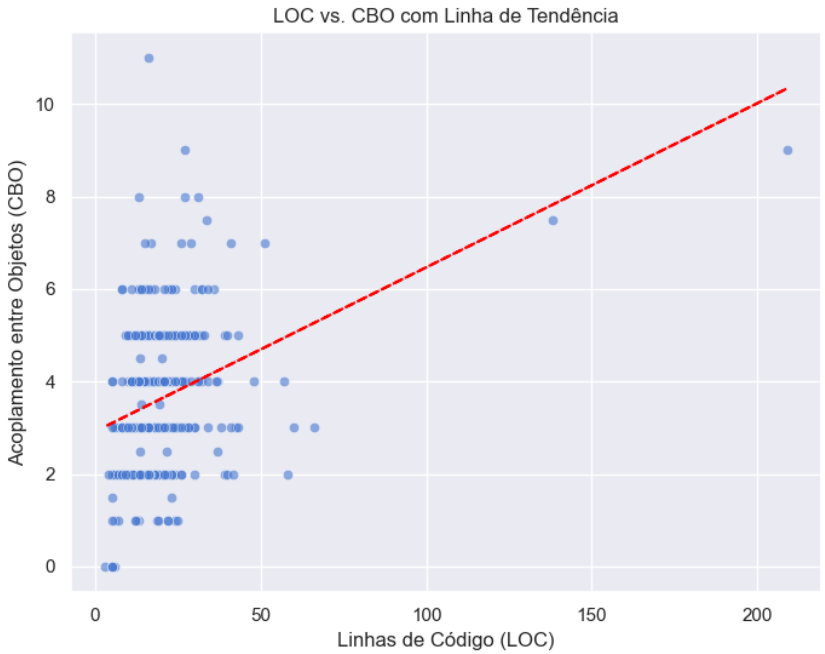


Figura 5: Gráfico de dispersão mostrando a relação entre as linhas de código (LOC) e o acoplamento (CBO) para uma amostra de repositórios. A linha de tendência indica uma relação positiva.

# 4 Conclusão

Este estudo teve como objetivo investigar a relação entre as características do processo de desenvolvimento de software e a qualidade interna do código em 1.000 dos repositórios Java mais populares do GitHub. Através da análise de métricas de processo (popularidade, maturidade, atividade e tamanho) e de produto (CBO, DIT e LCOM), o trabalho buscou validar ou refutar quatro hipóteses comuns na comunidade de desenvolvimento.

Os resultados revelaram que nem todas as percepções intuitivas sobre qualidade se sustentam empiricamente. Concluímos que a **popularidade** e a **maturidade** de um repositório não demonstraram ser indicadores confiáveis de uma maior qualidade interna do código. As hipóteses de que projetos mais populares ou mais antigos seriam intrinsecamente melhores foram refutadas, sugerindo que a complexidade e a gestão do débito técnico são fatores que independem dessas características.

Por outro lado, a análise confirmou parcialmente que uma **maior atividade** (medida em releases) está associada a uma ligeira degradação da qualidade, especificamente a um maior acoplamento (CBO). O achado mais conclusivo deste estudo, no entanto, foi a forte

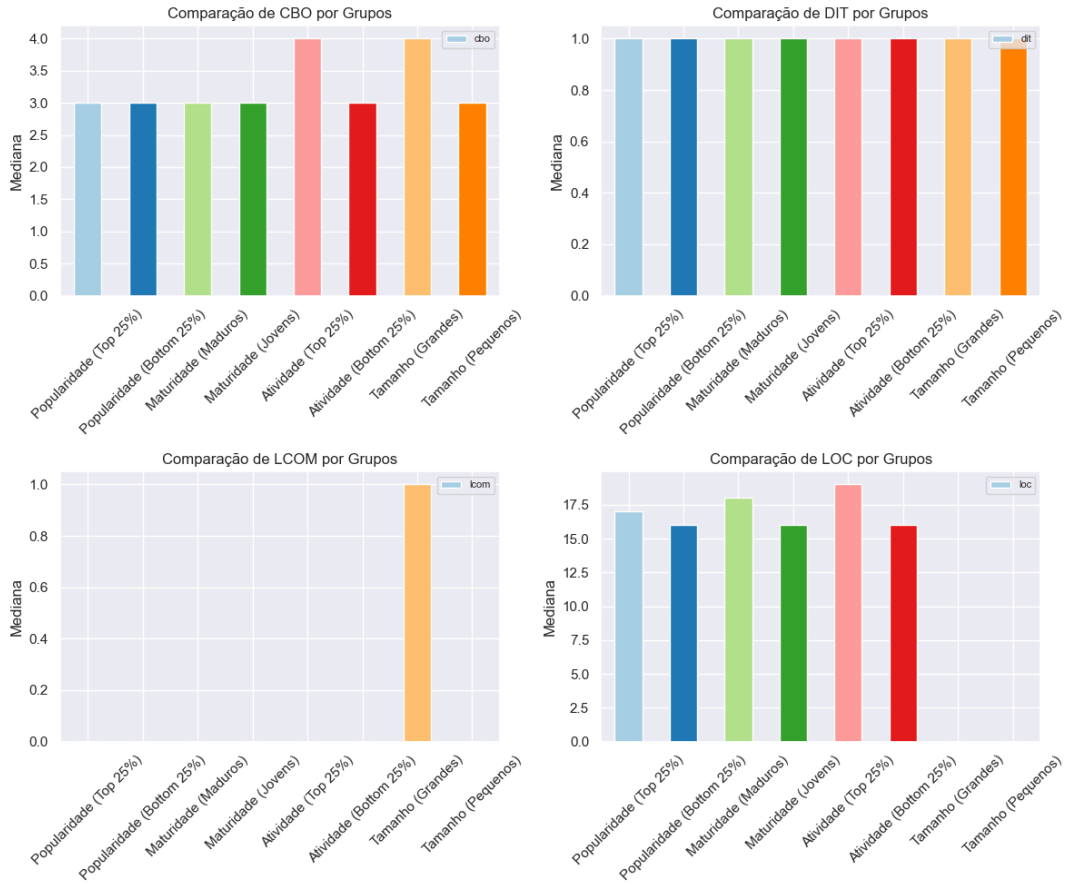


Figura 6: Comparação da mediana das métricas de qualidade (CBO, DIT, LCOM, LOC) entre grupos de popularidade, maturidade, atividade e tamanho.

evidência de que o **tamanho do repositório (LOC) está inversamente relacionado à sua qualidade**. Projetos maiores apresentaram, de forma consistente, maior acoplamento e menor coesão, confirmando a hipótese de que a complexidade escalar é um dos principais desafios para a manutenibilidade do software.

Como trabalhos futuros, sugere-se a expansão desta análise para incluir outras métricas de qualidade, como a complexidade ciclomática e a cobertura de testes, que poderiam oferecer uma visão mais holística da saúde de um projeto. Adicionalmente, replicar este estudo em ecossistemas de outras linguagens de programação, como Python ou JavaScript, seria valioso para verificar se os padrões aqui identificados são universais ou específicos da cultura de desenvolvimento Java.