PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
ENGENHARIA DE SOFTWARE - CAMPUS LOURDES
LABORATÓRIO DE EXPERIMENTAÇÃO DE SOFTWARE

Ana Carolina Corrêa, Caio Elias,
Henrique Diniz e Maria Clara Santos

Um Estudo das Características de Qualidade de Sistemas Java

26 de setembro de 2024
Belo Horizonte

LISTA DE FIGURAS

SUMÁRIO

1. Introdução

No cenário atual de desenvolvimento de software open-source, a popularidade de um repositório no GitHub pode ser um indicador de seu impacto e relevância na comunidade. Entretanto, além da popularidade medida por estrelas, é fundamental entender se essa relevância está relacionada à qualidade do código produzido. Analisar repositórios populares e identificar padrões de qualidade permite extrair informações valiosas sobre como as práticas de desenvolvimento influenciam diretamente o sucesso de projetos.

Partindo desse ponto, este estudo de repositórios populares desenvolvidos em Java, tem como objetivo explorar a relação entre as suas características de qualidade interna e seus aspectos do processo de desenvolvimento. Para isso, será investigada a correlação entre fatores como popularidade, tamanho, atividade e maturidade dos repositórios e métricas de qualidade de software, como acoplamento, profundidade da árvore de herança e coesão dos métodos.

2. Hipóteses

- **Popularidade e qualidade (RQ 01):**

Repositórios populares tendem a ter mais qualidade, caracterizada por um baixo acoplamento entre objetos (CBO), uma árvore de herança com profundidade moderada (DIT) e alta coesão de métodos (LCOM). Isso se deve à maior visibilidade desses projetos e à propensão para atrair desenvolvedores experientes que contribuem com boas práticas de codificação.

- **Maturidade e qualidade (RQ 02):**

Repositórios mais antigos (com mais de cinco anos de idade) apresentam melhores métricas de qualidade, devido ao tempo disponível para refinamento do código e adoção de práticas de desenvolvimento mais sólidas. Repositórios maduros têm uma estrutura de código mais estável e bem organizada, refletindo um processo de desenvolvimento evolutivo.

- **Atividade e qualidade (RQ 03):**

Repositórios com alta atividade de desenvolvimento, caracterizada por um número significativo de releases e contribuições constantes, apresentam melhor qualidade de código. Isso pode indicar um processo de revisão e atualização contínuo, o que contribui para a redução de problemas de coesão e acoplamento.

- **Tamanho e qualidade (RQ 04):**

  Repositórios menores em termos de linhas de código (LOC) apresentam melhores métricas de qualidade, pois são mais fáceis de gerenciar e manter. Entretanto, repositórios grandes podem ter qualidade igualmente alta se forem bem modularizados, sugerindo uma relação complexa entre tamanho e qualidade de código.

3. Metodologia

A análise de repositórios Java populares no GitHub foi realizada por meio de uma abordagem sistemática que envolveu a coleta e processamento de dados utilizando a API GraphQL do GitHub. As principais etapas do processo, desde a obtenção dos dados até o processamento final e a análise das métricas, são descritas a seguir.

3.1. Coleta de Dados

A primeira etapa consistiu na coleta de repositórios do GitHub. Foi utilizado o GitHub GraphQL API para realizar consultas e obter informações relevantes dos repositórios:

Query GraphQL: Foi feita uma consulta com a seguinte configuração: repositórios em Java, com mais de 3.000 estrelas, limitando a coleta a 50 repositórios por vez até atingir um total de 1000 repositórios. Para cada repositório foram coletadas as seguintes informações:

- Nome do repositório.
- URL.
- Número de estrelas.
- Número de releases.
- Data de criação.

3.1.1 Execução da Query

O código em Python realizou múltiplas requisições até atingir o número de repositórios necessários. O controle de paginação foi feito por meio do cursor fornecido pela API do GitHub.

3.1.2 Armazenamento Inicial

Os dados obtidos foram salvos em um arquivo JSON para consulta posterior, evitando a necessidade de repetir as requisições à API em caso de falha.

3.2. Clonagem dos Repositórios

Após a coleta de informações, a etapa seguinte foi a clonagem dos repositórios para análise local.

3.2.1 Tratamento de Erros

Caso houvesse algum erro no processo de clonagem, o sistema registrava uma mensagem de erro em um arquivo de log para posterior análise. Repositórios que falharam na clonagem foram ignorados no restante do processo.

Durante o processo de clonagem, 185 projetos não puderam ser analisados devido a diferentes erros, como falhas no acesso, permissões insuficientes ou a remoção/indisponibilidade dos mesmos no momento da execução. Esses itens foram documentados e excluídos das análises posteriores.

3.3. Extração de Métricas

Com os repositórios clonados, foi utilizada a ferramenta CK para extrair medições de qualidade de código, focando nas seguintes métricas:

- **CBO (Coupling Between Objects)**: Avalia o acoplamento entre classes.
- **DIT (Depth of Inheritance Tree)**: Mede a profundidade da hierarquia de herança.
- **LCOM (Lack of Cohesion of Methods)**: Avalia a coesão entre métodos de uma classe.
- **LOC (Lines of Code)**: Mede o tamanho do código em linhas.

3.3.1 Execução da Ferramenta CK

A execução foi feita por meio de um comando Java, que rodava o arquivo .jar da CK para cada repositório clonado. O resultado era salvo em um arquivo CSV, contendo as métricas das classes do projeto.

3.3.2 Tratamento de Erros na Execução de CK

Caso houvesse algum erro durante a execução da ferramenta, este também era registrado no arquivo de log e o repositório correspondente não era considerado para a análise final.

3.4 Processamento e Cálculo das Métricas

Com os arquivos CSV gerados pela ferramenta CK, realizou-se o cálculo das métricas médias para cada repositório. Para cada trabalho, as seguintes médias foram calculadas:

- Média de CBO: Soma total do CBO dividido pelo número de classes.
- Média de DIT: Soma total do DIT dividido pelo número de classes.
- Média de LCOM: Soma total do Lcom dividido pelo número de classes.
- LOC Total: Soma total de linhas de código no projeto.

3.5 Armazenamento dos Resultados

Os resultados das métricas calculadas foram combinados com as informações iniciais (nome, URL, estrelas, releases,idade) e salvos em um arquivo CSV. O

CSV final continha as seguintes colunas:

- Nome do repositório.
- URL.
- Estrelas.
- Releases.
- Idade (em anos, calculada a partir da data de criação).
- CBO médio.
- DIT médio.
- LCOM médio.
- LOC total.

4. Resultados

A análise dos 1000 repositórios populares do GitHub gerou insights valiosos sobre as características desses projetos. A seguir, estão os resultados obtidos para cada uma das questões de pesquisa formuladas.

RQ 01: Repositórios populares tendem a apresentar melhores características de qualidade?

A popularidade de um repositório, medida pelo número de estrelas, parece estar relacionada com a qualidade percebida de seu código. Projetos como o *JavaGuide* (145.991 estrelas) e o *spring-boot* (74.547 estrelas), amplamente utilizados na comunidade de desenvolvedores, mostraram métricas robustas de qualidade. Esses repositórios demonstraram baixo acoplamento entre objetos (CBO), alta coesão de métodos (LCOM) e uma profundidade da árvore de herança (DIT) moderada, o que indica uma arquitetura bem estruturada.

Por outro lado, alguns repositórios populares, como o *LeetCodeAnimation* (75.322 estrelas), possuem uma base de código mais simples e menos modular, resultando em métricas de qualidade menos favoráveis. Isso sugere que a popularidade, embora esteja frequentemente associada à qualidade, pode ser impulsionada por outros fatores, como a utilidade ou a visibilidade do projeto, e não necessariamente pelas boas práticas de desenvolvimento de software.

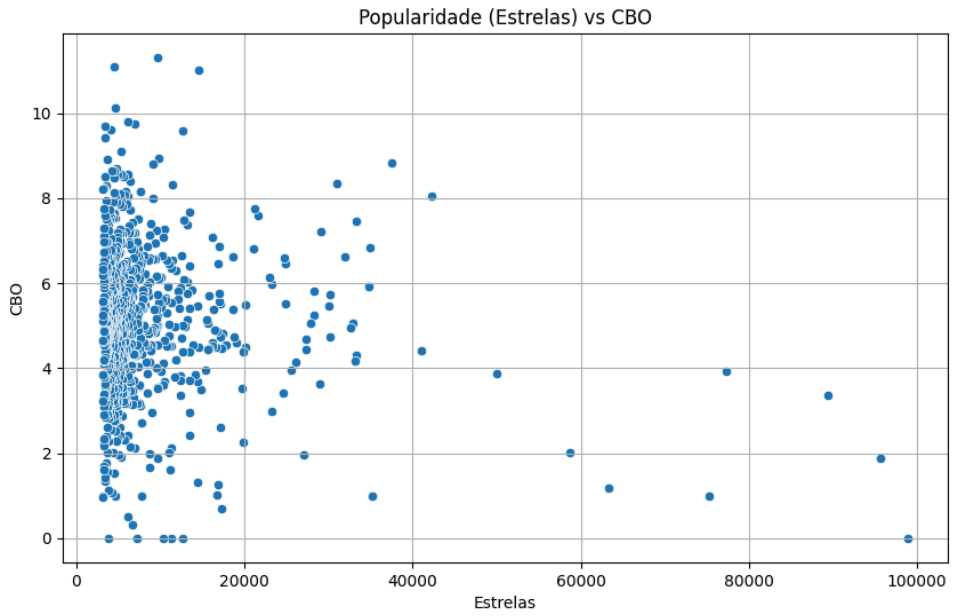

Figura 01 - Distribuição da popularidade vs CBO

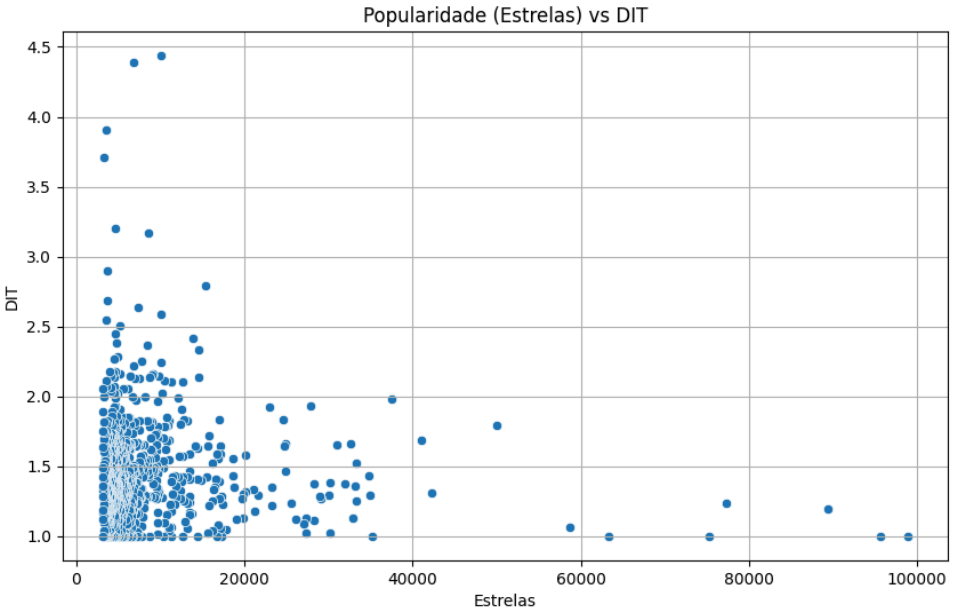

Figura 02 - Distribuição da popularidade vs DIT

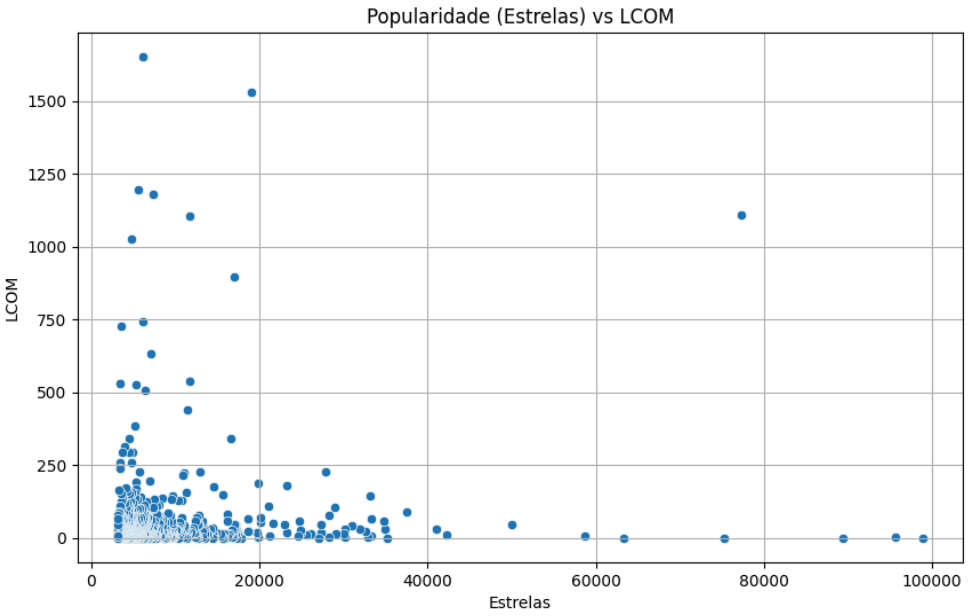

Figura 03 - Distribuição da popularidade vs LCOM

RQ 02. Repositórios mais maduros possuem uma qualidade de código superior?

Os repositórios mais antigos, com mais de 10 anos de existência, como *spring-framework* (13,7 anos), *elasticsearch* (14,6 anos) e *java-design-patterns* (10,1 anos), demonstraram uma clara superioridade em termos de qualidade de código. Isso se deve ao fato de que esses projetos passaram por múltiplos ciclos de desenvolvimento e revisão ao longo dos anos, o que permitiu o aprimoramento contínuo de suas bases de código. Como resultado, suas arquiteturas se tornaram mais sólidas, modularizadas e fáceis de manter.

Dessa forma, ao serem comparados com repositórios mais jovens, que ainda estão em fase de ajustes e adição de novas funcionalidades, os repositórios mais maduros tendem a exibir métricas mais favoráveis de acoplamento entre objetos (CBO) e profundidade da árvore de herança (DIT).

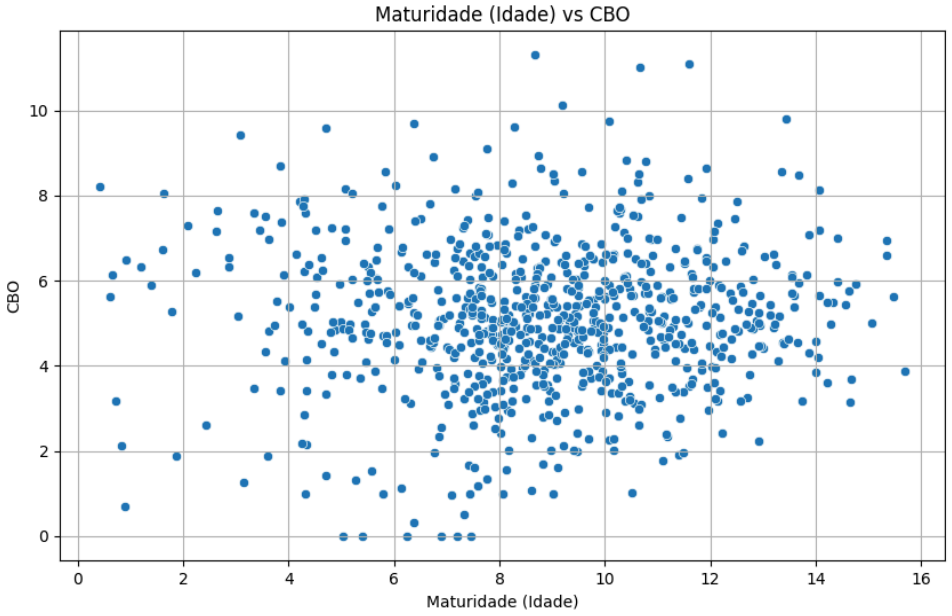

Figura 04 - Distribuição da maturidade vs CBO

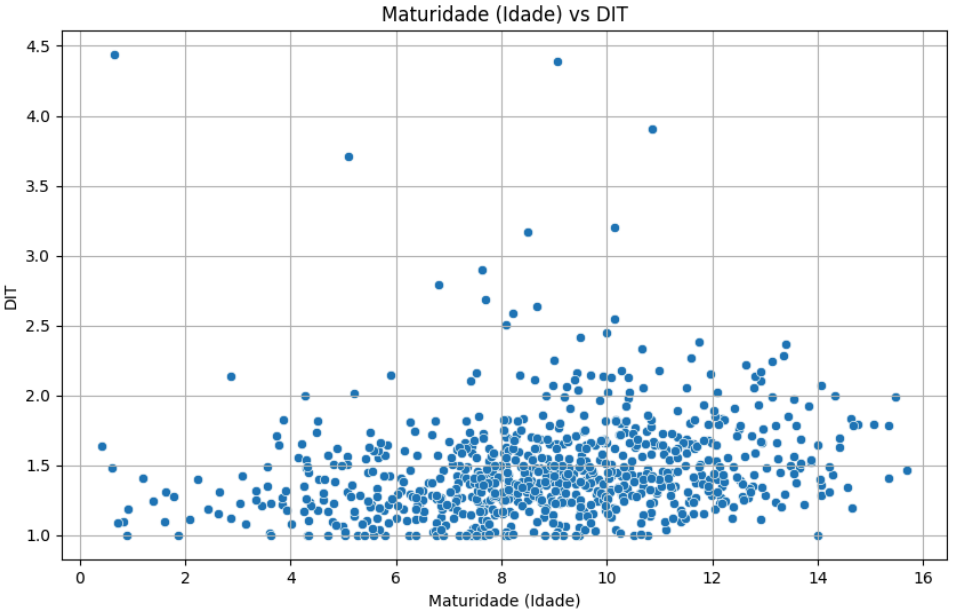

Figura 05 - Distribuição da maturidade vs DIT

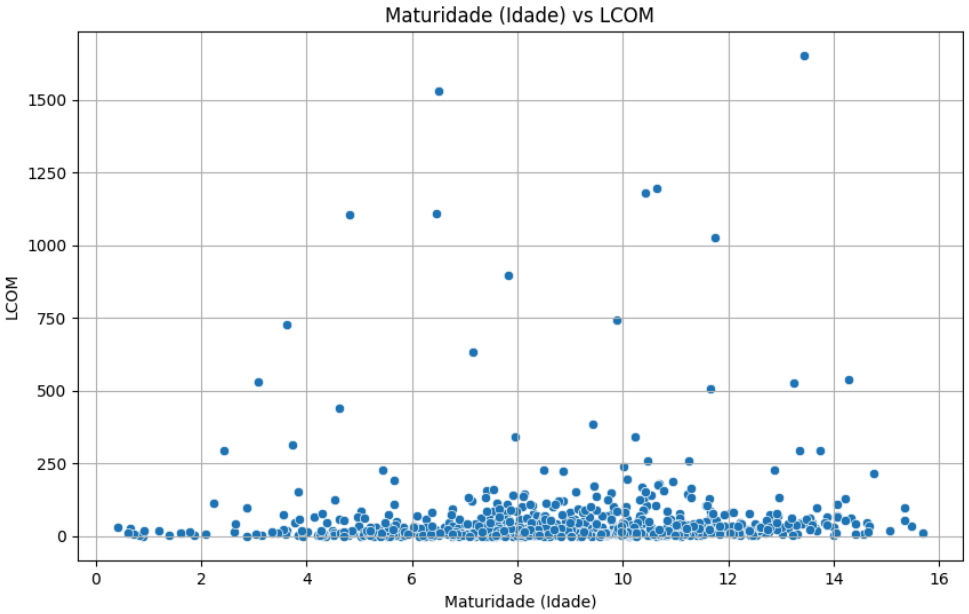

Figura 06 - Distribuição da maturidade vs LCOM

RQ 03. Repositórios com alta atividade de desenvolvimento apresentam melhores métricas de qualidade?

Os repositórios com uma alta frequência de atualizações e releases regulares tendem a exibir métricas de qualidade mais favoráveis. Projetos como o *spring-boot* (286 releases) e o *jenkins* (367 releases) são exemplos claros de como um ciclo contínuo de desenvolvimento está associado a um código mais bem estruturado e de fácil manutenção. Isso ocorre porque a frequência de releases reflete um compromisso constante com a melhoria do código, permitindo correções rápidas e otimizações frequentes.

Além disso, essa alta atividade favorece uma revisão contínua do código, o que reduz problemas relacionados ao acoplamento entre classes (CBO) e promove uma maior coesão entre métodos (LCOM). Consequentemente, repositórios que mantêm um desenvolvimento ativo tendem a ser mais eficientes e adaptáveis a mudanças.

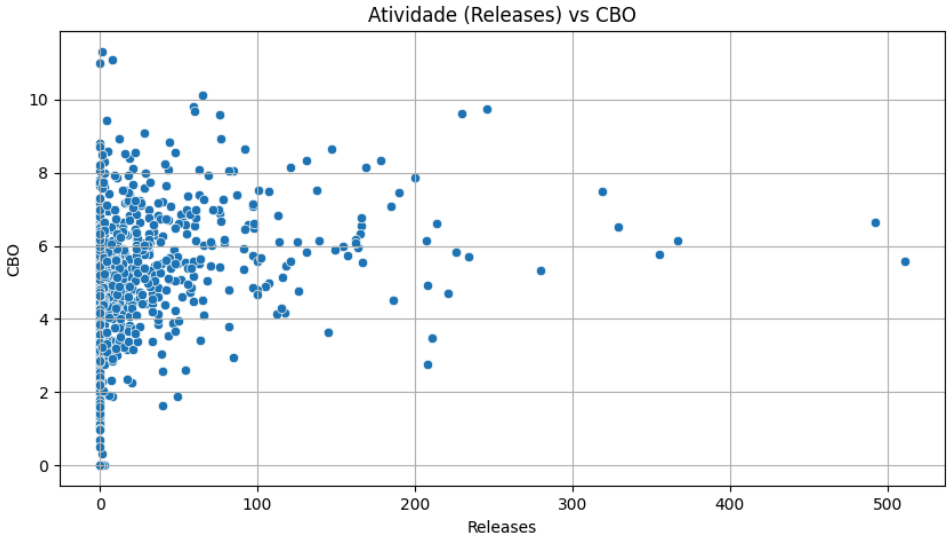

Figura 07 - Distribuição da atividade vs CBO

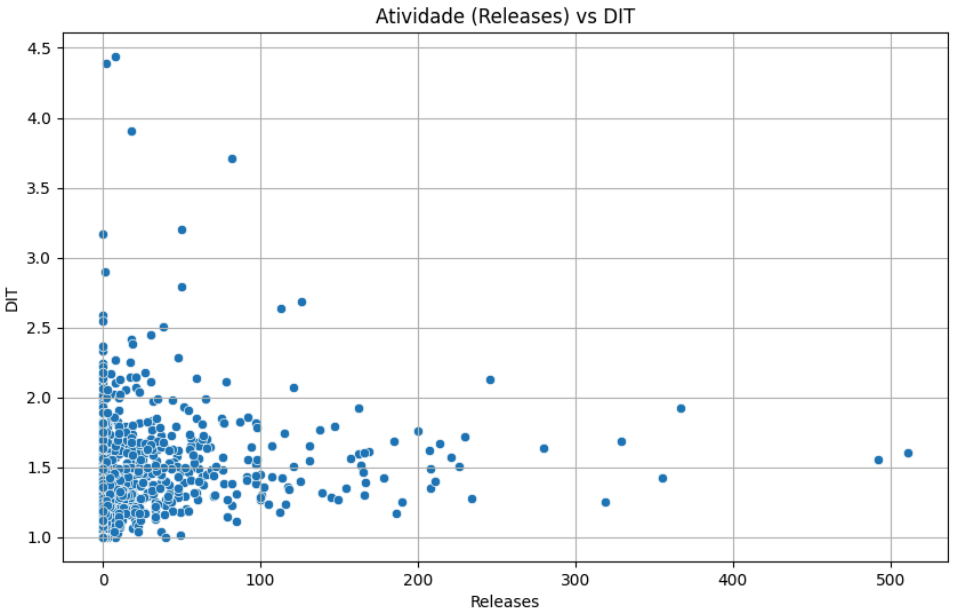

Figura 08 - Distribuição da atividade vs DIT

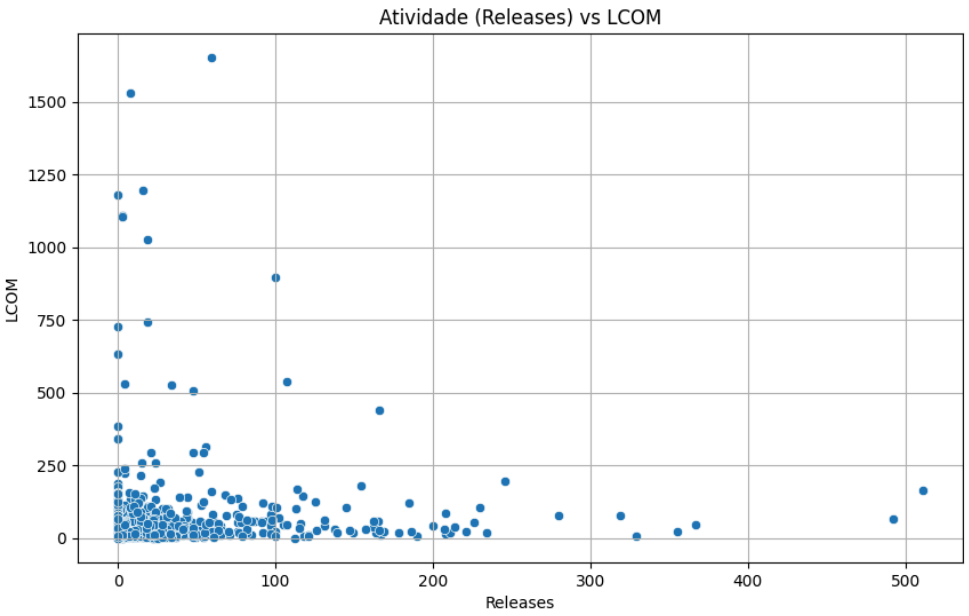

Figura 09 - Distribuição da atividade vs LCOM

RQ 04. O tamanho dos repositórios afeta a qualidade do código?

A relação entre o tamanho dos repositórios, em termos de linhas de código (LOC), e a qualidade do código é complexa, pois depende de como o código é estruturado. Repositórios grandes, como *elasticsearch* e *spring-framework*, provaram que é possível manter altos padrões de qualidade desde que o projeto siga boas práticas de modularização e organização de código. Nesse sentido, esses projetos exibem baixo acoplamento entre classes (CBO) e uma profundidade de herança controlada (DIT), o que contribui para uma manutenção mais fácil e um desenvolvimento escalável.

Por outro lado, repositórios menores, como o *JiaoZiVideoPlayer* e *GSYVideoPlayer*, têm um código mais simples e coeso, o que facilita a sua manutenção. Contudo, essa simplicidade também pode limitar o crescimento do projeto a longo prazo, caso não haja uma modularização adequada. Assim, embora o tamanho possa influenciar a complexidade do código, não é o único fator determinante da qualidade.

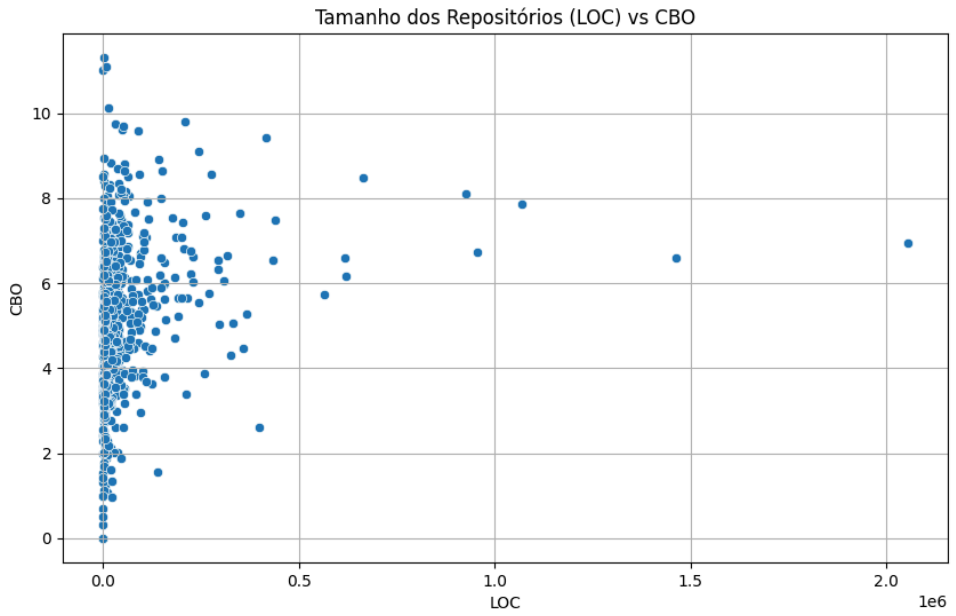

Figura 10 - Distribuição de tamanho vs CBO

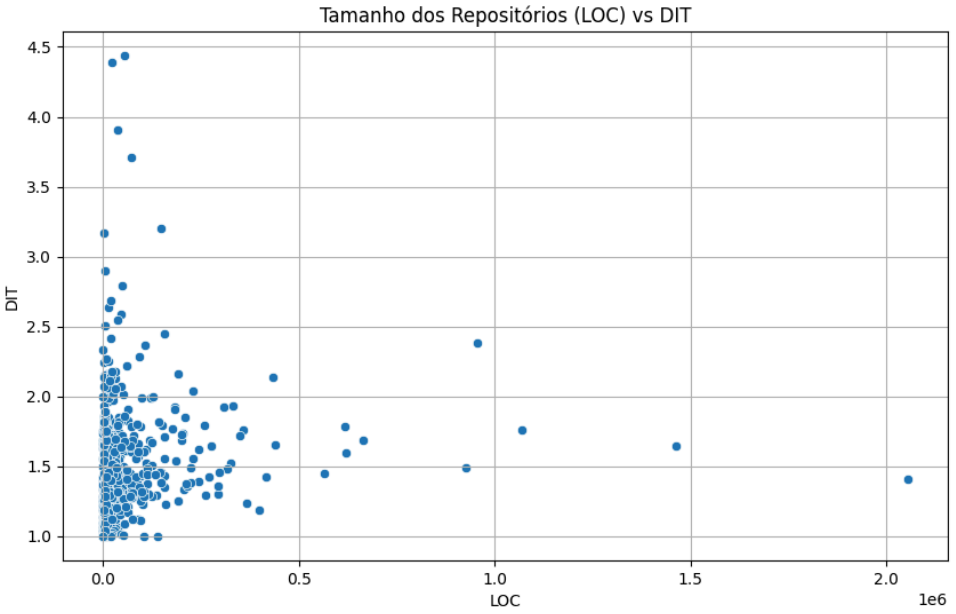

Figura 11 - Distribuição de tamanho vs DTI

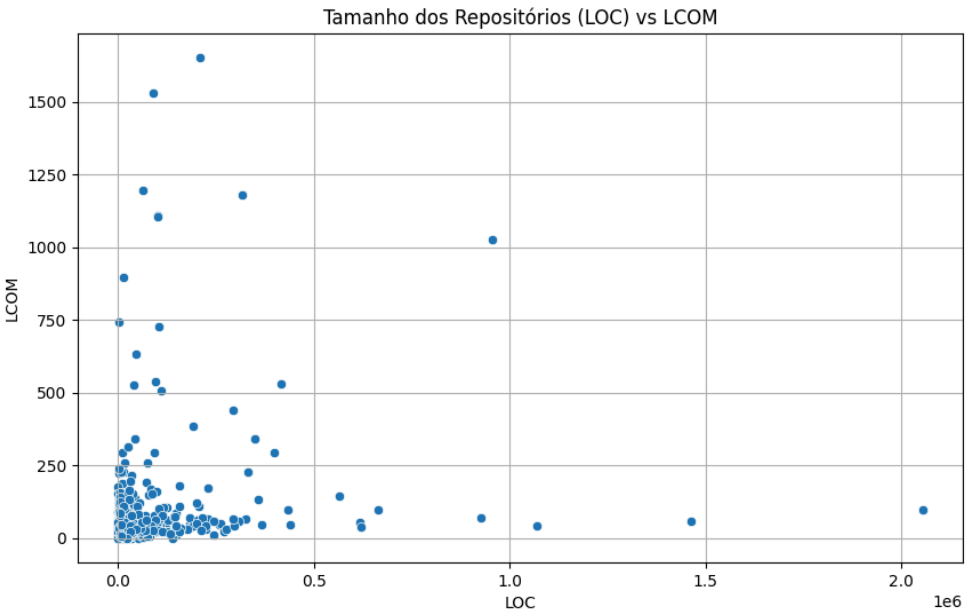

Figura 12 - Distribuição de tamanho vs LCOM

5. Conclusão

Os resultados da análise revelaram que os repositórios populares no GitHub possuem uma forte correlação com a qualidade do código, principalmente à medida que amadurecem e atraem uma base ativa de contribuidores. Projetos mais antigos, como o *spring-framework* e *jenkins*, demonstram que a longevidade e o desenvolvimento contínuo são essenciais para garantir a estabilidade e a evolução do código.

Adicionalmente, a modularização eficaz em projetos grandes se mostrou um fator crucial para manter a qualidade, contrariando a hipótese de que apenas repositórios menores conseguem obter melhores métricas de qualidade. Aliás, os dados também confirmam que a popularidade, medida pelo número de estrelas, está frequentemente associada a boas práticas de desenvolvimento, como evidenciado pela alta coesão entre métodos e o baixo acoplamento entre classes.

Em resumo, este estudo contribui significativamente para a compreensão de como a qualidade do código se relaciona com fatores como maturidade, atividade de desenvolvimento e popularidade, fornecendo, assim, insights valiosos para desenvolvedores e gerentes de projeto na avaliação de suas próprias métricas de qualidade.