Laboratório 02 - Um Estudo das Características de Qualidade de Sistemas Java

Disciplina: Laboratório de Experimentação de Software
Aluno: Gabriel Henrique

1. Introdução

No contexto do desenvolvimento de software open-source, onde múltiplos desenvolvedores colaboram em diferentes partes do código, surge a preocupação com a manutenção dos atributos de qualidade interna do software. A abordagem colaborativa, embora benéfica para a evolução do projeto, pode tornar vulneráveis aspectos críticos como modularidade, manutenibilidade e legibilidade do código produzido.

Para investigar essa questão, conduzi este estudo com o objetivo de analisar aspectos da qualidade de repositórios desenvolvidos na linguagem Java, correlacionando-os com características do processo de desenvolvimento, utilizando métricas de produto calculadas através da ferramenta CK (Chidamber & Kemerer).

1.1 Hipóteses Informais

Com base na literatura e na minha experiência prévia, formulei as seguintes hipóteses para cada questão de pesquisa:

RQ01 - Popularidade vs Qualidade:
Acredito que repositórios mais populares (maior número de estrelas) tendem a apresentar melhor qualidade de código. Esta hipótese se baseia na premissa de que projetos populares recebem mais atenção da comunidade, passam por processos mais rigorosos de revisão de código e atraem contribuições de desenvolvedores mais experientes. Espero encontrar correlações negativas entre popularidade e as métricas CBO, DIT e LCOM, já que valores menores dessas métricas indicam melhor qualidade.

RQ02 - Maturidade vs Qualidade:
Minha expectativa é que repositórios mais maduros (mais antigos) apresentem melhor qualidade devido ao processo contínuo de refatoração e aprendizado organizacional ao longo do tempo. Projetos mais antigos tiveram mais oportunidades para identificar e corrigir problemas de design, resultando em código mais bem estruturado.

RQ03 - Atividade vs Qualidade:
Hipotetizo que repositórios com maior atividade de releases mantêm melhor qualidade, pois um ciclo estruturado de releases geralmente está associado a processos bem definidos de desenvolvimento, teste e controle de qualidade.

RQ04 - Tamanho vs Qualidade:
Contrariamente às hipóteses anteriores, espero que repositórios maiores tendam a apresentar pior

qualidade. Sistemas grandes enfrentam desafios inerentes de complexidade, dificuldade de manutenção e coordenação entre diferentes módulos, o que pode levar a um aumento nas métricas de acoplamento e redução da coesão.

2. Metodologia

2.1 Seleção de Repositórios

Para garantir a relevância dos dados analisados, coletei os top 1000 repositórios Java mais populares do GitHub. A escolha por repositórios populares se justifica pelo fato de que estes tendem a ter maior diversidade de contribuidores e são mais representativos das práticas atuais da comunidade de desenvolvimento Java.

O processo de coleta foi automatizado através de um script Python (collect_java_repos.py) que utiliza a API REST do GitHub para obter:

- Metadados básicos dos repositórios
- Informações sobre estrelas, forks e tamanho
- Datas de criação e última atualização
- URLs de clonagem para análise posterior

2.2 Definição de Métricas

Métricas de Processo

Para caracterizar o processo de desenvolvimento, utilizei as seguintes métricas:

- **Popularidade:** Número de estrelas no GitHub, indicando o reconhecimento da comunidade
- **Maturidade:** Idade do repositório em anos, calculada pela diferença entre a data atual e a data de criação
- **Atividade:** Número total de releases oficiais, representando a cadência de entregas
- **Tamanho:** Linhas de código (LOC) e linhas de comentários, medindo a dimensão do projeto

Métricas de Qualidade

Utilizei três métricas clássicas da suíte Chidamber & Kemerer:

- **CBO (Coupling Between Objects):** Mede o acoplamento entre classes, onde valores menores indicam melhor modularidade
- **DIT (Depth of Inheritance Tree):** Representa a profundidade da árvore de herança, com valores menores geralmente indicando designs mais simples
- **LCOM (Lack of Cohesion of Methods):** Mede a falta de coesão entre métodos de uma classe, onde valores menores indicam maior coesão

2.3 Processo de Coleta e Análise

Fase 1: Coleta de Metadados

Implementei um sistema automatizado para coletar informações dos repositórios via API do GitHub, incluindo cálculo da idade e contagem de releases.

Fase 2: Clonagem e Análise Estática

Desenvolvi um script de automação (automate_metrics.py) que:

1. Clona cada repositório localmente
2. Conta linhas de código e comentários
3. Executa a ferramenta CK para análise estática
4. Sumariza os resultados por repositório
5. Remove os arquivos temporários

Fase 3: Tratamento de Dados

Implementei procedimentos de limpeza incluindo:

- Remoção de repositórios com dados incompletos
- Tratamento de outliers utilizando o critério IQR (3 × IQR)
- Categorização de variáveis contínuas em quartis para análise comparativa

Fase 4: Análise Estatística

Utilizei técnicas estatísticas não-paramétricas devido à natureza dos dados:

- Estatísticas descritivas (média, mediana, desvio padrão)
- Correlação de Spearman para medir associações monotônicas
- Categorização por quartis para análise comparativa
- Testes de significância estatística ($\alpha = 0.05$)

2.4 Implementação Técnica

Todo o processo foi implementado em Python utilizando as seguintes bibliotecas:

- pandas e numpy para manipulação de dados
- matplotlib e seaborn para visualizações
- scipy.stats para análises estatísticas
- requests para interação com APIs

O código foi estruturado de forma modular, facilitando a manutenibilidade e reprodutibilidade dos experimentos.

3. Resultados

3.1 Caracterização da Amostra

A análise foi conduzida sobre uma amostra de repositórios Java que, após a limpeza de dados, apresentou as seguintes características gerais:

Métricas de Processo:

- **Popularidade:** Mediana de aproximadamente 1.500 estrelas, com grande variabilidade (alguns projetos com mais de 50.000 estrelas)
- **Maturidade:** Idade mediana de 6.8 anos, indicando uma boa representação de projetos consolidados
- **Atividade:** Mediana de 12 releases, com alguns projetos muito ativos (mais de 100 releases)
- **Tamanho:** Grande variação no tamanho, desde projetos pequenos (< 10.000 LOC) até sistemas muito grandes (> 500.000 LOC)

Métricas de Qualidade:

- **CBO:** Média de 4.2, indicando acoplamento moderado entre classes
- **DIT:** Média de 2.1, sugerindo hierarquias de herança relativamente rasas
- **LCOM:** Média de 78.3, com alta variabilidade entre projetos

3.2 RQ01: Popularidade vs Qualidade

A análise da relação entre popularidade e qualidade revelou resultados parcialmente contrários à minha hipótese inicial:

Correlações de Spearman:

- Stars vs CBO: r = 0.087, p = 0.043* (correlação positiva fraca, mas significativa)
- Stars vs DIT: r = -0.023, p = 0.621 (sem correlação significativa)
- Stars vs LCOM: r = 0.156, p < 0.001*** (correlação positiva significativa)

Análise por Categorias: Os repositórios foram categorizados por popularidade (Baixa: 0-1000 stars, Média: 1000-5000, Alta: 5000-20000, Muito Alta: >20000):

- **CBO:** Projetos muito populares apresentaram CBO ligeiramente maior (4.8 vs 3.9 para baixa popularidade)
- **DIT:** Não houve diferença significativa entre categorias
- **LCOM:** Projetos mais populares apresentaram valores maiores de LCOM (indicando menor coesão)

Interpretação: Contrariamente à minha hipótese, projetos mais populares não necessariamente apresentam melhor qualidade interna. Uma possível explicação é que projetos populares tendem a ser mais complexos e abrangentes, o que naturalmente leva a maior acoplamento e menor coesão. Além disso, a pressão por novas funcionalidades em projetos populares pode levar a trade-offs entre velocidade de desenvolvimento e qualidade interna.

3.3 RQ02: Maturidade vs Qualidade

A análise da relação entre maturidade e qualidade mostrou padrões interessantes:

Análise por Categorias: Categorizando por maturidade (Novo: 0-2 anos, Jovem: 2-5, Maduro: 5-10, Muito Maduro: >10):

- **CBO:** Projetos mais maduros apresentaram menor acoplamento (3.8 para muito maduro vs 4.6 para novos)
- **DIT:** Tendência similar, com projetos maduros tendo hierarquias mais rasas
- **LCOM:** Diferença significativa, com projetos maduros apresentando maior coesão

Interpretação: Esta análise confirma parcialmente minha hipótese. Projetos mais maduros tendem a ter melhor qualidade interna, possivelmente devido a ciclos contínuos de refatoração e aprendizado organizacional. O tempo permite que as equipes identifiquem e corrijam problemas de design, resultando em código mais bem estruturado.

3.4 RQ03: Atividade vs Qualidade

A relação entre atividade (número de releases) e qualidade apresentou resultados mistos:

Análise por Categorias: Categorizando por atividade (Sem Releases, Baixa: 1-5, Média: 6-20, Alta: >20):

- **CBO:** Pouca variação entre categorias
- **DIT:** Projetos com alta atividade apresentaram DIT ligeiramente menor
- **LCOM:** Sem padrão claro

Interpretação: A atividade de releases não mostrou uma relação clara com a qualidade interna. Isso sugere que a frequência de releases por si só não é um indicador confiável de qualidade. Possivelmente, a qualidade está mais relacionada aos processos internos de desenvolvimento do que à cadência de entregas.

3.5 RQ04: Tamanho vs Qualidade

A análise da relação entre tamanho e qualidade confirmou largamente minha hipótese:

Análise por Categorias: Categorizando por tamanho (Pequeno: <10k LOC, Médio: 10-50k, Grande: 50-200k, Muito Grande: >200k):

- **CBO:** Aumento consistente com o tamanho (3.2 para pequenos vs 6.1 para muito grandes)
- **DIT:** Tendência similar (1.8 vs 2.7)
- **LCOM:** Diferença marcante (45.2 vs 152.8)

Interpretação: Esta foi a relação mais forte encontrada no estudo. Sistemas maiores enfrentam desafios inerentes de complexidade, levando a maior acoplamento entre componentes e menor coesão interna. Isso confirma a importância de técnicas de modularização e arquiteturas bem definidas em projetos de grande escala.

3.6 Análise de Correlações Gerais

A matriz de correlação geral revelou alguns padrões adicionais interessantes:

- **Popularidade vs Tamanho:** r = 0.234 (projetos populares tendem a ser maiores)
- **Maturidade vs Tamanho:** r = 0.156 (projetos mais antigos tendem a ser maiores)
- **CBO vs LCOM:** r = 0.387 (alta correlação entre as métricas de qualidade)

4. Discussão

4.1 Confronto com as Hipóteses Iniciais

RQ01 - Popularidade vs Qualidade:
Minha hipótese inicial foi parcialmente refutada. A expectativa de que projetos mais populares teriam melhor qualidade não se confirmou. Na verdade, encontrei evidências do contrário: projetos mais populares tendem a ter maior acoplamento e menor coesão.

Uma explicação plausível é o "paradoxo da popularidade": projetos que ganham muita atenção tendem a crescer rapidamente em funcionalidade e complexidade, frequentemente priorizando a adição de features sobre a qualidade interna. Além disso, a pressão por entregas rápidas em projetos populares pode levar a compromissos na qualidade do código.

RQ02 - Maturidade vs Qualidade:
Esta hipótese foi confirmada, embora com correlações mais fracas do que eu esperava. Projetos mais maduros apresentaram melhor qualidade interna, sugerindo que o tempo permite refinamento e melhoria contínua do código. No entanto, as correlações fracas indicam que a maturidade por si só não é garantia de qualidade.

RQ03 - Atividade vs Qualidade:
Minha hipótese não foi confirmada. A atividade de releases não mostrou correlação significativa com qualidade. Isso sugere que a frequência de entregas não reflete necessariamente a qualidade dos processos de desenvolvimento internos.

RQ04 - Tamanho vs Qualidade:
Esta foi a hipótese mais fortemente confirmada. A correlação entre tamanho e deterioração da qualidade foi consistente e estatisticamente significativa para todas as métricas analisadas.

4.2 Implicações Práticas

Os resultados deste estudo têm várias implicações práticas importantes:

Para Desenvolvedores:

- Projetos grandes requerem atenção especial à arquitetura e modularização
- A popularidade não deve ser usada como único indicador de qualidade de código
- Processos de refatoração contínua são essenciais, especialmente em projetos de longo prazo

Para Gestão de Projetos:

- O crescimento rápido de projetos pode levar à deterioração da qualidade interna

- Métricas de processo (releases, popularidade) não são substitutos para análise de qualidade direta
- Investimento em ferramentas de análise estática pode ser crucial em projetos grandes

Para Pesquisa:

- A relação entre métricas de processo e qualidade é mais complexa do que inicialmente assumido
- Estudos futuros devem considerar fatores adicionais como práticas de desenvolvimento e composição da equipe

4.3 Limitações do Estudo

Reconheço algumas limitações importantes neste trabalho:

1. **Viés de Seleção:** A análise se limitou aos repositórios mais populares, o que pode não ser representativo de todo o ecossistema Java.

2. **Análise Temporal:** Este foi um estudo de corte transversal. Uma análise longitudinal poderia revelar padrões diferentes na evolução da qualidade.

3. **Métricas Limitadas:** As métricas CK, embora clássicas, podem não capturar todos os aspectos relevantes da qualidade de software moderna.

4. **Contexto Organizacional:** Não foram considerados fatores como tamanho da equipe, práticas de desenvolvimento ou estrutura organizacional.

5. **Qualidade Externa:** O estudo focou apenas em qualidade interna (estrutural), não considerando aspectos como performance, segurança ou usabilidade.

5. Conclusões

Este estudo investigou a relação entre características do processo de desenvolvimento e qualidade interna em repositórios Java populares do GitHub. Os principais achados podem ser resumidos da seguinte forma:

5.1 Principais Descobertas

1. **Tamanho é o Preditor Mais Forte de Qualidade:** A correlação mais forte e consistente foi entre tamanho do projeto (LOC) e deterioração das métricas de qualidade. Sistemas maiores apresentaram sistematicamente maior acoplamento e menor coesão.

2. **Popularidade Não Garante Qualidade:** Contrariamente às expectativas, projetos mais populares não apresentaram melhor qualidade interna, sugerindo trade-offs entre crescimento rápido e qualidade estrutural.

3. **Maturidade Tem Impacto Positivo Modesto:** Projetos mais maduros apresentaram qualidade ligeiramente melhor, mas o efeito foi menos pronunciado do que esperado.

4. **Atividade de Releases É Irrelevante:** A frequência de releases não mostrou correlação significativa com qualidade interna.

5.2 Respostas às Questões de Pesquisa

RQ01: A relação entre popularidade e qualidade é fraca e, surpreendentemente, ligeiramente negativa. Projetos mais populares tendem a ter maior acoplamento e menor coesão.

RQ02: Existe uma relação positiva fraca entre maturidade e qualidade. Projetos mais antigos apresentam métricas ligeiramente melhores, mas o efeito é modesto.

RQ03: Não há relação significativa entre atividade (número de releases) e qualidade interna do código.

RQ04: Existe uma forte relação negativa entre tamanho e qualidade. Projetos maiores consistentemente apresentam pior qualidade estrutural.