PUC Minas
Instituto de Ciências Exatas e Informática

Pontifícia Universidade Católica de Minas Gerais
Departamento de Engenharia de Software e Sistemas de Informação

Curso de Engenharia de Software
Disciplina: Laboratório de Experimentação de Software
Professor: José Laerte Pires Xavier Junior

Relatório Final - Laboratório 02

Alunos: Guilherme Gabriel Silva Pereira e Lucas Ângelo Oliveira Martins Rocha.

1. Introdução

Neste trabalho são estudadas métricas de qualidade em relação a métricas de processo de sistemas populares open-source, com base nos 1.000 repositórios com maior número de estrelas e linguagem primária sendo Java, no GitHub.

As métricas de processo são a popularidade, definida pelo número de estrelas do repositório; tamanho (LOC, *Lines of Code*), que são as linhas de código do repositório; atividade, representada pela frequência de releases; por fim, maturidade do repositório pela idade em anos.

Já as métricas de qualidade do código, são o CBO (*Coupling between objects*) que representa o acoplamento entre objetos; o DIT (*Depth Inheritance Tree*) informando a profundidade de árvore de herança; por último o LCOM (*Lack of Cohesion of Methods*) que trás a falta de coesão entre os métodos. Tais métricas foram calculadas por meio de uma ferramenta de cálculo automatizado (CK Metrics).

A partir disto, algumas hipóteses iniciais para com relação ao estudo foram levantadas. São elas, relacionadas com a popularidade, que quanto mais estrelas o repositório possuir, menor será o CBO, pois, entende-se que desenvolvedores não gostam de projetos acoplados. Ademais, com relação a popularidade e DIT, quanto maior for o DIT, mais popular será, pelo fato do uso de heranças ser algo que a comunidade de desenvolvedores conhecem e utilizam. Por fim, quanto menor o LCOM mais popular será, devido ao fato que desenvolvedores gostam de projetos coesos.

Sobre o tamanho dos sistemas, entende-se que o CBO é menor, pois em sistemas grandes seria inviável dar suporte com muito acoplamento. Além disso, é quase certeira a hipótese de que quanto maior o sistema, maior será seu DIT, porque, tende-se a ser sistemas com objetos mais profundos e especificados, aumentando assim o seu tamanho. Ao fim, o

LCOM provavelmente será baixo à medida que o tamanho aumenta, pelo fato de ser um sistema mais específico e seus atributos e métodos estarem trabalhando de forma coesa.

Já em relação a atividade do repositório, considera-se que quanto maior for sua quantidade de releases, maior é seu CBO, pensando que se trata de um software que necessita de muitas correções, devido ao fato de que qualquer alteração afeta outros elementos, por causa do acoplamento. Em vista do DIT, quanto mais atividade o repositório tiver, maior será seu DIT, pelo motivo de se tiver objetos muito profundos na árvore, a alteração de quaisquer propriedades superiores, impactará diretamente nos inferiores, possivelmente necessitando do lançamento de uma nova release. Para concluir hipóteses da atividade, acredita-se que o número de release não possui nenhuma relação métrica com o LCOM de um software.

Para finalizar, acredita-se que a maturidade de um software está inversamente relacionado com seu CBO, pois, projetos antigos tendem a possuírem um menor acoplamento pelo fato de já possuírem um maior tempo de vida para refinos. Já o DIT de um repositório de código maduro, provavelmente será alto, pois quanto mais antigo um projeto, é possível que tenha utilizado mais herança modular, ocasionando a maior profundidade da herança. Por fim, o LCOM de um código com maturidade alta é baixo, pelo fato de ser mais bem refinado, os seus métodos também serão coesos.

2. Metodologia

Neste trabalho, foi desenvolvido um script na linguagem de programação Python, na versão 3.10.4. Este script exercê 2 funções principais para a análise de dados deste trabalho, sendo elas: busca massiva de dados de repositórios Java populares com extração das métricas de processo e análise automatizada de cada repositório por meio da ferramenta CK Metrics para capturar as métricas de qualidade de código. A partir destas métricas de processo e qualidade calculadas, foi feita uma planilha no Google Sheets, com a carga de dados *comma-separated values* (csv) gerada pelo script, sendo possível assim, gerar os gráficos e informações estáticas, no intuito de validar visualizar os dados e compará-los com as hipóteses levantadas inicialmente.

Com relação a busca massiva de dados de repositórios Java populares, o script lê informações dos 1.000 repositórios com a linguagem primária Java, do GitHub, com o maior número de estrelas decrescentes. Os dados buscados em cada repositório são o seu id, quantidade de estrelas, nome do criador com nome do repositório (identificador único), url, datas de criação (utilizada para cálculos anos) e total de *releases*. Todas estas informações

foram selecionadas para que fosse possível obter respostas para as análises das questões hipotéticas. A obtenção dos dados supra descritos foi realizada através da API 4.0 do Github, que segue a arquitetura GraphQL. Para isso, foi necessário fornecer o *personal token* do Github durante a coleta de dados, feita por meio da biblioteca *request* do Python. Para testes da API, foi utilizada a própria ferramenta de exploração da plataforma do GitHub, que fornece uma interface de consulta, com todos os atributos e entidades que podem ser buscados.

Na formatação dos dados retornados por esta API do GitHub, que são originalmente retornados em *JavaScript Object Notation (JSON)*, foram utilizadas principalmente as bibliotecas *json* para receber os dados e a *csv* para formatá-los e salvá-los em um arquivo no padrão csv. Durante esta formatação, é importante frisar que o cálculo de anos de idade do repositório foi pela data de criação (idade do repositório em dias). Desta forma, o arquivo csv onde os dados do estudo são armazenados, foi criado.

Para extração das métricas de qualidade de código, foi implementado um script Python funcionando paralelamente com 6 threads, que captura todas as urls salvas no csv pelo passo anterior, para ser possível clonar todo o repositório localmente, por meio de biblioteca do próprio Github. A partir disso, o diretório de cada repositório baixado era enviado para o jar do CK Metrics na versão 0.7.1, com os parâmetros para que ele calcula-se as métricas de código LOC, CBO, DIT, LCOM* (Lembrando que esta é uma versão aprimorada do LCOM original, disponibilizado pelo CK). Após isso, as métricas geradas pelo jar eram salvas em um outro arquivo csv, que é lido pelo script e captura as colunas das informações métricas necessárias, para que fossem salvas no arquivo de csv de repositórios, na devida linha para de informação para cada repositório, este arquivo sendo aquele criado no passo de busca massiva dos dados. Por fim, estes arquivos csv extras gerados pelo jar do CK Metrics e o repositório clonado eram deletados.

Vale considerar, que em 8 repositórios, a ferramenta CK Metrics não conseguiu calcular nenhuma métrica por algum erro interno, que foi salvo nos logs pelo script feito neste trabalho, além disso, outros 12 repositórios o CK Metrics não conseguiu calcular todas as 4 métricas de código necessárias, por motivos variados como por exemplo serem repositórios com anotações desconhecidas ou repositórios de exemplos de código para estudo. Diante disso, os seguintes 20 dados de repositórios representados na Figura 1, foram descartados da análise de resultados obtidos para que erros não interfiram nos gráficos, definindo assim um estudo sobre 980 repositórios Java.

id	stargazerCount	nameWithOwner	url	createdAt	releases	ageinyears	LOC	CBO	DIT	LCOM
MDEwOlRlcG9zaXRvcnkxMzI0NjQzOTU=	126912	Snailclimb/JavaGuide	https://github.com/Snailclimb/JavaGuide	2018-05-07T13:27:00Z	0	4.33	0			
MDEwOlRlcG9zaXRvcnkxNTE4MzQwNjI=	65278	doocs/advanced-java	https://github.com/doocs/advanced-java	2018-10-06T11:38:30Z	1	3.91	0			
MDEwOlRlcG9zaXRvcnkxNTQ2NDgxMTI=	23790	hollischuang/toBeTopJavaer	https://github.com/hollischuang/toBeTopJavaer	2018-10-25T09:54:54Z	0	3.86	0			
MDEwOlRlcG9zaXRvcnk4OTgyMjUzMQ==	15851	Anuken/Mindustry	https://github.com/Anuken/Mindustry	2017-04-30T01:24:29Z	140	5.35	0			
MDEwOlRlcG9zaXRvcnkxNDkxMjE5NTQ=	13995	openjdk/jdk	https://github.com/openjdk/jdk	2018-09-17T12:29:20Z	0	3.96				
MDEwOlRlcG9zaXRvcnkyMjE4NjI=	11440	projectlombok/lombok	https://github.com/projectlombok/lombok	2009-06-08T19:46:41Z	3	13.24				
MDEwOlRlcG9zaXRvcnkyMTc5NDMzNjE=	10844	doocs/source-code-hunter	https://github.com/doocs/source-code-hunter	2019-10-28T01:35:10Z	0	2.85	0			
MDEwOlRlcG9zaXRvcnkxMjkzMzI4MjY=	10725	frank-lam/fullstack-tutorial	https://github.com/frank-lam/fullstack-tutorial	2018-04-13T01:50:10Z	0	4.39	0			
R_kgDOHMKsgg	10507	Grasscutters/Grasscutter	https://github.com/Grasscutters/Grasscutter	2022-04-17T12:43:46Z	1	0.38	0			
MDEwOlRlcG9zaXRvcnkzNDg2NTE3NDY=	10106	chefyuan/algorithm-base	https://github.com/chefyuan/algorithm-base	2021-03-17T09:32:28Z	0	1.47	0			
MDEwOlRlcG9zaXRvcnk2NTg4NQ==	10102	aosp-mirror/platform_frameworks_base	https://github.com/aosp-mirror/platform_frameworks_base	2008-10-21T18:20:37Z	0	13.87				
MDEwOlRlcG9zaXRvcnkzMzIxODQxNA==	9571	react-native-camera/react-native-camera	https://github.com/react-native-camera/react-native-camera	2015-04-01T00:54:40Z	144	7.43	0			
MDEwOlRlcG9zaXRvcnkxMjQ5OTI1MQ==	7253	checkstyle/checkstyle	https://github.com/checkstyle/checkstyle	2013-08-31T02:05:05Z	119	9.01				
MDEwOlRlcG9zaXRvcnkxNjY1MTUwMjI=	6049	trinodb/trino	https://github.com/trinodb/trino	2019-01-19T06:38:14Z	0	3.62				
MDEwOlRlcG9zaXRvcnkxNjM4OTY4MQ==	5892	google/j2objc	https://github.com/google/j2objc	2014-01-30T20:19:56Z	39	8.59				
MDEwOlRlcG9zaXRvcnk5ODQ5OTI4Mw==	4199	CoI-E/Recaf	https://github.com/CoI-E/Recaf	2017-07-27T06:01:10Z	168	5.11				
MDEwOlRlcG9zaXRvcnkxNTU5ODczMjk=	3786	RedSpider1/concurrent	https://github.com/RedSpider1/concurrent	2018-11-03T13:49:36Z	0	3.84	0			
MDEwOlRlcG9zaXRvcnkxNzQyNjA2MDU=	3677	alibaba/dragonwell8	https://github.com/alibaba/dragonwell8	2019-03-07T02:57:45Z	19	3.5				
MDEwOlRlcG9zaXRvcnk1NjY2NDY0OA==	3641	NotFound9/interviewGuide	https://github.com/NotFound9/interviewGuide	2016-04-20T07:17:09Z	0	6.37	0			
MDEwOlRlcG9zaXRvcnkxNTY2MjczNjU=	3368	itwanger/toBeBetterJavaer	https://github.com/itwanger/toBeBetterJavaer	2018-11-08T00:35:56Z	0	3.82	0			

Figura 1. Dados de repositórios descartados por falhas no CK Metrics

A partir disso, com os dados no formato csv e a informação de idade em anos já calculada a partir da data de criação do repositório, juntamente com as demais métricas de processo e métricas de qualidade geradas pelo CK Metrics, foi possível manipular os dados em forma de gráficos e extrair informações estatísticas de forma mais simples. Para isso, foi utilizado o Google Sheets, que possibilitou gerar gráficos do tipo *Scatter plot* (gráficos de dispersão) para mostrar as relações entre variáveis das hipóteses. Sendo úteis para detectar tendências, padrões e outras relações ocultas nos dados e responder às questões.

A seguir, também usando o Google Sheets, foi feito o cálculo do coeficiente Spearman de correlação entre as variáveis. Para o cálculo dessa métrica, foi necessário fazer o ranqueamento dos conjuntos de dados analisados para cada variável, e utilizada a seguinte fórmula da Figura 2, logo abaixo. Sendo, nesta fórmula, *di* a diferença entre os *rankings* do elemento i, e *n* o número de dados.

$$r_R = 1 - \frac{6\Sigma_i {d_i}^2}{n(n^2-1)}$$

Figura 2. Equação do coeficiente de Spearman

Por fim, para extrair os indicadores desta métrica, foi usada a seguinte classificação: Para um módulo do coeficiente entre 0 e 0.15, a correlação é considerada inexistente/irrelevante. Para um módulo do coeficiente entre 0.15 e 0.45, é considerado que há uma correlação baixa. Para um módulo do coeficiente entre 0.45 e 0.75, é considerado que há uma correlação média. Por último, para um módulo do coeficiente maior que 0.75, é considerado que há uma correlação alta entre as duas variáveis.

3. Resultados obtidos

A partir da aplicação da metodologia descrita previamente, foi possível coletar os seguintes resultados.

Em relação à contagem de estrelas dos repositórios, as Figuras 3, 4 e 5 mostram o gráfico de dispersão em relação às métricas de qualidade CBO, DIT e LCOM, apresentando os coeficientes de correlação de Spearman de 0.12, -0.54 e 0.25, respectivamente.

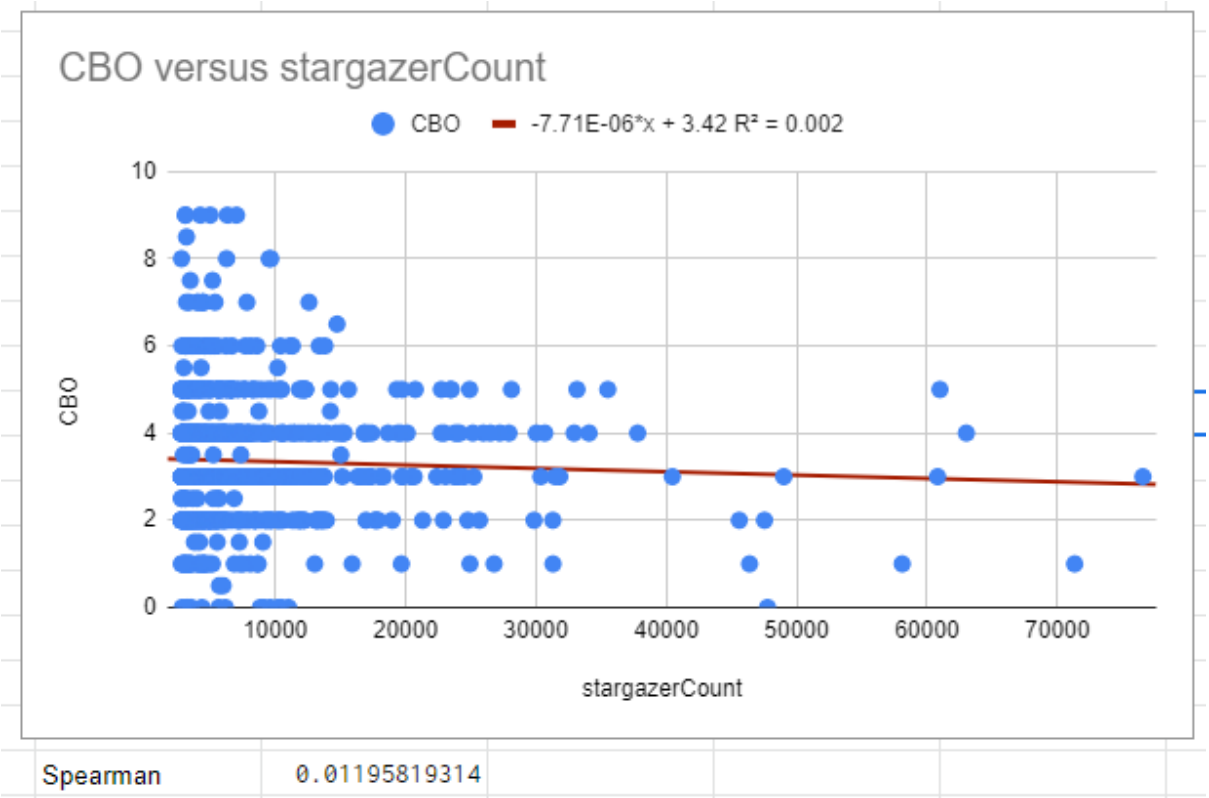
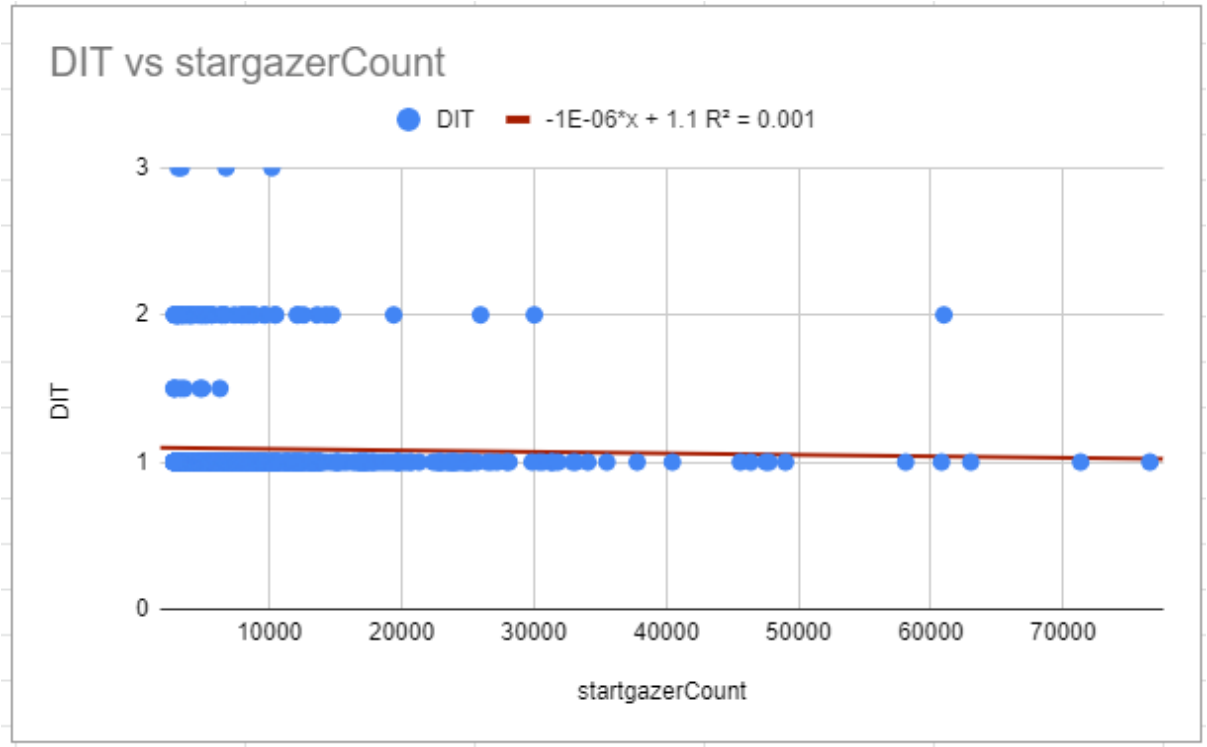
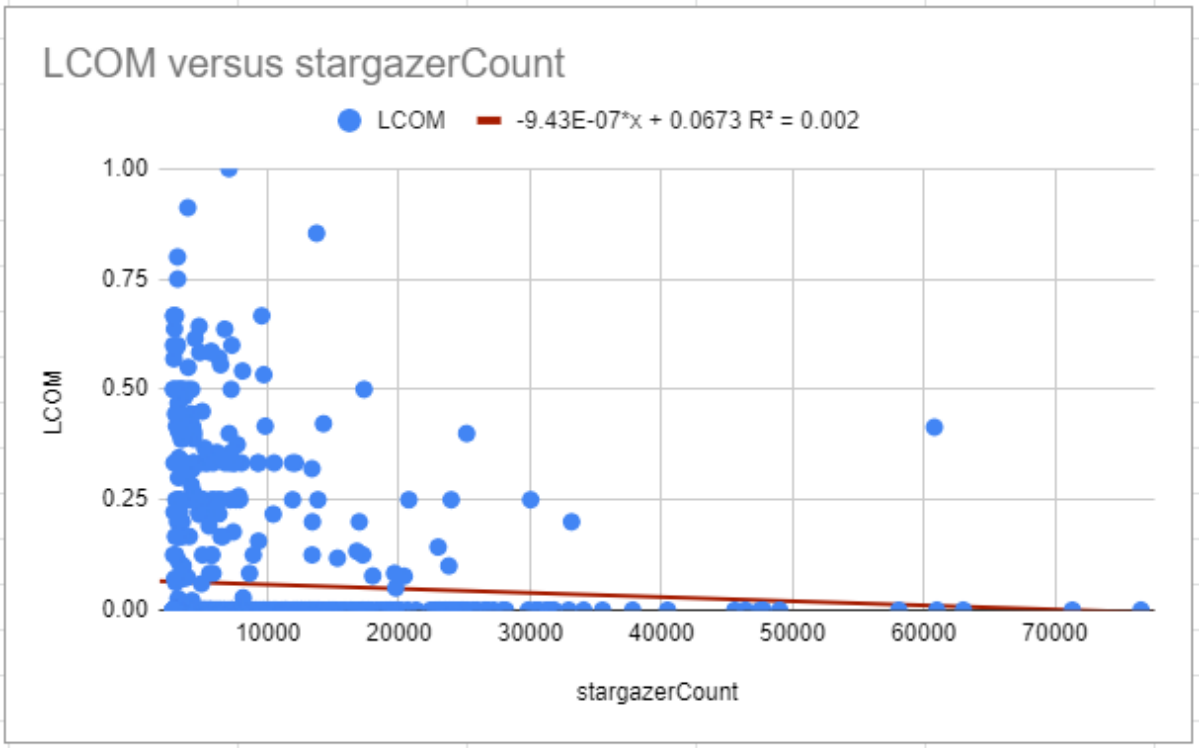


Figura 3. Gráfico de dispersão número de estrelas vs CBO



Spearman	-0.5393482505

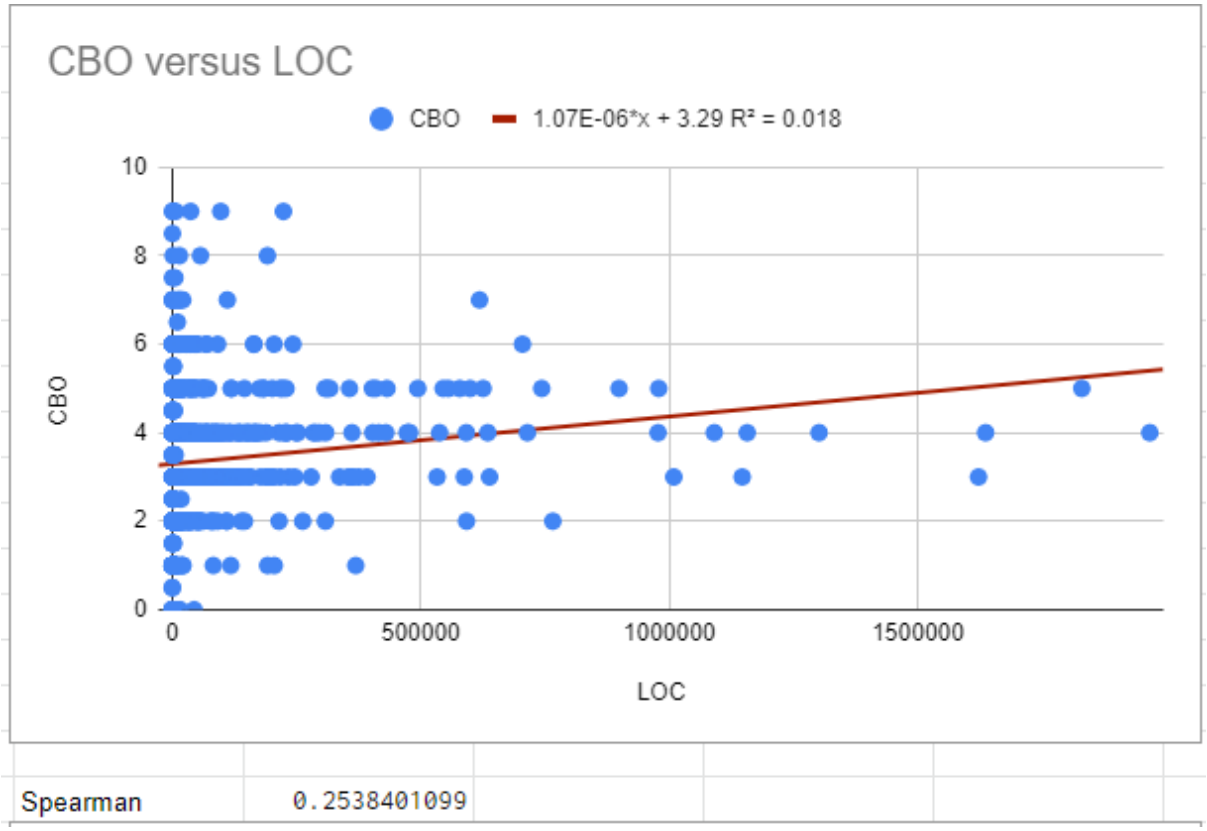
Figura 4. Gráfico de dispersão número de estrelas vs DIT



Spearman	-0.1873377564

Figura 5. Gráfico de dispersão número de estrelas vs LCOM

Já em relação à contagem total de linhas dos repositórios, as Figuras 6, 7 e 8 mostram o gráfico de dispersão em relação às métricas de qualidade CBO, DIT e LCOM, apresentando os coeficientes de correlação de Spearman de 0.25, -0.54 e -0.17, respectivamente.



Spearman	0.2538401099

Figura 6. Gráfico de dispersão número de linhas vs CBO

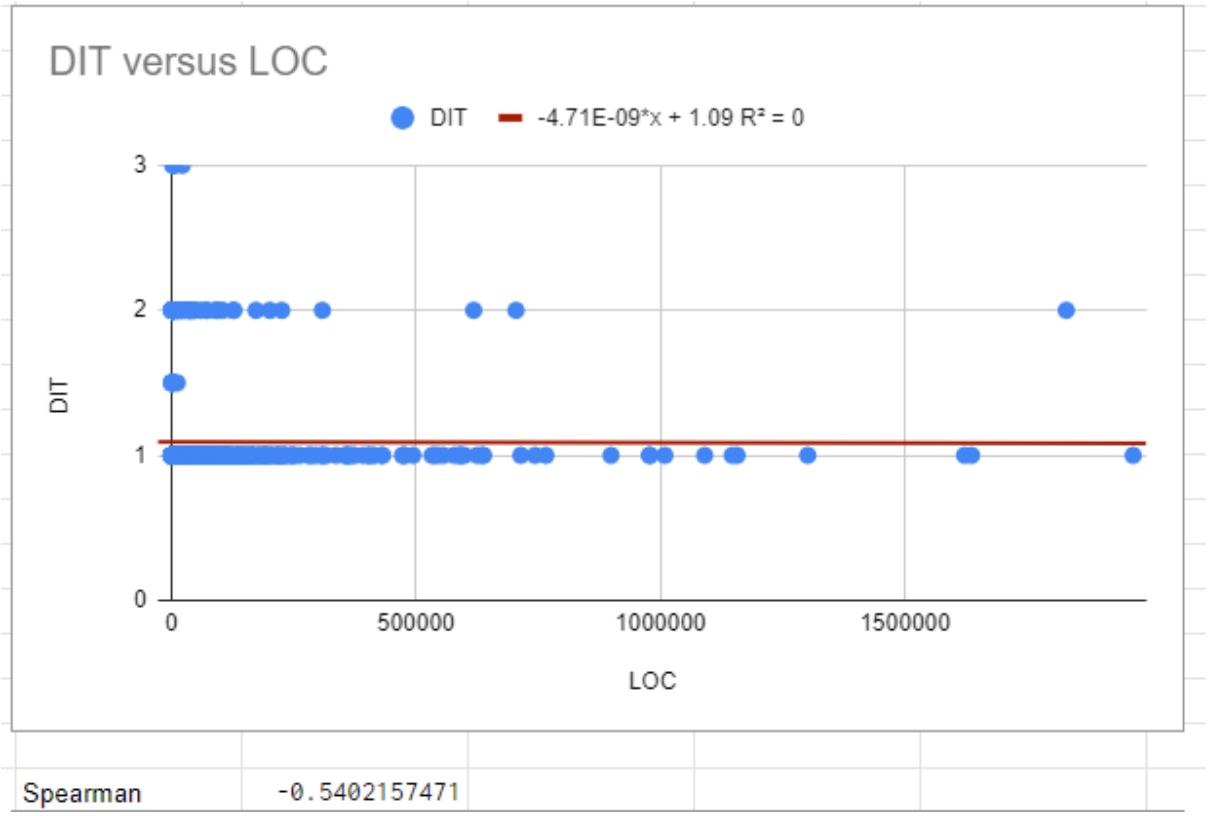


Figura 7. Gráfico de dispersão número de linhas vs DIT

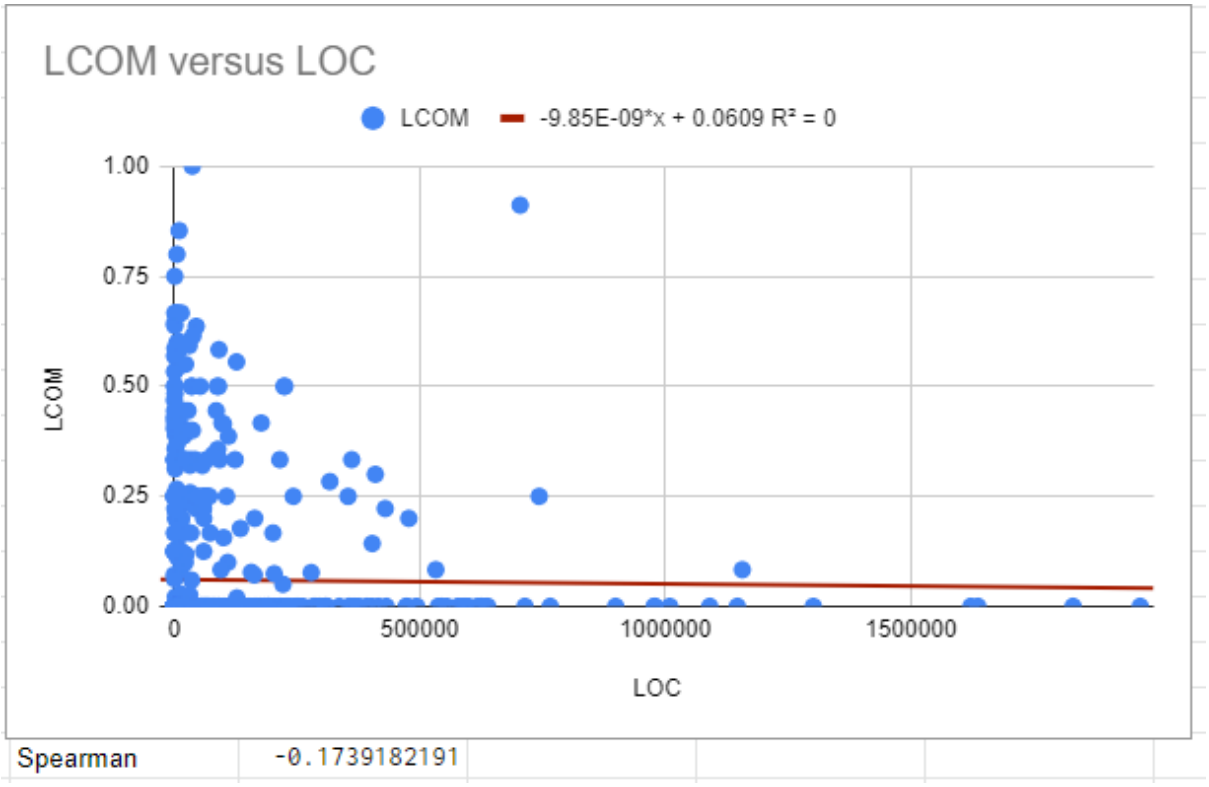
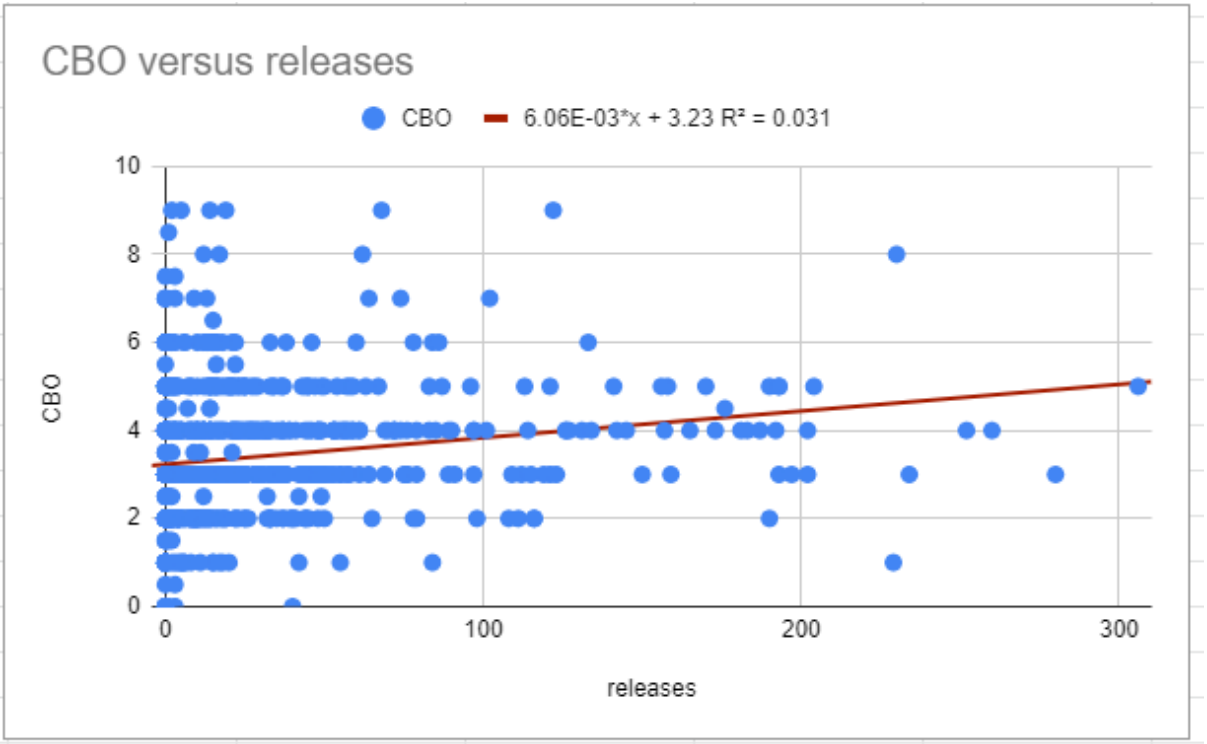


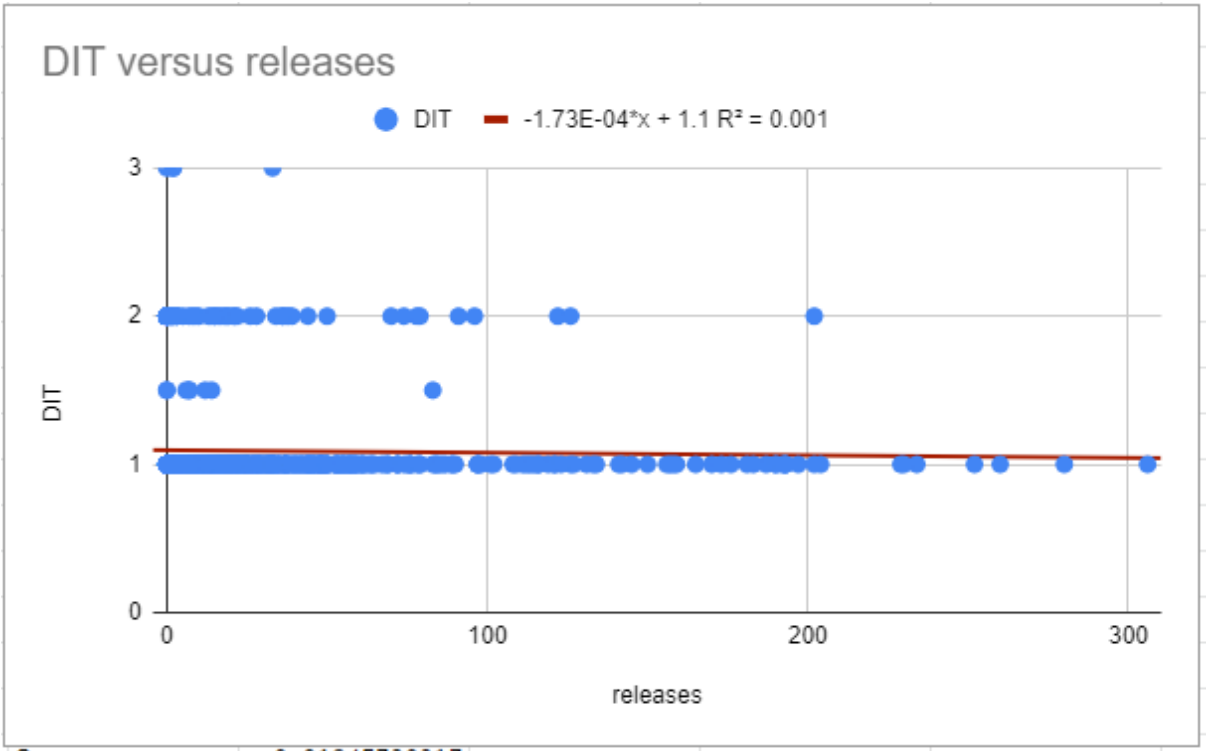
Figura 8. Gráfico de dispersão número de linhas vs LCOM

Agora falando do número de releases dos repositórios, as Figuras 9, 10 e 11 mostram o gráfico de dispersão em relação às métricas de qualidade CBO, DIT e LCOM, apresentando os coeficientes de correlação de Spearman de 0.44, 0.02 e -0.31, respectivamente.



Spearman	0.4414968664

Figura 9. Gráfico de dispersão número de releases vs CBO



Spearman	0.01845730317

Figura 10. Gráfico de dispersão número de releases vs DIT

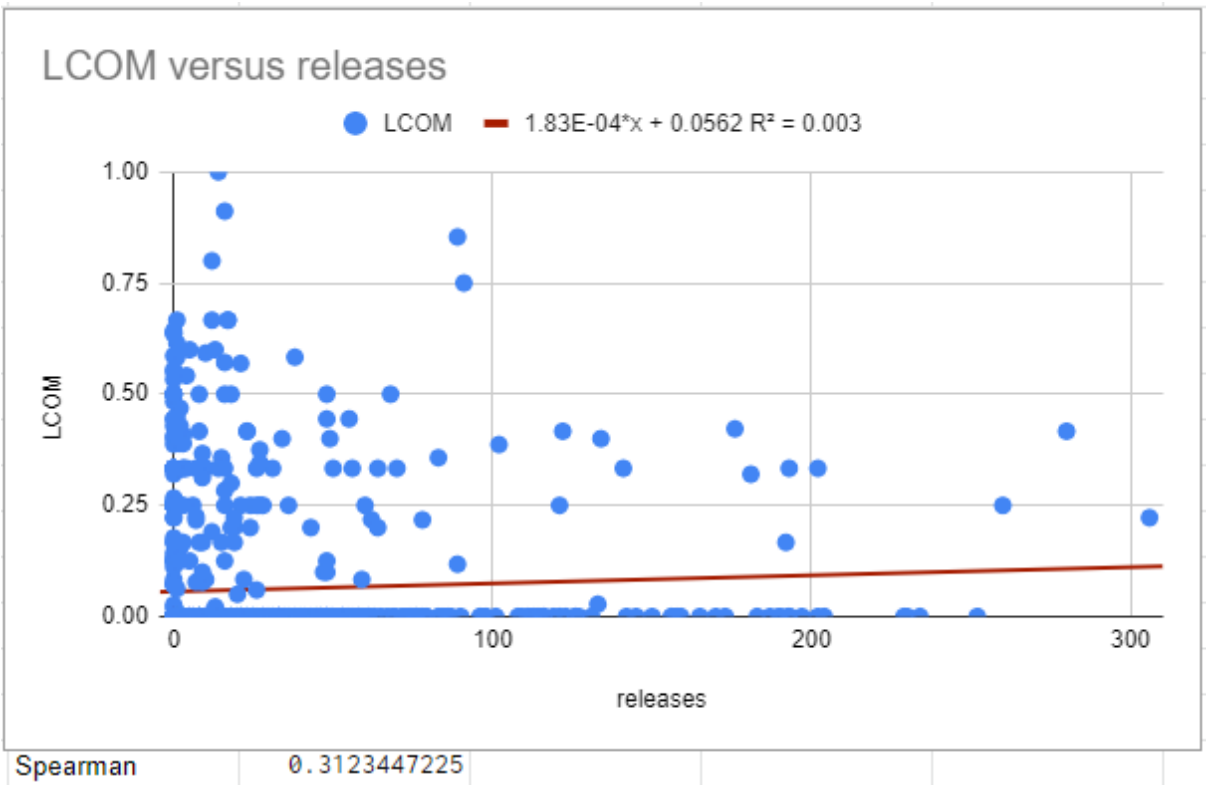
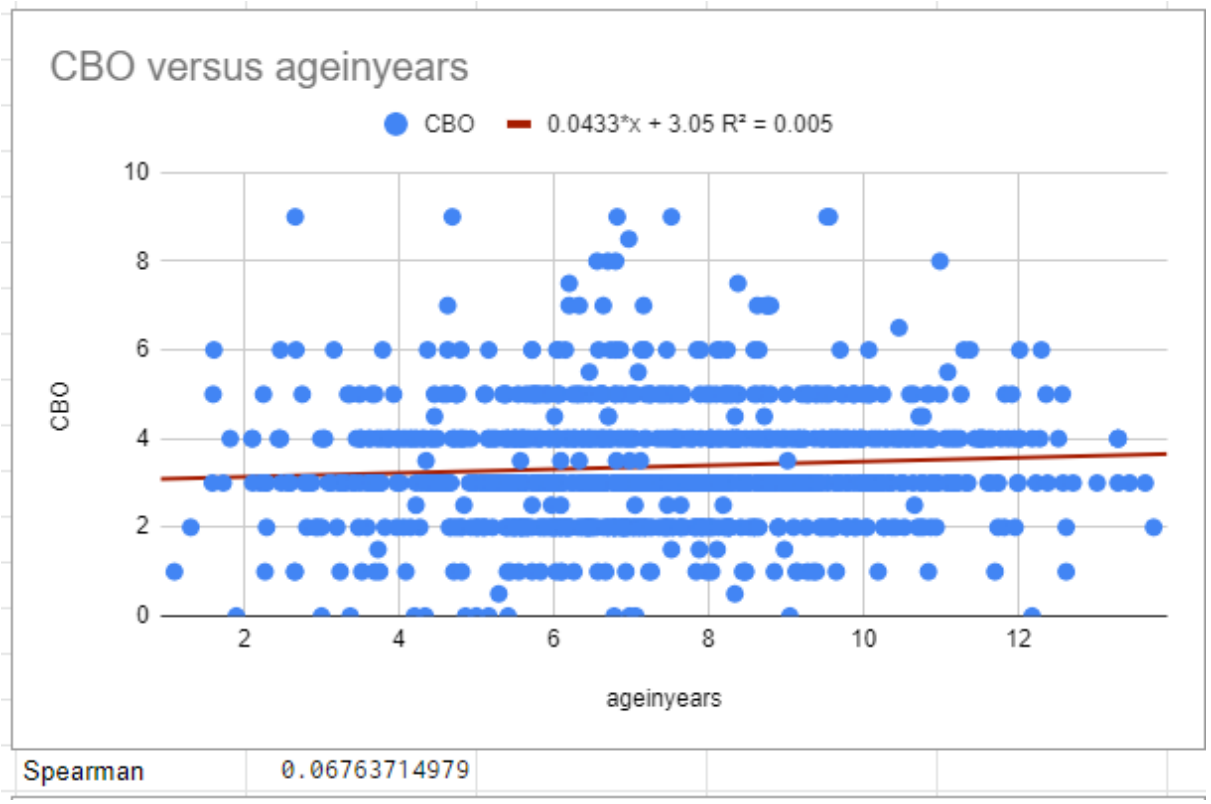


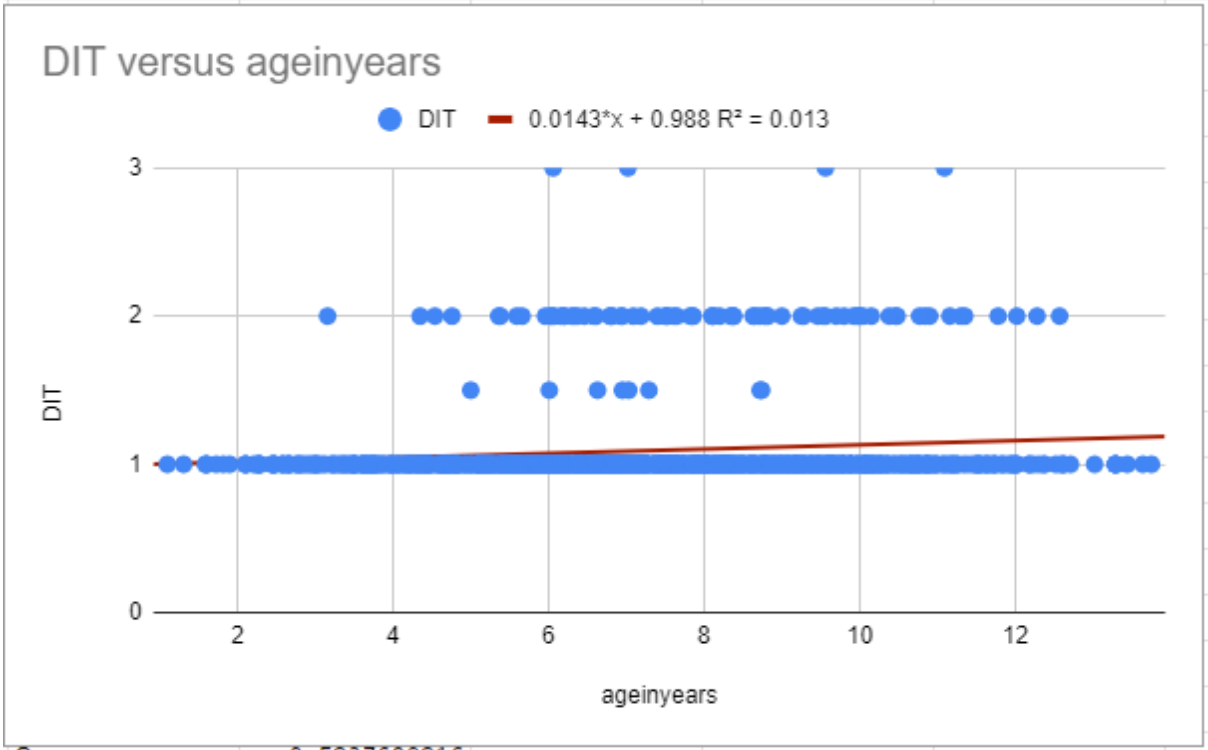
Figura 11. Gráfico de dispersão número de releases vs LCOM

Por último, em relação à idade em anos dos repositórios, as Figuras 12, 13 e 14 mostram o gráfico de dispersão em relação às métricas de qualidade CBO, DIT e LCOM, apresentando os coeficientes de correlação de Spearman de 0.07, -0.52 e -0.17, respectivamente.



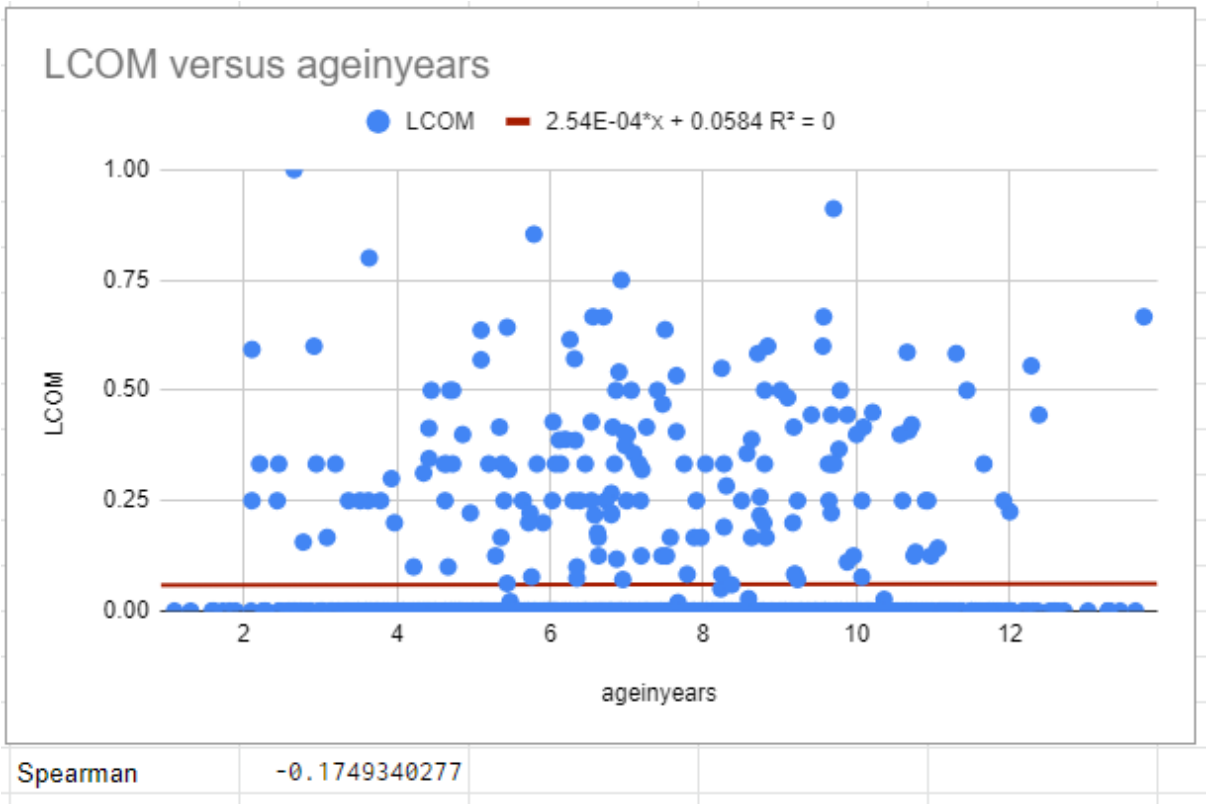
Spearman 0.06763714979

Figura 12. Gráfico de dispersão idade em anos vs CBO



Spearrman -0.5237609216

Figura 13. Gráfico de dispersão idade em anos vs CBO



Spearman	-0.1749340277

Figura 14. Gráfico de dispersão idade em anos vs CBO

4. Expectativas de resultados versus obtidos

A partir dos resultados coletados, é possível chegar a algumas conclusões sobre o tema pesquisado, relacionando com as hipóteses iniciais que foram levantadas no início da pesquisa.

Com relação a popularidade, foi possível perceber que não existe correlação entre o número de estrelas e o CBO do projeto. Ademais, com relação a popularidade e DIT, o resultado foi contrário à hipótese inicial, indicando que o número de estrelas tem uma correlação média com DIT, que é inversamente proporcional. Por fim, a relação entre popularidade e LCOM também surpreendeu, apesar da baixa correlação, ela é diretamente proporcional.

Sobre o tamanho dos sistemas, foi possível traçar uma pequena correlação direta entre o número de linhas e o CBO, sendo o contrário da hipótese inicial de que o CBO seria menor para grandes projetos. Além disso, a hipótese do DIT também foi contrariada, uma vez que foi possível constatar que essa métrica tem uma correlação inversa de nível médio com o

tamanho de linhas. Ao fim, apesar de um correlação baixa, o LCOM se comportou como o esperado, sendo inversamente proporcional ao tamanho do sistema.

Já em relação à atividade do repositório, a hipótese inicial do CBO foi provada, apresentando uma correlação direta, apesar desta ser baixa. Em vista do DIT, ele não apresenta nenhuma correlação com o número de releases. Para concluir os resultados de atividade, o LCOM apresentou uma relação inversa e baixa, o que diferiu da hipótese de que não haveria nenhuma correlação.

Para finalizar, fazendo análise em comparação com a maturidade de um software, ela não apresentou correlação com o CBO, o que contrariou a hipótese que as métricas tinham uma relação inversa. Já o DIT de um repositório de código maduro, tem uma tendência média a diminuir, outro indicador que não foi previsto nas hipóteses. Por fim, mesmo com uma relação baixa, a hipótese inicial em relação ao LCOM foi confirmada, pois apresentou uma correlação inversa.

Referências

[1] CK Metrics. Code metrics for Java code by means of static analysis. Disponível em: https://github.com/mauricioaniche/ck. Acesso em: 14 setembro de 2022.