Análise do impacto da expertise dos desenvolvedores no grau de manutenibilidade das contribuições ao HYPERLEDGER FABRIC

Moisés Cunha Pimentel[1], **Alcemir Rodrigues Santos**[1]

[1] Laboratório de Engenharia de Software (LES)
Universidade Estadual do Piauí (UESPI)
Piripiri – PI – Brasil

moisescunha.academic@gmail.com, alcemir@prp.uespi.br

Abstract. *Lehman's Software evolution Laws highlight the importance of maintain the ability to make changes in a software due the source-code quality decline over the years. However, only recently the software engineering researchers have turned their attention into the blockchain-orineted software. To the best of our knowledge, there is no evidence of the impact of the developers expertise in the level of maintainability. In this paper, we performed an empirical evaluation of it in the history of the* HYPERLEDGER FABRIC *evolution. Results show signs that the overload of the core developers may be affecting the quality of their contributions.*

Resumo. *As Leis da Evolução de Software de Lehman salientam a importância de manter-se a manutenibilidade de software devido ao declínio da qualidade do código-fonte ao longo do tempo. No entanto, somente recentemente pesquisadores da área de engenharia de software têm dado atenção à manutenção dos softwares orientados à blockchain. Para o melhor do nosso conhecimento, não existe evidência sobre o impacto do conhecimento dos desenvolvedores no grau de manutenibilidade de softwares orientados à blockchain. Este artigo conduziu um estudo empírico para avaliar este cenário na evolução do* HYPERLEDGER FABRIC. *Os resultados mostraram indícios de que a sobrecarga do time principal de desenvolvimento pode estar afetando a qualidade de suas contribuições.*

1. Introdução

As modificações feitas em código-fonte (*e.g.*, as alterações feitas com o propósito de adicionar novas funcionalidades ou correção de problemas) são conhecidas como a manutenção e evolução do software. A pesquisa em engenharia de software busca soluções eficientes para manter os projetos funcionando durante o seu ciclo de vida, pois é sabido que a manutenção prolonga a vida útil de um produto ou sistema [Misra 2008, de Oliveira et al. 2017]. A partir desta necessidade, define-se o termo "manutenibilidade", que refere-se a facilidade em que um software pode ser mantido e é conhecido como o atributo de qualidade de software. Desta maneira, se faz necessário mensurar o quão fácil é manter determinado software através de métricas associadas à manutenibilidade.

De fato, manutenção e manutenibilidade são termos diferentes, mas que estão interligados. Para Malhotra e Lata [Malhotra and Lata 2020], a manutenibilidade é a facilidade com que adicionamos, corrigimos ou alteramos o código. Basicamente, enquanto

a manutenção é ação de alterar o código, a manutenibilidade é o quão fácil é fazer essa alteração, permitindo uma avaliação e um retorno sobre a qualidade de um sistema.

Atualmente, existem inúmeros projetos de software baseados em *blockchain*. Estes projetos ficam cada vez maiores, à medida que novas funcionalidades e alterações são feitas. De fato, a modificação e aprimoramento do software tornam o código mais complexo e, assim, reduz a qualidade do software Oliveira *et al.* [de Oliveira et al. 2017]. O objetivo deste trabalho é analisar o grau de manutenibilidade do sistema orientado à *blockchain* conhecido como HYPERLEDGER FABRIC. Para isto, conduziu-se um estudo experimental para analisar a manutenibilidade desse sistema através de técnicas de mineração de software com a ferramenta PYDRILLER. Em conclusão, este artigo mostra indícios de que o grau de manutenibilidade do HYPERLEDGER FABRIC pode estar sofrendo alterações significativas ao longo do tempo.

O restante deste artigo está organizado como segue. A Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta a metodologia utilizada. Enquanto a Seção 4 apresenta os resultados obtidos, a Seção 5 discute sobre os dados obtidos e a Seção 6 enumera ameaças à validade do estudo. Por último, a Seção 7 aborda a conclusão deste artigo e aponta trabalhos futuros.

2. Trabalhos relacionados

Foi proposto em Heitlager *et al.* [Heitlager et al. 2007] um modelo de manutenibilidade conhecido como Software *Improvement Group - Maintainability Model* (SIG-MM), que calcula o índice de manutenibilidade através de aspectos gerais do projeto. Como complemento desse modelo, Di Biase *et al.* [di Biase et al. 2019] apresentou um modelo delta de manutenibilidade, que busca medir a manutenibilidade de código de granularidade fina. Esta é uma abordagem que analisa alterações de código individual, em vez de considerar toda a base de código como é feita no (SIG-MM), gerando dados mais precisos sobre pequenas alterações que os desenvolvedores fazem.

Samreen e Alalfi [Samreen and Alalfi 2023] acreditam que a manutenção de contratos inteligentes é muito única e diferente dos aplicativos de software tradicionais devido ao recurso principal da tecnologia *blockchain* para garantir a imutabilidade das informações. Ao analisar as aplicações descentralizadas (Dapps), verificou-se que elas são diferentes das aplicações tradicionais, possuindo um número maior de dependências de código em subcontratos ou bibliotecas. Ademais, foi observado uma grande clonagem de código em contratos inteligentes.

Observando-se por outra perspectiva, houve uma análise da similaridade entre a versão disponível no GitHub e a versão utilizada no Ethereum por Rodrigues *et al.* [Rodrigues et al. 2021]. Compreendeu-se a evolução de softwares de contratos inteligentes e observou-se que 51.8% dos projetos analisados foram enquadrados no Padrão de Evolução Delta, ou seja, aqueles cujo código-fonte dos contratos inteligentes encontrado no Etherscan apresenta baixa similaridade com o presente no Github desde o primeiro *commit*.

O presente estudo focou na análise do histórico de evolução de todo o software baseado em *blockchain*, diferentemente de Samreen e Alalfi [Samreen and Alalfi 2023], que analisaram somente os contratos inteligentes. Ademais, executou-se um estudo sobre a

evolução do grau de manutenibilidade, enquanto Rodrigues *et al.* [Rodrigues et al. 2021] fez um estudo sobre o tipo de evolução dos contratos inteligentes baseado em alguns padrões. Além disso, este trabalho lançou mão das métricas, limiares definidos e resultados do trabalho de Di Biase *et al.* [di Biase et al. 2019] para avaliação do grau de manutenibilidade.

3. Metodologia

O objetivo deste trabalho consistiu na **avaliação** da *evolução de manutenibilidade* do HYPERLEDGER FABRIC, **na perspectiva** *do engenheiro de software*, **com respeito** ao *grau de manutenibilidade de acordo com a métrica de variação de manutenibilidade encontrada na literatura*. Desta forma, definiu-se a seguinte questão de pesquisa:

QP: Qual o impacto da expertise dos desenvolvedores no grau de manutenibilidade de suas alterações no código-fonte de projetos orientados à *blockchain*?

É sabido da literatura, que dívidas técnicas são prejudiciais para a saúde do projeto ao logo do tempo e que custa caro o pagamento destas dívidas quando acumuladas por um tempo excessivo [Gomes et al. 2024]. Logo, ao responder-se nossa pergunta de pesquisa, seria possível identificar se ao logo do tempo o time de desenvolvimento tem aceitado alterações de que tem prejudicado o grau de manutenibilidade do projeto como um todo. Em seguida, apresenta-se detalhes de materiais e métodos utilizados na condução deste estudo.

3.1. Software alvo: HYPERLEDGER FABRIC

Neste trabalho, analisou-se a evolução da manutenibilidade do HYPERLEDGER FABRIC. O Fabric é um software orientado à *blockchain*, que implementa uma tecnologia distribuída de livro de razão corporativo, *i.e.* uma rede *blockchain* de propósito geral, que visa a construção de redes *blockchain* privadas.

Este projeto foi escolhido para a análise pois é o mais popular e antigo dentre os projetos *blockchain* de código-aberto[1]. Portanto, existe um longo histórico de alterações (*commits*) do código-fonte do HYPERLEDGER FABRIC disponível (desde 2016 à 2024). Além disso, existe um grande número de desenvolvedores com diferentes níveis de expertise, dentre aqueles que fizeram a evolução dele ao longo dos anos, uma vez que é mantido por uma comunidade de código-aberto.

3.2. Coleta, Extração e Processamento de dados

A Figura 1 apresenta um esquema representando as etapas do processo executado neste trabalho. O processo consistiu em 3 etapas: *(i)* a coleta do repositório; *(ii)* a extração dos dados; e *(iii)* o processamento dos dados. Toda a coleta, extração e processamento foram executadas com *scripts* `python` utilizando a biblioteca `pandas`[2] e `matplotlib`[3]. A

[1]O HYPERLEDGER FABRIC é mantido pela *Fundação Hyperledger*, com o foco em casos de uso empresarial e o objetivo de permitir a implementação e operação de aplicações de *blockchain*.

[2]O `pandas` é uma biblioteca de código aberto projetada para oferecer estruturas de dados e ferramentas de análise de dados eficientes e fáceis de usar.

[3]O `matplotlib` é uma das ferramentas mais populares e poderosas para a criação de gráficos e visualizações.

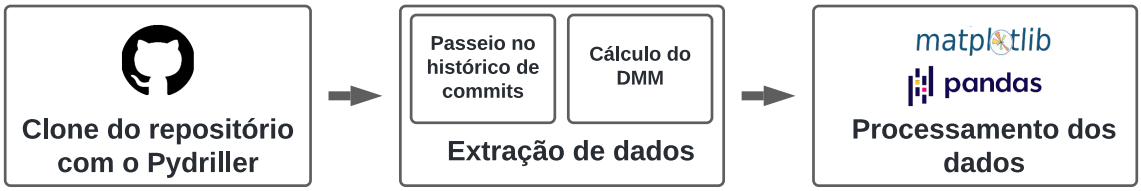

Figura 1. Processo de análise da evolução do HYPERLEDGER FABRIC.

estratégia de coleta do histórico de evolução do controle de versão foi dividida em duas maneiras. A primeira consistiu na coleta de todos os *commits* de cada *release* estável do HYPERLEDGER FABRIC. A segunda foi dividir todo o intervalo de desenvolvimento considerado em partes iguais de seis (6) meses e coletar os *commits* de cada intervalo. Passa-se a descrever cada uma destas estratégias em detalhes a seguir.

Estratégia 1 (divisão de *commits* por *release*): O repositório do HYPERLEDGER FABRIC possui *branches* para cada uma das *releases* estáveis do sistema. Cada uma das *branches* de *release* segue o seguinte padrão de nomenclatura **release-X.X**, onde "X" representa qualquer número. Por exemplo, a *branch* "release-1.0" segue esse padrão, mas a *branch* "feature/ca" não. Foram selecionados onze *branches* no total, e eles seguiram uma ordem crescente indo do "release-1.0" até o "release-2.5". Foram descartados os *commits* repetidos nas *releases*, ou seja, os *commits* analisados apenas poderiam participar de uma única *release*, sendo o critério de escolha a menor numeração da nomenclatura da *release*. Dessa forma, um mesmo *commit* presente em todas as *releases* será analisado somente na "release-1.0".

Estratégia 2 (divisão de *commits* por intervalo de tempo): A segunda estratégia consistiu em dividir o histórico de alterações em períodos iguais de duração. Cada intervalo possui 6 meses de dados dos *commits*, iniciando o primeiro período em 1º de julho de 2016 e finalizando o último em 1º de janeiro de 2024. A extração dos *commits* em cada *release* e período ocorreu com o auxílio do PYDRILLER [Spadini et al. 2018]. O PYDRILLER é uma ferramenta de mineração de repositórios que permite obter o histórico de alterações disponível em um dado repositório.

3.3. Métricas

Para responder às questões de pesquisa, considerou-se duas métricas: *(i)* o nível de expertise dos desenvolvedores; e *(ii)* o grau de manutenibilidade das mudanças deles. Em seguida, descreve-se cada uma delas.

Grau de contribuição dos desenvolvedores. Robles *et al.* [Robles et al. 2009] definiu a métrica *CoreTeam*, *i.e.* os desenvolvedores principais que efetuam a maioria dos *commits* de um sistema de código-aberto. Eles propuseram identificar os 10% ou 20% dos contribuidores que estão melhor ranqueados numa lista ordenada em ordem decrescente da quantidade de *commits* durante o período de interesse, considerando-os como principais. Neste estudo, considerou-se 20% dos desenvolvedores que mais contribuíram no período como o time principal.

Grau de manutenibilidade. Di Biase *et al.* [di Biase et al. 2019] define o *Delta Maintainability Score – DMM Score* partindo de cinco propriedades do sistema extraídas dos *commits* realizados durante a evolução do sistema de controle de versão. A Tabela

1 descreve cada uma destas propriedades. Neste estudo, no entanto, considerou-se apenas as propriedades implementadas[4] pelo PYDRILLER (*Unit Size*, *Unit Complexity* e *Unit Intefacing*), excluindo-se, portanto, as propriedades *Duplication* e *Module Coupling* do cálculo. O *DMM Score* é calculado a partir de uma média aritmética do valor calculado baseado nestas propriedades (Equação 1) e seu valor varia de 0 a 1, onde 0 representa a manutenibilidade mais baixa e 1 a mais alta.

$$\textit{DMM Score} = \frac{DS_{us} + DS_{uc} + DS_{ui}}{3} \tag{1}$$

Tabela 1. Propriedades utilizadas no cálculo do *DMM Score*. *(Adaptada de Di Biase et al. [di Biase et al. 2019].)*

Propriedade	Descrição	Critérios de baixo risco
Unit Size (DS_{us})	Tamanho das unidades de código-fonte, baseado no número de Linhas de Código (LOC), excluindo linhas consistindo apenas de espaços em branco ou comentários.	Unidades com no máximo 15 LOC
Unit Complexity (DS_{uc})	Grau de complexidade das unidades do código-fonte. A noção de unidade corresponde às menores partes executáveis do código-fonte, como métodos ou funções. A complexidade é medida usando a complexidade ciclomática de McCabe.	Unidades com Complexidade Ciclomática no máximo igual a 5
Duplication (DS_{d})	O grau de duplicação (textual) no código-fonte do produto de software. Uma linha de código é considerada redundante se fizer parte de um fragmento de código (maior que 6 linhas de código) repetido literalmente em pelo menos um outro local no código-fonte.	Nenhuma linha duplicada
Module Coupling (DS_{mc})	O acoplamento entre módulos é medido pelo número de dependências de entrada. A noção de módulo corresponde a um agrupamento de unidades relacionadas, normalmente um arquivo.	Módulos com no máximo 10 chamadas
Unit Interfacing (DS_{ui})	Tamanho das interfaces das unidades em termos do número de declarações de parâmetros de interface (parâmetros formais).	Unidades com no máximo 2 parâmetros são consideradas

4. Resultados

Nas seções a seguir, descreve-se os resultados obtidos a partir da coleta e processamento dos dados do HYPERLEDGER FABRIC. Primeiro, apresenta-se os dados de caracterização do histórico de desenvolvimento do sistema, considerando as duas estratégias de caracterização. Em seguida, apresenta-se a caracterização do nível de risco das contribuições efetuadas pelos dois grupos de contribuidores considerados.

4.1. Caracterização do processo de desenvolvimento

A Tabela 2 apresenta os dados de caracterização do histórico de desenvolvimento do sistema, considerando as duas estratégias de caracterização. As colunas indicadas com números indicam: uma *release* – no caso da Estratégia 1; ou um intervalo de seis meses – no caso da Estratégia 2. As linhas indicadas com `D` contém o número total de desenvolvedores envolvidos no desenvolvimento naquele intervalo. Mais especificamente, as linhas `DC` indicam o número de desenvolvedores *principais* e a as linhas `DP` indicam o número de desenvolvedores *periféricos* do intervalo. As linhas indicadas com a letra `C` contêm os

[4]O PYDRILLER foi construído para ser uma ferramenta multi-linguagem, e suas dependências dificultaram a implementação do cálculo de todas as propriedades consideradas no cálculo do *DMM Score* original [di Biase et al. 2019].

valores do número de *commits* daquele intervalo ou *release*. Para especificar a quantidade de *commits* para cada time, as linhas indicadas por `CC` englobam os números de *commits* dos desenvolvedores principais, enquanto as linhas `CP` incluem os números de *commits* dos periféricos.

A partir dos dados da Tabela 2, é possível perceber que, independente da estratégia de análise, o grupo de desenvolvedores principais (20% que mais contribuiu – `DC`) é também o grupo responsável por aproximadamente 80% das contribuições (`CC`). Este resultado corrobora com o resultado de Agrawal *et al.* [Agrawal et al. 2018], categorizando o HYPERLEDGER FABRIC como um projeto mantido por heróis. Neste caso, os heróis e o grupo de desenvolvedores principais são o mesmo grupo de desenvolvedores para este estudo.

Tabela 2. Caracterização dos intervalos considerados na análise do HYPERLEDGER FABRIC em cada uma das estratégias utilizadas.

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Estratégia 1 (divisão de *commits* por *releases*)															
D	137	107	77	70	118	154	61	109	85	107	71				
DC	27	21	15	14	23	30	12	21	17	21	14				
DP	110	86	62	56	95	124	49	88	68	86	57				
C	4285	1802	1402	928	1657	3864	367	728	566	580	302				
CC	3679	1487	1144	738	1429	3555	274	558	447	432	211				
CP	606	315	258	190	228	309	93	170	119	148	91				
Estratégia 2 (divisão de *commits* por intervalo de tempo)															
D	79	96	87	79	83	89	78	83	60	65	45	43	33	40	25
DC	15	19	17	15	16	17	15	16	12	13	9	8	6	8	5
DP	64	77	70	64	67	72	63	67	48	52	36	35	27	32	20
C	1226	2904	1301	1697	1701	2036	1452	736	448	296	169	109	134	165	154
CC	931	2343	1040	1365	1384	1810	1227	581	338	220	106	66	73	107	99
CP	295	561	261	332	317	226	225	155	110	76	63	43	61	58	55

C: número de commits; **D:** número total de desenvolvedores; **DC:** número de desenvolvedores core; **DP:** número desenvolvedores periféricos; **CC:** número de *commits* dos principais; **CP:** número de *commits* dos periféricos.

4.2. Caracterização do grau de manutenibilidade

Os resultados são apresentados por meio de quatro gráficos nas Figuras 2 e 3, sendo um para os resultados do *DMM Score*, e um para cada propriedade avaliada (*i.e.*, *Unit Size*, *Unit Complexity* e *Unit Interfacing*). A Figura 2 apresenta o resultado da estratégia de divisão por *releases*, em que no eixo `x` há 11 intervalos, enquanto A Figura 3 apresenta o resultado da estratégia de divisão por intervalo de tempo, onde há 15 intervalos. Cada intervalo apresenta um *box plot* para cada grupo de desenvolvedores considerados na extração de dados. A cor azul representa o grupo de desenvolvedores principais e a cor laranja o grupo de desenvolvedores periféricos.

Analisando mais a fundo o *DMM Score*, verifica-se que cada propriedade do sistema é diferente para cada estratégia nas Figuras 2 e 3. Na Figura 2, percebe-se que a maioria das medianas ficaram abaixo de $0,5$ para o *Unit Size*, ou seja, tiveram alterações mais distantes do perfil de baixo risco desta propriedade. Além disso, as propriedades *Unit Complexity* e *Unit Interfacing* revelaram medianas mais próximas do baixo risco, com valores acima de $0,7$ e $0,8$, respectivamente.

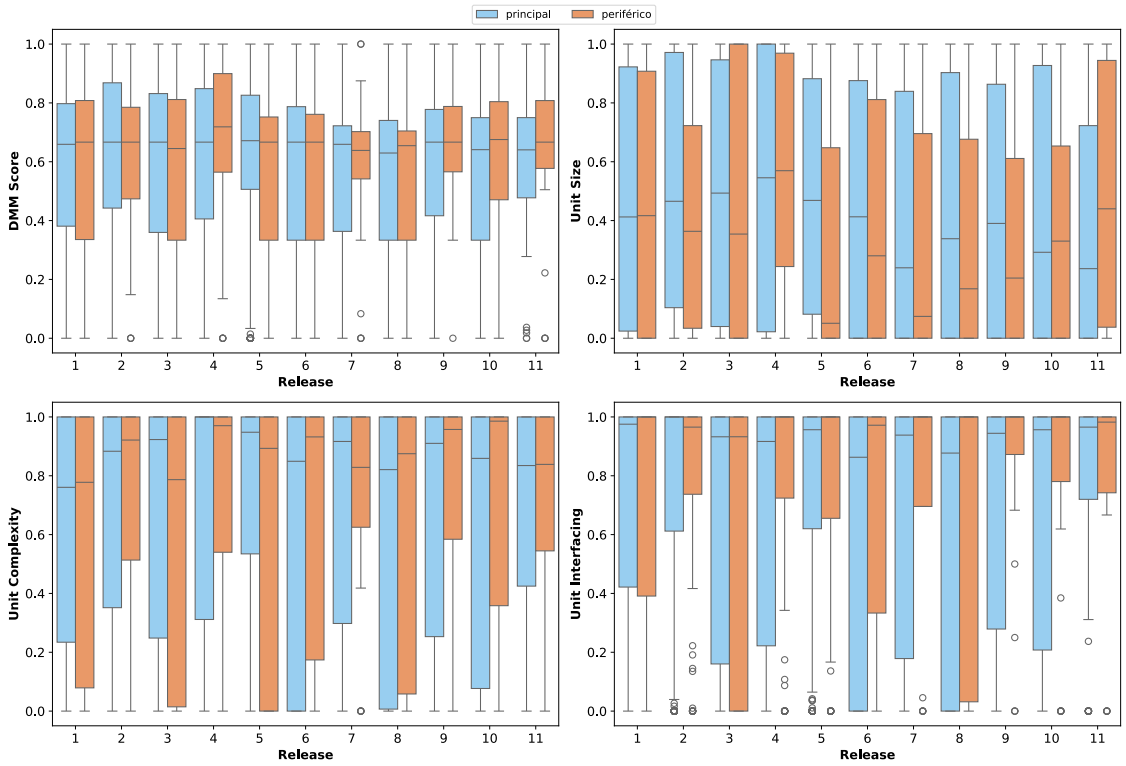

Figura 2. Distribuição dos resultados encontrados para valores do *DMM Score* e propriedades *Unit Size*, *Unit Complexity* e *Unit Interfacing* considerando a divisão de acordo com a Estratégia 1

A Figura 3 também exibiu o gráfico para a propriedade *Unit Size* com a maioria das medianas abaixo de $0,5$. Entretanto, os gráficos das propriedades *Unit Complexity* e *Unit Interfacing* revelaram uma situação diferente quando comparados com os gráficos da Figura 2. A primeira propriedade apresentou medianas acima de $0,5$, enquanto a segunda teve medianas acima de $0,7$. Estas duas medianas da Figura 3 são consideradas positivas a partir do perfil de baixo risco de cada propriedade.

Ao comparar os dois tipos de desenvolvedores no gráfico do *DMM Score* da Figura 3, nota-se que as medianas estão próximas até o 6° intervalo. Do 7° período em diante, percebe-se um aumento na mediana dos periféricos com exceção do 9° e 12° intervalos. Estes valores indicam que os desenvolvedores periféricos executaram alterações mais próximas do ideal de alta manutenibilidade.

Depois, executando a mesma análise para Figura 2, observa-se que as medianas estão semelhantes na 1°, 2°, 5°, 6° e 9° *releases*. A 3° e 7° *releases* tiveram valores levemente maiores para os desenvolvedores principais, enquanto a 4°, 8°, 10°, e 11° *releases* apresentaram medianas menores em comparação com o time dos periféricos. Nas últimas duas *releases* é visto uma manutenibilidade maior para os desenvolvedores periféricos.

Por fim, ao analisar a Figura 3, observa-se que todas as medianas ficaram acima do $0,5$ no gráfico do *DMM Score*. Isso indica que metade dos dados de cada *box plot* está mais próximo do baixo perfil de risco proposto por Di Biase *et al.* [di Biase et al. 2019]. A Figura 2 revela que a Estratégia 1 gerou valores ainda maiores de manutenibilidade,

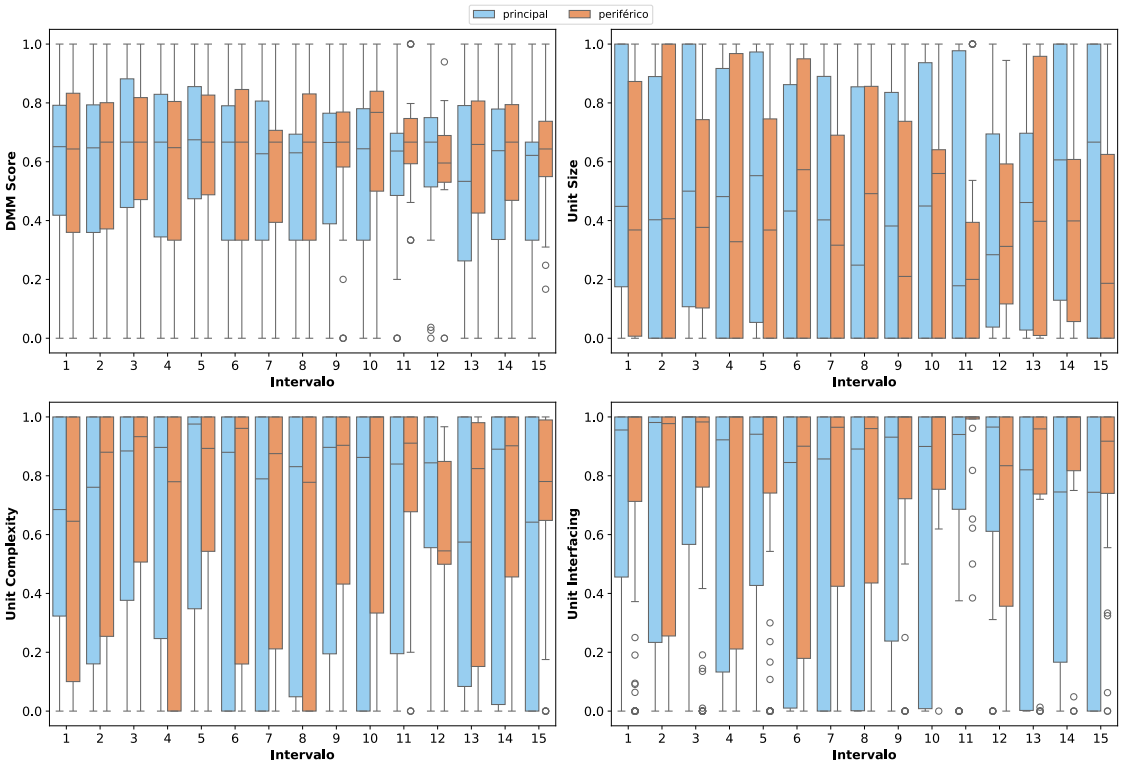

Figura 3. Distribuição dos resultados encontrados para valores do *DMM Score* e propriedades *Unit Size*, *Unit Complexity* e *Unit Interfacing* considerando a divisão de acordo com a Estratégia 2

onde todas as medianas ficaram acima de $0,6$. Ademais, todos máximos e mínimos de cada *blox plot* apresentado teve seus valores nos limites da métrica do *DMM Score*.

5. Discussão

Para conduzir a discussão dos resultados, deve-se retomar a questão de pesquisa: *"Qual o impacto da expertise dos desenvolvedores no grau de manutenibilidade de suas alterações no código-fonte de projetos orientados à blockchain?"*. Cada uma dessas estratégias apresentou bons valores no perfil de baixo risco na métrica do *DMM Score*, porém não retratando devidamente cada propriedade do sistema analisada individualmente. Tratando-se de uma média aritmética, o *DMM Score* omite algumas informações relevantes das propriedades que compõem a métrica, neste caso em específico o *Unit Size*, cuja medianas ficaram mais distante do perfil de baixo risco em ambas as estratégias. Para melhorar esse cenário, é recomendado o uso de refatoração para tornar as unidades de código menores e mais próximas do limiar de baixo risco como mostra a Tabela 1.

Cada grupo de desenvolvedores exibiu dados diferentes em todos os *box plots*. Percebe-se que o time dos principais é capaz de escrever unidades de código menores em relação ao grupo dos periféricos. Por outro lado, os resultados referentes às propriedades *Unit Complexity* e *Unit Interfacing* foram majoritariamente menores para os desenvolvedores principais. O fato de que o grupo de principais ter sido caracterizado também como heróis do projeto (ao fazer 80% das contribuições) como definido por Agrawal *et al.* [Agrawal et al. 2018], pode sugerir sobrecarga destes desenvolvedores.

No mesmo estudo, Agrawal *et al.* [Agrawal et al. 2018] mostrou que este tipo de concentração de trabalho é comum em projetos de média e grande escala. Além disso, eles concluíram que a presença desse tipo de heróis em projetos de software não impactam na taxa de *issues* fechadas e nem no tempo necessário para resolver *bugs* ou adicionar melhorias. No entanto, nossos resultados trazem indícios de que tais contribuições inserem mudanças mais distantes do perfil de baixo risco para a manutenibilidade, quando em comparação com as contribuições feitas pelo grupo de desenvolvedores periféricos.

Para reverter o cenário, o ideal seria diminuir a carga de trabalho entre trabalhadores, tornando-a mais balanceada. Isso abriria espaço para a melhoria das práticas de programação, tais como produzir unidades de código com complexidade ciclomática menor para aumentar o nível da propriedade *Unit Complexity*, e diminuir a quantidade de parâmetros formais das interfaces para progredir com a propriedade *Unit Interfacing*.

6. Ameaças à validade

Embora o máximo de cuidado seja tomado na preparação, execução e apresentação de estudos empíricos, é importante analisar as ameaças à sua validade. A seguir, apontamos algumas, bem como o que foi feito para minimizar os seus efeitos.

Primeiramente, a quantidade da amostra foi reduzida a um único software. Porém, tratando-se de um software mais antigo e com muitas contribuições, foi possível recuperar uma grande quantidade de intervalos em cada uma das estratégias de coleta de dados. Ademais, existe uma preocupação na implementação parcial do *DMM Score* pelo PYDRILLER. No entanto, como as propriedades do sistema são calculadas independente, é válido que haja a análise separada de cada propriedade disponível.

7. Conclusão e Trabalhos Futuros

Neste artigo, analisou-se o impacto da expertise dos desenvolvedores no grau de manutenibilidade das alterações no código-fonte do HYPERLEDGER FABRIC. Para isso, dividiu-se o histórico da evolução do sistema por *releases* e por intervalos fixos de tempo e mediu-se o grau de manutenibilidade com a métrica *DMM Score* identificada na literatura.

Os resultados mostraram que há uma necessidade de melhoria do código do HYPERLEDGER FABRIC, principalmente relacionado ao tamanho da unidade de código. Boas práticas sempre são bem-vindas, e a refatoração, diminuição da complexidade ciclomática e diminuição dos parâmetros formais são fortes aliados para aumentar a manutenibilidade de um software, além da remoção de duplicações e diminuição do acoplamento entre módulos.

Os resultados apontaram uma diferença no grau de manutenibilidade das mudanças realizadas no código-fonte por cada um dos grupos de desenvolvedores, independentemente do intervalo dos *commits* considerado para definir o time principal. O time dos periféricos conseguiu contribuir com códigos mais manuteníveis de acordo com as propriedades *Unit Complexity* e *Unit Interfacing*. Desta forma, os resultados inferiores do grupo principal podem estar ligados a sobrecarga destes desenvolvedores.

Como trabalhos futuros, é possível implementar o cálculo para os valores referentes às propriedades que não foram incluídas no cálculo do *DMM Score* pelo PYDRILLER.

Além disso, é possível ainda lançar mão de outras métricas para avaliar a evolução da manutenibilidade, bem como avaliar outros sistemas baseados em *blockchain* para maior generalização dos sistemas.

Referências

Agrawal, A., Rahman, A., Krishna, R., Sobran, A., e Menzies, T. (2018). We don't need another hero? the impact of "heroes"on software development. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP '18, page 245–253, New York, NY, USA. Association for Computing Machinery.

de Oliveira, R. P., Santos, A. R., de Almeida, E. S., e da Silva Gomes, G. S. (2017). Evaluating lehman's laws of software evolution within software product lines industrial projects. Journal of Systems and Software, 131:347–365.

di Biase, M., Rastogi, A., Bruntink, M., e van Deursen, A. (2019). The delta maintainability model: Measuring maintainability of fine-grained code changes. In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), pages 113–122. IEEE.

Gomes, T. B. S., de Moura Loiola, D. A., e Santos, A. R. (2024). Technical debt tools: a survey and an empirical evaluation. Journal of Software Engineering Research and Development, 12(1):8–1.

Heitlager, I., Kuipers, T., e Visser, J. (2007). A practical model for measuring maintainability. In 6th international conference on the quality of information and communications technology (QUATIC 2007), pages 30–39. IEEE.

Malhotra, R. e Lata, K. (2020). A systematic literature review on empirical studies towards prediction of software maintainability. Soft Computing, 24(21):16655–16677.

Misra, K. B. (2008). Maintenance engineering and maintainability: An introduction. Handbook of performability engineering, pages 755–772.

Robles, G., Gonzalez-Barahona, J. M., e Herraiz, I. (2009). Evolution of the core team of developers in libre software projects. In 2009 6th IEEE international working conference on mining software repositories, pages 167–170. IEEE.

Rodrigues, A., Araújo, A., Paixao, M., e Soares, P. (2021). Caracterizando a evolução de software de contratos inteligentes: Um estudo exploratório-descritivo utilizando github e etherscan. In Anais do IX Workshop de Visualização, Evolução e Manutenção de Software, pages 11–15, Porto Alegre, RS, Brasil. SBC.

Samreen, N. F. e Alalfi, M. H. (2023). An empirical study on the complexity, security and maintainability of ethereum-based decentralized applications (dapps). Blockchain: Research and Applications, 4(2):100120.

Spadini, D., Aniche, M., e Bacchelli, A. (2018). Pydriller: Python framework for mining software repositories. In The 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).