Caracterizando a evolução de software de contratos inteligentes: Um estudo exploratório-descritivo utilizando GitHub e Etherscan

Alan Rodrigues
Allysson Allex Araújo
alan.rodrigues434@gmail.com
allysson.araujo@crateus.ufc.br
Grupo de Estudos em Sistemas de Informação e Inovação Digital (GESID)
Universidade Federal do Ceará (UFC)
Crateús, Ceará, Brasil

Matheus Paixao
Pamella Soares
matheus.paixao@uece.br
pamella.soares@aluno.uece.br
Programa de Pós-Graduação em Ciência da Computação (PPGCC)
Universidade Estadual do Ceará (UECE)
Fortaleza, Ceará, Brasil

RESUMO

Blockchain tem sido enquadrada como uma nova infra-estrutura disruptiva baseada na internet. Parcela desse potencial advém do fortalecimento de plataformas públicas de *blockchain*, como a Ethereum, as quais viabilizam Aplicações Descentralizadas. Tais soluções são baseadas em contratos inteligentes (CIs) e lidam com restrições específicas que desafiam a evolução de software, como a imutabilidade de dados e o acesso transparente ao código-fonte. Em particular, a transparência de código pode ser observada através de ferramentas como o Etherscan, a qual provê acesso público a uma vasta quantidade de informações sobre os CIs implantados na Ethereum. Além disso, pode-se observar organizações oriundas desse ecossistema aderindo à prática de desenvolvimento *open source* dos CIs, incluindo o amplo uso do GitHub. Esse rico cenário motivou a necessidade de conduzir um estudo exploratório-descritivo baseado em mineração de repositórios de software para compreender a evolução de software de CIs através da avaliação de similaridade entre a versão disponível no GitHub e versão utilizada na Ethereum (e auditável via Etherscan). À luz de uma análise quali-quantititativa de 27 CIs, este artigo contribui ao 1) caracterizar quatro padrões que denotam diferentes comportamentos evolutivos dos CIs e 2) abordar um método experimental baseado em comparação de *strings* para avaliar a similaridade entre diferentes versões de CIs.

PALAVRAS-CHAVE

Contratos Inteligentes. Mineração de Repositórios. Ethereum. Etherscan. GitHub.

1 INTRODUÇÃO

O *blockchain*, que teve seu desenvolvimento difundido após a criação da criptomoeda Bitcoin [9], constitui uma das novas infraestruturas digitais baseadas na internet com amplo potencial disruptivo [12]. Em termos genéricos, o *blockchain* funciona como uma tecnologia distribuída e descentralizada capaz de armazenar criptograficamente um registro imutável de eventos lineares de transações [15]. No entanto, apesar de notoriamente impactante, o cenário de uso do Bitcoin é restrito, devido sua aplicação ter como foco principal o armazenamento e a transferência de valores. Visando contornar tal limitação, o surgimento da Ethereum, como uma nova plataforma de *blockchain* pública, ampliou as possibilidades de desenvolvimento de soluções baseadas em *blockchain* [6].

A Ethereum alavancou uma tecnologia chamada contratos inteligentes (CIs) (em inglês, *smart contracts*), tornando possível que programas *Turing-complete* possam ser executados na *blockchain* [5]. Em suma, CIs são códigos executáveis, armazenados de forma distribuída e que contém termos contratuais formalizados para realizar acordos de relacionamento [17]. Assim, por meio desta oportunidade tecnológica, o desenvolvimento de Aplicações Descentralizadas (do inglês, *Decentralized Applications* ou *dApps*) tem se fortalecido cada vez mais [13]. Nesse sentido, com a lapidação do ecossistema de soluções baseadas na Ethereum, ferramentas de apoio à comunidade também começaram a se consolidar no mercado, como é o caso do Etherscan[1] o qual funciona como *block explorer* da Ethereum e viabiliza qualquer pessoa visualizar as transações e informações referente a cada CI implantado.

Conforme pode-se pressupor, CIs apresentam características próprias que os diferenciam de "softwares tradicionais" [5]. Após compilado, o *bytecode* de um CI deve ser implantado no *blockchain*, onde o mesmo será posteriormente executado. Logo, uma vez que um CI é implantado no *blockchain*, ele nunca mais poderá ser alterado devido às propriedades criptográficas da tecnologia [12]. Em caso de atualização, por exemplo, um novo CI deve ser implantado em um novo nó do *blockchain*, onde todas as *dApps* que fazem uso deste CI devem ser atualizadas para referenciar o novo CI. Diferente do contexto usual no qual o software pode ser atualizado diretamente, para modificar um CI, os desenvolvedores o implantam novamente e descartam a versão antiga. Embora exista a possibilidade de desenvolver CIs atualizáveis usando `DelegateCall` ou destruindo os CIs com erros usando a função `SelfDestruct`, ambos os métodos podem aumentar os riscos de segurança, a desconfiança dos usuários e os custos de desenvolvimento [5].

Além da característica de imutabilidade, os CIs também lidam com restrições relacionadas à execução. Os CIs são executados de forma descentralizada, em máquinas denominadas "mineradores", onde o custo (usualmente em criptomoedas nativas da plataforma, como o Ether na Ethereum) de execução aumenta de acordo com a quantidade de instruções realizadas. Dito isso, há uma latente necessidade de atenção à complexidade do SC desenvolvido [7]. Ao analisar as diferenças dos CIs em contraste à "softwares tradicionais", constatam-se novos desafios à Engenharia de Software, incluindo aqueles relacionados à evolução de software [3].

[1] https://etherscan.io

Diante desse cenário intricado por novos desafios, verifica-se, por parte de envolvidos nesse ecossistema de soluções, a adesão às práticas de desenvolvimento *open source*, com destaque para o uso do GitHub [21]. Logo, assume-se que os CIs são evoluídos colaborativamente no GitHub até o estágio de *deploy* na Ethereum. Todavia, conforme alertado por Reibel *et al.*, é possível que as versões dos CIs no GitHub e na rede Ethereum difiram entre si [14]. Essa perspectiva, em especial, motiva questionamentos sobre como se dá a relação entre o CI exposto e desenvolvido no GitHub e o que realmente está sendo utilizado na Ethereum e, consequentemente, auditável via Etherscan. A pertinência dessa provocação se justifica ainda mais relevante tendo em vista as particularidades envolvidas no processo de evolução de software de CIs, como a natureza inalterável dos dados registrados em *blockchain* e a garantia da simplicidade para redução de custos operacionais em criptomoedas e transparência de acesso ao código-fonte do CI [4]. A partir desse panorama, ainda é válido destacar a presença de diferentes trabalhos [1, 10, 11, 18] com foco em desafios relativos ao desenvolvimento e manutenção de CIs, no entanto, nenhum foca exclusivamente na relação e similaridade entre o que está exposto no Etherscan e o que foi desenvolvido de forma *open source* no GitHub à luz da evolução de software.

Considerando a lacuna de pesquisa apresentada, este estudo objetiva responder a seguinte questão de pesquisa: *Como ocorre o processo de evolução de software em CIs a partir da relação entre repositórios de código-fonte disponíveis no GitHub e Etherscan?* Para responder tal indagação, conduziu-se um estudo exploratório-descritivo de natureza quali-quantitativa baseado em Mineração de Repositório de Software (MRS). O usufruto de MRS demonstra-se especialmente pertinente ao contexto deste trabalho devido à capacidade de investigar empiricamente e de forma sistemática repositórios de código-fonte para obtenção de *insights* sobre o processo de evolução de software [8]. Fundamentado a partir de um experimento computacional, o presente estudo investigou 27 CIs de projetos listados entre aqueles com maior valor de mercado de acordo com o *ranking* "ERC20 *Top Tokens*" do Etherscan e que, por sua vez, dispusessem de um repositório *open source* no GitHub.

Portanto, em termos de contribuições, este trabalho avança no entendimento da evolução de software de CIs ao 1) caracterizar padrões de evolução de software quanto a similaridade e comportamento dos projetos desenvolvidos de forma *open source* no GitHub em contraste à versão do CI disponível no Etherscan e 2) explorar um método experimental baseado em comparação de *strings* para análise de similaridade entre versões distintas de CIs.

2 ESTUDO EMPÍRICO

Nesta seção, apresenta-se o escopo metodológico e a análise dos resultados. Por fim, são esclarecidas as ameaças à validade.

2.1 Procedimentos Metodológicos

Posicionando-se como exploratório-descritivo, esse estudo busca proporcionar um melhor entendimento do fenômeno sob estudo através da investigação e descrição sobre como ocorre o processo de evolução de software de CIs a partir de dados e informações disponíveis nos repositórios provenientes do GitHub e Etherscan. Quanto ao escopo metodológico, optou-se por um experimento computacional baseado em MRS. Sob o ponto de vista analítico, esta pesquisa teve um enfoque quali-quantitativo devido a necessidade de uma visão mista em decorrência da confluência de elementos quantitativos e subjetivos obtidos durante a coleta de dados.

Conforme ilustrado na Figura 1, o presente trabalho adotou um plano metodológico baseado em quatro etapas principais. Inicialmente, na **Etapa 1**, realizou-se uma análise no Etherscan para identificação de CIs do tipo ERC-20 aptos a serem investigados. Tendo em vista as limitações de tempo e processamento de dados, estabeleceu-se uma amostra de 30 CIs oriundos da lista ERC-20 *Top Tokens* do Etherscan, ou seja, a lista dos projetos com maior valor de mercado *(market cap)* no dia da seleção (03/08/2020).

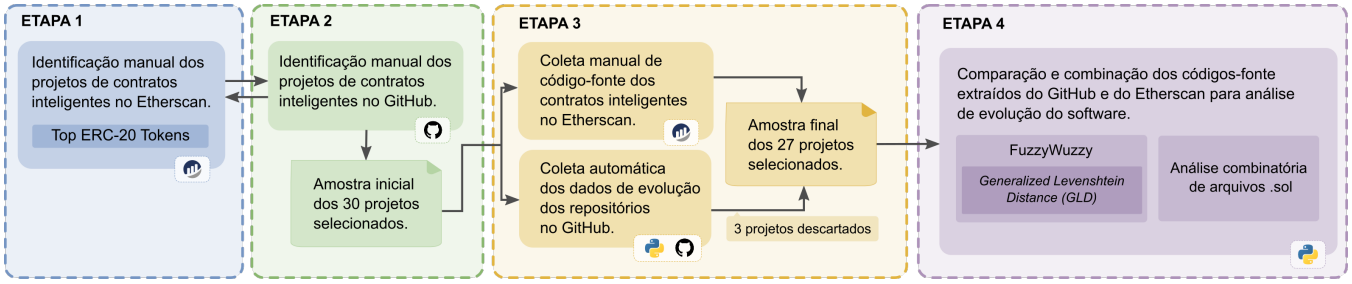
Em seguida, na **Etapa 2**, realizou-se, a partir dos CIs identificados, uma busca manual junto ao GitHub para identificação dos repositórios *open source*. Entretanto, foi possível constatar que parcela dos CIs não dispunham de repositórios no GitHub. Assim, substituiu-se os contratos em questão pelos contratos imediatamente subsequentes da lista ERC20 *Top Tokens* mas que também estivessem disponíveis no GitHub. Para se certificar da corretude do repositório selecionado, considerou-se os seguintes critérios: 1) O *nome do repositório* no GitHub deve ter alguma relação com o nome do CI no Etherscan; 2) O *tipo de linguagem de programação* descrito no repositório deve ser linguagem Solidity. Porém, tal condição não foi seguida em todos os casos devido o GitHub, muitas vezes, indicar a linguagem como sendo JavaScript. Nesse caso, realizou-se uma inspeção manual para certificar que a linguagem era Solidity; 3) Devem existir arquivos escritos no *formato de CIs* e com extensão "`.sol`"; 4) A *descrição do projeto*, quando disponível, deve ter relação com o CI disponível no Etherscan.

A **Etapa 3** objetivou concretizar a mineração de dados. O processo de coleta dos códigos-fonte dos CIs disponíveis no Etherscan foi realizado de forma manual, haja vista que o Etherscan fornece, através de sua versão web, fácil acesso ao arquivo[2] implementado em Solidity de cada CI disponível na Ethereum. Para cada projeto oriundo da amostra a ser investigado, i) realizou-se uma busca manual no site do Etherscan, ii) identificou-se o código-fonte do CI e, por fim, iii) realizou-se uma cópia do mesmo para uma base de dados interna para fins de análise. Em seguida, para coleta dos dados de evolução dos repositórios no GitHub, utilizou-se da API REST[3] disponibilizada pelo GitHub para a automatização do processo. Através de *scripts* em Python, coletou-se, para cada um dos 30 projetos no GitHub, a lista de *commits* e códigos-fonte dos CIs desde a início do projeto no GitHub. Assim, a partir da relação de *commits* obtidos, tornou-se possível o mapeamento de todas as versões do projeto ao longo de sua evolução. Todos os *scripts* e dados obtidos encontram-se no repositório de apoio deste trabalho [16].

A **Etapa 4**, de cunho analítico, teve o objetivo de desenvolver e aplicar um método experimental para comparar a similaridade da versão do CI disponível no Etherscan com todas as versões do referido CI disponíveis no GitHub ao longo de sua evolução. Para tal processo, primeiramente utilizou-se da função *Generalized Levenshtein Distance* (GLD) [20] para mensurar a porcentagem de similaridade de *string* entre os referidos arquivos. Em termos operacionais,

[2]https://etherscan.io/address/0x514910771af9ca656af840dff83e8264ecf986ca#code
[3]https://docs.github.com/pt/rest

Figura 1: Procedimentos Metodológicos



adotou-se a função `fuzz.token_sort_ratio()`, disponível no pacote FuzzyWuzzy da linguagem Python, que utiliza cálculo de GLD como princípio para realizar as comparações. As funções do tipo `fuzz.token` "*tokenizam*" as *strings* e as pré-processam, colocando-as em minúsculas e eliminando a pontuação. Em específico, no caso de `fuzz.token_sort_ratio()`, os *tokens* de *string* são classificados em ordem alfabética e depois agrupados [2]. Outro motivo para a utilização desta função foi devido ao fato da mesma verificar se as *strings* comparadas são as mesmas independentemente da posição do texto. Como *pretty printing*, também elaborou-se uma expressão regular em Python (disponível em [16]) para remoção dos espaços em branco e dos comentários nos códigos-fonte dos CIs.

Adicionalmente, outro procedimento relevante conduzido durante a Etapa 4 diz respeito ao fato de que, muitas vezes, o CI implantado na Ethereum é dividido em diferentes arquivos `.sol` durante o desenvolvimento. Logo, tornou-se necessário realização de uma análise combinatória entre todos os arquivos `.sol` oriundos de cada versão do projeto no GitHub. Para tal fim, implementou-se um *script* em Python o qual verifica, para cada *commit*, a combinação entre arquivos `.sol` que apresenta maior porcentagem de similaridade quando comparado a versão do CI disponível no Etherscan. Entretanto, durante esse processo, três projetos tiveram que ser descartados devido à complexidade computacional necessária para realizar todas as comparações (haja vista a quantidade de *commits* e arquivos `.sol` em cada projeto, bem como a limitação de processamento da máquina disponível para os testes – Intel Core i5 2.5 Ghz, 500 GB de HD e 8GB de RAM). No caso do projeto Aragon, tem-se 2, $104098964X10^{12}$ combinações possíveis. Já Bancor possui 1, $74613564X10^{21}$ possibilidades. Por fim, o Syntethix Network Token apresentou um total de 2, $48577844X10^{31}$ combinações.

2.2 Resultados e Análises

Esta seção busca discutir o comportamento evolutivo dos CIs sob estudo e, assim, apresentar uma categorização quanto aos padrões de evolução identificados. Para traçar a linha evolutiva de cada projeto, realizou-se uma análise combinatória entre todos os arquivos `.sol` de cada versão do projeto no GitHub para verificar a versão com maior similaridade quando comparado ao CI disponível no Etherscan. Como limiar de análise, para fins didáticos, considerou-se que comparações que apresentem uma similaridade igual ou superior a 90% serão classificadas como *alta similaridade*; caso contrário, *baixa similaridade*. Diante dessa concepção, torna-se possível a definição de quatro padrões de evolução:

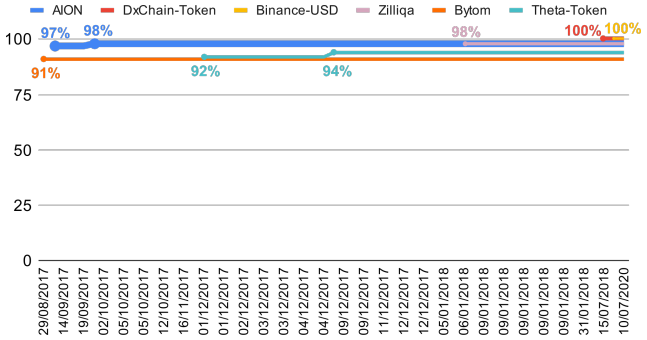
- **Padrão de Evolução Alpha**: O código-fonte dos CIs presentes no GitHub apresenta *alta similaridade* ao encontrado no Etherscan desde o primeiro *commit* e não muda até o último *commit* analisado;
- **Padrão de Evolução Beta**: O código-fonte dos CIs presentes no GitHub é de *alta similaridade* ao encontrado no Etherscan no início do desenvolvimento, porém em algum momento da evolução no GitHub é modificado e apresenta *baixa similaridade* até o último *commits* disponibilizado;
- **Padrão de Evolução Gama**: O código-fonte dos CIs presentes no GitHub apresenta *baixa similaridade* ao encontrado no Etherscan no início do desenvolvimento, porém em algum momento torna-se de *alta similaridade* e segue assim até o último *commit* analisado;
- **Padrão de Evolução Delta**: O código-fonte dos CIs presentes no GitHub apresenta *baixa similaridade* ao encontrado no Etherscan no início do desenvolvimento, e segue assim até o último *commit* analisado.

A partir dos padrões elencados, foi possível verificar que, ao todo, dos 27 projetos investigados, 14 deles são enquadrados como **Padrão de Evolução Delta**, representando assim, 51.8% da amostra. O **Padrão de Evolução Gama** foi a segunda categoria que apresentou mais projetos, contando com sete iniciativas (25.9%). Em seguida, com seis projetos (22.2%), tem-se o **Padrão de Evolução Alpha**. Ademais, nenhum projeto investigado foi classificado como pertencente ao **Padrão de Evolução Beta**, evidenciando, assim, a ausência de projetos cuja mudança destoa negativamente na similaridade. Dos 27 projetos analisados, 13 deles apresentaram acima de 90% de similaridade em algum momento da evolução do software. Por restrição de espaço, optou-se por disponibilizar mais detalhes descritivos sobre cada projeto no repositório de suporte [16].

Buscando aprofundar a análise dos padrões identificados, discute-se a seguir as evidências que denotam o comportamento diagnosticado para cada um. Primeiramente, a Figura 2 mostra uma visualização sobre a porcentagem de similaridade das versões disponíveis no GitHub, ao longo do desenvolvimento, quando comparadas à versão do Etherscan, referente aos projetos enquadrados como **Padrão de Evolução Alpha**. Conforme pode-se verificar, todos os projetos apresentam acima de 90% de similaridade desde seu início no GitHub. Os projetos Binance-USD, DxChain-Token e Zilliqa demonstram a porcentagem de similaridade estável durante todo o processo de evolução mapeado, respectivamente 100%, 100% e 98%. Em específico, o projeto Zilliqa não apresenta 100% de similaridade

porque o CI desenvolvido no GitHub foi decomposto em nove arquivos `.sol` separados, demandando linhas de código adicionais para declaração da versão do compilador e demais importações que, ao serem combinados, ocasionam na redução da similaridade. Para verificar a diferença de similaridade entre o CI disponível no GitHub e no Etherscan, realizou-se uma comparação manual considerando a versão com maior similaridade identificada no GitHub.

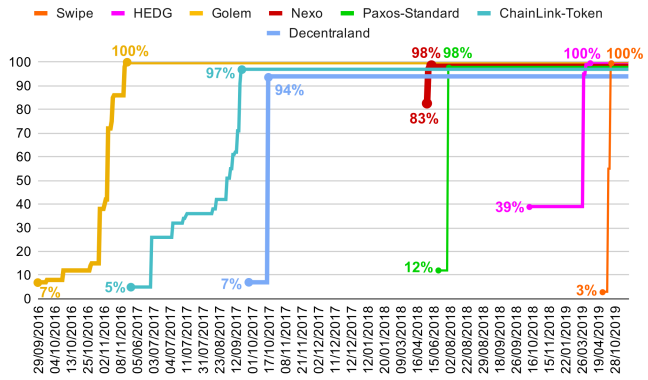
Figura 2: Análise de similaridade do Padrão Alpha



Ainda analisando a Figura 2, o Bytom também apresenta similaridade estável durante toda a evolução coletada, todavia é o menor valor (91%) dentre os projetos nesse padrão. Tal diferença se dá em decorrência do CI também ser desenvolvido em múltiplos arquivos `.sol` separados, mas também apresenta a substituição da função de conveniência `require` (que verifica invariantes, condições e lança exceções) por uma condicional `if` no corpo de três funções. Os projetos AION e Theta-Token apresentam um pequeno acréscimo de similaridade. No caso do AION, isso ocorreu devido às deleções do elemento `onlypayloadsize(2)` realizadas nas declarações de cinco funções, além da exclusão de um modificador. Ao analisar o Theta-Token, por sua vez, verificou-se que tal acréscimo se dá em virtude de deleções de três linhas no corpo da função `mint`, elevando a similaridade de 92% a 94%. De forma geral, pode-se constatar duas observações pertinentes A primeira, em relação ao processo de desenvolvimento dos CIs em arquivos separados e posteriormente agrupados para *deploy* na Ethereum. A segunda refere-se ao fato das modificações estarem atreladas à mudanças em diferentes funções, derivando assim uma perspectiva de trabalho futuro quanto ao entendimento da motivação de tais modificações.

Ao analisar a Figura 3 referente aos projetos categorizados como **Padrão de Evolução Gama**, verifica-se a presença de sete CIs cuja versão no GitHub apresenta inicialmente *baixa similaridade* em relação à versão disponível no Etherscan, porém em algum momento posterior torna-se de *alta similaridade* e segue assim até o último *commit* analisado. Nesse sentido, pode-se constatar que três projetos (Golem, HEDG e Swipe) apresentam 100% de similaridade, ou seja, evoluem até um estágio de total similaridade e permanecem assim. Por sua vez, a outra parcela dos projetos (ChainLink-Token, Decentraland, Nexo e Paxos-Standard) apresenta similaridade igual ou superior a 94%. Diante desse cenário, uma questão a se destacar diz respeito a dualidade de estratégias quanto ao processo de evolução de software. Enquanto nos projetos Golem e ChainLink-Token há uma progressão gradual de similaridade, os demais projetos apresentam *commits* "emblemáticos" cuja similaridade atinge um alto valor. Tal comportamento fica evidente, por exemplo, ao analisar o salto de 3% para 100% do CI referente ao projeto Swipe. Em síntese, verifica-se, a partir dos resultados atrelados ao Padrão de Evolução Gama, a presença de projetos cujo ritmo de evolução é mais constante e progressivo, enquanto outros se revelam menos graduais. Quanto aos projetos que não atingiram 100% de similaridade (Nexo e Paxos-Standard), ambos com 98% apresentam os mesmos motivos dos projetos do padrão anterior, ou seja, CIs desenvolvidos, respectivamente, em seis e quatro arquivos `.sol` separados, com várias declarações do modelo da versão do compilador e importações as quais causaram tal diferença entre as versões. Porém, no caso do Paxos, ainda há uma mudança entre a versão do compilador utilizada no GitHub (`solidity 0.4.21`) e a versão encontrada no Etherscan (`solidity 0.4.23`). Já para o projeto ChainLink-Token, com 97% de similaridade, além de conter as mesmas diferenças mencionadas anteriormente (nove arquivos `.sol`), apresenta a inclusão de duas funções (`contractfallback()` e `iscontract()`) no CI encontrado no Etherscan. Por fim, o projeto Decentraland, com 94% de similaridade, não atinge a similaridade máxima devido a modificação no nome de uma função (de `fakemana()` para `manatoken()`) e a exclusão de outra função chamada `setbalance()`.

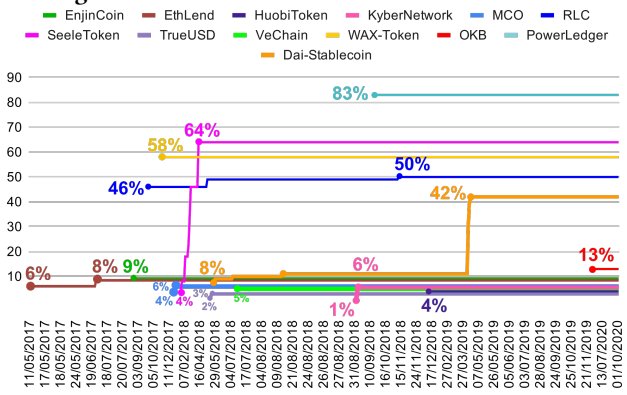
Figura 3: Análise de similaridade do Padrão Gama



Finalmente, o comportamento dos projetos caracterizados como **Padrão de Evolução Delta** é sintetizado através da Figura 4. Pode-se observar que os CIs analisados apresentam *baixa similaridade* ao longo de todo o período de evolução investigado no GitHub, ou seja, nenhuma versão identificada no GitHub apresenta similaridade acima de 90% em relação à versão disponível no Etherscan. Dentre os 14 projetos identificados, quatro (PowerLedger, SeeleToken, WAX-Token e RLC) apresentam grau de similaridade acima de 50%. Em específico, o projeto PowerLedger atinge 83% de similaridade, sendo assim o CI com maior similaridade dentre os investigados e classificados como Padrão de Evolução Delta. Observou-se, que tal divergência entre os CIs presentes no GitHub e Etherscan advém de importações, mudanças de versão de compilador e mudanças em funções como a remoção da `keyword` "`emit`", além de mudanças na declaração de funções. Reforçando uma divergência ainda maior entre o CI no GitHub e a versão disponível no Etherscan, os demais projetos apresentam similaridade inferior a 65%, sendo TrueUSD e HuobiToken com menores valores alcançados, respectivamente: 1%, 3% e 4%. Durante a análise manual, constatou-se que os três últimos projetos mencionados apresentaram zero similaridade, concluindo-se que estes CIs não condizem o do Etherscan. Diante de tal achado, hipotetiza-se um cenário com a presença de repositórios os quais

não foram necessariamente utilizados pelas empresas (seja pela adoção de outras plataformas ou abandono, por exemplo) deflagrando, assim, uma limitação metodológica na seleção de projetos.

Figura 4: Análise de similaridade do Padrão Delta



2.3 Ameaças à Validade

Baseando-se na classificação de Wohlin *et al.*, ameaças à validade foram identificadas [19]. Considera-se como *ameaça de conclusão* a falta de testes estatísticos e de hipótese para validar as observações. Apesar disso, todas as análises foram discutidas a partir de trabalhos similares ao estado da arte. Por sua vez, quanto à *ameaça de construção*, alguns conceitos teóricos de difícil compreensão podem ter afetado no *design* da metodologia. Porém, todas as etapas metodológicas foram extensamente discutidas entre os autores deste trabalho, onde foi possível revisar e ajustar as decisões do estudo empírico nos padrões de pesquisas da área. Em relação às *ameaças externas*, somente CIs da rede Ethereum foram estudados. Assim, tais observações não podem ser generalizadas para CIs em outras linguagens e plataformas. Ademais, destaca-se a possibilidade dos repositórios estarem com outro nome ou defasados propositalmente no GitHub pelos mantenedores. Dessa forma, a fim de certificar a corretude dos repositórios selecionados, estabeleceu-se um conjunto de quatro critérios, conforme relatado nos procedimentos metodológicos. Por fim, é possível que a quantidade de CIs analisados não seja suficiente para generalização das observações para todos os CIs implantados na rede Ethereum. Para minimizar essa ameaça, foram escolhidos os CIs de projetos com maior valor de mercado que tinham seus repositórios no GitHub disponíveis.

3 CONSIDERAÇÕES FINAIS

Sob a forma de uma pesquisa exploratória-descritiva baseada em MRS, este trabalho analisou o processo de evolução de software de 27 CIs implantados na plataforma Ethereum. Fundamentado por uma análise quali-quantitativa, este estudo contribui ao 1) caracterizar padrões de evolução de software quanto a similaridade e comportamento de projetos *open source* no GitHub em contraste à versão do CI disponível no Etherscan e 2) explorar um método experimental baseado no uso de expressão regular e análise combinatória para avaliar a similaridade entre versões de CIs.

Como implicações práticas, constatou-se que 13 projetos apresentaram acima de 90% de similaridade em algum momento da evolução do software. Em específico, evidenciou-se também a existência de diferentes práticas e comportamentos evolutivos relacionados aos projetos investigados como, por exemplo, progressões de desenvolvimento graduais (e não graduais) rumo ao *deploy* na Ethereum, decomposição do SC em múltiplos arquivos durante o período de desenvolvimento e diferentes estratégias de adoção do GitHub (apenas para exposição de código-fonte ou realmente como ferramenta de apoio ao desenvolvimento).

Em termos de trabalhos futuros, pretende-se expandir a quantidade de CIs analisados buscando ampliar o entendimento sobre a evolução *open source* deles; disponibilizar para a comunidade um *dataset* a partir de tal expansão; a partir de entrevistas com desenvolvedores, conduzir uma análise qualitativa adicional sobre elementos sociotécnicos e colaborativos associados aos repositórios de CIs; e um aprofundamento sobre as mudanças e o motivo das divergências de similaridades nas versões dos CIs.

REFERÊNCIAS

[1] Nemitari Ajienka, Peter Vangorp, and Andrea Capiluppi. 2020. An empirical analysis of source code metrics and smart contract resource consumption. *Journal of Software: Evolution and Process* 32, 10 (2020), e2267.

[2] Francisco Javier Carrera Arias. 2019. Fuzzy String Matching in Python. https://www.datacamp.com/community/tutorials/fuzzy-string-python Acessado em: 04/05/2021.

[3] Amiangshu Bosu, Anindya Iqbal, Rifat Shahriyar, and Partha Chakraborty. 2019. Understanding the motivations, challenges and needs of blockchain software developers: A survey. *Empirical Software Engineering* 24, 4 (2019), 2636–2673.

[4] Partha Chakraborty, Rifat Shahriyar, Anindya Iqbal, and Amiangshu Bosu. 2018. Understanding the software development practices of blockchain projects: a survey. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–10.

[5] Jiachi Chen, Xin Xia, David Lo, John Grundy, and Xiaohu Yang. 2020. Maintaining smart contracts on Ethereum: Issues, techniques, and future challenges. *arXiv preprint arXiv:2007.00286* (2020).

[6] Chris Dannen. 2017. *Introducing Ethereum and solidity*. Vol. 318. Springer.

[7] Giuseppe Destefanis, Michele Marchesi, Marco Ortu, Roberto Tonelli, Andrea Bracciali, and Robert Hierons. 2018. Smart contracts vulnerabilities: a call for blockchain software engineering?. In *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 19–25.

[8] Ahmed E Hassan. 2008. The road ahead for mining software repositories. In *2008 Frontiers of Software Maintenance*. IEEE, 48–57.

[9] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf

[10] Gustavo A Oliva, Ahmed E Hassan, and Zhen Ming Jack Jiang. 2020. An exploratory study of smart contracts in the Ethereum blockchain platform. *Empirical Software Engineering* (2020), 1–41.

[11] Andrea Pinna, Simona Ibba, Gavina Baralla, Roberto Tonelli, and Michele Marchesi. 2019. A massive analysis of ethereum smart contracts empirical study and code metrics. *IEEE Access* 7 (2019), 78194–78213.

[12] Massimo Ragnedda and Giuseppe Destefanis. 2019. *Blockchain and web 3.0: social, economic, and technological challenges*. Routledge.

[13] Siraj Raval. 2016. *Decentralized applications: harnessing Bitcoin's blockchain technology*. "O'Reilly Media, Inc.".

[14] Pierre Reibel, Haaroon Yousaf, and Sarah Meiklejohn. 2018. Why is a Ravencoin Like a TokenDesk? An Exploration of Code Diversity in the Cryptocurrency Landscape. *arXiv preprint arXiv:1810.08420* (2018).

[15] Marten Risius and Kai Spohrer. 2017. A blockchain research framework. *Business & Information Systems Engineering* 59, 6 (2017), 385–409.

[16] Alan Rodrigues, Allysson Allex Araújo, Matheus Paixao, and Pamella Sousa. 2021. Pacote de Replicação. https://zenodo.org/record/5216861

[17] Nick Szabo. 1996. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)* 18 (1996).

[18] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. Smartcheck: Static analysis of ethereum smart contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*. 9–16.

[19] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[20] Li Yujian and Liu Bo. 2007. A normalized Levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence* 29, 6 (2007), 1091–1095.

[21] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach D Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. 2019. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering* (2019).