UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE CRATEÚS

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

ALAN RODRIGUES CHAVES

CARACTERIZANDO A EVOLUÇÃO DE SOFTWARE DE CONTRATOS INTELIGENTES: UM ESTUDO EXPLORATÓRIO-DESCRITIVO UTILIZANDO GITHUB E ETHERSCAN

CRATEÚS

2021

ALAN RODRIGUES CHAVES

CARACTERIZANDO A EVOLUÇÃO DE SOFTWARE DE CONTRATOS INTELIGENTES: UM ESTUDO EXPLORATÓRIO-DESCRITIVO UTILIZANDO GITHUB E ETHERSCAN

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Allysson Allex Araújo

Coorientador: Prof. Dr. Matheus H. E. Paixão

CRATEÚS

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C438c Chaves, Alan Rodrigues.
Caracterizando a evolução de software de contratos inteligentes : Um Estudo Exploratório-descritivo utilizando GitHub e Etherscan / Alan Rodrigues Chaves. – 2021.
63 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Crateús, Curso de Sistemas de Informação, Crateús, 2021.
Orientação: Prof. Dr. Allysson Allex Araújo.
Coorientação: Prof. Dr. Matheus H. E. Paixão.

1. Contratos Inteligentes. 2. Mineração de Repositórios de Software. 3. Ethereum. 4. Etherscan. 5. GitHub. I. Título.

CDD 005

ALAN RODRIGUES CHAVES

CARACTERIZANDO A EVOLUÇÃO DE SOFTWARE DE CONTRATOS INTELIGENTES: UM ESTUDO EXPLORATÓRIO-DESCRITIVO UTILIZANDO GITHUB E ETHERSCAN

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus de Crateús da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Allysson Allex Araújo (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Matheus H. E. Paixão (Coorientador)
Universidade Estadual do Ceará (UECE)

Prof. Me. Francisco Anderson de Almada Gomes
Universidade Federal do Ceará (UFC)

Prof. Me. Lívio Antônio Melo Freire
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Agradeço, primeiramente, a minha família por estarem sempre ao meu lado e acreditarem em mim.

Agradeço a todos os professores que contribuíram na minha formação.

Aos amigos de graduação, em especial Luiza Ananda, por estar sempre ao meu lado e me ajudar a permanecer forte.

Aos amigos da vida por compartilharem comigo desta luta de conclusão da graduação.

Ao prof. Dr. Allysson Allex de Paula Araújo por me orientar e sempre me motivar ao longo do desenvolvimento do meu trabalho.

Ao prof. Dr. Matheus Henrique Esteves Paixão por aceitar ser meu coorientador e contribuir com seus conhecimentos em Mineração em Repositórios de Software para desenvolvimento deste trabalho.

“Nem sempre os mais talentosos se destacam:
São os persistentes que se sobressaem”

(Mary Kay Ash)

RESUMO

O *blockchain*, que iniciou seu desenvolvimento após a criação da criptomoeda Bitcoin, concebe uma das novas infraestruturas digitais baseadas na internet com grande potencial disruptivo. Parcela desse potencial advém da consolidação de plataformas públicas, como a Ethereum, a qual viabiliza o desenvolvimento de aplicações descentralizadas baseadas em *blockchain*. Tais soluções são baseadas em contratos inteligentes e precisam lidar com características próprias, como a imutabilidade de dados, a qual impacta diretamente na evolução do software, e a transparência quanto ao acesso ao código-fonte do contrato inteligente. Em específico, a transparência de código pode ser observada através de ferramentas como o Etherscan, a qual proporciona acesso público a uma vasta quantidade de informações dos contratos inteligentes utilizados na Ethereum. Nesse sentido, também percebe-se um movimento de adesão às práticas de desenvolvimento *open source*, com destaque para o uso do GitHub como plataforma social de hospedagem de código-fonte. Todavia, tal cenário motiva questionamentos sobre como se dá a relação entre o contrato inteligente exposto e desenvolvido no GitHub e o que realmente está sendo utilizado na Ethereum e, consequentemente, auditável via Etherscan. Sob a forma de uma pesquisa exploratória-descritiva baseada em Mineração em Repositório de Software, este estudo objetiva compreender como ocorre o processo de evolução de software em contratos inteligentes a partir da relação entre os códigos-fonte disponíveis no Etherscan e GitHub. A partir de uma análise quali-quantitativa, este trabalho contribui em três principais perspectivas: 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 contrato inteligente disponível no Etherscan, 2) analisar elementos colaborativos quanto ao desenvolvimento *open source* de contratos inteligentes e 3) implementar um método experimental baseado em comparação de *strings* para análise de similaridade entre versões de contratos inteligentes.

Palavras-chave: Contratos Inteligentes. Mineração de Repositórios de Software. Ethereum. Etherscan. GitHub.

ABSTRACT

Blockchain, which started its development after the creation of the Bitcoin cryptocurrency, conceives one of the new internet-based digital infrastructures with great disruptive potential. Part of this potential comes from the consolidation of public platforms, such as Ethereum, which enables the development of decentralized applications based on *blockchain*. Such solutions are based on smart contracts and need to deal with their own characteristics, such as the immutability of data, which directly impacts the evolution of the software, and a transparent access to the source code of the smart contract. In particular, code transparency can be observed through tools such as Etherscan, which provides public access to a vast amount of information from the smart contracts used in Ethereum. In this sense, there is also a movement of adherence to open source development practices, with emphasis on the use of GitHub as a social platform for hosting source code. However, this scenario motivates questions about how the relationship between the smart contract exposed and developed on GitHub and what is actually being used at Ethereum and, consequently, auditable via Etherscan, is carried out. In the form of an exploratory-descriptive research based on Mining Software Repository, this study aims to understand how the process of software evolution in smart contracts takes place based on the relationship between the source codes available on Etherscan and GitHub. From a quali-quantitative analysis, this work contributes in three main perspectives: 1) by characterizing software evolution models regarding the similarity and behavior of projects developed in an open source form on GitHub in contrast to the contract version available in Etherscan, 2) by analyzing collaborative elements regarding the open source development of smart contracts, and 3) by implementing an experimental method based on string comparison for comparing smart contracts similarity.

Keywords: Smart Contracts. Mining Software Repositories. Ethereum. Etherscan. GitHub.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CSS	*Cascading Style Sheets*
dApps	Decentralized Applications
EVM	*Ethereum Virtual Machine*
GLD	*Generalized Levenshtein Distance*
HTML	*Hypertext Markup Language*
ITS	*Issue Tracking Systems*
MRS	Mineração de Repositórios de Software
PoW	*Proof-of-Work*

SUMÁRIO

1 INTRODUÇÃO

A presente introdução contempla quatro elementos principais. Inicialmente, busca-se prover uma contextualização geral sobre os principais embasamentos teóricos que sustentam este trabalho. Em seguida, tem-se um afunilamento argumentativo com o intuito de evidenciar a justificativa desta pesquisa. Por fim, apresentam-se os objetivos e a estrutura do trabalho.

1.1 Contextualização

O *blockchain*, que teve seu desenvolvimento difundido após a criação da criptomoeda Bitcoin (NAKAMOTO, 2009), constitui uma das novas infraestruturas digitais baseadas na internet com amplo potencial disruptivo (RAGNEDDA; DESTEFANIS, 2019). Em termos genéricos, o *blockchain* funciona como uma tecnologia distribuída capaz de armazenar criptograficamente um registro de eventos lineares de transações entre atores em rede (RISIUS; SPOHRER, 2017). Em outras palavras, *blockchain* pode ser compreendido como um sistema em rede, digital e descentralizado, em que os usuários participam e colaboram para sua segurança, transparência e prestação de contas (YERMACK, 2017). Em decorrência da natureza descentralizada baseada em algoritmos de consenso, os próprios atores que compõem o *blockchain* conseguem fazer o papel de validador de transações, contornando, assim, a necessidade de uma entidade intermediadora para a realização e garantia da confiança de operações. Segundo Tapscott e Tapscott (2016), o *blockchain* está entre as tecnologias mais promissoras da atualidade e com grandes perspectivas de inovação em diversos segmentos da sociedade.

No entanto, apesar de disruptivo, o cenário de uso do Bitcoin é limitado, devido sua aplicação ter como foco principal o armazenamento e a transferência de valores (EFANOV; ROSCHIN, 2018). Visando contornar tal limitação, o surgimento da Ethereum, como uma nova plataforma de *blockchain*, ampliou as possibilidades de desenvolvimento de soluções baseadas em *blockchain* (DANNEN, 2017). A Ethereum alavancou uma tecnologia chamada contratos inteligentes (em inglês, *smart contracts*), tornando possível que programas *Turing-complete* fossem executados na própria *blockchain* (CHEN *et al.*, 2020). Em suma, contratos inteligentes são códigos executáveis, armazenados de forma distribuída e que contém termos contratuais formalizados para realizar acordos de relacionamento (SZABO, 1996). Tais contratos podem ser compreendidos como um sistema que libera ativos digitais para todas ou algumas das partes envolvidas, uma vez que regras pré-definidas tenham sido cumpridas (ETHEREUM, 2019).

Os contratos inteligentes possibilitaram a descentralização do controle lógico e das funções de pagamentos das aplicações (ANTONOPOULOS; WOOD, 2018) visto que a maioria dos sistemas anteriores à introdução desse conceito, eram centralizados, ou seja, controlados por uma única entidade. Assim, por meio desta oportunidade tecnológica, o desenvolvimento de Aplicações Descentralizadas (do inglês, Decentralized Applications (dApps) tem se fortalecido cada vez mais (RAVAL, 2016).

Os contratos inteligentes desenvolvidos para a rede Ethereum são codificados em uma linguagem de programação de alto nível denominada Solidity, cuja estrutura se assemelha ao JavaScript (DANNEN, 2017). Através da Solidity, viabiliza-se que desenvolvedores implementem contratos inteligentes e os compilem para *bytecode* na *Ethereum Virtual Machine* (EVM), antes da implantação no *blockchain* (HILDENBRANDT *et al.*, 2018). Assim, a EVM funciona essencialmente criando um nível de abstração entre o código de execução e a máquina executora, melhorando a portabilidade do software (HOLLANDER, 2019). Nesse sentido, com a lapidação do ecossistema de soluções baseadas na Ethereum, outras 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. Em suma, tal solução viabiliza a oportunidade de qualquer pessoa visualizar as transações realizadas pelos nós que compõem a rede, bem como obter o nome, descrição, quantidade de detentores do ativo, valor de mercado, transações de compra/venda e, inclusive, o próprio código-fonte em Solidity referente a cada contrato inteligente implantado na Ethereum.

Adicionalmente, há de se destacar que contratos inteligentes apresentam características próprias, que os diferenciam de outros tipos tradicionais de software (CHEN *et al.*, 2020). Após compilado, o *bytecode* de um contrato inteligente deve ser implantado no *blockchain*, onde o mesmo será posteriormente executado. Assim, pela natureza imutável do *blockchain*, uma vez que uma informação é registrada, ela nunca mais poderá ser alterada, o que também se aplica aos contratos inteligentes. Logo, uma vez que um contrato inteligente é implantado no *blockchain*, ele nunca mais poderá ser alterado devido às propriedades criptográficas da tecnologia (RAGNEDDA; DESTEFANIS, 2019; TONELLI *et al.*, 2018). Em caso de atualização, por exemplo, um novo contrato inteligente deve ser implantado em um novo nó do *blockchain*, onde todas as dApps que fazem uso deste contrato devem ser atualizadas para referenciar o novo contrato. Além da característica de imutabilidade, os contratos inteligentes também lidam com

[1] Disponível em: https://etherscan.io

restrições relacionadas à execução. Os contratos são executados de forma descentralizada, em máquinas denominadas “mineradores”, onde o custo (usualmente em criptomoedas nativas da plataforma) de execução aumenta de acordo com a quantidade de instruções realizadas. Assim, o tamanho de seu código-fonte e a quantidade de transações realizadas não devem exceder restrições específicas de domínio (DESTEFANIS *et al.*, 2018). Dessa forma, ao analisar as diferenças dos contratos inteligentes com relação a software tradicionais, constatam-se novos e específicos desafios à engenharia de software, incluindo aqueles relacionados à evolução de software (PORRU *et al.*, 2017; BOSU *et al.*, 2019).

1.2 Justificativa

Conforme conceituado por Lehman e Ramil (2003), a evolução do software refere-se ao fato de que o sistema deve evoluir para atender às mudanças nas necessidades do usuário e pode ser acionada por diversos problemas, incluindo relatórios e correção de *bugs* de software encontrados na operação. Depois que um sistema foi implantado, ele inevitavelmente precisa mudar para permanecer útil (MENS *et al.*, 2005). Assim, os desenvolvedores adicionam novos recursos, corrigem erros anteriores e se adéquam aos novos requisitos, tecnologias e volatilidade do conhecimento à medida que o tempo evolui (RAJLICH, 2014). Porém, todas essas mudanças que ocorrem após o desenvolvimento inicial são impossíveis quando os contratos são publicados em um *blockchain*, sendo imutáveis a partir desse ponto (SILLABER *et al.*, 2020). Ou seja, ao contrário dos softwares tradicionais que podem ser atualizados diretamente, para manter um contrato inteligente, os desenvolvedores precisam reimplantá-lo e descartar a versão antiga. Embora os desenvolvedores possam desenvolver contratos atualizáveis usando `DelegateCall` ou destruindo os contratos com erros usando a função `SelfDestruct`, ambos os métodos podem aumentar os riscos de segurança, a não confiança dos usuários de contratos inteligentes e os custos de desenvolvimento (CHEN *et al.*, 2020).

Verifica-se desde o início da década de 2000 uma forte ascensão do desenvolvimento *open source*, ou seja, uma metodologia de desenvolvimento colaborativa cujo código-fonte é disponibilizado publicamente para revisão e produção pela comunidade (CHAU, 2020; KOGUT; METIU, 2001). Tal fenômeno também se faz presente no contexto de contratos inteligentes, utilizando especialmente plataformas sociais de hospedagem de código-fonte como, por exemplo, o GitHub (ZOU *et al.*, 2019). Nesse cenário, pressupõe-se que os contratos inteligentes são desenvolvidos dentro dessas plataformas até o estágio de *deploy* na Ethereum. Todavia, conforme

alertado por Reibel *et al.* (2018), é possível que as versões dos contratos inteligentes no GitHub e na rede Ethereum podem diferir entre si. Tal cenário, em especial, motiva questionamentos sobre como se dá a relação entre o contrato inteligente 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 *open source* de contratos inteligentes, como, por exemplo, a natureza inalterável dos dados registrados em *blockchain*, a garantia da simplicidade para redução de custos operacionais em criptomoedas e transparência de acesso ao código-fonte do contrato inteligente (CHAKRABORTY *et al.*, 2018). Nesse cenário, é válido destacar a presença de trabalhos prévios (TIKHOMIROV *et al.*, 2018; PINNA *et al.*, 2019; OLIVA *et al.*, 2020; AJIENKA *et al.*, 2020) com foco em desafios relativos ao desenvolvimento e manutenção de contratos inteligentes, 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.

Englobando a motivação e a lacuna de pesquisa acima, este estudo tem o propósito de responder a seguinte pergunta de pesquisa: *Como ocorre o processo de evolução de software em contratos inteligentes a partir da relação entre repositórios de código-fonte disponíveis no Etherscan e GitHub?* Para responder tal indagação, conduziu-se um estudo exploratório-descritivo de natureza quali-quantitativa baseado em Mineração de Repositório de Software (MRS). A MRS tem o intuito de investigar dados presentes em repositórios de software com o intuito de extrair informações pertinentes que ajudem a melhorar sistemas computacionais (HASSAN, 2008). Fundamentado a partir de um experimento computacional, o presente estudo investigou 27 contratos inteligentes 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, também dispusessem de um repositório *open source* no GitHub.

Em termos de contribuições, este trabalho avança no entendimento da evolução de software no contexto de contratos inteligentes 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 contrato inteligente disponível no Etherscan, 2) analisar elementos colaborativos quanto ao desenvolvimento *open source* de contratos inteligentes, como, por exemplo, a relação entre *commits* e colaboradores e o engajamento da comunidade em relação ao *report* e resolução de *issues* e 3) implementar um método experimental baseado em comparação de *strings* para análise de similaridade entre versões de contratos inteligentes.

1.3 Objetivos

Investigar o processo de evolução de software de contratos inteligentes a partir de dados e informações disponíveis nos repositórios Etherscan e GitHub. Em relação aos objetivos específicos, tem-se:

- Compreender a dinâmica de evolução de software a partir da caracterização de padrões os quais evidenciem o comportamento evolutivo dos projetos investigados;
- Analisar elementos colaborativos (colaboradores, *commits* e *issues*) quanto ao desenvolvimento *open source* de projetos de contratos inteligentes;
- Implementar um método experimental baseado em comparação de *strings* para análise de similaridade entre versões de contratos inteligentes;
- Definir um protocolo de pesquisa para, a partir das informações sobre um contrato inteligente no Etherscan, ser capaz de identificar seu repositório no GitHub, quando disponível.

1.4 Estrutura do Trabalho

No capítulo inicial do presente trabalho foram descritos seu contexto, motivação e objetivos, introduzindo, assim, a pesquisa desenvolvida. A versão atual da pesquisa contará ainda com a seguinte estrutura:

- Capítulo 2 - Fundamentação Teórica: tem por objetivo discutir os principais conceitos acerca das áreas que constituem este trabalho: *Blockchain*, Contratos Inteligentes, Ethereum e Mineração de Repositórios de Software;
- Capítulo 3 - Trabalhos Relacionados: busca-se analisar os principais trabalhos que ajudaram na construção desta pesquisa;
- Capítulo 4 - Estudo Empírico: detalha-se os procedimentos metodológicos, bem como as análises dos resultados obtidos;
- Capítulo 5 - Ameaças à Validade: discute-se as ameaças identificadas para os resultados deste trabalho e as estratégias adotadas para mitigar tais questões;
- Capítulo 6 - Considerações Finais: apresenta-se as contribuições, limitações e trabalhos futuros desta pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação para a compreensão dos elementos teóricos utilizados nesta pesquisa. A princípio, na Seção 2.1, são elucidados os fundamentos da tecnologia *blockchain*, enquanto na Seção 2.2, explana-se sobre as características da plataforma Ethereum, incluindo os fundamentos de Contratos Inteligentes e Aplicações Descentralizadas. Posteriormente, na Seção 2.3, discute-se os elementos teóricos que embasam a Evolução de Software. Finalmente, a Mineração em Repositórios de Software é apresentada na Seção 2.4.

2.1 Blockchain

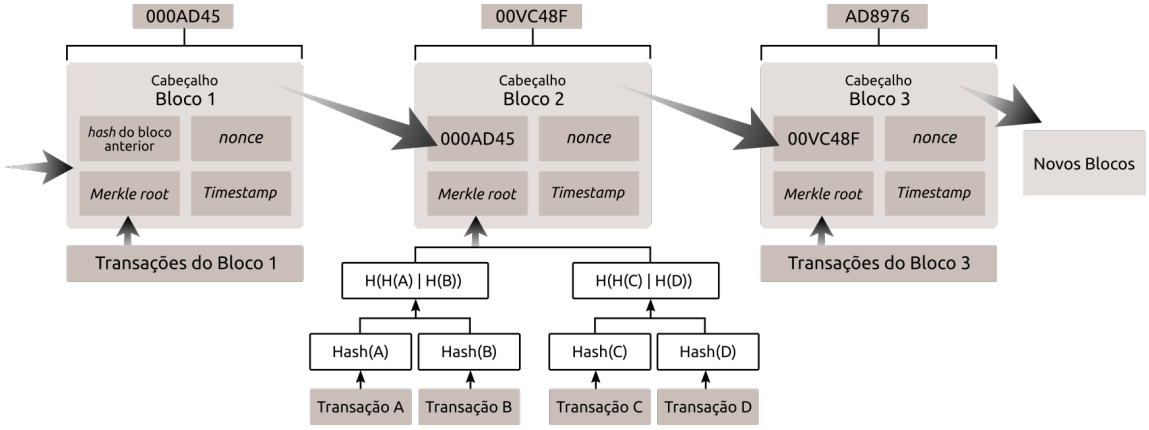
Blockchain é uma tecnologia que faz uso de uma arquitetura distribuída e descentralizada para registrar transações de maneira que um registro não possa ser alterado retroativamente, tornando, assim, este registro imutável (ALVES *et al.*, 2018). Em outras palavras, o *blockchain* funciona como um sistema em rede, digital e descentralizado onde os usuários participam e colaboram para sua segurança, transparência e prestação de contas (YERMACK, 2017). De acordo com Greve *et al.* (2018), o *blockchain* implementa uma máquina de estados replicada para a manutenção consistente de um estado global compartilhado por um conjunto de pares distribuídos numa rede *peer-to-peer*. Todos os nós possuem e mantêm uma réplica do registro de transações efetuadas, abstraído sob a forma de um livro-razão distribuído (em inglês, *distributed ledger*), que é imutável, pode ser verificado e auditado, e está sempre disponível. Entre as motivações para o uso de um *distributed ledger* tem-se a segurança e, consequentemente, a confiança na rede, inclusive por permitir a auditoria deste histórico de transações, ou seja, a confrontação das novas transações com o histórico de transações (NAKAMOTO *et al.*, 2008).

Wüst e Gervais (2018) elencam uma série de propriedades atreladas aos *distributed ledgers*. A **verificabilidade pública** permite que qualquer pessoa verifique a exatidão do estado do sistema. Em um livro-razão distribuído, cada transição de estado é confirmada por verificadores (por exemplo, mineradores em Bitcoin), que podem ser um conjunto restrito de participantes. Qualquer observador, no entanto, pode verificar que o estado do livro-razão foi alterado de acordo com o protocolo e todos os observadores eventualmente terão a mesma visão do livro-razão, pelo menos até um determinado nível. Por sua vez, a **transparência dos dados** junto com o processo de atualização do estado são requisitos para a verificabilidade pública. No entanto, a quantidade de informação que é transparente para um observador pode diferir e nem

todo participante precisa ter acesso a todas as informações. Já a **privacidade** é uma propriedade importante de qualquer sistema. Existe uma tensão inerente entre privacidade e transparência, porém a privacidade é certamente mais fácil de conseguir em um sistema centralizado porque a transparência e a verificabilidade pública não são necessárias para o funcionamento do sistema. A **integridade das informações** garante que as informações sejam protegidas contra modificações não autorizadas, ou seja, os dados recuperados estão corretos. Assim, tem-se que a integridade da informação está intimamente ligada à verificabilidade pública. Dito isso, se um sistema fornece verificabilidade pública, qualquer um pode verificar a integridade dos dados. Adicionalmente, a **redundância de dados** é uma forma de proteção dos dados, onde os mesmos encontram-se replicados em mais de uma máquina, reduzindo a possibilidade dos dados serem perdidos em caso de falha do sistema. A propriedade **trust anchor** define quem representa a autoridade mais alta que pode conceder e revogar o acesso de leitura e gravação de um dado.

Conforme Figura 1, o *blockchain* é formada por uma cadeia de blocos cujo encadeamento é feito quando o *hash* do bloco prévio é adicionado ao conteúdo do bloco atual. Este *hash* é uma impressão digital desses dados e bloqueia os blocos em ordem cronológica fazendo com que não haja a possibilidade de que uma transação seja alterada sem alterar seu bloco e todos os blocos prévios (AITZHAN; SVETINOVIC, 2018). Assim, o bloco é formado por um conjunto de transações organizadas em uma estrutura de dados definida como *Merkle Tree* (CHEN *et al.*, 2019). Nessa estrutura, as transações são agrupadas em pares e o *hash* de cada uma das transações é armazenado em um nó pai e, assim, sucessivamente até o nó raiz da *Merkle Tree* (NARAYANAN *et al.*, 2016). Por sua vez, o apontador *hash* da *Merkle Tree*, composta pelas transações, é armazenado no cabeçalho do bloco.

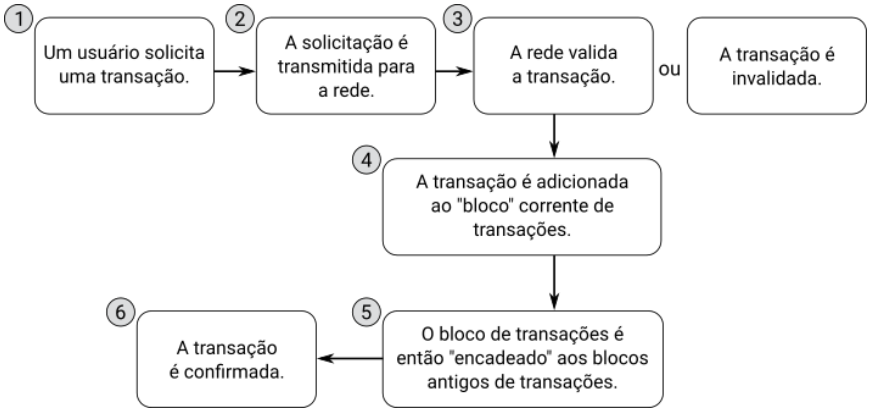
Figura 1 – Representação simplificada de um *blockchain*.



Fonte: Adaptado de Chen *et al.* (2019).

Por caracterizar-se como rede distribuída, o comportamento do *blockchain* é controlado através do voto coletivo de nós anônimos (AITZHAN; SVETINOVIC, 2018). Contudo, os nós individuais podem travar ou comportar-se maliciosamente, por exemplo. Por esse motivo, a execução de um protocolo de consenso tolerante a falhas como parte do funcionamento dos *blockchains* pode garantir que todos os nós concordem com a ordem em que as entradas são anexadas ao *blockchain* (CACHIN; VUKOLIĆ, 2017). O conceito de "consenso" é relacionado ao problema de sincronização de estados em sistemas distribuídos, com o intuito de que diferentes participantes da rede podem concordar com um mesmo estado. A falta de uma entidade central para fazer estas decisões é uma das qualidades das redes *blockchain*, pois isso favorece a não censura e a independência de autoridades que definem as permissões de acesso a dados. De forma resumida, a Figura 2 mostra as etapas gerais do funcionamento de um *blockchain*.

Figura 2 – Funcionamento de um blockchain



Fonte: Adaptado de Laurence (2019).

Cada *blockchain* pode operar com um ou mais tipos de algoritmos de consenso, os quais podem ser determinados pela ameaça esperada e o grau de confiança que a rede possui nos nós que participam da *blockchain*. No caso das redes Bitcoin e Ethereum, utiliza-se atualmente o algoritmo de consenso denominado *Proof-of-Work* (PoW) (LAURENCE, 2019). Em suma, tal protocolo de consenso funciona da seguinte maneira: como cada bloco tem um de seus campos presentes denominado de `nonce`, o nó minerador, através de um algoritmo de força bruta, testa diversos números sucessivamente até que o bloco seja considerado correto. Para que um bloco seja aceito, o nó minerador deve computar o valor da *hash* do bloco inteiro, onde o valor resultante deve ser menor que a dificuldade imposta pela rede naquele período. O principal ganho da PoW é que não existe uma estratégia mais eficiente de achar um valor para o `nonce` que não seja através de enumerar todas as possibilidades e que sua checagem é trivial e barata, ou seja, torna-se difícil forjar blocos e menos complexo checar a corretude deles.

Além disso, diferentes tipos ou categorias de *blockchains* surgiram com o objetivo de lidar com necessidades de negócios distintas. De forma análoga à computação em nuvem, existem *blockchains públicos* (como Bitcoin) onde todos podem acessar e atualizar, existem *blockchains privados* (por exemplo, Ripple) em que apenas um grupo limitado dentro uma organização é capaz de acessar e atualizar, e, finalmente, existe um terceiro tipo, um *consórcio de blockchains* (por exemplo, R3 Consortium) que são usados em colaboração com outros (BAMBARA; ALLEN, 2018). Dessa forma, apesar de ter sido inicialmente desenvolvida como a principal tecnologia de autenticação e verificação de transações do Bitcoin (NAKAMOTO *et al.*, 2008), o *blockchain*, como uma tecnologia, tem atraído atenção além do propósito de transações financeiras, abordando outras possibilidades de negócios, como, por exemplo, propriedade inteligente, internet das coisas, suprimentos gestão da cadeia, saúde, propriedade e distribuição de *royalties* e organizações autônomas descentralizadas (WÜST; GERVAIS, 2018). Nesse sentido, a plataforma de *blockchain* Ethereum tem se posicionado como a principal plataforma pública para desenvolvimento de aplicações descentralizadas baseadas em contratos inteligentes.

2.2 Ethereum

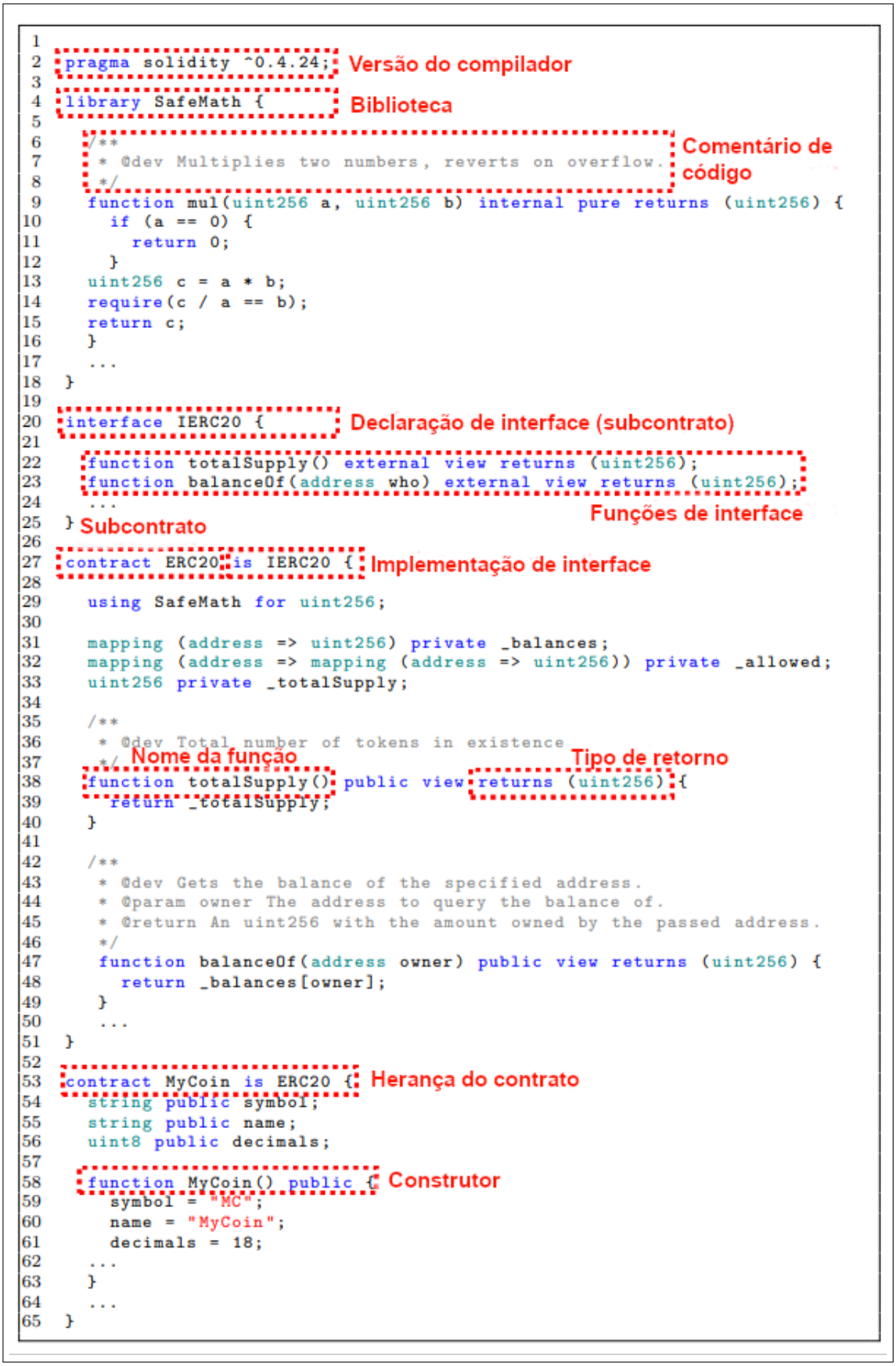
O avanço da plataforma Ethereum ascendeu a tecnologia *blockchain* para um novo patamar (PINNA *et al.*, 2019; ETHEREUM, 2019). De acordo com Kolb *et al.* (2020), o *blockchain* da Ethereum é público e opera de forma muito semelhante ao Bitcoin. As identidades são representadas como chaves públicas e autenticam suas transações com a chave privada correspondente. Cada chave pública está associada a um saldo de *tokens* digitais nativos sob a forma de criptomoeda, Ether, que são usados para compensar os mineradores pelo processamento de transações em um protocolo estabelecido como mecanismo de consenso.

Percebe-se a disrupção ocasionada pela Ethereum ao prover uma máquina virtual *Turing-complete*, programável, baseada em *blockchain* para executar código de software escrito especificamente para o ambiente distribuído, descentralizado e imutável (WOOD *et al.*, 2014; PINNA *et al.*, 2019). Tal projeto foi concebido para aproveitar os benefícios do *blockchain* e implementar automaticamente as restrições onde duas partes podem acordar ao assinar um contrato em um ambiente sem confiança, sendo esses códigos do software conhecidos como Contratos Inteligentes (em inglês, *smart contracts*) (ANTONOPOULOS; WOOD, 2018). O termo contrato inteligente foi introduzido por Szabo (1996) quando ele descreveu como a execução de contratos por computador entre duas partes pode ser garantida sem a necessidade de

terceiros para intermediação. Os contratos inteligentes são acordos de auto-execução, ou seja, contratos como no mundo real, mas expressos como um programa de computador cuja execução impõe os termos por meio de uma sequência lógica de etapas (AJIENKA *et al.*, 2020; PINNA *et al.*, 2019). O código-fonte dos contratos inteligentes manipula variáveis da mesma forma que os programas tradicionais, no entanto, o modelo descentralizado de contratos imutáveis implica que a execução e a saída de um contrato são validadas por cada participante do sistema e, portanto, nenhuma parte é independente do controle do dinheiro (DESTEFANIS *et al.*, 2018). Entretanto, uma limitação é a sua complexidade, pois cada operação realizada tem um custo associado e as transações têm um valor máximo que podem gastar (REIBEL *et al.*, 2018). Por outro lado, o fato de não demandar de uma terceira parte confiável para garantir a realização de uma transação, ocasiona a redução de custos operacionais. Em suma, o código de um contrato inteligente é armazenado, verificado e evoluído em um *blockchain* (STARK, 2016).

Tonelli *et al.* (2018) explana que um Contrato Inteligente na Ethereum normalmente mantém seu código executável e um estado que consiste em: um armazenamento privado e a quantidade de Ether que contém, ou seja, o saldo de contrato. Os usuários podem transferir Ether usando transações, como no Bitcoin e, adicionalmente, podem invocar contratos usando transações que invocam contratos. Conceitualmente, a Ethereum pode ser vista como uma "grande máquina de estado baseada em transações", onde seu estado é atualizado após cada transação e armazenado no *blockchain* (ANTONOPOULOS; WOOD, 2018). No nível mais baixo, o código de um contrato inteligente é uma linguagem de *bytecode* baseada em pilha, executada na *Ethereum Virtual Machine* (EVM) em cada nó. Os desenvolvedores de contrato inteligentes definem as regras usando linguagens de programação de alto nível. A linguagem adotada na Ethereum é Solidity (uma linguagem semelhante a JavaScript), que é compilada em *bytecode* para a EVM. Uma vez que um contrato inteligente é criado em um endereço X, é possível invocá-lo enviando uma transação de convocação de contrato para o endereço X (GRISHCHENKO *et al.*, 2018). Assim, uma transação de invocação de contrato normalmente inclui: o pagamento pela execução do contrato (em Ether) e dados de entrada para a invocação (AJIENKA *et al.*, 2020). Um exemplo ilustrativo de contrato inteligente é mostrado na Figura 3. Segundo Szabo (1996), a nível conceitual, os contratos inteligentes consistem em três partes: 1) o código de um programa que se torna a expressão de uma lógica contratual; 2) o conjunto de mensagens que o programa pode receber e que representam os eventos que ativam o contrato; 3) o conjunto de métodos que ativam as reações previstas pela lógica contratual.

Figura 3 – Exemplo de contrato inteligente

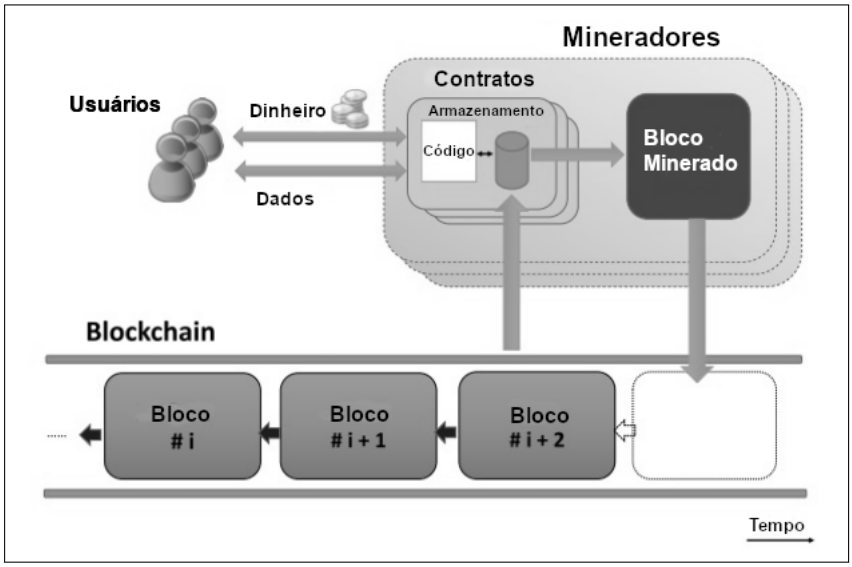



```
1
2  pragma solidity ^0.4.24;
3
4  library SafeMath {
5
6    /**
7     * @dev Multiplies two numbers, reverts on overflow.
8     */
9    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
10     if (a == 0) {
11       return 0;
12     }
13   uint256 c = a * b;
14   require(c / a == b);
15   return c;
16   }
17   ...
18 }
19
20 interface IERC20 {
21
22   function totalSupply() external view returns (uint256);
23   function balanceOf(address who) external view returns (uint256);
24   ...
25 }
26
27 contract ERC20 is IERC20 {
28
29   using SafeMath for uint256;
30
31   mapping (address => uint256) private _balances;
32   mapping (address => mapping (address => uint256)) private _allowed;
33   uint256 private _totalSupply;
34
35   /**
36    * @dev Total number of tokens in existence
37    */
38   function totalSupply() public view returns (uint256) {
39     return _totalSupply;
40   }
41
42   /**
43    * @dev Gets the balance of the specified address.
44    * @param owner The address to query the balance of.
45    * @return An uint256 with the amount owned by the passed address.
46    */
47   function balanceOf(address owner) public view returns (uint256) {
48     return _balances[owner];
49   }
50   ...
51 }
52
53 contract MyCoin is ERC20 {
54   string public symbol;
55   string public name;
56   uint8 public decimals;
57
58   function MyCoin() public {
59     symbol = "MC";
60     name = "MyCoin";
61     decimals = 18;
62   ...
63   }
64   ...
65 }
```

Fonte: Adaptado de Oliva *et al.* (2020).

A Figura 4 representa uma visão geral de um sistema de contrato inteligente, em que estado do contrato é armazenado em um *blockchain* público, como a Ethereum. Um contrato inteligente é executado por uma rede de mineradores que atingem o consenso sobre o resultado da execução e atualizam o estado do contrato no *blockchain*. Os usuários podem enviar dinheiro ou dados para um contrato; ou receber dinheiro ou dados de um contrato (DELMOLINO *et al.*, 2016). Zumkeller (2018) salienta o funcionamento de um contrato inteligente da seguinte maneira: a validação de nós executa contratos inteligentes em sua máquina virtual, processando transações que possuem uma conta de contrato como destinatário. Essas transações contêm dados para o contrato inteligente e, o mais importante, o gás, que é necessário para cada etapa computacional de um contrato inteligente. O conceito de gás foi introduzido para limitar a execução do código, uma vez que cada nó de mineração na rede está processando esses contratos inteligentes, o que leva a um alto esforço computacional. Assim, ao chamar uma função de contrato inteligente, é necessário incluir uma criptomoeda nativa na transação, que é então gasta durante o tempo de execução do contrato inteligente. Quando o gás fornecido esgota durante o tempo de execução, o cálculo é interrompido e a transação não é mais processada. Uma transação para um contrato inteligente inicializa sua transição do estado do contrato anterior para o novo, com os dados sendo gravados no armazenamento do contrato inteligente. Durante o tempo de execução, o contrato inteligente realiza cálculos complexos e pode interagir com outras contas enviando mensagens (para chamar outros contratos inteligentes) ou transferindo criptomoedas (para contas externas e de contratos).

Figura 4 – Sistema de contrato inteligente



Fonte: Adaptado de Delmolino *et al.* (2016)

O objetivo da Ethereum é permitir que os desenvolvedores criem aplicações arbitrárias baseados em consenso, que tenham escalabilidade, padronização, funcionalidade completa, facilidade de desenvolvimento e interoperabilidade oferecida por esses diferentes paradigmas (BUTERIN *et al.*, 2014). Tais soluções possibilitaram a ascensão das Aplicações Descentralizadas (em inglês, *Decentralized Applications* ou dApps), as quais têm como principal característica o fato de que não há um único servidor ou entidade controlando-os como em um modelo cliente-servidor. No contexto da Ethereum, os dApps têm como base o uso de *blockchain* como armazenamento e processamento, através da implementação de contratos inteligentes (METCALFE, 2020). Segundo o BlockhainHub (2019), a maneira mais fácil de compreender como uma aplicação descentralizada funciona é entendendo e comparando com aplicações tradicionais, ou seja, uma Aplicação web tradicional usa *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript para renderizar uma página. Ele também precisará capturar detalhes de um banco de dados utilizando uma *Application Programming Interface* (API). Quando um usuário realiza login no Facebook, a página chama uma API para pegar seus dados pessoais e exibi-los na página. De forma semelhante a um aplicativo da web convencional, o Front End de um dApp usa exatamente a mesma tecnologia para renderizar a página. A única diferença importante é que, em vez de uma API se conectar a um banco de dados, você tem um contrato inteligente conectado a um *blockchain*. De acordo com Raval (2016) e Metcalfe (2020), os dApps podem ser implementados em diferentes dispositivos utilizando as mesmas ferramentas e linguagens de programação como qualquer outra aplicação a diferença é que precisam atender aos seguintes critérios:

- **Código aberto**: A aplicação deve ser totalmente de código aberto e operar autonomamente, de maneira que não haja uma autoridade central monopolizando os *tokens* da rede e que sejam possíveis verificação por parte de terceiros.
- **Descentralizada**: Dados e registros devem ser armazenados em um *blockchain* ou rede P2P de forma a evitar pontos centrais de falha.
- **Incentivo por meio de *tokens***: dApps usam *tokens* criptográficos para acessar as aplicações, realizar transações e prover recompensas.
- **Algoritmo/Protocolo**: Inclusão de mecanismos de consenso, como *Proof-of-Work*, *Proof-of-Stake* ou mesmo o próprio adaptado para o dApp.

2.3 Evolução de Software

Segundo Pressman (2005), pode-se definir software como um programa de computador que, quando executado, fornece características, funções e desempenhos desejados. De acordo com Zaidman *et al.* (2008), o desenvolvimento de software é uma atividade multidisciplinar, onde uma série de artefatos devem ser criados e mantidos de forma síncrona. Logo, produtos de software costumam estar em constante transformação, quer seja para o acréscimo de novas funcionalidades ou somente para correção de algum *bug* encontrado por a equipe de teste ou usuários finais. Lehman (1979), por sua vez, destaca que um programa em uso sofre, inerentemente, mudanças contínuas ou se torna progressivamente menos útil. Como Frederick Brooks colocou em seu famoso artigo "*No silver bullet*" (BROOKS; KUGLER, 1987) , "todo software de sucesso é alterado".

Nesse contexto, a evolução do software é a sequência de mudanças em um sistema de software ao longo de sua vida útil, que abrange desenvolvimento e manutenção, garantindo que o mesmo continue confiável e flexível (COOK *et al.*, 2000). Como consequência da evolução, os sistemas podem aumentar ou diminuir em complexidade e, em desenvolvimento de software, é amplamente aceito que a complexidade apenas aumenta à medida que evoluem no tempo (VOINEA, 2007). Os primeiros estudos sobre evolução de software datam desde o final dos anos 60, embora o termo tenha sido usado da forma como é conhecido hoje apenas nos anos 70 (LEHMAN; RAMIL, 2003). Okwu e Onyeje (2014) ressaltam que, apesar da importância do tema, a evolução do software tem recebido pouca atenção e a quantidade de informações disponíveis para os pesquisadores é limitada

Segundo Diehl (2007), existem duas áreas na evolução do software. A primeira, chamada de "Design para Mudança", tem o objetivo de criar regras e arquiteturas para facilitar a tarefa de mudar um sistema de software ao longo do tempo, abordando questões como reconfiguração, adaptação, extensão, depuração, otimização, avaliação (medição da mutabilidade) e gerenciamento de projeto. A segunda, chamada de "Análise de Históricos" de software, diz que durante a vida útil de um sistema de software, muitas versões serão produzidas. A análise do código-fonte dessas versões, bem como a documentação e outras meta-informações, pode revelar regularidades e anomalias no processo de desenvolvimento do sistema em questão.

De acordo com Tripathy e Naik (2014), Lehman e seus colaboradores formularam algumas observações e as introduziram como "Leis de Evolução". Essas leis são o resultado de estudos da evolução em larga escala de sistemas proprietários proprietários ou software de código

fechado. Tais leis dizem respeito a uma categoria de software chamados de *E-type systems*. Dito isso, as oito leis de evolução propostas por Lehman (LEHMAN, 1996) são explicadas resumidamente a seguir:

1. **Mudança contínua**: a menos que um sistema seja continuamente modificado para satisfazer as necessidades emergentes dos usuários, o sistema se torna cada vez menos útil.
2. **Aumento da complexidade**: a menos que um trabalho adicional seja feito para reduzir explicitamente a complexidade de um sistema, o sistema se tornará cada vez mais complexo devido a mudanças relacionadas à manutenção.
3. **Autorregulação**: o processo de evolução é auto regulado no sentido de que as medidas dos produtos e processos, que são produzidos durante a evolução, seguem distribuições próximas da normal.
4. **Conservação da estabilidade organizacional**: a taxa média de atividade global efetiva em um sistema em evolução é quase constante ao longo da vida útil do sistema. Em outras palavras, a quantidade média de esforço adicional necessária para produzir uma nova versão é quase a mesma.
5. **Conservação de familiaridade**: à medida que um sistema evolui, todos os tipos de pessoal, ou seja, desenvolvedores e usuários, por exemplo, devem obter um nível desejado de compreensão do conteúdo e comportamento do sistema para realizar uma evolução satisfatória. Um grande crescimento incremental em uma versão reduz essa compreensão. Portanto, o crescimento incremental médio em um sistema em evolução permanece quase o mesmo.
6. **Crescimento contínuo**: conforme o tempo passa, o conteúdo funcional de um sistema é continuamente aumentado para satisfazer as necessidades do usuário.
7. **Qualidade em declínio**: a menos que o design de um sistema seja cuidadosamente ajustado e adaptado a novos ambientes operacionais, as qualidades do sistema serão percebidas como em declínio ao longo da vida útil do sistema.
8. **Sistema de feedback**: o processo de evolução do sistema envolve *feedbacks multi-loop*, *feedbacks* multi-agente e *feedbacks* multinível entre diferentes tipos de atividades. Os desenvolvedores devem reconhecer essas interações complexas a fim de desenvolverem continuamente um sistema para fornecer mais funcionalidades e níveis mais elevados de qualidades.

A atualização regular do software junto com sua modernização é muito importante, ainda mais quando várias organizações investem muito dinheiro em seus sistemas de software, pois os consideram recursos críticos para seus negócios (UKESSAYS, 2012). Para todas as empresas que produzem e consomem software, existe uma grande necessidade de evolução (LEHMAN; RAMIL, 2003). A relevância da evolução do software está aumentando devido ao fato de que esta evolução é inevitável (MENS; KLEIN, 2012). Conforme salientado por Kagdi (2008), "as mudanças são as forças centrais que impulsionam a evolução do software e, portanto, não é surpreendente que um esforço primordial tenha sido (e deveria ser) dedicado à comunidade de engenharia de software para compreender, estimar e gerenciar sistematicamente as mudanças nos artefatos de software".

2.4 Mineração de Repositórios de Software

Com o advento de novas funcionalidades, correções de *bugs*, melhoria de *design* e refatoramento de código, é comum haver um aumento na quantidade de informações armazenadas referentes a várias versões do software. Os modernos processos de desenvolvimento de software frequentemente envolvem vários desenvolvedores e equipes de desenvolvimento, às vezes residindo em diferentes continentes e fusos horários (PONCIN *et al.*, 2011). A comunicação e a coordenação em tais projetos requerem suporte por meio de vários tipos de repositórios de software (HASSAN, 2008). Nesse sentido, os repositórios de *software* exercem um papel fundamental, pois servem como um local de armazenamento mantido, *online* ou *offline*, por várias organizações de desenvolvimento de software onde pacotes de software, código-fonte, *bugs* e muitas outras informações relacionadas ao software e seu processo de desenvolvimento são mantidas (SIDDIQUI; AHMAD, 2018).

Segundo Kagdi (2008), informações históricas e valiosas armazenadas em repositórios de software fornecem uma grande oportunidade de adquirir conhecimento e ajudar no monitoramento de projetos e produtos complexos sem interferir nas atividades de desenvolvimento e prazos. A premissa é que investigações empíricas e sistemáticas de repositórios lançarão uma nova luz sobre o processo de evolução de software e as mudanças que ocorrem ao longo do tempo, revelando informações pertinentes, relacionamentos ou tendências sobre características evolutivas específicas do sistema (KAGDI *et al.*, 2007). Conforme explanado no Quadro 1, esses dados são alocados em diversos tipos de repositórios com diferentes propósitos.

Quadro 1 – Exemplos de Repositórios de Software

Repositório	**Descrição**
Repositórios de controle de versão	Esses repositórios registram o histórico de desenvolvimento de um projeto. Eles acompanham todas as alterações no código-fonte, juntamente com metadados sobre cada alteração, por exemplo, o nome do desenvolvedor que realizou a alteração, a hora em que a alteração foi realizada e uma pequena mensagem descrevendo a alteração. Os repositórios de controle de versão são os repositórios mais comumente disponíveis e usados em projetos de software. CVS, Git , Subversion, Perforce e ClearCase são exemplos de repositórios de controle de versão que são usados na prática.
Issue Tracking Systems (ITS)	ITS é um sistema de computador que ajuda a manter e acima de tudo gerenciar listas de *issues*, que possam ocorrer no desenvolvimento de um software (BERTRAM *et al.*, 2009). Nos ITSs pode-se compartilhar as informações com a equipe, ter uma visão geral instantânea do estado do software, decidir com habilidade sobre a liberação, definir e atualizar a importância de correções e ajustes individuais, e ter um histórico de mudanças registrado (JANÁK, 2009). Bugzilla e Jira são exemplos de ITS.
Comunicações Arquivadas	Esses repositórios rastreiam discussões sobre várias facetas de um projeto de software para o longo de sua existência. Listas de e-mails, chats de IRC e mensagens instantâneas são exemplos de comunicações arquivadas sobre um projeto.
Logs de implantação	Esses repositórios registram informações sobre a execução de uma única implantação de um aplicativo de software ou implantações diferentes dos mesmos aplicativos. Por exemplo, os *logs* de implantação podem registrar as mensagens de erro relatadas por um aplicativo em vários sites de implantação. A disponibilidade de *logs* de implantação continua a aumentar em uma taxa rápida devido ao seu uso para resolução remota de problemas.
Repositórios de código	Arquivam o código fonte e inúmeros repositórios de controle de versão de diferentes projetos. O Sourceforge e o GitHub são exemplos de grandes repositórios de código.

Fonte: Hassan (2008).

Esses repositórios contêm uma riqueza de informações e fornecem uma visão única do caminho evolutivo real percorrido para realizar um sistema de software (KAGDI *et al.*, 2007). No entanto, os dados neles armazenados podem encontrar-se de forma aglomerada, muitas vezes precisando de técnicas apropriadas de coleta para serem extraídos e, subsequentemente, utilizados para algum propósito. Verifica-se atualmente o desenvolvimento de um amplo espectro de abordagens na Engenharia de Software que visam extrair informações pertinentes e descobrir relações e tendências de repositórios de software no contexto da evolução de sistemas (FARIAS *et al.*, 2016). Usufruindo de tais informações, os profissionais podem depender menos de sua intuição e experiência e depender mais de dados históricos e de campo (HASSAN, 2008).

Portanto, o armazenamento dos referidos dados é de suma importância para empresas desenvolvedoras de software, pois, através deles, usando de procedimentos de Mineração de Dados, torna-se possível aprimorar o processo o desenvolvimento de um novo produto tendo como base experiências anteriores. Além disso, compreender o processo de evolução do software é uma tarefa difícil e grandes sistemas tendem a ter uma longa história de desenvolvimento, com muitos desenvolvedores diferentes trabalhando em diferentes partes do sistema. É comum que nenhum desenvolvedor conheça todo o código-fonte do projeto. Por causa disso, a ideia de uma

análise manual de todo o software é impraticável (SOKOL *et al.*, 2013). Logo, a Mineração de Dados exerce um suporte fundamental ao viabilizar a obtenção e exploração de uma grande quantidade de dados a fim de encontrar padrões, para que, posteriormente, seja possível uma adequação desses dados e assim seja possível a sua disponibilização para determinados fins de estudos (SIDDIQUI; AHMAD, 2018). Por tal motivo, a Mineração de Dados tem se demonstrado uma aliada fundamental para a análise massiva de repositórios de software.

A Mineração de Repositórios de Software (MRS) é uma área que analisa os ricos dados disponíveis em repositórios de software para descobrir informações interessantes e acionáveis sobre sistemas e projetos (HASSAN, 2008). Nas últimas décadas, constata-se um constante crescimento da quantidade de dados e, embora muitas informações estejam disponíveis, muitas encontram-se escondidas na vasta coleção de dados brutos (VANDECRUYS *et al.*, 2008). Através da MRS, torna-se cada vez mais fácil para desenvolvedores usufruir das informações oriundas de múltiplas versões anteriores de sistemas e de projetos desenvolvidos previamente (CHATURVEDI *et al.*, 2013). Tal conhecimento pode trazer grandes melhorias para evolução de software, como descobertas e correções de *bugs*, melhorias de interface, aumento de desempenho, ganho no tempo de desenvolvimento, etc (ZAIDMAN *et al.*, 2008).

Apesar de ser um campo que traz uma quantidade significativa de benefícios, a MRS não era uma área muito explorada até recentemente. Segundo Hassan (2008), dois motivos ocasionavam tal desinteresse. O primeiro era o fato de que a maior parte dos grandes repositórios, nos quais se poderiam extrair alguma informação relevante, pertenciam a instituições privadas que não compartilhavam o acesso aos dados. O outro se dava devido a parcela dos repositórios que não era projetada para automatizar a extração da grande quantidade de dados, o que tornava o procedimento limitado. Todavia, com a disseminação e fortalecimento de Sistemas de Código Aberto (em inglês, *Open Source Systems*), possibilitou o acesso a uma rica diversidade de repositórios de sistemas (FELLER *et al.*, 2002). Profissionais e pesquisadores de software estão reconhecendo os benefícios de minerar essas informações para apoiar a manutenção de sistemas de software, melhorar o *design* de software e validar empiricamente novas ideias e técnicas (CHATURVEDI *et al.*, 2013). Diante de tal progresso, a MRSvem somando conquistas significantes, como a criação de técnicas para automatizar e melhorar a extração de informações de repositórios e a descoberta de novas técnicas para minerar informações importantes a partir de repositórios públicos, como o GitHub e Etherscan (HASSAN, 2008; D'AMBROS *et al.*, 2008).

2.4.1 GitHub

O GitHub é um site de hospedagem de código colaborativo construído sobre o sistema de controle de versão do *Git* que introduziu um modelo "*fork and pull*", no qual permite que contribuidores em potencial do projeto façam uma cópia pessoal de qualquer projeto público onde possam fazer alterações, adicionar ou alterar a funcionalidade, sem afetar o código na *branch* original e, em seguida, enviar uma *pull request* quando desejarem que o mantenedor do projeto "puxe" suas mudanças para a *branch* principal (TSAY *et al.*, 2014). O GitHub não está somente oferecendo um serviço de hospedagem de código, como seus concorrentes faziam há muito tempo, mas também uma ferramenta online, fácil de usar e barata para desenvolvimento de software colaborativo a qual conta com muitos recursos de suporte a comunidade (LIMA *et al.*, 2014). Ele possui uma espécie de rede social, que permite a seus usuários seguirem outros usuários, e a visualizarem um *feed* de informações sobre projetos de interesse do usuário. O *feed* apresenta um histórico recente de *following* (Usuário A segue usuário B), *watching* (A está acompanhando qualquer repositório de B), *commits*, *issues*, solicitações de *pull* e ações de comentário do repositório em questão (DABBISH *et al.*, 2012). Além disso, o GitHub possui um ambiente que integra outras funcionalidades como *wiki*, rastreamento de *issues* e revisão de código (THUNG *et al.*, 2013; YU *et al.*, 2014). Atualmente, o GitHub hospeda mais de 100 milhões de projetos mantidos por mais de 56 milhões de desenvolvedores registrados no mundo todo (GITHUB, 2021).

Kalliamvakou *et al.* (2014) relata que os recursos sociais integrados e a disponibilidade de metadados por meio de uma API acessível tornaram o GitHub muito atraente para os pesquisadores de engenharia de software, fazendo com que realizem estudos qualitativos e quantitativos na plataforma. Esta API é projetada para retornar todas as informações sobre um usuário, incluindo repositórios e suas características (CHATZIASIMIDIS; STAMELOS, 2015). Os estudos qualitativos se concentraram em como os desenvolvedores usam os recursos sociais do GitHub para formar impressões e tirar conclusões sobre outros desenvolvedores (KALLIAMVAKOU *et al.*, 2014). Além disso, as atividades dos projetos permitem avaliar o sucesso, o desempenho e as possíveis oportunidades de colaboração. Já os estudos quantitativos têm como objetivo coletar sistematicamente os dados disponíveis publicamente do GitHub e usá-los para investigar práticas de desenvolvimento no ambiente do mesmo (KALLIAMVAKOU *et al.*, 2014).

2.4.2 Etherscan

O Etherscan é atualmente a ferramenta *block explorer* mais utilizada para obter-se dados do *blockchain* da plataforma Ethereum. É operada e desenvolvida por uma equipe independente, com os tipos de informações descentralizadas e aplicações de infra-estrutura que a Ethereum possibilita (ETHERSCAN, 2017). Um *block explorer* é basicamente um mecanismo de pesquisa que permite aos usuários investigar, confirmar e validar transações que ocorreram em um *blockchain* (ETHERSCAN, 2017). Contudo, precisa ser notado que não pode-se usar um *block explorer* de um *blockchain* em outro. Por exemplo, você não pode rastrear transações do Litecoin com um *block explorer* do Bitcoin (KHATWANI, 2018).

O Etherscan disponibiliza para seus usuários dados relativos a *tokens* digitais, no qual dividem-se em diferentes categorias, como ERC20 e ERC721. Para ambas categorias pode-se encontrar uma lista de transferências, onde a mesma contém todas as transferências realizadas no *blockchain* atualizada a cada segundo. Em cada transferência, disponibilizam-se dados como o *hash* da transação, *hash* do usuário que transferiu e o *hash* do destinatário, além do referente valor transferido e qual token foi utilizado na transação. Referindo-se novamente as categorias, ambas possuem um *ranking* contendo os *Top Tokens* de cada uma, ou seja, a lista ordenada com todos os *tokens*, iniciando com aqueles que possuem um maior valor de mercado. Tal lista disponibiliza o nome do *token*, preço, porcentagem de variação do valor, valor de mercado e a quantidade de detentores do *token*.

Usuários podem ainda encontrar os códigos-fonte dos *tokens* operados na plataforma. Para realizar tal ação, o usuário pode efetuar uma pesquisa utilizando o nome do mesmo ou encontrá-los através das listas de transferências ou *top tokens*, coletando-os manualmente. Todavia, a plataforma também disponibiliza uma API para coleta de dados onde é possível realizar buscas e salvar dados utilizando linguagens de programação. Para a utilização da API o usuário necessita realizar um cadastro na plataforma ao qual lhe disponibilizará uma *Api-Key Token*, que deverá ser utilizada para a obtenção dos dados. Além disso, a API poderá ser utilizada por desenvolvedores de aplicações descentralizadas, possibilitando o envio de seus contratos, a fim de serem disponibilizados e auditados por toda a comunidade.

3 TRABALHOS RELACIONADOS

Com o objetivo de esclarecer os trabalhos relacionados à presente pesquisa, apresenta-se a seguir uma discussão focada nas pesquisas empíricas e exploratórias que tratam sobre contratos inteligentes à luz dos conceitos de manutenção e evolução de software. Por fim, realiza-se uma síntese comparativa entre as referências visando evidenciar a lacuna de pesquisa explorada neste trabalho.

Chen *et al.* (2020) apresentam um amplo estudo empírico sobre manutenção de contratos inteligentes na Ethereum. Para realização da pesquisa, primeiro conduziu-se uma revisão sistemática da literatura para analisar 131 artigos de pesquisa relacionados a contratos inteligentes publicados de 2014 a 2020 com o objetivo de verificar uma possível desatualização de publicações anteriores e checar a possibilidade de haver uma lacuna prática entre a academia e a indústria. Em uma segunda etapa, realizou-se uma pesquisa online com desenvolvedores de contratos inteligentes no GitHub para validar as descobertas, o que gerou um total de 165 respostas úteis recebidas. Inicialmente, eles identificaram 10 tipos de problemas relacionados à manutenção corretiva, adaptativa e preventiva de contratos inteligentes e outros 5 problemas relacionados ao processo de manutenção geral. Os autores também resumiram os métodos de manutenção usados atualmente para contratos inteligentes de 41 publicações e os dividiram em três categorias, métodos de verificação *offline*, métodos de verificação *online* e outros métodos. De acordo com o estudo, verificou-se que existem ricas perspectivas de linhas de pesquisas futuras, sendo assim, propostas sugestões tanto para desenvolvedores de contratos inteligentes como para pesquisadores. O estudo contribuiu através da apresentação de *insights* para apoiar desenvolvedores de contratos inteligentes a manterem melhor seus projetos, bem como também destacou algumas orientações importantes para pesquisas futuras ao qual servem para melhorar o ecossistema Ethereum.

Oliva *et al.* (2020) tiveram como objetivo ter um entendimento mais amplo de todos os contratos que são atualmente implantados no Ethereum. Em particular, eles esclarecem com que frequência os contratos são usados, em relação ao nível de atividade, o que eles fazem, sua categoria e quão complexos são seus códigos-fonte. Para conduzir este estudo, eles extraíram e cruzaram dados de quatro fontes: Ethereum *dataset* na plataforma Google BigData, Etherscan, estados de *dApps* e *Coin MarketCap*. Em termos de resultados, os autores verificaram que o nível de atividade está concentrado em um subconjunto muito pequeno dos contratos e quase três quartos deles são verificados. Apesar da crescente utilização de aplicações movidas o *blockchain*,

a principal aplicação de contratos inteligentes ainda é restrita ao gerenciamento de *tokens*. O códigos-fonte de contratos de alta atividade que foram verificados são pequenos, geralmente incluem pelo menos 2 bibliotecas/subcontratos e são amplamente documentados.

Tonelli *et al.* (2018) estudam a hipótese de que, devido serem desenvolvidos para posterior execução em um *blockchain*, o código de contratos inteligentes deve satisfazer diversas restrições. Tais restrições são, inclusive, diferentes das tradicionalmente enfrentadas na Engenharia de Software. Dessa forma, o tamanho do código, os recursos computacionais, a interação entre as diferentes partes do software poderão ser todos limitados e não possuírem a mesma liberdade de desenvolvimento, quando comparados com softwares tradicionais. A pesquisa investigou doze mil contratos inteligentes, escritos em Solidity, e implantados na Ethereum. Os autores analisaram as estatísticas de um conjunto de métricas de software relacionadas aos contratos inteligentes e posteriormente compararam às métricas extraídas de projetos de software tradicionais. Os resultados obtidos mostraram que geralmente as métricas de contratos inteligentes têm faixas mais restritas do que as métricas correspondentes em sistemas de software tradicionais. Entretanto, algumas métricas, como linhas de código, seguem uma distribuição normal que se assemelha ao comportamento já encontrado em software tradicionais.

Por sua vez, Ajienka *et al.* (2020) examinam contratos inteligentes de 11 projetos de software, orientados em *blockchain*, da Ethereum hospedados em GitHub e avaliam os recursos necessários para sua implantação, ou seja, o gás usado. Para cada um desses contratos, também são extraídas um conjunto de métricas orientadas a objetos, para avaliar suas características estruturais. Os resultados obtidos mostram uma correlação estatisticamente significativa entre algumas das métricas utilizadas em Orientação a Objetos e os recursos consumidos na rede Ethereum ao implantar o contrato inteligente. Esses resultados são significativos e demonstram um impacto nas práticas de desenvolvimento de contratos inteligentes. Em um nível mais alto, os resultados guiam os profissionais sobre as mudanças estruturais ou refatorações que podem ser feitas para minimizar os recursos de implantação. Por fim, para os desenvolvedores de contratos inteligentes, as métricas extraídas demonstram-se úteis para informar a quantidade de gás que poderão destinar para a execução do mesmo.

Pinna *et al.* (2019) também realizam um amplo estudo empírico sobre contratos inteligentes implantados na Ethereum. O objetivo da pesquisa foi analisar os resultados empíricos sobre recursos de contratos inteligentes, transações de contratos inteligentes dentro do *blockchain*, o papel da comunidade de desenvolvimento e as características do código-fonte. Os autores

coletaram um conjunto de mais de dez mil códigos-fonte de contratos inteligentes e um conjunto de dados de metadados sobre sua interação com o *blockchain*, dados estes disponíveis no Etherscan. Os dados coletados foram examinados computando estatísticas diferentes sobre políticas de nomenclatura, o saldo em Ether de cada contrato inteligente, número de transações, funções e outras métricas que caracterizam o uso e a finalidade dos contratos inteligentes. Ao final, verificou-se que o número de transações e os saldos seguem a distribuição da "*power-law*" e as métricas do código fonte exibem, em média, valores inferiores às métricas correspondentes ao software padrão, porém com grandes variações. Ao dar ênfase aos 20 contratos inteligentes com o maior número de transações, descobriu-se que a maioria deles representa contratos inteligentes de escopo financeiro e alguns deles têm histórias peculiares de desenvolvimento de software por trás deles. Os resultados mostram que o software presente no *blockchain* está mudando e evoluindo rapidamente e não é mais dedicado apenas a criptomoedas, mas à computação de propósito geral.

Os trabalhos apresentados serviram como referências para a presente pesquisa tendo em vista o escopo exploratório com foco na plataforma Ethereum. Tais artigos também contribuíram para evidenciar a relevância quanto ao estudo de aspectos relacionados à evolução de software no contexto de contratos inteligentes. Verificou-se, nesse sentido, que as pesquisas são bastante recentes e que, consequentemente, muitos desafios ainda se fazem presentes. Logo, os estudos previamente discutidos investigam os desafios relativos ao desenvolvimento de contratos inteligentes, no entanto, nenhum deles foca exclusivamente na compreensão da relação e similaridade entre o que está exposto no Etherscan e o que foi desenvolvido de forma *open source* no GitHub. A pertinência dessa análise se justifica dada a relevância em compreender as particularidades envolvidas no processo de evolução *open source* de contratos inteligentes em decorrência da natureza inalterável da tecnologia *blockchain*, bem como aprofundar os estudo sobre possíveis dissonâncias entre o contrato inteligente exposto no GitHub e o que realmente está sendo utilizado na Ethereum.

4 ESTUDO EMPÍRICO

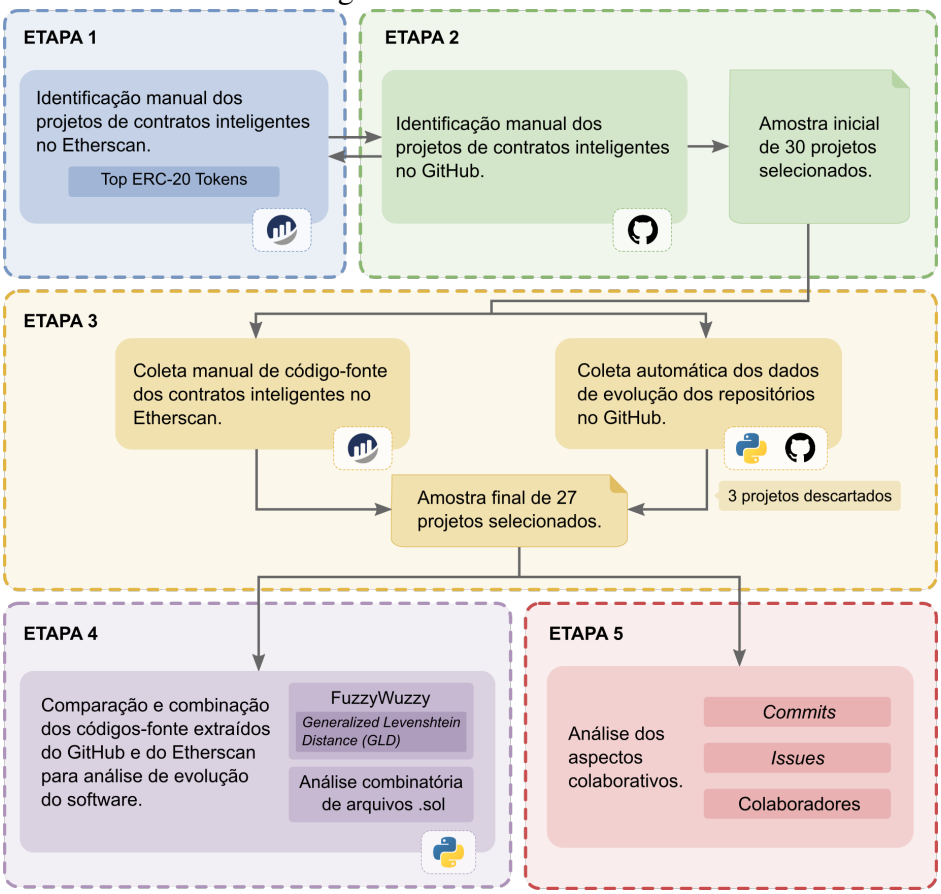
Neste capítulo apresenta-se inicialmente os procedimentos metodológicos adotados nesta pesquisa. Em seguida, tem-se uma seção exclusiva para discussão dos resultados obtidos.

4.1 Procedimentos Metodológicos

Os contratos inteligentes implantados na Ethereum apresentam um ciclo de vida de software diferente do usual quando comparados a projetos tradicionais. Além da característica da imutabilidade de dados a qual impacta diretamente na evolução do software, o processo de desenvolvimento de contratos inteligentes também deve lidar com diferentes demandas intrínsecas ao domínio em questão, como a garantia da simplicidade, a incidência de custos operacionais em criptomoedas e a transparência quanto o acesso ao código-fonte do contrato inteligente. Em específico, a transparência de código pode ser observada através de ferramentas como o Etherscan, o qual proporciona acesso público a uma vasta quantidade de informações dos contratos inteligentes utilizados na Ethereum. Ademais, tendo em vista a ascensão do desenvolvimento *open source*, é de se constatar que uma parcela considerável dos contratos inteligentes também se façam presentes em plataformas sociais de hospedagem de código-fonte aberto, com destaque para o GitHub.

Diante de tal contexto, este estudo se posiciona como uma pesquisa exploratória-descritiva cujo objetivo consiste em proporcionar um melhor entendimento da temática através da investigação e descrição sobre como ocorre o processo de evolução de software de contratos inteligentes a partir de dados e informações disponíveis nos repositórios provenientes do Etherscan e GitHub. Quanto ao escopo metodológico adotado, optou-se por um experimento computacional baseado em Mineração de Repositórios 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 em um domínio recente, como o de contratos inteligentes. Sob o ponto de vista analítico, esta pesquisa optou por um enfoque orientado por uma perspectiva quali-quantitativa devido a necessidade de uma visão mista em decorrência da confluência de elementos quantitativos e subjetivos obtidos durante a coleta de dados.

Figura 5 – Procedimentos Metodológicos



Fonte: Autoria própria.

Conforme ilustrado na Figura 5, o presente trabalho adotou um plano metodológico baseado em cinco etapas principais. Inicialmente, na **Etapa 1** (tracejada com cor azul), realizou-se uma análise no Etherscan para identificação de contratos inteligentes 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 contratos inteligentes 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). O valor de mercado se refere ao preço em que o ativo está sendo comercializado, isto é, o quanto o mercado está pagando para sua obtenção (INVEST, 2020).

Em seguida, na **Etapa 2** (tracejado com cor verde), realizou-se, a partir dos contratos inteligentes identificados anteriormente, uma busca manual junto à plataforma GitHub para identificação dos repositórios *open source* oficiais de tais contratos. Nesse sentido, foi possível constatar que parcela dos projetos selecionados na Etapa 1 não estavam disponíveis no GitHub. Dessa forma, substituíram-se os contratos em questão pelos contratos imediatamente subsequentes da lista ERC20 *Top Tokens* mas que também estivessem disponíveis no GitHub, visando, assim, totalizar a amostra de 30 contratos inteligentes. Para se certificar da corretude

do repositório selecionado via GitHub, considerou-se os seguintes critérios como protocolo de pesquisa para identificação:

1. O *nome do repositório* no GitHub deve ter alguma relação com o nome do contrato presente 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 demonstrar a linguagem como sendo JavaScript. Neste caso, foi realizada uma inspeção manual para certificação de que a linguagem prioritária era Solidity;
3. Devem existir arquivos escritos no *formato de contratos inteligentes* e com extensão ‘‘`.sol`’’;
4. A *descrição do projeto*, quando disponível, deve ter relação com o contrato inteligente disponível no Etherscan.

Em posse da lista de contratos inteligentes dos projetos (disponíveis no Etherscan e GitHub, respectivamente) a serem investigados, deu-se início a **Etapa 3** (tracejado amarelo) com o objetivo de concretizar o processo de mineração de dados. O processo de coleta dos códigos-fonte dos contratos inteligentes disponíveis no Etherscan foi realizado de forma manual, haja vista que o Etherscan fornece, através de sua versão web, fácil acesso ao arquivo[1] implementado em Solidity de cada contrato disponível na Ethereum. Assim, 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 contrato inteligente 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[2] disponibilizada pelo próprio GitHub para a automatização do processo. Através de *scripts* em Python (disponíveis no Apêndices A e B), coletou-se, para cada um dos 30 projetos no GitHub, a lista de *commits* e códigos-fonte dos contratos inteligentes, lista de colaboradores, dados relacionados às *issues* como seus respectivos números, título, nome do colaborador que submeteu, *status* (aberta ou fechada), data de criação, data de atualização e data de fechamento (se a *issue* estiver fechada), além da quantidade de *commits* de cada colaborador desde a início do projeto no GitHub. Com exceção dos códigos-fonte dos contratos inteligentes os quais foram armazenados em arquivos individuais do tipo `.sol` (Solidity), todos os demais dados foram armazenados em arquivos `.csv` visando facilitar a gestão dos dados obtidos. Assim, a partir da

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

relação de *commits* obtidos, torna-se possível o mapeamento de todas as versões do projeto ao longo de sua evolução histórica, haja vista que as respectivas versões dos códigos-fonte também foram coletadas. Todos os *scripts* e dados obtidos encontram-se disponíveis no repositório de apoio deste trabalho (RODRIGUES *et al.*, 2021).

Por sua vez, a **Etapa 4** (tracejado roxo), de cunho analítico, teve o objetivo de desenvolver e aplicar um método experimental para comparar a similaridade da versão do contrato inteligente disponível no Etherscan com todas as versões do referido contrato disponíveis no GitHub ao longo de sua evolução. Primeiramente, utilizou-se de uma função de similaridade de *string* para mensurar a porcentagem de similaridade entre os referidos arquivos. Segundo Yujian e Bo (2007), a *Generalized Levenshtein Distance* (GLD) é medida mais promissora para comparar *strings* que utiliza várias operações de edição, geralmente incluindo a exclusão, inserção e substituição de símbolos individuais. Essa medida, também chamada de "distância de edição", pode ser definida como o custo mínimo de transformação de uma *string* em outra por meio de uma sequência de operações de edição ponderadas. Em termos operacionais, 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 (ARIAS, 2019). 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, por exemplo, a frase "Nosso mundo" será dada como igual a "Mundo nosso". Como parte do método proposto, elaborou-se uma expressão regular em Python (disponível no Apêndice C) para remoção dos espaços em branco e dos comentários nos códigos-fonte dos contratos inteligentes visando possibilitar uma comparação mais adequada, processo este conhecido como *pretty printing* (JONGE, 2002).

Outro procedimento relevante conduzido durante a Etapa 4 diz respeito ao fato de que, muitas vezes, o contrato inteligente implantado na Ethereum é dividido em diferentes arquivos `.sol` durante o processo de desenvolvimento no GitHub. Assim, 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 (disponível no Apêndice D) o qual verifica, para cada *commit*, a combinação entre arquivos `.sol` que apresenta maior porcentagem de similaridade quando comparado a versão do contrato inteligente disponível no

Etherscan. Como resultado da Etapa 4, tem-se a categorização quanto ao comportamento de evolução de software dos projetos e a investigação qualitativa sobre as diferenças observadas nos códigos-fonte. Entretanto, durante esse processo verificou-se a necessidade de descartar três projetos (Aragon, Bancor e Synthetix Network Token) em decorrência da complexidade computacional necessária para realizar todas as comparações haja vista a quantidade de *commits* e arquivos `.sol` em cada projeto. No caso do Aragon, há seis *commits* com 44 arquivos `.sol`, onde no pior dos casos, para cada *commit*, tem-se C_{22}^{44} gerando um total de $2,104098964X10^{12}$ combinações possíveis. Já Bancor possui dois *commits* com um total de 74 arquivos `.sol`. Logo, tem-se no pior dos casos C_{37}^{74} com um total de $1,74613564X10^{21}$ combinações possíveis. Por fim, o Syntethix Network Token possui quatro *commits* com 108 arquivos e, para cada um, no pior dos casos, tem-se C_{54}^{108}, ou seja, um total de $2,48577844X10^{31}$ combinações possíveis.

Finalmente, na **Etapa 5** (tracejado em vermelho), tem-se a análise dos aspectos colaborativos oriundos dos dados quantitativos e qualitativos obtidos a partir do repositório no GitHub de cada projeto. A partir de tal análise, tornou-se possível verificar elementos associados ao processo de desenvolvimento como, projetos com mais atividade em termos de *commits*, participação de colaboradores e resolução de *issues*.

4.2 Resultados e Análises

Este estudo traz duas principais perspectivas de análises. A primeira refere-se a caracterização dos padrões de evolução dos contratos inteligentes objetivando, assim, compreender a similaridade e comportamento dos contratos inteligentes desenvolvidos de forma *open source* no GitHub em contraste à versão disponível no Etherscan. A segunda vertente de análise traz uma discussão dos aspectos colaborativos oriundos dos dados obtidos do repositório de cada projeto no GitHub. Nesse sentido, busca-se analisar a relação entre *commits* e colaboradores, bem como o engajamento da comunidade em relação ao *report* e resolução de *issues*.

4.2.1 Caracterização dos Padrões de Evolução

Conforme destacado por Cook *et al.* (2000), a evolução do software diz respeito a sequência de mudanças em um software ao longo de sua vida útil, abrangendo, assim, atividades de desenvolvimento e manutenção. A partir de tal percepção, esta seção busca discutir o comportamento evolutivo dos projetos sob estudo e, assim, apresentar uma categorização quanto

aos padrões de evolução no contexto de contratos inteligentes. Para traçar essa linha evolutiva de cada projeto, realizou-se uma análise combinatória entre todos os arquivos `.sol` de cada versão do projeto no GitHub visando verificar a similaridade quando comparado à versão do contrato inteligente disponível no Etherscan. Como limiar de análise, considerou-se que comparações que apresentem uma similaridade igual ou superior a 90% serão classificadas como *alta similaridade*. Caso contrário, considerar-se-á de *baixa similaridade*. Diante dessa concepção, torna-se possível a definição de quatro macro categorias que denotam padrões de evolução para os contratos inteligentes a partir de dados disponíveis nos repositórios via GitHub e Etherscan:

- **Padrão de Evolução Alpha**: O código-fonte dos contratos 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 dos contratos no GitHub é de *alta similaridade* ao encontrado no Etherscan no início do desenvolvimento, porém é modificado em algum momento da evolução no GitHub e apresenta *baixa similaridade* até o último *commit*;
- **Padrão de Evolução Gama**: O código-fonte dos contratos 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 contratos 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 de evolução previamente 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 7 iniciativas (25.9%). Em seguida, com 6 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 apresentam acima de 90% de similaridade em algum momento da evolução do software. Diante de tais cenários, apresenta-se na Tabela 1 uma visão geral sobre os projetos investigados, incluindo o padrão de evolução identificado, nome do projeto, posição no ranking ERC-20 *Top Tokens* na data da coleta, descrição do projeto, data de *deploy* de acordo com o Etherscan, data de início no GitHub e valor da máxima similaridade obtida ao longo de todo período de evolução.

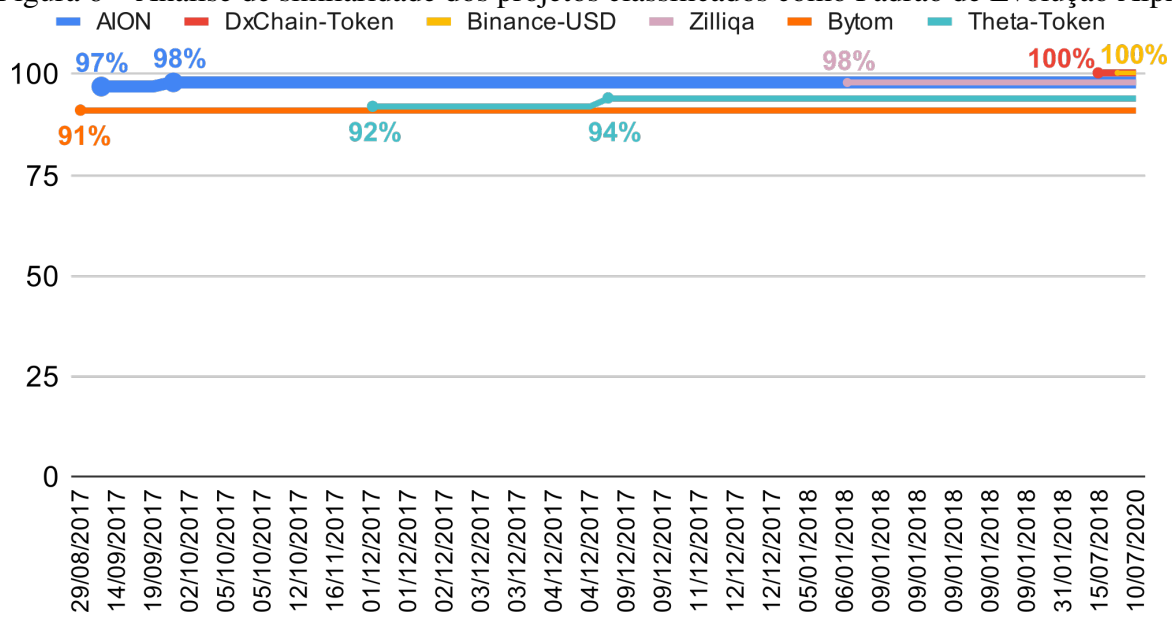
Tabela 1 – Visão geral dos projetos analisados

Padrão de Evolução	Projetos	Ranking ERC-20 Top Tokens	Descrição do Projeto	Data de deploy Etherscan	Data de início no GitHub	Máxima similaridade
Alpha	Binance-USD	31	Binance USD (BUSD) é um stablecoin respaldado em dólares, emitido e custodiado pela Paxos Trust Company	09/09/2019	10/07/2020	100
	DxChain-Token	33	Fornece soluções para problemas de big data	20/07/2018	15/07/2018	100
	AION	78	Um sistema *blockchain* de várias camadas projetado para resolver questões não resolvidas de escalabilidade, privacidade e interoperabilidade em redes *blockchain*	11/10/2017	10/09/2017	98
	Zilliqa	30	Zilliqa é uma plataforma de *blockchain* pública de alto rendimento projetada para escalar para milhares de transações por segundo.	12/01/2018	06/01/2018	98
	Bytom	38	Transfira ativos do mundo atômico para o byteworld	22/08/2017	29/08/2017	91
	Theta-Token	29	Uma rede ponto a ponto descentralizada que visa oferecer entrega de vídeo aprimorada a custos mais baixos.	06/03/2019	01/12/2017	94
Gama	Golem	68	Golem vai criar o primeiro mercado global descentralizado para poder de computação	18/11/2020	29/09/2016	100
	HEDG	8	HedgeTrade é uma plataforma onde os traders compartilham seus conhecimentos.	30/03/2019	11/10/2018	100
	Nexo	36	Empréstimos instantâneos garantidos por criptografia	20/04/2018	18/04/2018	98
	Paxos-Standard	25	O Paxos Standard é um criptocivo regulado totalmente garantido 1: 1 pelo dólar americano, aprovado e regulamentado pelo NYDFS.	08/09/2018	16/07/2018	98
	Swipe	45	Swipe é uma carteira de criptomoeda e cartão de débito que permite aos usuários gastar suas criptomoedas em todo o mundo.	16/08/2019	08/07/2019	100
	ChainLink-Token	3	Um middleware baseado em *blockchain*, agindo como uma ponte entre os contratos inteligentes de criptomoeda, feeds de dados, APIs e pagamentos de contas bancárias tradicionais.	23/09/2017	30/05/2017	97
	Decentraland	67	Decentraland é uma plataforma de realidade virtual alimentada pelo *blockchain* Ethereum. Os usuários podem criar, experimentar e monetizar conteúdo e aplicativos	15/08/2017	24/09/2017	94
Delta	EnjinCoin	34	Criptomoeda personalizável e plataforma de bens virtuais para jogos.	03/10/2017	22/08/2017	9
	EthLend	12	Aave é um protocolo de código aberto e não custodial para ganhar juros sobre depósitos e pedir ativos emprestados. Ele também oferece acesso a empréstimos flash altamente inovadores, que permitem que os desenvolvedores tomem emprestado de forma instantânea e fácil; nenhuma garantia necessária.	17/09/2017	09/05/2017	8
	HuobiToken	6	Huobi Global é um grupo de serviços financeiros de criptomoeda líder mundial.	27/02/2018	11/12/2018	4
	KyberNetwork	22	KyberNetwork é um novo sistema que permite a troca e conversão de ativos digitais.	12/09/2017	01/09/2018	6
	MCO	62	Crypto.com, a plataforma pioneira de pagamentos e criptomoedas, busca acelerar a transição do mundo para a criptomoeda.	26/06/2017	03/01/2018	6
	RLC	52	Computação em nuvem distribuída baseada em *blockchain*	18/04/2017	03/10/2017	50
	SeeleToken	124	A Seele criou o "Algoritmo de consenso neural". Ele transforma o problema de consenso em uma solicitação assíncrona, processando e classificando os dados em um ambiente de grande escala baseado em "números micro-reais".	04/04/2018	06/02/2018	64
	TrueUSD	26	Um stablecoin regulado, independente de câmbio, garantido 1 por 1 com dólares americanos.	18/12/2018	09/05/2018	3
	VeChain	7	Visa conectar a tecnologia *blockchain* ao mundo real, bem como integração IoT avançada.	15/08/2017	11/07/2018	5
	WAX-Token	64	Mercado global descentralizado para ativos virtuais.	30/12/2017	11/12/2017	58
	OKB	17	Trocas de ativos digitáis	Data não disponível	16/03/2020	13
	PowerLedger	90	Power Ledger é um mercado ponto a ponto para energia renovável.	11/09/2017	23/09/2018	83
	Reputation	27	Augur combina a magia dos mercados de previsão com o poder de uma rede descentralizada para criar uma ferramenta de previsão incrivelmente precisa	07/12/2018	09/05/2017	8
	Dai-Stablecoin	20	Multi-Collateral Dai, traz uma série de recursos novos e interessantes, como suporte para novos tipos de colaterais CDP e Dai Savings Rate.	14/11/2019	28/05/2018	42

Fonte: Autoria própria.

Buscando aprofundar a caracterização dos padrões de evolução identificados, são discutidas a seguir as evidências que denotam o comportamento diagnosticado para cada padrão. Nesse sentido, a Figura 6 traz uma visualização sobre a porcentagem de similaridade das versões disponíveis no GitHub, ao longo do processo de desenvolvimento, quando comparadas à versão do Etherscan, referente aos projetos enquadrados como **Padrão de Evolução Alpha**, ou seja, cujo código-fonte presente no GitHub apresenta *alta similaridade* ao do Etherscan desde o início do projeto no GitHub e nunca reduz. Conforme pode-se verificar, todos os projetos apresentam acima de 90% de similaridade desde o início do projeto 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 contrato inteligente desenvolvido no GitHub foi decomposto em nove arquivos `.sol` separados, demandando, assim, linhas de código adicionais para declaração da versão do compilador e demais importações, que ao serem combinados, causaram uma redução da similaridade. Para realizar essa verificação de diferença de similaridade entre o contrato disponível no GitHub e no Etherscan, realizou-se uma comparação manual entre a versão com maior similaridade identificada no GitHub. Dessa forma, tornou-se possível checar especificamente as diferenças de implementação.

Figura 6 – Análise de similaridade dos projetos classificados como Padrão de Evolução Alpha



Fonte: Autoria própria.

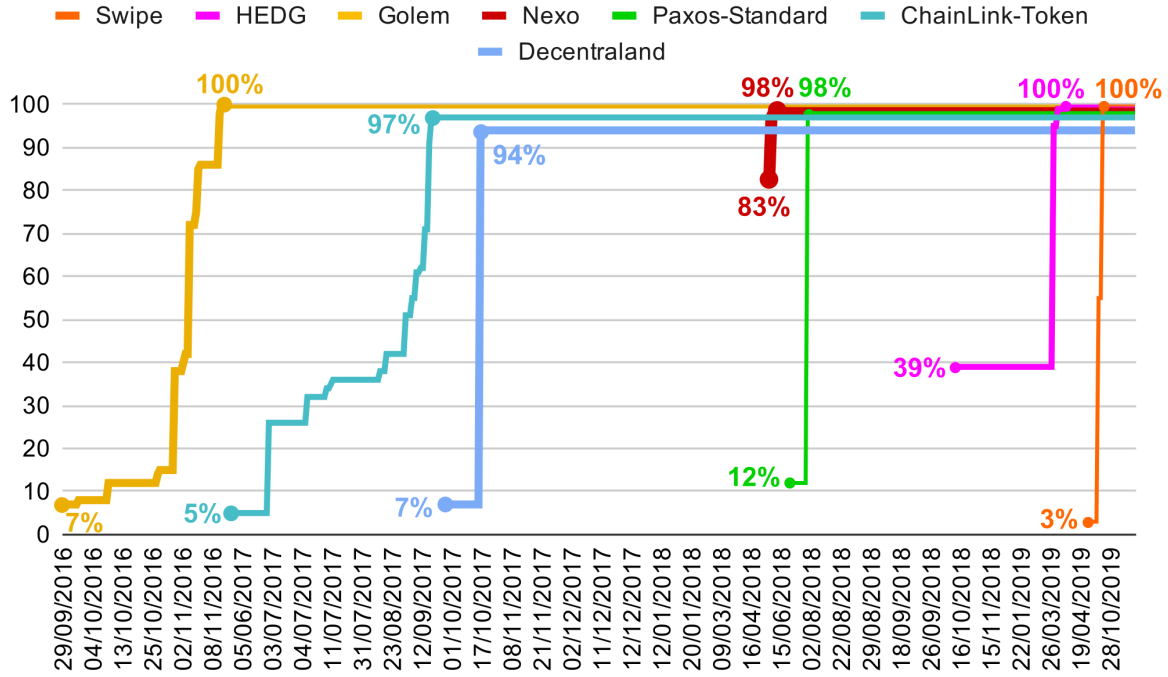
Ademais, conforme identificado na Figura 6, o Bytom também apresenta similaridade estável durante toda a evolução coletada, todavia é o menor valor (91%) dentre os projetos

nessa categoria. O motivo dessa diferença se dá em decorrência do contrato 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 contratos inteligentes em arquivos separados e posteriormente agrupados para *deploy* na Ethereum. A segunda observação refere-se ao fato das modificações estarem atreladas à mudanças em diferentes funções, mesmo ainda não sendo claro o motivo de tais modificações.

Ao analisar a Figura 7 referente aos projetos categorizados como **Padrão de Evolução Gama**, verifica-se a presença de sete contratos 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 do cenário apresentado, uma questão pertinente a se destacar diz respeito a dualidade de estratégias quanto ao processo de evolução. 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 contrato inteligente 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, contratos 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 contrato 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 7 – Análise de similaridade dos projetos classificados como Padrão de Evolução Gama

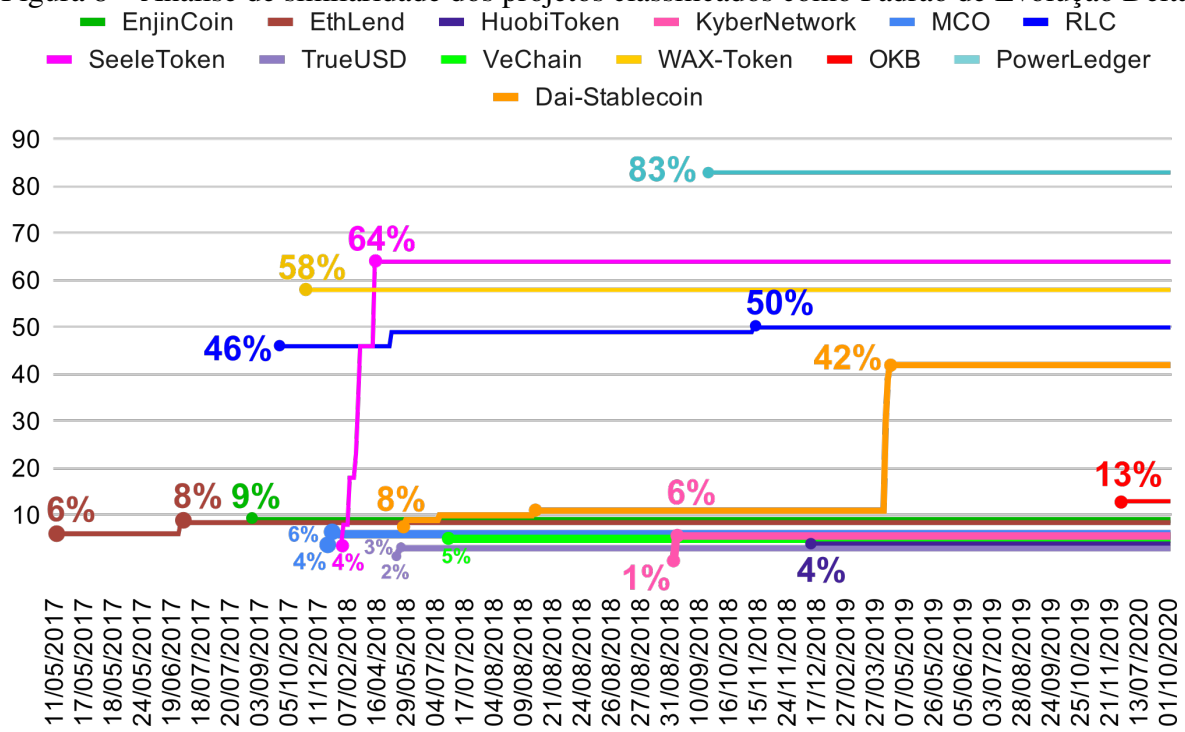


Fonte: Autoria própria.

Finalmente, o comportamento dos projetos caracterizados como **Padrão de Evolução Delta** é sintetizado através da Figura 8. Pode-se observar que os contratos inteligentes 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 contrato inteligente com maior similaridade dentre os investigados e classificados como Padrão de Evolução Delta. Nesse caso, em específico, observou-se que tal divergência entre os contratos 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 contrato inteligente 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, levando a conclusão de que estes contratos não condizem com os que foram encontrados no Etherscan. Diante de tal resultado, percebe-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 da metodologia de seleção de projetos apresentada nos procedimentos metodológicos dessa pesquisa.

Figura 8 – Análise de similaridade dos projetos classificados como Padrão de Evolução Delta



Fonte: Autoria própria.

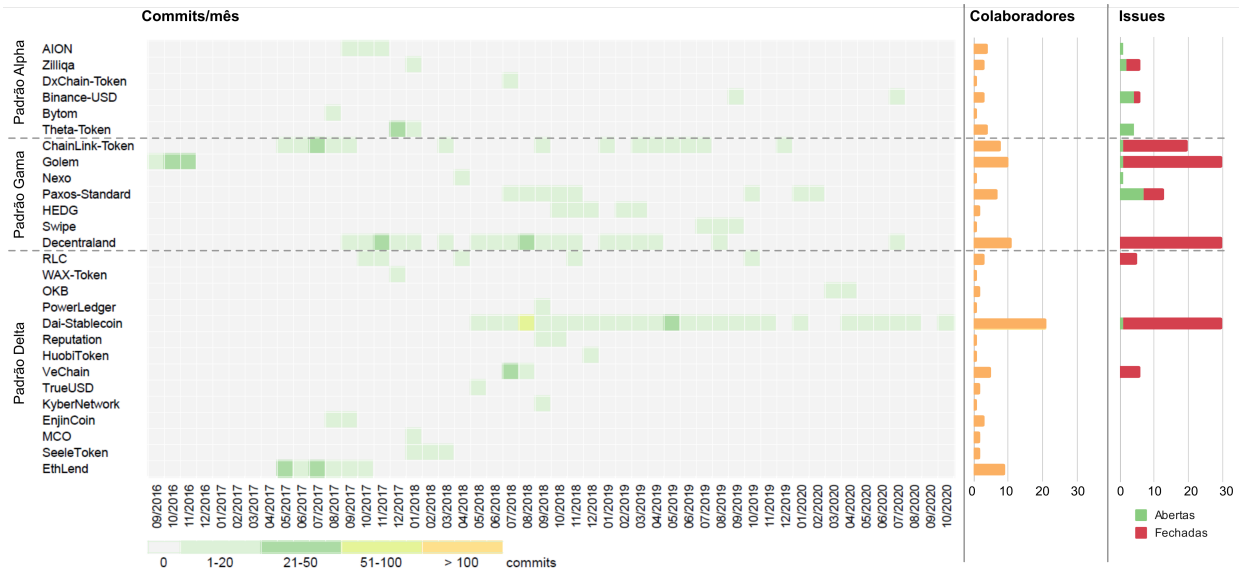
4.2.2 Análise de Aspectos Colaborativos

Prikladnicki *et al.* (2013) ressaltam que softwares são criados *por* e *para* pessoas, e entender os aspectos cooperativos inerentes ao desenvolvimento de software demonstra-se crucial para compreender como os métodos e ferramentas são utilizados e, consequentemente, melhorar criação e manutenção de sistemas. Diante de tal motivação, essa seção visa analisar os aspectos colaborativos oriundos dos dados obtidos a partir do repositório no GitHub de cada projeto investigado. Essa análise será pautada por duas questões principais: 1) a relação entre

commits e colaboradores e 2) a resolução de *issues*. Para a primeira perspectiva, portanto, tem-se a oportunidade de analisar o nível de atividade de cada projeto. Quanto à segunda questão, tem-se uma discussão sobre o engajamento da comunidade em relação ao *report* de *issues*, bem como atenção empregada na resolução das mesmas.

A Figura 9 apresenta, para cada projeto, uma visão geral sobre a quantidade de *commits*, quantidade de colaboradores e quantidade de *issues*. Destaca-se também que todos os dados granulares encontram-se disponíveis no Apêndice E e no repositório de apoio da presente pesquisa (RODRIGUES *et al.*, 2021). Conforme pode-se perceber, há uma ampla diversidade de composições quanto aos dados coletados. Ao analisar a quantidade de commits, constata-se que 50% dos projetos apresentam menos de 11 commits, enquanto 26% possuem mais de 100 commits. Os três projetos com mais *commits* foram Dai-Stablecoin (586), Golem (365) e Decentraland (333). Por sua vez, tais projetos são os que apresentaram a maior quantidade de colaboradores: 21, 10 e 11, respectivamente. Em contrapartida, pode-se verificar a presença de dois projetos (Byton e DxChain-Token) com somente 1 *commit* e 1 colaborador denotando, assim, ausência de interações. Porém, há de se destacar que nos dois projetos se constatou uma alta similaridade, de 92% e 98%, respectivamente. Revela-se, portanto, a incidência de cenários com o mero propósito de criação do repositório, mesmo com a falta de engajamento da comunidade. Dessa forma, verifica-se o uso do GitHub como mecanismo de exposição do código e não como apoio ao processo de desenvolvimento.

Figura 9 – Análise de *commits*, colaboradores e *issues* por padrão de evolução



Fonte: Autoria própria.

Quanto ao tempo de desenvolvimento e nível de atividade dos projetos, verifica-se que, por exemplo, no caso do AION, houve um intenso progresso de desenvolvimento (15 commits) entre os meses de Julho e Setembro de 2017. Já o projeto Binance-USD possui apenas dois commits, um com data de Setembro de 2019 e outro somente em Julho de 2020, onde não se verificou modificações no código-fonte do contrato. O motivo do *commit* realizado em 2020 foi para correção do arquivo `package.json` visando a redução de vulnerabilidades. O projeto Zilliqa, por sua vez, dispõe de 7 *commits* ao todo, cobrindo o período de 06/01/2018 e 09/01/2018. Por fim, o projeto DxChain-Token possui somente um *commit* no GitHub realizado no dia 15/07/2018.

Ainda na Figura 9, é possível analisar para cada projeto a relação entre quantidade de *issues* abertas e fechadas. Nesse sentido, pode-se verificar que 76% dos projetos possuem menos do que sete *issues*. Por outro lado, 20% dos projetos investigados apresentaram 30 *issues* reportadas. Os três projetos com maior quantidade de *issues* foram Dai-Stablecoin (30), Golem (30) e Decentraland (30). Analisando em específico a relação entre *issues* abertas e fechadas para projetos com mais de uma issue, verifica-se que em 70% dos casos a quantidade de *issues* fechadas se demonstrou superior a quantidade de *issues* abertas. Todavia, em particular, o Paxos-Standard (13) apresentou uma situação com mais *issues* abertas (7) que fechadas (6). Contudo, é válido destacar que as 6 *issues* em aberto eram oriundas de um bot automático para atualizações de dependências do projeto. Já os projetos AION (1), Theta-Token (4) e Nexo (1) não apresentaram nenhuma *issue* fechada. Ao analisar especificamente esses projetos, verificou-se a incidência de *issues* relacionadas a remoção de dependências (AION) obsoletas e configuração de arquivos `.yaml` (Nexo), bem como, no caso do Theta-Token, havia uma dúvida sobre o *software development kit* utilizado pela carteira web, correções textuais, solicitação para adicionar fonte do contrato e ABI ao Etherscan e comentário no código do contrato inteligente a respeito da criação de tokens.

5 AMEAÇAS À VALIDADE

Como todos estudos empíricos e experimentais em engenharia de software, este apresenta algumas ameaças à sua validade. Neste capítulo, tais ameaças são discutidas de acordo com a classificação seminal de Wohlin *et al.* (2012): conclusão, construção, internas e externas. Para cada ameaça, são discutidas as ações tomadas para sua minimização.

Ameaças de conclusão se referem a cenários ou situações que afetem a habilidade do pesquisador em interpretar corretamente os resultados de um experimento, estudo ou intervenção. No contexto deste estudo, tais ameaças se manifestam pela falta de testes estatísticos e de hipótese para validar as observações. Para mitigar tal ameaça, todas as observações aqui realizadas e reportadas foram discutidas e validadas a partir de trabalhos relacionados e trabalhos similares do estado da arte.

Ameaças de construção estão relacionadas a problemas de generalização dos resultados do estudo para os conceitos e teorias nas quais o estudo se baseia. No contexto deste estudo, tal ameaça se manifesta a partir de um mal entendimento dos fundamentos teóricos, levando a um mal design da metodologia do estudo. Para minimizar tal ameaça, todas as etapas de *design* do estudo foram extensamente discutidas entre os autores dessa pesquisa, onde as decisões foram devidamente revisadas e ajustadas para atender e se aproximar dos estudos empíricos e experimentais do estado da arte.

Ameaças internas dizem respeito a fatores que possam influenciar a causalidade de uma certa observação sem o conhecimento prévio do pesquisador, de forma a enviesar os resultados. Este trabalho não busca estabelecer nenhuma relação de causalidade entre os diferentes aspectos estudados, onde somente são apresentados os dados tais quais os mesmos foram coletados e processados. Assim, tais ameaças não se aplicam.

Ameaças externas se referem a motivos nos quais as observações do estudo não possam ser generalizadas para uma maior população. No contexto deste trabalho, somente contratos inteligentes da rede Ethereum foram estudados. Assim, tais observações não podem ser generalizadas para contratos inteligentes desenvolvidos em outras linguagens e outras plataformas.

Além disso, só foram estudados os históricos de evolução para contratos inteligentes cujo código-fonte está disponível no GitHub. É possível que as observações aqui realizadas não sejam as mesmas para contratos inteligentes cujo desenvolvimento e evolução ocorrem em outras plataformas. Porém, como o GitHub é a maior plataforma para desenvolvimento de

software colaborativo, acredita-se que tal ameaça é pequena. Ademais, é importante destacar a possibilidade dos repositórios no GitHub estarem defasados propositalmente pelas organizações mantenedoras dos contratos inteligentes analisados ou, inclusive, estar definida com outro nome no GitHub. Para mitigar tal risco e se certificar da corretude dos repositórios selecionados, a presente pesquisa estabeleceu um conjunto de quatro critérios (conforme relatado nos procedimentos metodológicos).

É possível que a quantidade de contratos inteligentes estudados neste trabalho não seja suficiente para generalizar as observações para todos os contratos inteligentes disponíveis na rede Ethereum. Para minimizar essa ameaça, foram escolhidos os contratos inteligentes de projetos com maior valor de mercado que tinham seus repositórios no GitHub disponíveis. Apesar de reconhecer que trata-se um número reduzido de contratos inteligentes, acredita-se que eles conseguem generalizar pelo menos a parcela de contratos inteligentes desta categoria tendo em vista a reincidência de determinados comportamentos entre os padrões de evolução analisados.

6 CONSIDERAÇÕES FINAIS

A construção de soluções baseadas em *blockchain* tem impactado diferentes segmentos da sociedade. Uma parcela considerável desse impacto advém da disseminação de contratos inteligentes, os quais possibilitaram fortalecer o desenvolvimento de aplicações descentralizadas e, subsequentemente, expandir a adoção de *blockchain* para além do domínio das criptomoedas. Todavia, o processo de desenvolvimento relacionado a contratos inteligentes lida com características diferentes de outros projetos tradicionais de Engenharia de Software. Além da característica da imutabilidade de dados, a qual impacta frontalmente na manutenção do software, a implementação de contratos inteligentes também deve lidar com diferentes demandas intrínsecas ao domínio em questão, como a garantia da simplicidade, a incidência de custos operacionais em criptomoedas e a transparência quanto o acesso ao código-fonte do contrato inteligente. Nesse sentido, pode-se destacar a relevância em compreender as particularidades envolvidas na evolução de software de contratos inteligentes, haja vista a oportunidade em aprimorar o processo de desenvolvimento em decorrência da natureza inalterável da tecnologia *blockchain* e, assim, revelar informações pertinentes, relacionamentos ou tendências sobre características evolutivas específicas.

Sob a forma de uma pesquisa exploratória-descritiva baseada em Mineração em Repositório de Software, este estudo teve como objetivo investigar o processo de evolução de software de 27 contratos inteligentes implantados na plataforma Ethereum e, consequentemente, disponíveis no *block explorer* Etherscan. Em prol da compreensão histórico-evolutiva de cada contrato à luz do fenômeno *open source*, optou-se por considerar apenas projetos que também tivessem os repositórios abertos no GitHub. A partir de uma análise quali-quantitativa, este trabalho contribuiu em três principais perspectivas: 1) caracterizou padrões de evolução de software quanto a similaridade e comportamento dos projetos desenvolvidos de forma *open source* no GitHub em contraste à versão do contrato inteligente disponível no Etherscan, 2) analisou elementos colaborativos quanto ao desenvolvimento *open source* de contratos inteligentes, como relação entre *commits* e colaboradores, bem como o engajamento da comunidade em relação ao *report* e resolução de *issues* e 3) relatou o uso de um método experimental baseado em comparação de *strings* para análise de similaridade entre versões de contratos inteligentes.

Como principais achados práticos oriundos desta pesquisa, destaca-se que, a partir dos padrões de evolução levantados, 14 projetos foram enquadrados na categoria cujo contrato inteligente disponível no GitHub apresenta *baixa similaridade* em relação a versão do Etherscan

e em nenhum momento atinge uma *alta similaridade*. Por sua vez, 6 projetos foram categorizados em um padrão no qual o código-fonte dos contratos presentes no GitHub apresenta *alta similaridade* ao encontrado no Etherscan desde o primeiro *commit* e não muda até o último *commit* analisado. Verificou-se também a presença de 7 projetos que o código-fonte dos contratos 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. Por fim, nenhum projeto investigado foi classificado como pertencente ao padrão de evolução que iniciam com *alta similaridade*, porém, em algum momento posterior, apresentam *baixa similaridade*. À luz de tais resultados, constatou-se também que, dos 27 projetos analisados, 13 deles apresentaram acima de 90% de similaridade em algum momento da evolução do software. Nesse sentido, salientou-se algumas práticas recorrentes quanto ao desenvolvimento de contratos inteligentes como, por exemplo, a decomposição em arquivos diferentes. Quanto à análise de aspectos colaborativos, foi possível evidenciar comportamentos distintos quanto aos projetos identificados: enquanto existem projetos com considerável nível de atividade (em termos de *commits*, colaboradores e *issues*) no respectivo repositório no GitHub, existem iniciativas que demonstram aderir ao GitHub apenas como mecanismo de exposição do código-fonte, tendo em vista a falta de mobilização constatada.

Em termos de trabalhos futuros, pretende-se 1) expandir a quantidade de contratos inteligentes analisados buscando ampliar o entendimento sobre a evolução *open source* de contratos inteligentes, 2) disponibilizar para a comunidade um *dataset* a partir de tal expansão e 3) conduzir uma análise qualitativa adicional sobre elementos sociotécnicos associados aos repositórios como, por exemplo, um aprofundamento sobre as mudanças e as divergências de similaridades entre as versões dos contratos inteligentes.

REFERÊNCIAS

AITZHAN, N. Z.; SVETINOVIC, D. Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. **IEEE Transactions on Dependable and Secure Computing**, IEEE, v. 15, n. 5, p. 840–852, 2018.

AJIENKA, N.; VANGORP, P.; CAPILUPPI, A. An empirical analysis of source code metrics and smart contract resource consumption. **Journal of Software: Evolution and Process**, Wiley Online Library, v. 32, n. 10, p. e2267, 2020.

ALVES, P. H.; LAIGNER, R.; NASSER, R.; ROBICHEZ, G.; LOPES, H.; KALINOWSKI, M. Desmistificando blockchain: Conceitos e aplicaçoes. **es). Computação e Sociedade. Rio de Janeiro: Sociedade Brasileira de Computação**, p. 1–24, 2018.

ANTONOPOULOS, A. M.; WOOD, G. **Mastering ethereum: building smart contracts and dapps**. [S.l.]: O'reilly Media, 2018.

ARIAS, F. J. C. **Fuzzy String Matching in Python**. 2019. Acessado em: 04/05/2021. Disponível em: <https://www.datacamp.com/community/tutorials/fuzzy-string-python>.

BAMBARA, J. J.; ALLEN, P. R. **Blockchain - A Practical Guide to Developing Business, Law, and Technology Solutions**. [S.l.]: McGraw-Hill Education, 2018.

BERTRAM, D.; VOIDA, A.; GREENBERG, S.; WALKER, R. Communication, collaboration, and bugs: The social nature of issue tracking in software engineering. In: . [S.l.: s.n.], 2009.

BLOCKHAINHUB. **Decentralized Applications – dApps**. 2019. Acessado em: 31/05/19. Disponível em: <https://blockchainhub.net/decentralized-applications-dapps/>.

BOSU, A.; IQBAL, A.; SHAHRIYAR, R.; CHAKRABORTY, P. Understanding the motivations, challenges and needs of blockchain software developers: A survey. **Empirical Software Engineering**, Springer, v. 24, n. 4, p. 2636–2673, 2019.

BROOKS, F.; KUGLER, H. **No silver bullet**. [S.l.]: April, 1987.

BUTERIN, V. *et al.* Ethereum white paper: a next generation smart contract & decentralized application platform. **First version**, 2014.

CACHIN, C.; VUKOLIĆ, M. Blockchain consensus protocols in the wild. **arXiv preprint arXiv:1707.01873**, 2017.

CHAKRABORTY, P.; SHAHRIYAR, R.; IQBAL, A.; BOSU, A. 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**. [S.l.: s.n.], 2018. p. 1–10.

CHATURVEDI, K. K.; SING, V.; SINGH, P. Tools in mining software repositories. In: IEEE. **2013 13th International Conference on Computational Science and Its Applications**. [S.l.], 2013. p. 89–98.

CHATZIASIMIDIS, F.; STAMELOS, I. Data collection and analysis of github repositories and users. In: IEEE. **2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)**. [S.l.], 2015. p. 1–6.

CHAU, J. **A Code Reputation System Using AI and Blockchain Technology**. Tese (Doutorado) — San Jose State University, 2020.

CHEN, J.; XIA, X.; LO, D.; GRUNDY, J.; YANG, X. Maintaining smart contracts on ethereum: Issues, techniques, and future challenges. **arXiv preprint arXiv:2007.00286**, 2020.

CHEN, Y.-C.; CHOU, Y.-P.; CHOU, Y.-C. An image authentication scheme using merkle tree mechanisms. **Future Internet**, Multidisciplinary Digital Publishing Institute, v. 11, n. 7, p. 149, 2019.

COOK, S.; JI, H.; HARRISON, R. Software evolution and software evolvability. **University of Reading, UK**, Citeseer, p. 1–12, 2000.

DABBISH, L.; STUART, C.; TSAY, J.; HERBSLEB, J. Social coding in github: transparency and collaboration in an open software repository. In: ACM. **Proceedings of the ACM 2012 conference on computer supported cooperative work**. [S.l.], 2012. p. 1277–1286.

DANNEN, C. **Introducing Ethereum and solidity**. [S.l.]: Springer, 2017. v. 318.

DELMOLINO, K.; ARNETT, M.; KOSBA, A.; MILLER, A.; SHI, E. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In: SPRINGER. **International Conference on Financial Cryptography and Data Security**. [S.l.], 2016. p. 79–94.

DESTEFANIS, G.; MARCHESI, M.; ORTU, M.; TONELLI, R.; BRACCIALI, A.; HIERONS, R. Smart contracts vulnerabilities: a call for blockchain software engineering? In: IEEE. **2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)**. [S.l.], 2018. p. 19–25.

DIEHL, S. **Software visualization: visualizing the structure, behaviour, and evolution of software**. [S.l.]: Springer Science & Business Media, 2007.

D'AMBROS, M.; GALL, H.; LANZA, M.; PINZGER, M. Analysing software repositories to understand software evolution. In: **Software evolution**. [S.l.]: Springer, 2008. p. 37–67.

EFANOV, D.; ROSCHIN, P. The all-pervasiveness of the blockchain technology. **Procedia Computer Science**, Elsevier, v. 123, p. 116–121, 2018.

ETHEREUM, W. **White Paper**. 2019. Acessado em: 31/05/19. Disponível em: <https://github.com/ethereum/wiki/wiki/White-Paper#applications>.

ETHERSCAN. **What is Etherscan?** 2017. Acessado em: 04/06/19. Disponível em: <https://etherscancom.freshdesk.com/support/solutions/articles/35000022140-what-is-etherscan->.

FARIAS, M. A. de F.; NOVAIS, R.; JÚNIOR, M. C.; CARVALHO, L. P. da S.; MENDONÇA, M.; SPÍNOLA, R. O. A systematic mapping study on mining software repositories. In: **Proceedings of the 31st Annual ACM Symposium on Applied Computing**. [S.l.: s.n.], 2016. p. 1472–1479.

FELLER, J.; FITZGERALD, B. *et al.* **Understanding open source software development**. [S.l.]: Addison-Wesley London, 2002.

GITHUB. **Welcome home, developers**. 2021. Acessado em: 22/04/2021. Disponível em: <https://github.com/>.

GREVE, F.; SAMPAIO, L.; ABIJAUDE, J.; COUTINHO, A.; VALCY, I.; QUEIROZ, S. Blockchain e a revolução do consenso sob demanda. **Livro de Minicursos do SBRC**, v. 1, p. 1–52, 2018.

GRISHCHENKO, I.; MAFFEI, M.; SCHNEIDEWIND, C. Foundations and tools for the static analysis of ethereum smart contracts. In: SPRINGER. **International Conference on Computer Aided Verification**. [S.l.], 2018. p. 51–78.

HASSAN, A. E. The road ahead for mining software repositories. In: IEEE. **2008 Frontiers of Software Maintenance**. [S.l.], 2008. p. 48–57.

HILDENBRANDT, E.; SAXENA, M.; RODRIGUES, N.; ZHU, X.; DAIAN, P.; GUTH, D.; MOORE, B.; PARK, D.; ZHANG, Y.; STEFANESCU, A. *et al.* Kevm: A complete formal semantics of the ethereum virtual machine. In: IEEE. **2018 IEEE 31st Computer Security Foundations Symposium (CSF)**. [S.l.], 2018. p. 204–217.

HOLLANDER, L. **The Ethereum Virtual Machine — How does it work?** 2019. Acessado em: 15/06/19. Disponível em: <https://medium.com/mycrypto/the-ethereum-virtual-machine-how-does-it-work-9abac2b7c9e>.

INVEST, S. **Valor de Mercado**. 2020. Acessado em: 04/05/2021. Disponível em: <https://statusinvest.com.br/termos/v/valor-de-mercado>.

JANÁK, J. **Issue tracking systems**. Tese (Doutorado) — Masarykova univerzita, Fakulta informatiky, 2009.

JONGE, M. D. Pretty-printing for software reengineering. In: IEEE. **International Conference on Software Maintenance, 2002. Proceedings.** [S.l.], 2002. p. 550–559.

KAGDI, H.; COLLARD, M. L.; MALETIC, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. **Journal of software maintenance and evolution: Research and practice**, Wiley Online Library, v. 19, n. 2, p. 77–131, 2007.

KAGDI, H. H. **Mining software repositories to support software evolution**. Tese (Doutorado) — Kent State University, 2008.

KALLIAMVAKOU, E.; GOUSIOS, G.; BLINCOE, K.; SINGER, L.; GERMAN, D. M.; DAMIAN, D. The promises and perils of mining github. In: ACM. **Proceedings of the 11th working conference on mining software repositories**. [S.l.], 2014. p. 92–101.

KHATWANI, S. **What Is A Block/Blockchain Explorer?** 2018. Acessado em: 04/06/19. Disponível em: <https://coinsutra.com/blockchain-explorer/>.

KOGUT, B.; METIU, A. Open-source software development and distributed innovation. **Oxford review of economic policy**, Oxford University Press, v. 17, n. 2, p. 248–264, 2001.

KOLB, J.; ABDELBAKY, M.; KATZ, R. H.; CULLER, D. E. Core concepts, challenges, and future directions in blockchain: a centralized tutorial. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 1, p. 1–39, 2020.

LAURENCE, T. **Blockchain for dummies**. [S.l.]: John Wiley & Sons, 2019.

LEHMAN, M. M. On understanding laws, evolution, and conservation in the large-program life cycle. **Journal of Systems and Software**, Elsevier, v. 1, p. 213–221, 1979.

LEHMAN, M. M. Laws of software evolution revisited. In: SPRINGER. **European Workshop on Software Process Technology**. [S.l.], 1996. p. 108–124.

LEHMAN, M. M.; RAMIL, J. F. Software evolution—background, theory, practice. **Information Processing Letters**, Elsevier, v. 88, n. 1-2, p. 33–44, 2003.

LIMA, A.; ROSSI, L.; MUSOLESI, M. Coding together at scale: Github as a collaborative social network. In: **Proceedings of the International AAAI Conference on Web and Social Media**. [S.l.: s.n.], 2014. v. 8, n. 1.

MENS, T.; KLEIN, J. Evolving software-introduction to the special theme. **ERCIM News**, ERCIM, v. 88, p. 8–9, 2012.

MENS, T.; WERMELINGER, M.; DUCASSE, S.; DEMEYER, S.; HIRSCHFELD, R.; JAZAYERI, M. Challenges in software evolution. In: IEEE. **Eighth International Workshop on Principles of Software Evolution (IWPSE'05)**. [S.l.], 2005. p. 13–22.

METCALFE, W. Ethereum, smart contracts, dapps. In: **Blockchain and Crypt Currency**. [S.l.]: Springer, Singapore, 2020. p. 77–93.

NAKAMOTO, S. **Bitcoin: A peer-to-peer electronic cash system**. 2009. Disponível em: <http://www.bitcoin.org/bitcoin.pdf>. Acesso em: 28 set. 2019.

NAKAMOTO, S. *et al.* Bitcoin: A peer-to-peer electronic cash system. Working Paper, 2008.

NARAYANAN, A.; BONNEAU, J.; FELTEN, E.; MILLER, A.; GOLDFEDER, S. **Bitcoin and cryptocurrency technologies:** a comprehensive introduction. New Jersey: Princeton University Press, 2016. 328 p.

OKWU, P.; ONYEJE, I. Software evolution: past, present and future. **American Journal of Engineering Research (AJER)**, Citeseer, v. 3, n. 05, p. 21–28, 2014.

OLIVA, G. A.; HASSAN, A. E.; JIANG, Z. M. J. An exploratory study of smart contracts in the ethereum blockchain platform. **Empirical Software Engineering**, Springer, p. 1–41, 2020.

PINNA, A.; IBBA, S.; BARALLA, G.; TONELLI, R.; MARCHESI, M. A massive analysis of ethereum smart contracts empirical study and code metrics. **IEEE Access**, IEEE, v. 7, p. 78194–78213, 2019.

PONCIN, W.; SEREBRENIK, A.; BRAND, M. V. D. Process mining software repositories. In: IEEE. **2011 15th European Conference on Software Maintenance and Reengineering**. [S.l.], 2011. p. 5–14.

PORRU, S.; PINNA, A.; MARCHESI, M.; TONELLI, R. Blockchain-oriented software engineering: challenges and new directions. In: IEEE. **2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)**. [S.l.], 2017. p. 169–171.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. [S.l.]: Palgrave Macmillan, 2005.

PRIKLADNICKI, R.; DITTRICH, Y.; SHARP, H.; SOUZA, C. D.; CATALDO, M.; HODA, R. Cooperative and human aspects of software engineering: Chase 2013. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 38, n. 5, p. 34–37, 2013.

RAGNEDDA, M.; DESTEFANIS, G. **Blockchain and web 3.0: social, economic, and technological challenges**. [S.l.]: Routledge, 2019.

RAJLICH, V. Software evolution and maintenance. In: **Future of Software Engineering Proceedings**. [S.l.: s.n.], 2014. p. 133–144.

RAVAL, S. **Decentralized applications: harnessing Bitcoin's blockchain technology**. [S.l.]: "O'Reilly Media, Inc.", 2016.

REIBEL, P.; YOUSAF, H.; MEIKLEJOHN, S. Why is a ravencoin like a tokendesk? an exploration of code diversity in the cryptocurrency landscape. **arXiv preprint arXiv:1810.08420**, 2018.

RISIUS, M.; SPOHRER, K. A blockchain research framework. **Business & Information Systems Engineering**, Springer, v. 59, n. 6, p. 385–409, 2017.

RODRIGUES, A.; ARAUJO, A. A.; PAIXAO, M. H. E. **Repositório de apoio**. 2021. Disponível em: <https://github.com/gesid/smart-contracts-software-evolution>.

SIDDIQUI, T.; AHMAD, A. Data mining tools and techniques for mining software repositories: A systematic review. **Big Data Analytics**, Springer, p. 717–726, 2018.

SILLABER, C.; WALTL, B.; TREIBLMAIER, H.; GALLERSDÖRFER, U.; FELDERER, M. Laying the foundation for smart contract development: an integrated engineering process model. **Information Systems and e-Business Management**, Springer, p. 1–20, 2020.

SOKOL, F. Z.; ANICHE, M. F.; GEROSA, M. A. Metricminer: Supporting researchers in mining software repositories. In: IEEE. **2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)**. [S.l.], 2013. p. 142–146.

STARK, J. **Making sense of blockchain smart contracts**. 2016. Acessado em: 01/06/19. Disponível em: <https://www.coindesk.com/making-sense-smart-contracts>.

SZABO, N. Smart contracts: building blocks for digital markets. **EXTROPY: The Journal of Transhumanist Thought,(16)**, v. 18, 1996.

TAPSCOTT, D.; TAPSCOTT, A. The impact of the blockchain goes beyond financial services. **Harvard Business Review**, v. 10, n. 7, 2016.

THUNG, F.; BISSYANDE, T. F.; LO, D.; JIANG, L. Network structure of social coding in github. In: IEEE. **2013 17th European Conference on Software Maintenance and Reengineering**. [S.l.], 2013. p. 323–326.

TIKHOMIROV, S.; VOSKRESENSKAYA, E.; IVANITSKIY, I.; TAKHAVIEV, R.; MARCHENKO, E.; ALEXANDROV, Y. Smartcheck: Static analysis of ethereum smart contracts. In: **Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain**. [S.l.: s.n.], 2018. p. 9–16.

TONELLI, R.; DESTEFANIS, G.; MARCHESI, M.; ORTU, M. Smart contracts software metrics: A first study. arxiv 2018. **arXiv preprint arXiv:1802.01517**, 2018.

TRIPATHY, P.; NAIK, K. **Software evolution and maintenance: a practitioner's approach**. [S.l.]: John Wiley & Sons, 2014.

TSAY, J.; DABBISH, L.; HERBSLEB, J. Influence of social and technical factors for evaluating contribution in github. In: **Proceedings of the 36th international conference on Software engineering**. [S.l.: s.n.], 2014. p. 356–366.

UKESSAYS. **Software Evolution Process**. 2012. Acessado em: 23/04/2021. Disponível em: <https://www.ukessays.com/essays/information-systems/software-evolution-process.php?vref=1>.

VANDECRUYS, O.; MARTENS, D.; BAESENS, B.; MUES, C.; BACKER, M. D.; HAESEN, R. Mining software repositories for comprehensible software fault prediction models. **Journal of Systems and software**, Elsevier, v. 81, n. 5, p. 823–839, 2008.

VOINEA, S.-L. **Software evolution visualization**. [S.l.]: Citeseer, 2007.

WOHLIN, C.; RUNESON, P.; HST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLN, A. **Experimentation in Software Engineering**. [S.l.]: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434, 9783642290435.

WOOD, G. *et al.* Ethereum: A secure decentralised generalised transaction ledger. **Ethereum project yellow paper**, v. 151, n. 2014, p. 1–32, 2014.

WÜST, K.; GERVAIS, A. Do you need a blockchain? In: IEEE. **2018 Crypto Valley Conference on Blockchain Technology (CVCBT)**. [S.l.], 2018. p. 45–54.

YERMACK, D. Corporate governance and blockchains. **Review of Finance**, Oxford University Press, v. 21, n. 1, p. 7–31, 2017.

YU, Y.; YIN, G.; WANG, H.; WANG, T. Exploring the patterns of social behavior in github. In: **Proceedings of the 1st international workshop on crowd-based software development methods and technologies**. [S.l.: s.n.], 2014. p. 31–36.

YUJIAN, L.; BO, L. A normalized levenshtein distance metric. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 29, n. 6, p. 1091–1095, 2007.

ZAIDMAN, A.; ROMPAEY, B. V.; DEMEYER, S.; DEURSEN, A. V. Mining software repositories to study co-evolution of production & test code. In: IEEE. **2008 1st international conference on software testing, verification, and validation**. [S.l.], 2008. p. 220–229.

ZOU, W.; LO, D.; KOCHHAR, P. S.; LE, X.-B. D.; XIA, X.; FENG, Y.; CHEN, Z.; XU, B. Smart contract development: Challenges and opportunities. **IEEE Transactions on Software Engineering**, IEEE, 2019.

ZUMKELLER, S. **Using Smart Contracts for Digital Services: A Feasibility Study based on Service Level Agreements**. 2018.

APÊNDICE A – *SCRIPT* UTILIZADO PARA OBTENÇÃO DE REPOSITÓRIOS E *COMMITS*

```
import subprocess
import glob
import os
from fuzzywuzzy import fuzz
import git

#Lista com exemplos de links para repositórios
repos = [
"https://github.com/smartcontractkit/LinkToken.git",
"https://github.com/HuobiRussia/HuobiTokenRussia.git",
"https://github.com/vechain/thor-builtins.git",
]
#Lista com exemplos nomes dos repositórios
tokens = [
"ChainLink-Token",
"HuobiToken",
"VeChain",
]

#Criação de diretórios para projetos
for token in tokens:
        path = "github_repos/"
        try:
            os.mkdir(path+token)
        except OSError:
            print ("Creation of the directory %s failed" % path)

#Clonagem de repositórios para respectivos diretórios
count = 0
for repo, token in zip(repos, tokens):
        subprocess.Popen(["git", "-C", "github_repos/"+tokens[count],"clone", repo, token], stdout=subprocess.PIPE).communicate()
        count= count+1

#Obtenção de commits para cada repositório
for token in tokens:
    g = git.Git("github_repos/"+token+"/"+token)
    loginfo = g.log('--first-parent','--pretty="%H"')
    commits_file = open("github_commits/" + token + "_commits.txt", "w")
    for line in loginfo:
        commits_file.write(line.strip('"'))
    commits_file.close()


```

Fonte: Autoria própria.

APÊNDICE B – *SCRIPT* UTILIZADO PARA OBTENÇÃO DE DADOS SOBRE COLABORADORES, *COMMITS* E *ISSUES*.

```
import subprocess
import glob
import os
from fuzzywuzzy import fuzz
import git
import requests
import json
import pandas as pd

 #Lista com exemplos nomes dos repositórios
tokens = [
"ChainLink-Token",
"HuobiToken",
"VeChain",
]
#Lista com exemplos caminhos para repositórios
repos = [
"/smartcontractkit/LinkToken",
"/HuobiRussia/HuobiTokenRussia",
"/vechain/thor-builtins",

]

#Função para obter a lista de contribuidores de cada repositório e quantidade de commits de cada contribuidor
def get_contributors_insights(contributors, token):
    data = subprocess.Popen(["git", "-C", "github_repos/" + token + "/" + token, "shortlog",  "-s", "-n"], stdout=subprocess.PIPE)
    for line in data.stdout:
        contributors.write(','+line.decode().replace('\t', ', '))

#Função para obter a quantidade total de contribuidores
def get_contributors_amount(repo):
    url='https://api.github.com/repos'
    print(url + repo + "/collaborators")
    response = requests.get(url + repo + "/issues?state=all&direction=asc").json()

    data = []
    for x in response:
        data.append(str(x["user"]["login"]))

    count = 0
    contributors = []
    for line in data:
        if line in contributors:
            continue
        else:
            contributors.append(line)
            count = count + 1

    return count

#Função para obter a quantidade total de commits
def get_commits_insights(token):
    data = subprocess.Popen(["git", "-C", "github_repos/" + token + "/" + token, "rev-list",  "--all", "--count"], stdout=subprocess.PIPE, universal_newlines=True).communicate()
    return data[0]

#Função para obter insights sobre issues
def get_issues_insights(repo, insights):
    url='https://api.github.com/repos'
    print(url + repo + "/issues?state=all")
    response = requests.get(url + repo + "/issues?state=all&direction=asc").json()

    for x in response:
        insights.write(','+str(x['number']) + ',' + str(x['title']) + ',' + str(x["user"]["login"]) + ',' +
        str(x['state']) + ',' + str(x['created_at']) + ',' + str(x['updated_at']) + ',' +
        str(x['closed_at'])+'\n')

#Função para obter insights sobre issues
def get_timestamp(token, project_timestamp):

    commits = open("github_commits/" + token  + "_commits.txt")
    for commit in commits:
        commit = commit.replace("\n","")
        data = str(subprocess.Popen(["git", "-C", "github_repos/" + token + "/" + token, "show",  "-s", "--format=" + "%ci" , commit], stdout=subprocess.PIPE).communicate())
        project_timestamp.write("," + commit + "," + data[3:-16] + "\n")
    commits.close()

    return



```

Fonte: Autoria própria.

APÊNDICE C – EXPRESSÃO REGULAR UTILIZADA PARA REMOÇÃO DOS ESPAÇOS EM BRANCO E DOS COMENTÁRIOS NOS CÓDIGOS-FONTE

```
1 patternII = r"(\".*?\"|\'.*?\')|(/\*.*?\*/|\-.*?)|(//.*?[\s\S]$|\s.*?)"
```

Fonte: Autoria própria.

APÊNDICE D – *SCRIPT* UTILIZADO PARA REALIZAÇÃO DE COMBINAÇÕES E COMPARAÇÕES ENTRE ARQUIVOS .SOL

```python
import subprocess
import glob
import fnmatch
import os
from fuzzywuzzy import fuzz
import re
import itertools

#Função para realizar combinações entre códigos solidity do Github e em seguida comparações com o código do Etherscan
def combine_and_compare(token, commit, pattern, patternII, etherscan_code):
    solidity_files_in_folder = []

    for solidity_file in glob.glob("github_contracts_without_comments/" + token + "/" + commit + "/" + "**/*.sol", recursive=True):
        solidity_files_in_folder.append(solidity_file)

    n = len(solidity_files_in_folder)
    #
    count = 1
    while(count <=n):
        result = list(itertools.combinations(solidity_files_in_folder, count))
        filE = open("results_combinations/" + token + "/" + commit + "/" + "combinations_of_" + str(count) + '.csv', "w")
        filE.write("similarity,combination\n")
        for item in result:
            solidity_code = join_solidity_codes(item)
            solidity_code = remove_blank_spaces(solidity_code,patternII)

            combination = []
            for line in item:
                combination.append(line.split('/')[-1])

            # mais opções para calcular similaridades em: https://www.datacamp.com/community/tutorials/fuzzy-string-python
            similarity = fuzz.token_sort_ratio(etherscan_code, solidity_code)
            filE.write(str(similarity) + "," + str(combination) + "\n")

        filE.close()

        count += 1

    return
```

Fonte: Autoria própria.

APÊNDICE E – DADOS GRANULARES REFERENTES AO NÚMERO TOTAL DE *COMMITS*, COLABORADORES E *ISSUES*

Padrão de Evolução	Projetos	Commits	Colaboradores	Issues
Alpha	Binance-USD	7	3	6
	DxChain-Token	1	1	0
	AION	51	4	1
	Zilliqa	8	3	6
	Theta-Token	36	4	4
	Bytom	1	1	0
Gama	Golem	365	10	30
	HEDG	25	2	0
	Swipe	5	1	0
	Nexo	3	1	1
	Paxos-Standard	40	7	13
	ChainLink-Token	175	8	20
	Decentraland	333	11	30
Delta	PowerLedger	7	1	0
	SeeleToken	38	2	0
	WAX-Token	10	1	0
	RLC	39	3	5
	Dai-Stablecoin	586	21	30
	OKB	2	2	0
	EnjinCoin	5	3	0
	EthLend	173	9	0
	Reputation	4	1	0
	KyberNetwork	2	1	0
	MCO	8	2	0
	VeChain	25	5	6
	HuobiToken	2	1	0
	TrueUSD	4	2	0

Fonte: Autoria própria.

APÊNDICE F – DADOS GRANULARES REFERENTES ÀS *ISSUES* ABERTAS E FECHADAS

Padrão de Evolução	Projetos	Total de Issues	Abertas	Fechadas
Alpha	Binance-USD	6	4	2
	DxChain-Token	0	0	0
	AION	1	1	0
	Zilliqa	6	2	4
	Theta-Token	4	4	0
	Bytom	0	0	0
Gama	Golem	30	1	29
	HEDG	0	0	0
	Swipe	0	0	0
	Nexo	1	1	0
	Paxos-Standard	13	7	6
	ChainLink-Token	20	1	19
	Decentraland	30	0	30
Delta	PowerLedger	0	0	0
	SeeleToken	0	0	0
	WAX-Token	0	0	0
	RLC	5	0	5
	Dai-Stablecoin	30	1	29
	OKB	0	0	0
	EnjinCoin	0	0	0
	EthLend	0	0	0
	Reputation	0	0	0
	KyberNetwork	0	0	0
	MCO	0	0	0
	VeChain	6	0	6
	HuobiToken	0	0	0
	TrueUSD	0	0	0

Fonte: Autoria própria.