Informatics

Comparison of Smart Contract Platforms for Decentralized Applications Development

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Ammar Voloder, BSc
Matrikelnummer 01527915

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn Monika di Angelo

Wien, 27. Juli 2023

Ammar Voloder   Monika di Angelo

Informatics

Comparison of Smart Contract Platforms for Decentralized Applications Development

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Ammar Voloder, BSc
Registration Number 01527915

to the Faculty of Informatics

at the TU Wien

Advisor: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn Monika di Angelo

Vienna, 27th July, 2023

Ammar Voloder | Monika di Angelo

Erklärung zur Verfassung der Arbeit

Ammar Voloder, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 27. Juli 2023

Ammar Voloder

Kurzfassung

Die Blockchain-Technologie hat sich seit der Einführung von Bitcoin kontinuierlich weiterentwickelt. Mit dem Aufkommen von Ethereum im Jahr 2014 legten Programme mit einem vordefinierten Regelsatz, sogenannte 'Smart Contracts', den Grundstein für die Blockchain-Technologie über Kryptowährungen hinaus. Anschließend wurde die Blockchain-Technologie in verschiedenen Sektoren eingesetzt, darunter Lieferkettenmanagement, Immobilien, Gesundheitswesen und viele mehr, und bot zahlreiche Vorteile wie Effizienz, Transparenz und Unveränderlichkeit. In den folgenden Jahren entstanden zahlreiche weitere Smart-Contract-Plattformen, darunter Neo, EOS, Cardano und Algorand, die Alternativen zu Ethereum darstellten. Dies stellt Unternehmen und Entwickler vor die Herausforderung, die für ihre Anforderungen am besten geeignete Plattform auszuwählen. Wir erstellen einen Kriterienkatalog und vergleichen diese fünf Plattformen aus Entwicklungssicht. Dieser Katalog umfasst Schlüsselmerkmale wie die Verfügbarkeit von Dokumentationen, die Installationsfreundlichkeit, Sprachunterstützung und das Engagement der Community. Wir haben drei praktische Anwendungsfälle identifiziert: einen fungiblen Token, einen nicht-fungiblen Token und ein einfaches Supply-Chain-Management-System. Die Implementierung dieser Anwendungsfälle auf allen fünf Platformen ermöglicht es uns, die Stärken und Schwächen jeder Plattform zu erkennen und sie aus praktischer Sicht anhand zuvor definierter Kriterien zu vergleichen. Während der Implementierung wurden verschiedene Aspekte aufgezeichnet, wie Aufwand, Vertragslänge, Anzahl der Schwierigkeiten und eine Schätzung des erforderlichen Erfahrungsniveau, um einen gegebenen Anwendungsfall auf einer bestimmten Plattform zu entwickeln. Während wir die Unterschiede zwischen den Plattformen erforschen, ihre Stärken und Schwächen bewerten und die möglichen Auswirkungen für Entwickler betonen, bietet unsere Forschung wertvolle Erkenntnisse, um Entwicklern bei der Auswahl der am besten geeigneten Plattform zu helfen.

Abstract

Blockchain technology has been continually evolving since the inception of Bitcoin. With Ethereum's emergence in 2014, programs with a predefined set of rules, known as 'Smart Contracts,' established the foundation of blockchain technology beyond cryptocurrencies. Subsequently, blockchain technology began to be applied in diverse sectors, including supply chain management, real estate, healthcare, and many more, offering numerous benefits such as efficiency, transparency, and immutability. In the years that followed, a variety of other smart contract platforms, including Neo, EOS, Cardano, and Algorand, came into existence, presenting alternatives to Ethereum. This poses a challenge for companies and developers when selecting the most suitable platform for their needs.
We establish a catalogue of criteria and compare these five platforms from a development perspective. This catalogue encompasses key features such as documentation availability, ease of installation, language support, and community engagement. We identify three practical use-cases: a fungible token, a non-fungible token, and a simple supply chain management system. Implementing these use-cases across all platforms allows us to discern each platform's strengths and weaknesses and to compare them from the practical point of view using previously defined criteria. During the implementation, we record various aspects such as effort, contract length, number of difficulties, and an estimation of the experience level required to develop a given use-case on a specific platform. As we explore the distinctions between the platforms, evaluate their strengths and weaknesses, and underscore the potential implications for developers, our research provides valuable insights to aid developers in selecting the most appropriate platform.

Contents

CHAPTER 1

Introduction

Motivation

In 1982 David Chaum first proposed a protocol similar to blockchain, as we know it today, in his dissertation "Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups" [33]. Although the protocol itself has been developed and improved over the years throughout various scientific studies, the first blockchain was conceptualized in 2008 when Satoshi Nakamoto created Bitcoin. In "Bitcoin: A peer-to-peer electronic cash system" [52] Nakamoto introduced the fundamental notions of blockchain technology providing the foundation of today's smart contract platforms.

As the only purpose of Bitcoin was to serve as a digital currency, at that time, blockchain has often been wrongly identified as Bitcoin. In the following years, as the technology evolved and Vitalik Buterin developed Ethereum, it was clear that there were other use cases eligible for blockchain, and not just cryptocurrencies. Ethereum is a smart contract-based blockchain, which enables the development of different kinds of applications.

Computer scientist Nick Szabo defined the term "smart contract" in the 90s as a "set of promises, specified in digital form, including protocols within which the parties perform on these promises" [59]. Just like the blockchain did not find its adoption before Nakamoto created Bitcoin, the term "smart contract" only became identified as a general-purpose computation on a blockchain or distributed ledger with the development of Ethereum in 2015.

Ever since Ethereum was built and many applications have been developed on it, notable issues arose concerning scalability of the network, transaction fees, etc. These led to the development of many interesting smart contract platforms that should provide an alternative to Ethereum and potentially solve issues that Ethereum has.

Problem Statement and Research Questions

Ethereum, one of the most well-known platforms for smart contracts, has certainly played a critical role in advancing blockchain technology. Despite its pioneering status and popularity, however, Ethereum may not necessarily be the most suitable choice for every type of application. In the ever-evolving blockchain landscape, developers and companies must carefully consider a plethora of alternatives to determine the best fit for their specific use case. Over recent years, several of these alternative smart contract platforms have gained prominence, adding to the complexity of choice for developers or enterprises looking to create decentralized applications (dApps). Given this diversity, it becomes crucial to examine the unique offerings of each platform, their strengths and weaknesses, and the specific contexts in which they excel.

Some of these platforms are designed to focus on particular industries or applications. For instance, VeChain is purpose-built for supply chain management, aiming to enhance traceability and operational efficiency. In contrast, other platforms like Neo, Algorand, Cardano, and EOS have a broader focus. Rather than catering to a particular use case, these platforms provide a more general-purpose infrastructure for developing a wide range of dApps. Existing comparisons of these platforms often focus on technical and performance aspects without covering a wide range of platforms. This thesis aims to close this gap and provide a structured comparison of several use-case-agnostic smart contract platforms from a developer's perspective, including Ethereum, Neo, Algorand, Cardano, and EOS.

For an effective comparison of the selected platforms, each must undergo a thorough evaluation where the following questions are addressed:

- **RQ:** Which aspects of platforms are relevant for this specific group?
- **RQ:** What are the key differences of the selected platforms?
- **RQ:** What are the main strengths and weaknesses of the selected platforms?

Methodology

Systematic literature review

Initially, a systematic literature review of Ethereum, Neo, Algorand, Cardano, and EOS as smart contract platforms is necessary to understand the fundamental concepts, properties, and limitations inherent to these platforms.

Creation of a catalog of criteria for comparison (technical perspective)

To derive a catalog of criteria for comparison, a further review of existing literature, wherein the main research topic has been the comparison between these platforms, is required. It is important to note that not all platforms will be compared simultaneously. However, for the successful creation of a criteria catalog, it is sufficient that at least some of the platforms are compared.

Definition of three common use cases for decentralized applications

Once the literature review is completed, three common use cases for decentralized applications will be defined. Each use case will be implemented across all platforms, serving as a test input for the subsequent comparison.

Creation of catalog of criteria for comparison (developer's perspective)

After defining the use cases and gaining an understanding of the various platforms, a catalog of criteria from a developer's perspective will be established. This catalog will encapsulate key properties that should be taken into consideration when selecting a smart contract platform for developing a decentralized application. It is intended to provide developers with a clearer view of the ease, or potential difficulty, associated with using a particular platform (or blockchain) for a specific use case. Criteria such as ease of setup, availability of documentation, and examples of contracts, among others, will be included in the catalog. The creation of two such catalogues of criteria will provide an answer to the first research question.

Implementation of use cases on selected platforms

Defining a catalog of criteria from a developer's perspective is a critical step before beginning to implement use cases on various platforms. As these use cases are implemented, we will record our personal experiences with each platform, guided by the criteria outlined in the previously defined catalog.

Evaluation of the selected platforms based on derived criteria (developer's perspective)

Ultimately, platforms will be evaluated and compared based on the findings. The main differences will be elaborated upon, including their strengths and weaknesses. This should provide meaningful insight for developers when choosing a platform for their decentralized application. Through this discussion, the second and third research questions will be answered.

Structure

The remainder of the thesis is structured as follows: Chapter 2 presents related work in which smart contract platforms were compared. We summarize the significant properties in these comparisons, i.e., the criteria used to compare these platforms. This allows for a better understanding and enables us to derive our catalog of criteria. In Chapter 3, we define two criteria catalogues from technical and development perspectives. We describe the meaning of each criterion and its significance in the choice of the platform. Chapter 4 briefly compares selected smart contract platforms using the technical catalogue defined in the previous chapter. Chapter 5 outlines three use-cases we plan to implement across various platforms, allowing us to compare these platforms and evaluate their results. This chapter discusses the use-cases for fungible tokens, non-fungible tokens, and supply chain management. Furthermore, it presents three straightforward scenarios that, once accomplished, would mark the successful completion of these use-cases. In Chapter 6, the author's educational background and professional experience are noted to lend a certain degree of credibility to the results. Furthermore, the comparison metric for implementation is defined, and all software versions used are documented to ensure reproducibility. Chapter 7 provides a detailed and structured comparison of selected platforms based on the development perspective catalogue defined in Chapter 3. In Chapter 8, we consolidate the findings from Chapters 6 and 7, discussing the key differences between the platforms and examining the strengths and weaknesses of each one. Furthermore, we group factors that may influence the choice of the platforms and emphasize limitations and future research opportunities. Finally, Chapter 9 concludes the thesis.

CHAPTER 2

Related Work

This chapter addresses related work in the context of a comparison of different smart contract platforms. As this paper covers following platforms: Ethereum, Neo, EOS, Cardano, Algorand, it presents scientific works that have contributed to the writing of this paper, by either discussing some criteria or discussing numerous use-cases of a certain or multiple smart contract platform(s).

In their article "Comparison of Ethereum, Hyperledger Fabric and Corda", Valenta Martin and Sandner Philipp [60] discuss and give a brief analysis of the most notable differences between Ethereum, Hyperledger Fabric, and Corda. They are arguing how the development of both Fabric and Corda were driven by specific use-cases, whereby Ethereum "present itself utterly independent of any specific field of application". They have also provided criteria to note the differences between the platforms, using governance, mode of operation, and consensus among others.

It is worth mentioning that some of the younger platforms are trying to solve problems that Ethereum has that have not been solved yet, i.e., blockchain's trilemma. Ethereum's creator, Vitalik Buterin, said that having two out of three properties (scalability, security, decentralization) is achievable, but having all of them simultaneously is hard. That being said, it is clear that most blockchain's whitepapers or other scientific papers compare different platforms to Ethereum.

EOS is a platform that tackled Ethereum's scaling issue. Xu et al. stated that EOS is "...a platform for smart contract development and decentralized storage in an attempt to address the common scalability issues found in popular blockchain systems such as Ethereum and Bitcoin." [62] In "EOS.IO blockchain data analysis" Song et al. examine transaction data in the EOS blockchain and analyze the data in the Ethereum and EOS

chains from a complex network perspective. [57]
In their recent work "Blockchain Development Platforms: Performance Comparison" Iman Dernayka and Ali Chehab compare EOS and Ethereum to help developers select the most appropriate platform as a Blockchain back-end for their applications.[37] However, they only evaluated the performance of the respective blockchains by triggering and measuring basic operations from the decentralized application they have developed on both platforms. They have also covered an architectural comparison between EOS and Ethereum, but the structural comparison has not been done.

Bareis constructed a structural catalogue of criteria for comparison of Ethereum and NEO in his work "Comparison of Ethereum and NEO as smart contract platforms". [30] The catalogue consists of 4 main categories having several sub-categories. These are *Project, Blockchain Properties, Platform and Development and Execution and Operation.* The catalogue was created by structured literature research and will be used as a reference to describe different platforms from a technical point of view in this paper as well. This will be described in more detail in the next chapter (Chapter 3), section 3.1

Mogavero et al. compare Algorand with Ethereum concerning the blockchain's trilemma as Algorand claimed to have resolved Buterin's concerns. [48] Algorand's founder Silvio Micali says that Algorand technology satisfies all three properties simultaneously. It provides high scalability by obtaining up to 125x the Bitcoin's transaction throughput while offering the possibility to run smart contracts with limited but non-trivial capabilities to run decentralized computations. An important feature of Algorand is that transactions are finalized as soon as they appear on-chain since Algorand's blockchain forks only with negligible probability. One of the criteria was to measure costs and computations in blockchains by observing computational effectiveness, since in some cases, it can be connected with scalability.
As they wanted to answer the question *"Which blockchain is more appropriate to run a certain natural decentralized computation?"* they have also used a natural use-case - auctions, that require decentralized computations and traditionally involve a trusted party.[39]

Yongpu et al. analyze several different blockchains, namely Binance Smart Chain, Polkadot, Cardano, Avalanche, Polygon, Tron, Elrond, Cosmos, and Fantom, and compare them in terms of their degree of decentralization. [42] They believe that behind the excellent performance of many popular blockchains attempting to solve the blockchain trilemma, there is a sacrifice of decentralization. On the other hand, Niss et al. evaluate the performance and scalability of blockchain protocols using an existing framework. Their paper aimed to highlight the differences in performance between Ethereum, which was utilizing Proof-of-Work at that time, and Cardano, a Proof-of-Stake blockchain. [53]

As it can be seen, these papers compare a single or multiple (two in most cases) platform(s) to Ethereum, by either using some specific criteria (transaction speed, execution costs, etc.) or a specific use-case. However, there is not a single paper that discusses and compares multiple platforms to each other from a development perspective, i.e. that analyzes what platform is more suitable for a certain use-case and in that way help a developer choose an appropriate smart contract platform. To tackle this and to better understand how different platforms work, in the following chapters we will create another catalogue of criteria and compare these platforms from a development point of view.

CHAPTER 3

Catalogues of Criteria

We base our detailed structured comparison of different smart contract platforms on a set of carefully selected criteria. Since the focus of this work is on the development perspective, we divide the criteria into two catalogues, one for technical aspects and one for development aspects. In this chapter, we describe the criteria of the two catalogues that we use for our comparison.

3.1 Technical Perspective

The criteria regarding the technical perspective are taken from [30],[31], where the author did systematic literature research in order to derive a comprehensive catalogue of criteria for the technical comparison of Ethereum and NEO as smart contract platforms. Even though development aspects were not a focus in that work, some criteria pertain to this aspect. Therefore, we slightly regroup the criteria from [30], [31]. For the development aspect, we also take the relevant criteria from [30], [31] and add a few new ones.

3.1.1 Project

The Project category comprises three criteria: Objectives, Origin and Organization, and Governance. It specifies what are the objectives of each smart contract platform, who or what organization governs the project and in which direction does the platform develop.

Objectives

This criterion reflects the main objectives of the selected platform and explores what are the use-cases that a certain platform concentrates on. As important as it is to know which use-case and target group your application is being built for, it is crucial to pick a smart contract platform that suits your needs best.

Origin and Organization

The origin-country of a blockchain and the organization behind it also play an important role when deciding which platform for smart contracts to choose as there might be many legal requirements and regulations that need to be dealt with. A couple of examples of how important this might be are multiple bans of cryptocurrencies and mining in India and China in 2021 only [27], [26], [54]. Although this may not be crucial for the development of decentralized applications, it still has to be considered as a potential risk.

Governance

Governance is a term that relates to the decision-making process of a platform for smart contracts, i.e., who can propose changes and how a decision on which changes will be implemented is made. Governance mechanisms can be off-chain and on-chain. Off-chain governance mechanism includes public discussions, proposals, and collectivelly aggreed-upon dates, whereas on-chain includes stakeholders that vote with native coins to make changes to the blockchain directly. [28]

3.1.2 Blockchain Properties

The development of smart contracts highly depends on the underlying blockchain. Therefore it is important to discuss and elaborate on blockchain properties as some of these properties affect operation costs, transaction volumes, and future integration and development of the application.

Consensus Protocol

An important criterion that influences operation costs and transaction volumes is a consensus protocol. The consensus mechanism guarantees blockchain integrity, i.e., that a state, value, or any other piece of information is correct and has been agreed upon by most nodes.

Interoperability

Interoperability represents the possibility of one blockchain to interact with and use services of another blockchain. These interactions could relate to data transfer, verification of the data, transfer of digital assets, or use of the computational capacity of another blockchain network. There are several ways to achieve interoperability as stated by [43]:

- Interaction between a legacy system and a blockchain platform.
- Interaction between two blockchain platforms.
- Interaction between smart contracts of a decentralized application.

It is an important criterion to consider since it improves blockchain scalability and helps achieve a higher degree of decentralization. Thus, when developing decentralized applications blockchain interoperability could be beneficial to reach a higher number of users and a better distribution of a decentralized application. [50]

3.1.3 Execution and Operation

The execution and operation section covers several blockchain properties that could influence smart contracts execution and the functionality of a decentralized application.

Block Time

Block time refers to the approximate time it takes to produce a new block in a blockchain. This criterion also affects transaction confirmation speed measured in transactions per second. There is a simple and effective way to lower the block time by increasing the block size. However, there is a constant debate whether this may affect the security of a decentralized network. On the other hand, higher block time means enough time to update nodes in a network and lower the number of rejected blocks.

Block Confirmation Time

Block confirmation time is defined as the time between the moment a blockchain transaction has been submitted to the blockchain network and the moment when it is included in the block. Depending on a network's architecture, this time can be reduced by offering higher transaction fees. As Bareis stated in his work, the block confirmation time influences the application's design and thus needs to be taken into consideration when designing decentralized applications. [30]

Throughput

Throughput is described as the number of processed transactions per one second. It is usually affected by the platform's underlying mechanisms, like consensus protocol or network configuration, i.e., if it is a private or public network. Picking the right platform is crucial if scalability is one of the important aspects of a decentralized application. By knowing the throughput of the network it can be seen whether the network satisfies users' needs or not.

Execution Costs

Many blockchain networks charge a certain fee for executing smart contracts. These fees are influenced by the usage of the computational resources of the network. They vary based on several factors, e.g., what operation is being executed, if the network is congested at the moment and the users need to pay higher fees for their transactions to be processed, etc. Some operations are more expensive than others. That being said, for example, in Ethereum an operation on permanent storage is much more expensive than an arithmetic operation. [61]

Furthermore, fees are charged in a native currency of a blockchain network, e.g., Ether in Ethereum, Matic in Polygon Network, etc. Another point that influences execution costs is the volatility of the native currency at the moment of transaction execution.

When evaluating a smart-contract platform, execution costs and the platform's fee model are helpful to be able to predict future costs of an application.

3.2 Development Perspective

The criteria covered in this section give an overview of the key aspects of smart contract platforms from a non-technical point of view. In addition to the technical aspects, non-technical ones might even play a more important role when choosing a suitable smart contract platform for a particular use-case. As previously said, some of the criteria are taken from [Bar19], [BDAS20], whereas others are added based on personal work experience as a software developer. The comparison of different smart contract platforms based on these criteria will be conducted in Chapter 7.

3.2.1 Blockchain Properties

The blockchain properties section consists of three criteria and covers some of the most significant aspects when it comes to the ease of a concrete blockchain setup and use, which could highly affect the choice of a certain smart contract platform.

Chain Availability

One of the conditions for successfully deploying decentralized applications on a blockchain, is the mainnet availability. If the mainnet of the blockchain is not available yet, the application can not be used or its usage is very limited. On the other side, to have a successful lunch of the application, the application should be well tested beforehand. Blockchains' testnets represent blockchain environments, similar to the mainnets, in which users can test their applications. The importance of testing is immense since the blockchain is immutable and, as such, transactions, once mined, cannot be reversed, which could lead to severe consequences. Projects that have multiple testnets available allow users to test their application on multiple environments, and as such ensure the robustness of the software. Hence, this criterion discusses the availability of mainnet and testnets of different blockchains.

Permission Type

The permission type criterion denotes whether the chain is public, or private, i.e. permissionless or permissioned. The term "public blockchain" refers to a blockchain that does not require authentication of participant identities nor any kind of authorization of participant's rights. On the contrary, "private blockchain" refers to a blockchain where both, authentication and authorization are implied. Since blockchain is a highly use-case-specific technology, this criterion might affect the decision about the choice of a smart contract platform based on the underlying chain.[44]

Local Blockchain Instance Setup

Besides test networks, developers usually run the application on a local blockchain instance before deploying it to the mainnet. This is another way of testing functionality and the correctness of the application. This criterion refers to how hard it is to set up such a network and get it up and running. It also compares and discusses if there are any limitations and prerequisites that may impede the development of an application on a specific platform.

3.2.2 Official Documentation and Support

This section covers two criteria: Language Support and Documentation. Apart from blockchain types and other technical and architectural aspects of a decentralized application, these two criteria might be the most important ones for developers when deciding which platform to pick. In the following subsections, we explain what will be compared and evaluated.

Language Support

Every platform uses its programming language. Many of these platforms also support multiple languages, making it easier for developers to choose a platform that promotes a language they are familiar with. In case developers have to learn a new language, that could incur additional costs for a responsible organization. Therefore, this criterion compares platforms with regard to programming languages. It answers questions like:

- What is the main language of the platform?
- Are other languages supported?
- What languages exactly are supported?

Documentation

Apart from supported languages, how well a particular platform is documented plays a crucial role. If a chosen platform is not well documented, developers might struggle to implement and deliver a decentralized application on time or at all. This could have negative consequences, either for a company or the application itself, increasing further costs. This criterion refers to the availability of the official documentation and its quality and will be evaluated based on the following aspects:

- Is the smart contract platform as a concept explained?
- Is the necessary tech stack given and explained?
- Are some of the basic topics covered (e.g., developing, compiling, testing, deploying smart contracts)
- Are some advanced topics for more complex applications and use-cases covered?

3.2.3 Extended Documentation

The extended documentation refers to further pieces of information including unofficial parts. It comprises contributions from the community, exemplary contracts, educational material, and tutorials. These are all means of support where the developers could look for any help needed.

Community

Most of the time, community support is the support of the highest value for developers. When it comes to developing decentralized applications on different smart contract platforms, it can be that developers can ask questions directly to the team responsible for the development of a specific platform and get help in that way. Although this is possible, it might not always be necessary. The community is represented by active users, contributors, and platform enthusiasts. This criterion compares platforms based on their social media engagement (e.g., Twitter, Facebook), active forums (e.g., Reddit), chat rooms, and groups (e.g., Discord, Telegram). It assesses how active the community is based on the time when the last question was asked and answered and how long it takes to get the question answered.

Exemplary Contracts

When developing a decentralized application, it can be helpful to have some exemplary contracts to rely on or some contracts that developers and the community have checked and verified against any security flaws. This criterion answers the question of their existence and availability.

Educational Material and Tutorials

Some platforms provide additional tutorials, where the active users of a specific platform publish their code-base or the full implementation of a particular topic (e.g., crowdfunding application). As for the "Exemplary Contracts" subsection, questions about their existence and availability will be addressed and answered in this paper.

3.2.4 Platform Development

Every platform has its development tools, testing environments, and standards. The development tools, so as testing environments, help developers build and test decentralized applications. If the platform implements specific standards, it is beneficial to understand them and implement them adequately.

Tools

Despite the increasing popularity, smart contract development remains somewhat limited due to its special design and application. [63] According to a survey conducted in [63] one of the challenges that many developers face is that the existing tools for development are still very basic. Besides, in the follow-up survey with developers, 88% of developers agreed that it is challenging to debug a smart contract and that there is a lack of powerful interactive debuggers compared to traditional software development. Consequently, this means that there is a need for improved tools suitable for smart contract development, e.g., specialized IDEs and debuggers.

This section discusses and elaborates on the availability of different tools for a particular smart contract platform that should make it easier to develop a decentralized application. Otherwise, the lack of adequate tools may cause higher costs, a steeper learning curve, and an increased number of bugs.

Standards

A smart contract standard is a set of rules that smart contracts must comply with. These standards improve smart contracts' security, interoperability, and re-usability and therefore play an important role when evaluating smart contract platforms. [31] This section discusses which standards exist, which use-case they are used for, and which smart contract platform implements them.

CHAPTER 4

Comparison of Platforms from a Technical Perspective

In this chapter, we will briefly explore the technical perspectives of the various platforms under discussion, comparing them based on the catalogue of criteria developed by Bareis et al. [30]

4.1 Objectives

All platforms we discuss in our thesis are classified as general-purpose blockchains. This means they are designed to support a wide range of decentralized applications (dApps) rather than being optimized for a specific use case. Ethereum's main objective is to provide security with decentralization [18], as well as Cardano's and Algorand's. [40] [5]. On the other hand, EOS emphasizes scalability and high throughput as its main advantages [15], while Neo focuses on digital assets and identities. [25]

4.2 Origin and Organization

The Ethereum Foundation was established in Switzerland as a non-profit organization in 2014, with the primary objective of bolstering Ethereum's development. In China, the Neo Foundation was founded to act as a strategic guide for the Neo platform. This organization is tasked with laying out the strategic goals, driving adoption, and ensuring the platform's technological advancement, all while fostering a vibrant and self-sustaining ecosystem [25] The Algorand blockchain is backed by Algorand, a company based in Boston, Massachusetts, as well as by the non-profit Algorand Foundation. In 2017, the private company Block.one developed and launched the EOS blockchain and later released it as open-source software. IOHK, the private company, founded by Charles Hoskinson and Jeremy Wood, developed and released Cardano blockchain to the public in 2017.

4.3 Governance

As stated in Ethereum's whitepaper, Ethereum governance takes place off-chain, with various stakeholders involved in the process. The stakeholder list comprises several entities, such as Ether holders, application users, node operators, Ethereum Improvement Proposal (EIP) authors, validators, protocol developers, and application/tooling developers.[18] Similarly, Neo is also community-driven, where NEO holders participate in governance by voting for a Neo Council to oversee the management of the Neo blockchain. [24] Though Cardano is administered by three independent entities - the Cardano Foundation, IOHK, and Emurgo - its governance is also dictated by its community. Token holders, as network stakeholders, are entitled and incentivized to vote on proposals for the development or upgrade of the blockchain and its ecosystem. [6] Unlike Ethereum, the majority of EOS governance occurs on-chain. While anyone is free to submit new policy or code change proposals, these must receive a majority vote before being incorporated into the codebase. Nonetheless, discussions related to governance, which can influence decisions made on-chain, still occur in an off-chain environment. [8] Contrarily, the Algorand network is governed by the Algorand Foundation. However, the foundation is devoted to further decentralizing the network and entrusting more decision-making power to the broader Algorand community. [2]

4.4 Consensus Protocol

All selected platforms utilize similar consensus mechanisms, specifically variants of proof-of-stake (PoS). The Neo protocol employs a Delegated Byzantine Fault Tolerant (dBFT) consensus mechanism, while Algorand uses a decentralized Byzantine Agreement that leverages pure proof-of-stake (Pure PoS). Both Ethereum and Cardano employ a traditional PoS mechanism, and EOS has implemented delegated proof-of-stake (dPoS). The above-mentioned consensus mechanisms vary in the methods they use to select validators and the type of security models they employ. Moreover, each of these consensus mechanisms has its trade-offs between security, decentralization, and scalability.

4.5 Interoperability

Neo achieves cross-chain interoperability with its NeoX protocol, which is divided into two components: cross-chain asset exchange and cross-chain distributed transactions. Algorand has developed a new standard, Algorand State Proofs (ASP), to enable interoperability between blockchains. It utilizes what is referred to as 'light clients'—efficient software that tracks the state of the blockchain—to provide a simple, trustless interface for inter-blockchain communication. [4] Similarly, EOS has also developed its Inter-Blockchain Communication Protocol (IBC). This IBC mechanism is facilitated through a suite of smart contracts, specifically the IBC Bridge and the Wrap/Lock Token. [9] On the other hand, Ethereum achieves interoperability through bridges and Layer 2 solutions. Concurrently, Input Output Global (IOG) provides a sidechain toolkit and technical specifications to enable interoperability on the Cardano platform. [5]

4.6 Block Time

Among the blockchain platforms in our selection, two exhibit significantly different block times, while the rest have more comparable durations. EOS has the fastest block production time, generating a new block every 0.5 seconds. [58] Algorand ranks as the second-best platform, with a block production time of approximately 3.6 seconds. [20] Cardano, Neo, and Ethereum have more comparable times. The block creation time for Ethereum is roughly 15 seconds, although it can peak at times to around 30 seconds. [49] Cardano produces a new block every 20 seconds, whereas Neo does it in approximately 15 seconds.

4.7 Block Confirmation Time

Confirmation time refers to the duration between the moment a transaction is submitted to a blockchain network and the moment it is officially recorded in a confirmed block. In blockchains using Proof-of-Work (PoW) or Proof-of-Stake (PoS) consensus mechanisms, transactions finality is achieved over time in a probabilistic way by utilizing confirmations. In Neo, each block requires agreement from a 2/3 majority of consensus nodes before it is committed to the blockchain. Because each potential block must receive full network consensus in this manner before acceptance, transactions in Neo achieve absolute finality as soon as they are confirmed within a single block. [25] For the same reason, the Algorand blockchain has also an instant transaction finality, meaning that the chain is fast to confirm new blocks. [20] However, Ethereum's Proof-of-Stake protocol stipulates that for a block to be incorporated into the blockchain, it must be staked with Ether by at least 2/3 of the validators. Hence, the average time to achieve transaction finality is 14 minutes. In EOS the Last Irreversible Block (LIB) is the block that has achieved 15/21 consensus and it takes about 3 minutes to finalize it. On the other side, Cardano documentation says that the transaction finality can be achieved in approximately one day according to Ouroboros consensus design.

4.8 Throughput

One major issue that blockchain faces is scalability. Ethereum currently processes 24 transactions per second on average. On the other hand, Algorand's documentation asserts that it can process 6,000 transactions per second. Recent studies by Algorand indicate a new benchmark in transaction processing, demonstrating its capability to handle 10,000 transactions per second. [3] When it comes to Neo, using the dBFT consensus mechanism, which generates a block in about 15 to 20 seconds, the transaction throughput is gauged to reach up to approximately 1,000 transactions per second. [25] In its whitepaper, EOS maintains that its blockchain possesses the potential to scale up to one million transactions per second [12]. Nevertheless, the company behind it announced in 2018 that they have achieved 6,000 transactions per second. Regrettably, we could not find any more recent studies or performance tests. Cardano can accommodate 250 transactions per second [46], and it is currently developing a layer-2 solution, the Hydra protocol. This upgrade should enhance scalability and enable Cardano to process up to 1,000 transactions per second. [19]

4.9 Execution Costs

Execution costs refer to the expenses required to perform computational operations on the blockchain. Most platforms impose these costs in their native currencies, which means the total costs can fluctuate based on the currency's value when converted to fiat money. Ethereum's fees are paid based on the gas required to complete a transaction. Gas is a measurement unit for the computational effort required to complete a specific operation on Ethereum. It can be calculated as follows:

$$\textit{total fee} = \textit{units of gas used} * (\textit{base fee} + \textit{priority fee})$$

where *base fee* is set by protocol and *priority fee* is a value the user sets as a tip to the validator. [16] It is also important to mention that fees usually depend on network congestion.
Neo blockchain introduces similar calculation mechanism having system fees (instruction fee, storage fee, system call fee) and network fees (network byte fee, script verification fee). On the other side, Algorand does not have a gas concept. The minimum fee on the Algorand blockchain is 1,000 microAlgos or 0.001 Algo. The size of the transaction determines fees, and users have the option to increase these fees to prioritize their transactions, especially during peak network traffic when blocks are consistently filled. Cardano uses its fee system to cover transaction fees and long-term storage costs of transactions. These fees are constructed around two constants (dependence of the transaction cost on the size of the transaction (a) and payable fee (b)), hence minimal fees are calculated as follows:

$$\textit{total fee} = a * \textit{size}(tx) + b$$

EOS introduced a different concept when compared to all the other platforms. Namely, EOS tokens can be sent or received, and applications built on its network can be used without incurring fees. EOS token holders leverage the resources of the blockchain to interact with and operate applications on the network. The protocol periodically generates new EOS tokens to reward those entities that run the network, effectively replacing transaction fees with inflation. This system forms an ecosystem where the network bears the cost of fees rather than charging individual users for each transaction they make. [7]

CHAPTER 5

Use-case Definitions

5.1 Fungible Token

Because of its nature, blockchain technology is well-suited for empowering financial transactions. A transaction, once validated, becomes irreversible, permanent, verifiable, and secure on the blockchain. Bitcoin is the first application of a digital token (currency) powered by blockchain technology. With the emergence of Ethereum in 2015, the growth of decentralized applications began to increase, thereby contributing to the creation of digital tokens that developers used to interact with decentralized applications. This led to a new era where any scarce asset, and not only currency, could become tokenized, thus further expanding blockchain development and usage. [34]

Fungible tokens, per definition, are blockchain tokens that usually have either a fixed supply or follow a specific and transparent supply schedule. Furthermore, they are interchangeable and equal in value, therefore being a trusted means of conducting transactions in the blockchain.
In blockchain technology, we differentiate two major types of tokens: currencies and tokens. Currencies, also known as coins, are units of a value usually native to a blockchain. It is used to incentivize the network of participants to use the blockchain (e.g., Bitcoin, Ethereum, Litecoin).

On the other side, tokens are governed by a smart contracts and are built on top of the protocol, having other functionalities, rather then being just an exchange of value. Tokens can have different utilities and their value depends on the value of underlying assets or services that these tokens represent. [45].
Not many years passed until developers realized they can tokenize projects and sell tokens to fund their projects. This is how new way of fundraising was born - Initial Coin Offerings (ICOs). [34]

Even nowadays, this is the most common use-case for blockchain. According to Coingecko there are 12,906 distinct tokens/coins. [1]. This is the first use-case we will be implementing and comparing different smart contract platforms.

A simple scenario is defined as follows:

1. A token with the name "SimpleToken", the symbol "STK", having a fixed supply, will be created and deployed to the blockchain.
2. The balance of the creator's address is checked and verified to contain a certain number of STK tokens (e.g., 1000).
3. A new address will be created and verified to contain 0 STK tokens.
4. 300 STK tokens will be transferred to the newly created address.
5. The newly created address will be verified to contain 300 STK tokens.
6. The creator's address will be verified to contain 700 STK tokens.

After all 6 steps have been successfully processed and verified, the use-case is considered done and successfull.

Methods and functionalities of fungible tokens are derived from the analysis of different platforms and their token implementations. Every fungible token should adhere to the following rules:

1. A token should have a defined **name** (e.g. Ethereum)
2. A token should have a defined **symbol** (e.g. ETH)
3. A token should have a defined **supply** (e.g. 10 000)

Following list represent list of minimum functionalities each token should contain:

1. Method: *Total Supply*

input value	empty
return value	number
description	Returns a total number of coins in circulation

[1]https://www.coingecko.com/; Last accessed on: 12.09.2022

2. Method: *Balance*

input value	address of an account
return value	number
description	Returns an amount of tokens owned by the *account* address

3. Method: *Transfer*

input value	sender address, recipient address, amount of tokens to transfer
return value	boolean
description	Transfers a specified amount of tokens from *sender's* address to the *recipient's* address, if possible. The amount is then deducted from the sender's account.

5.2 Non-fungible Token

As the Internet evolved throughout the last two decades, a new era of digital content creation began, and with time such content became available to almost everyone. Furthermore, becoming available to the majority, it became the preferred and most widespread means of advertising, marketing, and business.

Despite digitizing content through online, legal platforms (e.g., Google, Youtube), there has always been and still is the creation and sharing of pirated content (e.g., on black-market sites). Deductively, it is challenging for the original authors (e.g., artists, musicians, brands) to verify and profit from their digital content. [35]

One application of blockchain technology that tackles this problem is non-fungible tokens (NFTs). In contrast to a fungible token, a non-fungible token is a unique digital asset stored on the blockchain and is not interchangeable with other digital assets. In addition, it has different properties contained in its metadata, which determine the difference in appearance, scarcity, rarity, utility, and many other factors that could affect the value of a token. For example, a hat signed by a famous basketball player has more value than the same hat in a regular shop, despite all other similar characteristics and appearance.

One of the main advantages of blockchain is that everything is immutable and publicly verifiable by anyone once saved to the blockchain. Thus, NFTs can be efficiently used to record the ownership of digital media (e.g., images, videos) or real-world items (e.g., paintings, music records). In that way, creators can claim their ownership of an NFT and collect royalties not only from the initial sale but also from future sales. Furthermore, all the details of reselling, royalties, and ownership rights are sealed in and regulated by a smart contract.

We used the previously described use-case to explain non-fungible tokens, how they can be relevant, and what importance they have for day-to-day life. As we already know, many blockchains support fungible tokens. However, there is also a strong indication of the growth of blockchains supporting non-fungible tokens. Therefore we decided to construct a simple use-case where we will transfer and confirm the ownership of the non-fungible token to compare different smart contract platforms, i.e., blockchains.

A simple scenario is defined as follows:

1. A token with the name "NFTToken", and the total supply of 1 (since it is a unique token) will be minted.
2. The new address will be created and verified to have 0 NFT tokens.
3. The freshly minted NFTToken will be transferred to the newly created address.
4. The verification of ownership will take place. The newly created address ought to be the owner.
5. The address that created the token should not posses the NFTToken anymore.

As before, after all 5 steps have been successfully processed and verified, the use-case is considered done and successful.

Methods and functionalities of non-fungible tokens are derived from the analysis of different platforms and their token implementations.
Every non-fungible token should adhere to the following rules:

1. A token should have a defined **name** (e.g. Ethereum)
2. A token should have a defined **symbol** (e.g. ETH)
3. A token should have a defined **unique identifier** (e.g. tokenURI)

Following list represent list of minimum functionalities each token should contain:

1. Method: *Owner*

input value	token ID
return value	string
description	Returns the address of the account that is owner of the NFT with the given *token ID*.

2. Method: *Balance*

input value	address of an account
return value	number
description	Returns an amount of tokens owned by the *account* address

3. Method: *Transfer*

input value	sender address, recipient address, token ID
return value	boolean
description	Transfers a token with the given *token ID* from *sender's* address to the *recipient's* address, if possible.

5.3 Supply Chain Management

Supply chain management, represents the management of the flow of goods and services between multiple organizations and locations (i.e., a production flow) from the beginning - starting with the raw components to the very end - by delivering goods and services to the final consumer. In this way, a network of businesses and relationships is created to cut the total costs, increase efficiency, drive customer value, and fulfill demand.

The created network consists of several business entities, among which are producers, suppliers, freight forwarders, warehouses, distributors, and retailers, all working together, as mentioned above, to acquire raw materials, transform the materials into a specified final product, and deliver the product to the final customers. [38]

Due to globalization, a modern supply chain has evolved into highly complex value networks making its management and control more difficult. Each of these participants in a supply chain process has to keep its record of transactions and do proper data management to ensure efficiency. With such a high number of involved parties, there is a high probability of erroneous and maliciously altered data. Furthermore, it becomes challenging to trace the history of food, verify the source of raw materials and maintain the visibility of a product and merchandise as they move along the supply chain, which is becoming increasingly important for modern companies. [55]

Such a high motivation for traceability and transparency comes from many different reasons, among which are the attestation of product origin, quality and identity, compliance with the international regulations, compliance with the international standards and certifications, marketing strategies, and most important, reacting or preventing any possible sanitary outbreak due to food safety issues. [36]

Blockchain, as a distributed shared ledger technology, could help tackle this issue and provide complete transparency and traceability of products even with an enormously high number of stakeholders, with the help of its consensus mechanism and shared ledger. As known, every node on a blockchain platform maintains and validates transactions in the shared ledger. Therefore, all stakeholders would have a duplicate of all transactional records and access permission to monitor the process flow. [32]

With its characteristics, blockchain significantly reduces the need for centralized trusted authorities that operate and maintain the whole system, allowing customers to inspect the chain of custody and transactions from raw materials to the end sale. [56]

The blockchain is still in its early stages, and there always exist difficulties concerning organizational, regulatory, technological, and policy-oriented aspects of supply chain management application. However, we decided to implement this use-case as one with high potential and one of the clearest examples of blockchain applications. In that way, we compare different smart contract platforms.

Hence, we defined the following actors and the simple scenario of a coffee supply chain:

1. **Farmer** is in charge of harvesting, processing, packing, selling, and shipping coffee beans to a distributor.
2. **Distributor** is in charge of buying coffee beans from a farmer and distributing them to a retailer.
3. **Retailer** receives coffee beans from a distributor and makes them available for end consumers.
4. **Consumer** purchases coffee beans from a retailer.

Additionally, there are different states of the supply chain we defined to denote how far the coffee is along the chain:

Harvested, **Processed**, **Packed**, **ForSale**, **Sold**, **Shipped**, **Received**, **Purchased**.

The following points describe the use-case scenario and a simple supply chain management process. The use-case is considered successful if all steps are correctly executed. The figure below the description (Figure 5.1) represents the state diagram of the scenario.

1. A farmer harvests coffee beans and marks them ’harvested’.
2. After being harvested, a farmer processes coffee beans and marks them ’processed’.
3. After processing, a farmer packs coffee beans and changes their state to ’packed’.
4. After being processed and packed, a farmer wants to sell coffee beans, changing their state to ’for sale’.
5. A distributor buys coffee beans and sets their state to ’sold’.
6. Afterwards, a distributor ships coffee beans and sets the state to ’shipped’.
7. Once beans have been shipped, a receiver can mark them ’received’.
8. In the end, a customer can purchase coffee beans which sets the state of coffee beans to ’purchased’.

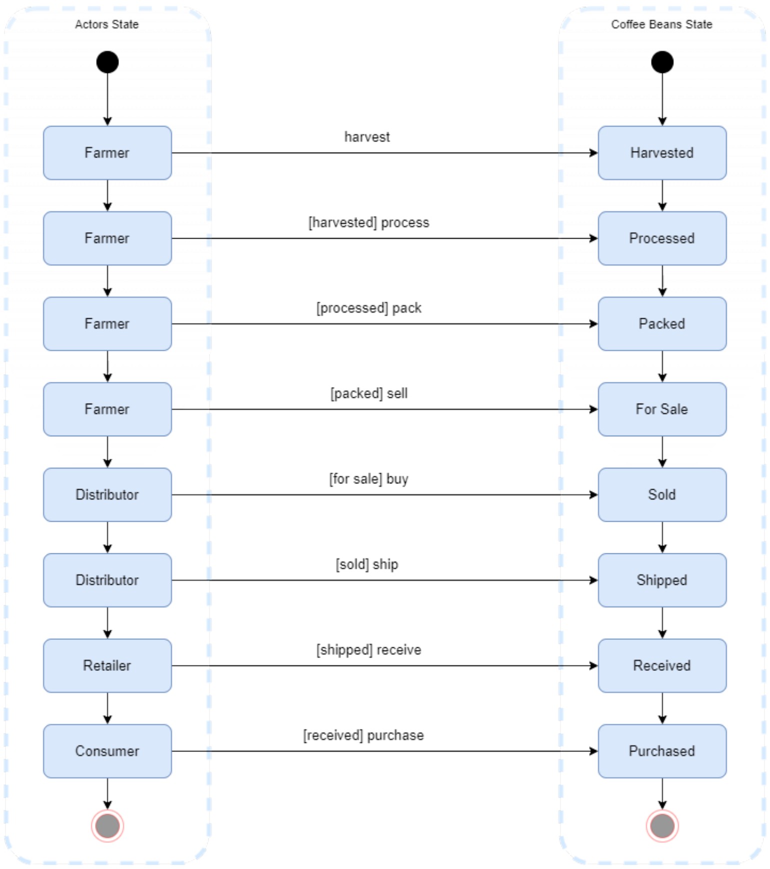

Figure 5.1: State diagram of a simplified coffee supply chain scenario

Furthermore, every coffee item (not coffee bean) should have the following properties:

1. *sku* - Stock Keeping Unit
2. *upc* - Universal Product Code, generated by a farmer, verifiable by consumer
3. *ownerID* - a blockchain address of a current owner
4. *farmerID* - a blockchain address of a farmer
5. *productID* - a unique product id - a combination of sku and upc
6. *itemState* - a state of an item
7. *productPrice* - a price of an item
8. *distributorID* - a blockchain address of a distributor
9. *retailerID* - a blockchain address of a retailer
10. *consumerID* - a blockchain address of a consumer

The use-case should also implement the following methods, whereby the authorization has been omitted here but will be implemented according to the specifications of a specific platform.

1. Method: *Harvest Item*

input value	universal product code
return value	void
description	Adds a new item as a part of the harvest and sets the item's state to '*harvested.*'

2. Method: *Process Item*

input value	universal product code
return value	void
description	Sets the state of the item with the given *universal process code* to '*processed.*'

3. Method: *Pack Item*

input value	universal product code
return value	void
description	Sets the state of the item with the given *universal process code* to '*packed.*'

4. Method: *Sell Item*

input value	universal product code, price
return value	void
description	Sets the price of the item with the given *universal process code* and changes its state to '*for sale*.'

5. Method: *Buy Item*

input value	universal product code
return value	void
description	Checks if a distributor offered enough money to buy the item with the given *universal product code*; if positive, changes the ownership of the item to the *distributor*; transfers the money from *distributor* to the *farmer* and sets the state of the item to '*sold*'.

6. Method: *Ship Item*

input value	universal product code
return value	void
description	Sets the state of the item with the given *universal process code* to '*shipped*.'

7. Method: *Receive Item*

input value	universal product code
return value	void
description	Sets the state of the item with the given *universal process code* to '*received*' and changes the ownership of the item to the *retailer*

8. Method: *Purchase Item*

input value	universal product code
return value	void
description	Sets the state of the item with the given *universal process code* to '*purchased*' and changes the ownership of the item to the *consumer*

CHAPTER 6

Implementation

In the previous chapter, we defined three use-cases, with the help of which we will compare different blockchains based on our previously described catalog of criteria from a development perspective.
This chapter introduces us to the implementation details of the selected use-cases, tools used, and their versions so that the reproduction could be possible at any time. It also states the author's personal and professional experience to support the results and facts presented below and to help the reader understand how we deducted the results. In addition, it introduces several categories that are used as a comparison metric and also defines measurements for each category.

6.1 Personal experience

The author of the research paper has completed the Bachelor's program in Software and Information Engineering at the Vienna University of Technology and attends the Master's program in the area of Software Engineering and Internet Computing at the same university. Besides, he has worked as a Java software developer in a multinational technology company for over three years, focusing on supply chain management. Furthermore, at the time of writing, he works as a blockchain developer for another international company, providing diverse solutions concerning non-fungible tokens and supply chain management.

6.2 Comparison metric

To assess and describe the implementation we defined following categories:
Contract length, *Time to code*, *Experience level* and *Difficulty occurrence*.

It is essential to mention that categories are assessed based on the same measurement scale (Low, Medium, High), but each scale has a different meaning in every category. We describe the details and the context of each category and measurement scale below.

Contract length

Although the length of a contract does not necessarily mean more complexity, it could introduce potential security risks and higher maintenance. This metric refers to the number of lines of code. Since we have clearly defined the basic functionalities of each use-case in Chapter 6, this category could be helpful to see how blockchains implement different functions to complete a specific task (e.g., some blockchains might use a library or have a built-in function). Thus, it might denote how much code is boilerplate code. It is also firmly interconnected with other relevant categories and could indicate ease of use, time to code, experience, and maintenance levels.
By analyzing implementations of the aforementioned use-cases we determined the following measurements:

- **Low** - refers to contracts that have less than 100 lines of code.
- **Medium** - refers to contracts that have between 100 and 300 lines of code.
- **High** - refers to contracts that have above 300 lines of code.

Time spent

This metric is undoubtedly not crucial when choosing a blockchain for development. However, it could indicate how steep a learning curve might be or how complicated it is to utilize a particular chain. Like the previous metric, time spent is influenced by many factors, among which are the developer's experience level, the complexity of a use-case, and the frequency of difficulty occurrence. More specifically, this metric defines how many working hours were necessary to set up everything, write the code and get it running. Considering the definition, we introduce the following measurements:

- **Low** - less then 10 working hours.
- **Medium** - between 10 and 20 working hours.
- **High** - more than 20 working hours.

Experience level

Experience level is a metric that refers to years of experience needed in a software engineering field to understand and implement a specific use-case by using either known or unknown technology. In this case, the emphasis is on a general software engineering experience and not specifically on the experience in the blockchain field. The metric,

again, could suggest if the particular blockchain is more complicated to setup and use than the other; how complex is the use-case, but it is also highly dependent on an individual's knowledge.
It comprises following measurements:

- **Low** - 0 to 3 years of experience

  A developer is familiar with the basics of software engineering and is capable of applying the knowledge to implement simple tasks.

- **Medium** - 3 to 5 years of experience

  A developer has a solid knowledge of software engineering practices and can apply them to implement more complex tasks. He/She is also familiar with different technologies and thus can grasp the basics of the new technology faster.

- **High** - more than 5 years of experience

  A developer has a deep knowledge of software engineering practices. As a result, he/she can utilize multiple platforms, tools and technologies to design, plan and implement complex tasks.

Difficulty occurrence

All blockchain projects assessed in this research paper are stable, although the technology is relatively new. Therefore, when choosing a blockchain for a project, it is significant to have access to resources to reduce the number of possible difficulties. Difficulty occurrence represents a metric defined as a frequency of difficulties encountered during the implementation of selected use-cases. A difficulty, in this case, is defined as follows:

- Lack of specific documentation
- Unclear documentation
- Non-existing examples
- Lack of an active community
- OS compatibility
- Any code-related limitations (e.g., compiler compatibility)

As in previous cases, we define measurements as follows:

- **Low** - refers to 0 to 5 difficulties encountered
- **Medium** - refers to 5 to 10 difficulties encountered
- **High** - refers to more than 10 difficulties encountered

Some blockchains are structurally designed to support certain use-cases better than others. Therefore, by comparing blockchains against a specific use-case instead of overall, we can derive what blockchain would be a better fit for particular case. Additionally, not all use-cases are of the same complexity and it would be hard to make a general comparison based on the collected data. The following tables represent the comparison of blockchains against three use-cases.

	Contract length	Time spent	Experience level	Difficulty occurrence
Ethereum	Low	Low	Medium	Low
Neo	Low	Low	Medium	Medium
Cardano	Low	Medium	Low	Medium
EOS	Medium	Medium	Medium	High
Algorand	Medium	Low	Low	Medium

Table 6.1: Comparison of different blockchains with regard to "Fungible token" use-case

	Contract length	Time spent	Experience level	Difficulty occurrence
Ethereum	Low	Low	Medium	Low
Neo	Medium	Low	Medium	Medium
Cardano	Low	Medium	Medium	High
EOS	High	Medium	Medium	High
Algorand	Medium	Low	Low	Medium

Table 6.2: Comparison of different blockchains with regard to "Non-fungible token" use-case

	Contract length	Time spent	Experience level	Difficulty occurrence
Ethereum	Medium	Low	Medium	Low
Neo	Medium	Medium	Medium	Medium
Cardano	-	High	Medium	High
EOS	High	Medium	Medium	High
Algorand	High	Medium	Medium	Low

Table 6.3: Comparison of different blockchains with regard to "Supply chain management" use-case

As stated, it is necessary to know what tools have been used and the exact versions to successfully reproduce each of the selected use-cases. Therefore, the following tables represent tools used for a particular chain and their versions.

	Node	npm	Truffle	Solidity	Ganache
Version	v16.13.2	v8.4.1	v5.4.32	v0.8.11	v2.5.4

Table 6.4: Ethereum Stack

	Node	npm	Sandbox	algod	postgres	Docker Desktop	Beaker
Version	v16.13.2	8.4.1	no release	v3.4.2 stable	v13.6	4.6.0	1.0

Table 6.5: Algorand Stack

	Node	npm	neo-express	.NET
Version	v16.13.2	v8.4.1	v.3.1.46	5.0 SDK

Table 6.6: Neo Stack

	WSL	Ubuntu	GHCup Haskell	ghc	cabal
Version	2	20.04 LTS	v.0.1.17.8	v8.10.7	v.3.6.2.0

Table 6.7: Cardano Stack

	WSL	Ubuntu	EOSIO binaries	EOSIO cdt
Version	2	20.04 LTS	v2.1.0-1	v1.8.0-1

Table 6.8: EOS Stack

CHAPTER 7

Comparison of Platforms from a Development Perspective

After describing the implementation details and categorizing the results in the previous chapter, this chapter compares platforms for smart contracts using the catalogue of criteria from a development perspective, as defined in Chapter 4. We analyze selected platforms and document our findings and experience. Afterwards, we discuss those findings in Chapter 9.

7.1 Blockchain Properties

7.1.1 Chain Availability

Networks represent different environments of a specific blockchain that can be used for testing, developing, and production use-cases. As explained in one of the previous chapters, having a working blockchain network is of utter importance to the specific smart contract platform, i.e., to the whole ecosystem being built around the platform.
Ethereum has a working main network (*mainnet*) which is a production network that processes and saves all transactions to Ethereum's blockchain. It is a public network where everyone can create, read and validate executed transactions. In the main network, Ethereum's native token ETH is used as payment for transaction costs. By deploying to the mainnet, applications can leverage the full potential of decentralization. In addition to the functional main network, Ethereum also possesses several test networks (*testnets*), where users can deploy and test their applications before they go live. It is worth mentioning that these networks are production-like environments. However, unlike the main network, they do not use the real ETH token, i.e., the ETH token does not have a real value on test networks and is usually acquired from faucets. This is, in most cases, how main and test networks work and how they utilize the blockchain's native

token. Ethereum always had several test networks where some of them started by using proof-of-authority consensus mechanisms, while others used proof-of-work. To be more specific, Ethereum's test networks are following:

Proof-of-Authority (PoA)

- Kovan - the old proof-of-authority network for those who are still running OpenEthereum clients
- Rinkeby - the proof-of-authority network for those running old versions of Geth client
- Goerli - once proof-of-authority test network

Proof-of-Work (PoW)

- Ropsten - once proof-of-work test network
- Sepolia - once proof-of-work test network

These were some of the latest Ethereum test networks, with their respective consensus mechanisms. However, after Ethereum transitioned from Proof-of-Work to Proof-of-Stake (also known as Ethereum Merge), some test networks also transitioned to the new consensus mechanism, while others became deprecated.[16] This is the current list of Ethereum's test networks, indicating both active and deprecated ones:

Proof-of-Stake (PoS)

- Sepolia - the network that transitioned to the PoS mechanism. It represents Ethereum's default testnet for application development.
- Goerli - another network that transitioned to the PoS mechanism. It is the recommended default testnet for testing of validating and staking.

Deprecated Networks

- Ropsten
- Rinkeby
- Kovan

The **Neo** blockchain uses, similar to Ethereum's Proof-of-Stake (PoS), a highly advanced consensus mechanism called Delegated Byzantine Fault Tolerance (dBFT). Although testing on multiple environments ensures a software's robustness, Neo has only one test network in addition to its main network. [21]
The economic model of Neo consists of two native tokens, NEO and GAS, which significantly differs from Ethereum's model. NEO represents the primary means for payment settlement on the platform and also represents the right to manage the network, i.e., the right to vote for bookkeeping and the right to change Neo network parameters. On the other side, GAS is used to pay transaction fees and incentivize consensus nodes to generate new blocks. [25] This dual token model has its advantages which will be discussed in the next chapter.

The core service daemon, *nodeos*, operates on each **EOSIO** node and can be customized to execute smart contracts, authenticate transactions, generate blocks that encompass legitimate transactions, and verify blocks for their addition to the blockchain. Based on a specific goal and how nodeos is configured, it is possible to setup following development environments in **EOS**:

1. Local Single-Node Testnet
2. Local Multi-Node Testnet

Besides EOS also have EOSIO Public Blockchain representing main network and several test networks:

1. Official Testnet - offline at the time of the writing
2. Third-party test networks
    - Jungle Testnet
    - CryptoKylin Testnet
    - Telos Testnet

The EOS Public Blockchain operates on Delegated Proof of Stake, whereas additional information about EOS test networks were not available.[15] [13]

Similar to the NEO blockchain, the **Algorand** blockchain employs a decentralized Byzantine Agreement protocol that utilizes pure proof of stake (Pure POS). Additionally, like most smart contract platforms, Algorand provides public networks for anyone to use. These networks encompass the following:

1. MainNet - The main network of Algorand, wherein Algo tokens are utilized as a form of currency

2. TestNet - The test network with conditions that closely mirror real-world scenarios, on which applications are deployed before being launched on the main network
3. BetaNet - The beta network where applications can be deployed with access to the newest protocol-level features

The final platform in our discussion, which holds equal importance, is **Cardano**. In addition to its main network (public network), Cardano provides various early-stage and pre-production networks that enable active testing of core Cardano features before deployment on the mainnet. To join Cardano test networks, one must configure the Cardano node and Command Line Interface (CLI), create keys and addresses, and fund them with the test currency (ADA). Below is a list of all the test environments available according to the Cardano's official documentation page [5]:

1. Early-stage testing networks
    - Devnet - a network designed for early involvement and testing of functionality before a release candidate becomes mature. It is intended for projects such as DApps that want to explore new features as soon as they are released.
    - Preview - a network environment that is used for extended testing of release candidates and for conducting more comprehensive test scenarios.
2. Late-stage testing networks
    - Pre-production - the most advanced network environment utilized for testing that closely resembles a production (mainnet) environment. It is intended for exchanges, SPOs, pre-deployment DApps, and wallets that want to test release functionality before deployment on mainnet.

7.1.2 Permission Type

Different companies have different policies and use-cases regarding the blockchain, so they will have different approaches when selecting a suitable smart contract platform. As described in Chapter 4, we refer to a public and private blockchain. Furthermore, we explain the exact meaning, advantages, and disadvantages and elaborate on their potential usages to understand and clearly distinguish between the two types.

Private blockchains are permissioned networks and are often used in business settings where only a specific group of people or organizations are allowed to read and write to the blockchain. This group participates as validators or members. Organizations may opt for a private blockchain for several reasons:

- **Enhanced privacy**: Private blockchains offer enhanced privacy to participants, as transactions are not visible to the public

- **Customizability**: These types of chains can be customized to meet the specific needs and requirements of an organization
- **Improved security**: Since only authorized parties can access a private blockchain, it can be more secure than a public blockchain
- **Increased speed**: They can potentially be faster than public blockchains since they can prioritize transactions and do not have a need to incentivize their validators

Overall, private blockchains can be a good fit for organizations that need to maintain control over their data and transactions and do not need public blockchain's transparency and censorship resistance. This way, participants still utilize distributed system technology while maintaining and controlling other aspects of an enterprise.

However, private blockchains have limitations that need to be considered when choosing between public and private chains. Some limitations are listed and explained below:

- **Centralization**: These chains are typically owned and operated by a specific organization, which makes them more centralized in comparison with public blockchains
- **Limited scalability**: It can become hard to scale private blockchains since they have a smaller and limited number of operating nodes
- **Limited interoperability**: By being owned by a single organization, with a focus on privacy, it is hard for private chains to become interoperable and utilize blockchain technology's full potential
- **High costs**: Setting up and maintaining a private blockchain can come with high costs, as it requires a lot of resources and technical knowledge.

On the contrary, public blockchains have different advantages and challenges, offering a better utilization of blockchain technology with couple trade-offs regarding privacy and control.

Public blockchains are:

- **Highly available**: Public blockchains have many nodes and are distributed across a large network of computers, meaning they can continue to operate even if some nodes go offline
- **Decentralized**: Public blockchains are not controlled by a single entity or an organization, making them resistant to censorship and tampering
- **Transparent**: All transactions on a network are visible to everyone

- **Interoperable**: Being public and decentralized facilitates higher interoperability with other chains and fully taking advantage of blockchain technology
- **Accessible**: Public blockchains are accessible to everyone without additional costs making them easy to integrate

Some of the features of public blockchains might be a drawback for a particular use-case for the following reasons

- **Slow performance**: Public blockchains, as large as they are, might be less efficient and slower than private ones, as they require a larger number of nodes to reach consensus
- **Lack of privacy**: Being transparent and open to everyone does not offer the same level of privacy as private chains
- **Scalability**: They can struggle to handle a large number of transactions, which can lead to slow transaction times and high fees

These are only some of the issues one might encounter when using a public blockchain. Hence, by analyzing the benefits and drawbacks of public and private blockchains, we can conclude that there is no one-size-fits-all approach. Instead, it depends on the combination of functional and non-functional requirements.
Ethereum's main and test networks are public since they are accessible to anyone with an internet connection. However, there are possibilities to use Ethereum as a private network. One is the so-called "Consortium Network", where a pre-defined set of trusted nodes controls the consensus process. This approach, as specified above, has its benefits and drawbacks. Another approach presented by Ethereum is a Development Network, i.e., a private network where an application can be deployed before it is deployed to a public instance of blockchain.[16] Of course, such a network means creating a local blockchain instance, but this approach will be discussed in detail in the next section.

Neo also has publicly available main and test networks. In addition to the test network, Neo suggests testing an application on a local private chain. In this case, there are two options when setting up a private chain, i.e., setting it up with one node or multiple nodes. Setting up a private chain with the selected set of multiple trusted nodes, where strict rules for reaching a consensus can be applied, makes it possible to have a consortium network. [21]

In contrast to Neo and Ethereum, **EOS** does not offer any additional details concerning the organization of a blockchain structure for a closed network of stakeholders, specifically a private blockchain. Besides the information related to official test networks and the main network mentioned earlier, EOS provides a list of blockchain networks operating on EOSIO Software. These networks suit particular business use-cases and processes. Some of them are listed below.

- BOSCore - providing users with easy-to-access and easy-to-use blockchain services
- COFFE Network - EOSIO platform form
- eosiofinex - a scalable and transparent platform for digital asset trading

Due to its high level of configurability and flexibility, EOSIO software promotes the development of customized blockchain networks built on top of it rather than offering a range of deployment options. However, this might lead to higher effort and costs to achieve desired results.

Algorand offers two methods for creating a private blockchain. The first option involves establishing a network of nodes limited to select participants, such as a specific group of trusted individuals or organizations. These nodes can be customized to adhere to particular rules and consensus mechanisms based on specific requirements. Alternatively, Silvio Micali, Sergey Gorbunov, and Maurice Herlihy introduced the Co-Chain architecture. In their research paper, Micali and his colleagues argue that permissioned blockchains can risk isolating their users. As a result, enabling interoperability between public and private blockchains is beneficial for organizations. This approach allows them to maintain control over their sensitive data on a private blockchain while also securely engaging with the broader world through an interoperable public blockchain. [47] Below is the list what Algorand Co-Chains provide:

- Public chain independence
- Its own consensus algorithm
- Fine-grained access control
- Interoperation with Algorand main chain to communicate and transact with other co-chains
- Its own validators
- Private transactions

Cardano, sharing similarities with Ethereum, also offers the capability to establish a consortium network on its blockchain, allowing a private group of stakeholders to communicate and collaborate through the platform. To establish a consortium network, it is necessary to set up and configure Cardano nodes where each organization would need to run its own node. In addition, the consensus mechanism should also be defined to suit the specific needs of the consortium.

7.1.3 Local Blockchain Instance Setup

When developing a prototype of a decentralized application or experimenting with blockchain technology, organization or single parties could benefit from a secure, fast, and cost-effective way to explore the capabilities of technology. To make this possible, most smart contract platforms offer a local blockchain instance. In this way, an iterative way of building and testing decentralized applications becomes smoother as it decreases costs and time to interact with the blockchain network.
In **Ethereum**, local blockchain instances are also called development networks. So these are Ethereum clients explicitly designed for local development.[16] Benefits of running a local instance instead of running a standard node locally are:

- Local blockchain automatically creates and fills accounts with ETH that can be used for testing
- It provides enhanced debugging and logging functionality
- It processes transactions in order as they arrive, with no delay

There are couple of options to set up a local blockchain instance - some of them are listed below:

- Ganache - is Ethereum simulator that is used to run, test and execute commands
- Hardhat Network - comes with a built-in Ethereum environment for professionals
- Local Beacon Chains - there exists several Ethereum consensus clients that offer built-in tools for spinning up local beacon chain for testing purposes

Additionally, Ethereum offers a variety of tools and frameworks that contain a basic stack needed to get started, including a local blockchain instance. We will discuss these in detail in the future.
To develop and test use-cases we defined previously, we chose a framework called Truffle Suite. Truffle Suite comes with a built-in Ganache client and is one of the most popular frameworks for Ethereum development. There were no limitations or issues that could impede the implementation, and the setup went without any issues by following the official documentation provided by Truffle.

As discussed previously, **Neo** offers multiple ways to set up a local blockchain instance. These include setting up and running a private chain with one node or setting up and running a private chain with multiple nodes (consortium network). Since we are not building a consortium network and for our use-cases, one node is sufficient, we chose the first option. Furthermore, Neo offers multiple options when setting a private blockchain with one node. These are

1. Neo-CLI
2. neo-express

Neo-CLI is a command line interface developed by Neo and supports generating blocks without consensus nodes. It can also be used to build a private blockchain with one node from scratch, which is a more complicated process and require additional setups. On the other side, neo-express offers a much simpler process to run a private blockchain. It is a private network optimized for development scenarios and provides wallet and asset management, smart contract management, and blockchain instance management, among other features. [23] We chose the second option (neo-express) as it required less setup and was easier to get the blockchain up and running. In addition, even though the tool is not thoroughly documented, it offers clear instructions on how to start.

EOS defines two sets of instructions for setting up the local EOSIO development environment. The faster and easier method is a pre-configured development environment Quickstart Web IDE which uses Gitpod.io and Docker. The IDE provides a personal single-node EOSIO blockchain running in a docker container and is accessible from a web browser. On the other hand, there is a possibility to set up the environment from the ground up. First, it is necessary to run the local blockchain instance that runs *nodeos*[1]. Further steps include installing EOSIO binaries, EOSIO Contracts Development Kit (CDT), which acts as a compiler, creating wallets, starting keosd[2] and nodeos, and in the end, creating development accounts.

Algorand suggests using AlgoKit to start a local network and develop, test, and deploy smart contracts on the Algorand blockchain. AlgoKit LocalNet feature runs a sandboxed private Algorand network on a local machine and includes tasks such as starting, stopping, resetting, and managing the network. This capability enables modification and engagement with the personalized Algorand network without any concern of needing to finance TestNet accounts, the risk of the information provided being exposed to the public, or requiring an active internet connection after the network's initiation.

Cardano employs the term "local testnet" for a local blockchain instance. There are several options for creating a local test network. One approach involves using Plutip (a tool for private network creation), while another relies on Nix (a tool for package management and system configuration). Furthermore, Plutip offers multiple methods to facilitate the creation of a private network to run Plutus contracts. These methods are listed below:

- Utilizing the Plutip tool in a configuration that includes an executable for initiating a private network and establishing wallets with preloaded funds.

[1] *Nodeos* is the core EOSIO node deamon
[2] *Keosd* is a key manager deamon for storing private keys and signing digital messages

- Utilizing a different branch of Plutip, which provides a customized configuration for hard forks and directions on how to perform them.
- Executing Plutip alongside cardano-transaction-lib, which offers a declarative interface to local clusters for integration in test suites.

7.2 Official Documentation and Support

Choosing a technical stack is an important initial step in software development. Developers and architects need to select a combination of tools, languages, frameworks, and libraries that meet their specific project needs. Familiarity with programming languages is often a factor in the selection process. Choosing well-documented, up-to-date frameworks and libraries with active development is crucial, as it minimizes the learning curve, reduces time and costs, and ensures security updates. The selection of a smart-contract platform should not be any different. Therefore this section concentrates on what languages the specific platform supports and how well it is documented.

7.2.1 Language Support

As stated in **Ethereum's** official documentation, smart contracts in Ethereum can be written in developer-friendly languages. Specifically, developers that are familiar with Python or any curly-bracket language could easily pick up the two most active and maintained languages

1. **Solidity**
2. **Vyper**

Solidity is a high-level, object-oriented programming language. Although it uses ECMAScript-like syntax, it is statically typed (i.e., the variable type is/need to be known at compile time), unlike JavaScript, which is a dynamically typed language. Solidity offers a variety of complex features and concepts like the inheritance of other contracts, user-defined types, and the creation of custom libraries to be reused by other contracts.

On the other side, Vyper is a Python-like programming language with strong typing. Unlike Solidity, it has a limited set of features to provide an additional layer of security. Hence, Vyper does not support concepts among which are inheritance, modifiers, recursive calling, and function overloading.
Although Ethereum lists these two as the most popular and well-maintained programming languages, there are also several others for more experienced developers or those who want to help test new languages still under development. The following list shortly describes these languages and their purpose. [16]

- **Yul**

  - A low-level programming language
  - Intermediate language for Ethereum
  - Supports Ethereum Virtual Machine (EVM) and Ethereum flavored WebAssembly (Ewasm)
  - Allows more fine-grained control over the EVM's resources

- **Yul+**
  - Can be seen as an experimental upgrade to Yul with additional features
  - Purposefully designed for an optimistic rollup contract

- **FE**
  - The new programming language in its early stages, released in 2021
  - Statically typed language for EVM
  - Based on Python and Rust and aims to be easy to learn.

Similar to Ethereum Virtual Machine, **Neo** also utilizes Neo Virtual Machine (NeoVM) for executing smart contracts. According to Neo Documentation, the Neo Compiler allows source code written in Java or C# to be converted into a standardized NeoVM instruction set, enabling cross-platform compatibility. Furthermore, Neo's documentation briefly mentions other languages that can be used to develop smart contracts, like Go, Python, and JavaScript. While there is limited information available regarding the maintenance status of specific programming languages, it is known that the infrastructure for C# and Python in the current ecosystem is highly advanced, and developers have access to various compilers. [21]

As Jansen et al. observe in their publication, NEO has not created a new programming language for creating smart contracts. Instead, it focused on the integration of already established and widely-used programming languages. This approach aims to increase accessibility to the blockchain community by providing interfaces to the blockchain using languages that developers are already familiar with, therefore reducing the need for developers to learn new languages to interact with the technology. Furthermore, this approach helps to reduce the risks associated with smart contract development, as developers can use their usual IDE debugging tools and are less likely to introduce bugs that could affect the contract's behavior during runtime.[41]

EOSIO supports multiple programming languages, including C++ and WebAssembly (WASM). C++ is the primary language used to develop smart contracts for EOSIO and the language used to develop the EOSIO core software. WASM is a binary instruction format that is designed to be executed in a virtual machine. It is used to create portable, efficient, and secure code that can be run on various platforms, including EOSIO. With WASM, developers can write smart contracts in languages such as C++ and JavaScript, which can be compiled to WASM bytecode. That being said, EOSIO also offers Software Development Kits (SDK) for different programming languages, enabling the interaction with the EOSIO blockchain using Javascript, Java, and Swift.[14]

The Algorand Virtual Machine (AVM) is a central component of the **Algorand** blockchain, providing a virtual execution environment for processing transactions and executing smart contracts on the network. The Algorand Virtual Machine (AVM) interprets the Transaction Execution Approval Language (TEAL) to process transactions and execute smart contracts. TEAL is a low-level language, and it is not a general-purpose programming language like Python, JavaScript, or Java. Instead, TEAL is designed to be efficient, secure, and deterministic, focusing on providing predictable transaction execution on the Algorand blockchain. The official Algorand documentation states that, although it is possible to write smart contracts directly in TEAL, developers may prefer to write them using the PyTeal library in Python, which provides a more familiar syntax. [1] Algorand blockchain also provides several SDKs that enable external applications written in different programming languages to interact with the blockchain. These SDKs support various languages, including Java, Go, JavaScript, and Python.

While **Cardano** provides a variety of programming languages for developing smart contracts, numerous options are either tailored for particular purposes or still under development, making them unsuitable for production use. The two most commonly utilized languages in Cardano are Marlowe and Plutus. Marlowe is a non-Turing complete, domain-specific language (DSL) that enables users to develop decentralized applications primarily focusing on financial contracts. Marlowe can be used through JavaScript, Haskell, or Blockly, providing multiple options for developers. However, Marlowe is also one of the languages that are not advisable to be used on the mainnet, since its audit needs to be completed. Alternatively, Plutus is a general-purpose language for creating smart contracts. Based on Haskell, a functional programming language, Plutus inherits features like strong typing and mathematical rigor, ensuring security, reliability, and maintainability in smart contract development. Therefore, Plutus is suitable for developers familiar with Haskell or those who desire a high level of control and flexibility when creating smart contracts. Among other less popular languages, there are Aiken and opshin. Aiken is a new programming language for developing smart contracts based on modern languages such as Gleam, Rust, and Elm. It is specifically used for creating the on-chain validator scripts. As a result, any off-chain code for generating transactions can be written in other languages, such as JavaScript, Python, and more. Opshin, on the other hand, is a Python-compatible programming language and, therefore, suitable for Python developers.

	Main Language(s)	Support other languages	Other languages
Ethereum	Solidity, Vyper	Yes	Yul, Yul+, FE
Neo	C#, Python	Yes	Java, Go, JavaScript
Cardano	Marlowe, Plutus	Yes	Aiken, opshin
EOS	C++	Yes	Java, Swift, Javascript
Algorand	TEAL	Yes	Python

Table 7.1: Summary of supported programming languages on a specific platform

7.2.2 Documentation

In Chapter 4, we elaborated on why documentation plays an important role when choosing a smart contract platform. We also presented several aspects based on which we will assess how good the documentation is. These aspects include:

- Basic concepts (e.g., smart contract)
- Technology stack (e.g., what technologies are used)
- Basic development concepts (e.g., how-to develop, compile, test, deploy)
- Advanced topics (e.g., complex applications, use-cases)

Ethereum's documentation for developers provides a rich set of well-organized resources. It classifies its documentation into the following three categories

Foundational Topics gives a detailed overview of basic concepts starting from the introduction to blockchain, covering what Ethereum is, what Ether is, and slowly going deeper, explaining some of the terminology, including nodes, transactions, and accounts.

Ethereum Stack introduces us to the fundamental components of Ethereum that can help understand the different ways of how to integrate it in software projects. The Smart Contract section is divided into several subcategories, each covering specific elements of smart contract technology. The subcategories are of a technical nature and include

- Smart Contract languages - covers programming languages

- Smart Contract anatomy - covers variable types, data types, functions
- Libraries - covers already available libraries
- Testing - covers technical details of testing a smart contract
- Deploying - covers steps on how to deploy the contract
- Smart Contract security - covers a security aspect, access control and audits

These are just some smart contract subcategories we found relevant to our evaluation process. In addition, Ethereum Stack offers insights in different development networks and frameworks we discussed previously that can be used to integrate Ethereum into software development.

Advanced Topics part of Ethereum's documentation gives a comprehensive overview of different concepts that could be useful for more complex and sophisticated applications. For example, it discusses the need for bridges, their benefits, trade-offs, and security and risk aspects. It also elaborates on standards, which we will discuss later on. Among other topics, oracles and diverse scaling solutions and approaches like sharding, Layer2 scaling, rollups and side-chains are well documented.
As said in the beginning, Ethereum's documentation is a valuable, well-organized resource that could benefit developers of all skill levels. It is easy to follow and easy to understand, which could be the most critical aspect for developers who either develop on or integrate Ethereum in their software system.

Although founded around the same time, the difference between Ethereum's and **Neo's** documentation could not be neglected. Neo's documentation is classified into following categories: *Getting Started, Neo Basics, Neo Node, Development Guide, Reference and Support for Exchanges*

Getting Started immediately starts with an example of releasing a NEP17 asset on the Neo blockchain, setting up a local environment, compiling, deploying and invoking a contract on a local chain. Then, it gives step-by-step instructions on preparing a wallet and needed tools for creating a contract project without detailed explanation what the NEP17 standard represents, let alone smart contract as a concept. However, in the beginning, Neo's documentation provides Glossary, briefly describing the terms (smart contract, NEP17)

Neo Basics covers the basics of the Neo blockchain. These are further divided into four subcategories: *Main Concepts, Consensus, Governance and Incentives, NeoVM.* The first subcategory explains the theory behind the block, Neo's native tokens, different types of cryptography algorithms wallets and transactions. Consensus category gives a good theoretical background about Neo's consensus mechanism, the algorithm behind it and the system model.

Neo Node section offers insights on two full-node programs: Neo-CLI and Neo-GUI. It compares the two ways of running a full-node, explains them in detail and provide instructions on setting them up and running.

Development Guide starts with types of Neo networks. We already discussed the Main and Test networks Neo offers. In addition, it also gives guidance on writing, deploying, and invoking smart contracts. However, Neo does not explain a concept of a smart contract, but it immediately dives into contract property, storage property, and technical details, including data types in C#, events and triggers. In these sections, Neo explains its two token standards, NEP-11 and NEP-17.

Reference and **Support for Exchanges** sections provide API definitions and guidance on integrating a Neo node with an exchange.
Concerning advanced topics, Neo only describes its Neo Oracle Service without explaining such concepts in general.

The EOSIO Developer Portal serves as a documentation platform for **EOSIO**, offering a diverse selection of technical product resources that are designed and maintained by the EOSIO blockchain developer community. The introductory section gives an overview of the platform and toolchain's fundamental concepts, including a detailed explanation of a command line interface, a key management system, and the core EOSIO node daemon. Subsequently, the core concepts that comprise the EOSIO ecosystem, such as accounts, wallets, and permissions, are elaborated on. This is followed by a discussion of technical features, including the stacking mechanism, WebAssembly C++ compilation, and upgradability. The term "Smart Contract" is shortly described as part of core concepts.

The Getting Started Guide focuses on configuring the local environment and creating and deploying a "Hello World" smart contract. In contrast, the **Smart Contract Guides** section provides a more extensive explanation of smart contracts and introduces the concept of tokens. It also provides specific instructions for deploying, issuing, and transferring tokens. Furthermore, the section discusses crucial smart contract security concepts, such as secure coding practices, smart contract auditing, and managing vulnerabilities and exploits. Additionally, the Smart Contract Guides section covers certain EOSIO-specific smart contract concepts, including inline action and secondary indices.

While the platform does not offer many examples on writing smart contracts using C++, it does have a specific section **Software Manuals** that provides sample applications for various programming language SDKs.
Beside providing a couple of tutorials for different games, that might represent a more complex use-case, EOSIO does not provide any documentation on more advanced topics.

Similar to the previously mentioned platforms, **Algorand** has a dedicated developer portal that provides structured and organized documentation on various topics. The documentation is divided into several sections, each covering a specific topic.

The **Get Started** section provides theoretical background followed by practical examples on tokenization, including fungible and non-fungible tokens. In addition, the section provides information on blockchain basics, such as what a blockchain is, what a dApp is, and why Algorand is a good choice for developing blockchain-based applications.

Within the **Get Details** section of Algorand's developer portal, several subsections focus on specific concepts related to the Algorand blockchain. These subsections provide detailed explanations of topics such as accounts, transactions, and smart contracts. More specifically, the subsection "Smart Contracts" contains details on how to start writing a smart contracts, build them and deploy on the blockchain network. Additionally, these subsections also include more advanced concepts for experienced developers, e.g., account rekeying, offline transaction signatures, atomic transfers. Other sections that might could be beneficial include SDKs for various languages, REST APIs, CLI tools and guidelines on how to run a node.

The documentation provided by Algorand is extensive and provides a wealth of detailed technical instructions. However, due to its vast scope, it is not feasible to comprehensively describe every aspect. Therefore, this research paper will focus solely on the essential elements of the documentation pertinent to developers beginning to work with the Algorand platform.

The documentation for **Cardano** ecosystem is extensive and rich. Unlike some previous platforms, Cardano distinguishes between ecosystem documentation and a portal for developers. The official documentation for the ecosystem covers a theoretical background related to the Cardano blockchain. In contrast, the developer portal supports developers in integrating the Cardano blockchain by offering more in-depth technical explanations of concepts and components.

Similar to Algorand, **New to Cardano** section introduces us to the basic terms such as blockchain, cryptocurrency and explain why Cardano is a good choice. It also explains what is a smart contract and gives an introduction to the available tools to develop smart contracts. The rest of the documentation covers different topics specific to the Cardano chain, e.g., Cardano architecture, components, sidechains, etc. Since our emphasis in this paper is on the more technically-oriented aspects, we transitioned to the developer's portal.

Again, in the **Create Smart Contract** section, an overview of smart contracts is provided, and the available programming languages are listed. Each subsection contains more detailed explanations of a specific language, i.e., it gives instructions on developing, deploying, and testing smart contracts. However, these subsections provide many external resources and lack the structure to progress easier.

As of the **Advanced Topics**, the developer portal covers various aspects such as "Security Best Practices" and "Build with Transaction Metadata." Additionally, it features a segment on operating stake pools. However, the developer's portal does not cover more advanced topics from a practical standpoint, although they are covered in the ecosystem documentation (e.g., sidechains).

7.3 Extended Documentation

The following types of unofficial documentation might be of greater significance than official documentation for developers developing on a particular platform. This could be a strong indication of the level of commitment from the team supporting the platform, as providing such information could results in increased platform usage and reliability.

7.3.1 Community

Many platforms have dedicated community sections on their websites, showcasing various ways to get involved and upcoming events or to ask questions and contribute to the platform. Some of the reasons why community is so important for particular platforms are listed below:

- Adoption: A strong community can drive adoption
- Ideas and Feedback: A community can generate new ideas and provide feedback, thus help to improve the platform
- Development: A large community can lead to more development of a platform making it more valuable
- Support: A thriving community can provide support to developers and users of the platform, helping them overcome potential technical challenges

Ethereum offers multiple ways to get involved. For example, they offer online communities, different Ethereum events, and various ways to contribute to a project or seek grants. In this section, we focus on the online community and examine its activity level and interaction modes.
Ethereum community is divided into three categories

1. Forums

   Various Reddit forums with over 4 million members are available to users and developers. Some of them are listed below

- **r/ethereum** (Main Ethereum subreddit, 1.6m builders)
- **r/ethfinance** (Community for Ethereum investors, 86.1K members)
- **r/ethdev** (Focused on Ethereum development, 99K members)
- **r/ethtrader** (Trends and Market analysis, 2.3m members)

2. Chat Rooms

   Ethereum also has multiple discord chat rooms for different purposes. They are counting over 70 thousand members in total.

   - **Ethereum Cat Herders** (Community offering project management support to Ethereum development)
   - **Ethereum Hackers** (Community run by ETHGlobal)
   - **CryptoDevs** (Development focused community)
   - **EthStaker** Discord
   - **Solidity Gitter** (Chat for Solidity development)

3. Youtube and Twitter

   - Ethereum Foundation (Youtube, 62.2K subscribers)
   - Twitter (Official account of the Ethereum foundation, 3m followers)
   - Twitter (The portal to Ethereum, global community account, 31.5K followers)

To analyze the responsiveness of the community, we asked a use-case specific question that we had. Since Ethereum has its own Stackexchange, we decided to ask our question there. It is important to mention that our profile did not have any reputation, so we took into account that we might have a limited reach. However, the question was answered after one day and the discussion continued until we solved our problem.

Neo has multiple means to interact with their community. They have several social media accounts with a large number of followers dedicated to content posting, i.e., news, monthly reports, and announcements. Currently, for this purpose, they are using

- Twitter (425.6K followers)
- Facebook (29K followers)
- Medium (1.7K followers)

Furthermore, they also have different channels where the same kind of posts are possible, but they primarily serve as a two-way communication channel between the community and the team. These include

- Reddit (∼117K members)
- Telegram (∼8.3K members)
- Discord (∼8K members)

In addition to the official community channels presented on the Neo page, we also searched for different terms and tags on one of the most popular sites for developers - Stackoverflow [3] to even better try to gauge the usage of the platform.
We present you the terms we searched for and the number of results retrieved. The term in square brackets refers to a tag ([]), whereas the word refers to a keyword.

- [Neo] - 0 results
- Neo [smartcontracts] - 7 results
- Neo [blockchain] - 14 results

Again, we posted a specific question related to our task during the implementation of one of the use-cases. Since Discord had the most online members at the moment, we decided to ask our question on this platform. The question referred to implementing an NFT contract from the official documentation page, where we asked someone to confirm if the contract is valid and up-to-date. After several days of no response, we reached out to one of the group members, who had been asking similar questions, and resolved the issue through private discussion.

EOSIO leverages many different channels to interact with its community of developers, enthusiasts, entrepreneurs. Some of the official channels are represented by following mediums:

- Youtube (∼13.5K subscribers)
- Twitter (∼254K followers)

Regarding channels where the aforementioned audience can communicate, EOSIO has multiple Telegram channels for different purposes. So with that, they are running:

- EOS General Chat (∼10.4K members)
- Meetup Channels:
  - EOS New York
  - EOS London

[3] `https://stackoverflow.com/` - Last accessed: 30/01/2023

  - EOS Nairobi
  - EOS Hong Kong
  - EOS Nation
- EOS Developers
  - EOS Index
  - EOS Game Developers
  - ...

However, it is important to state that EOSIO has many channels listed, but not all are up-to-date, and many have expired. On the other side, EOS Network Foundation also provides several means of communication:

- Youtube (∼4.8K subscribers)
- Twitter (∼27.8K followers)
- Discord (∼12.6K members)
- Telegram (∼2.6K members)

In line with the previously mentioned platforms, **Algorand** also provides a comparable range of communication channels. These are listed below:

- Youtube (∼16.1K subscribers)
- Twitter (∼333.1K followers)
- Facebook (∼130K followers)

Communication channels:

- Discord (∼37K members)
- Forum (unknown number of users)
- Telegram (∼20.5K members)
- Reddit (∼70.9K members)

It is crucial to recognize that Algorand is supported by a dynamic and highly responsive community that readily extends help and assistance for a diverse range of inquiries and concerns. As a result, there was no necessity to seek solutions for issues on external platforms such as StackOverflow or StackExchange.

Cardano has a massive community with numerous groups of people speaking different languages (e.g., China Cardano Telegram, South Korea Telegram, and more) who are interested in the project. However, this paper focuses on official community channels and those dedicated to developers. Below are the official channels associated with the Cardano Foundation:

- Telegram (∼65.8K members)
- Facebook (∼230K followers)
- Youtube (∼25.9K subscribers)
- Twitter (∼1.3M followers)
- Reddit (∼690K members)
- Discord (∼10.2K members)

The channels listed above serve the entire community, while the following channels are designed to unite the developer community and provide support:

- IOG Technical Discord (∼11.9K members)
- Telegram (∼1.8K members)
- Developer Portal Discord (∼1K members)
- Reddit (∼13.3K members)

Moreover, the Cardano Foundation maintains a developer forum, though the precise membership count remains undisclosed. It also supports Cardano StackExchange, which features approximately 2,600 questions. For comparison, Ethereum StackExchange comprises over 50,000 questions.

7.3.2 Exemplary Contracts

As mentioned in one of the previous sections, **Ethereum** offers a deep dive into smart contract topics, providing essential information and resources that serve as a guide when developing smart contracts. One of the resources that provide exemplary contracts is OpenZeppelin. OpenZeppelin is a company that develops and maintains a library for secure smart contract development. Their GitHub page states they are "built on a solid foundation of community-vetted code." [4]

[4] `https://github.com/OpenZeppelin/openzeppelin-contracts`

Their smart contract examples and implementations can be categorized as follows:

- Access Control - provides contract module for implementing ownership in smart contracts
- Tokens - different kinds of token implementations (e.g., ERC-20, ERC-721)
- Utilities - generic useful tools (e.g., Cryptography, Math, Payment)

Neo has a dedicated section called "Examples and Tutorials", authored by their global developer community. However, not many examples are available. As they support multiple programming languages, many NEP-17 contracts examples are implemented using these languages (Python, C#, Go, Java).[22] They also cover a couple of other topics like

- Hello World Python Smart Contract
- Token Sale Python Smart Contract
- Wrapped NEO Python Smart Contract
- Using Oracles in Smart Contracts

Developers can find reference smart contracts for common use-cases in **EOSIO's** section "For Developers", leading to a GitHub repository where some of the contracts are stored. As stated in their GitHub README file: "The design of the EOSIO blockchain calls for a number of smart contracts that are run at a privileged permission level in order to support functions such as block producer registration and voting, token staking for CPU and network bandwidth, RAM purchasing, multi-sig, etc. These smart contracts are referred to as the bios, system, msig, wrap (formerly known as sudo) and token contracts." [11]
Although these contracts represent helpful contracts for more complex use-case implementations, these are not exemplary contracts for implementing certain concepts or utilizing some of the EOSIO functionality (e.g., storage, indices, etc.).

The **Algorand** platform includes official examples in various programming languages as part of explanation of a particular topic, such as Atomic Transfers or Algorand Standard Assets. The section "Writing Smart Contracts" covers PyTeal and the Beaker framework. PyTeal official documentation page [5] provides tutorials for several aspects, e.g.:

- Atomic Swap
- Split Payment

[5] https://pyteal.readthedocs.io/en/latest/examples.html

- Periodic Payment
- Voting
- Asset
- Security Token

In addition, the Beaker framework [6] also offers its own set of illustrative examples showcasing the implementation of diverse components. Some of these examples cover:

- Boxes
- Account Storage
- States
- Structures

The documentation of **Cardano** related to programming languages is divided into sections. Each section represents a language and includes several examples of contracts in that specific language. For instance, as part of its "Playground", Marlowe offers pre-defined examples written in Javascript, Haskell, Marlowe, and Blocky. As Marlove is designed for financial contracts, these examples are based on simple financial use-cases, such as:

- Escrow
- Escrow with Collateral
- Zero Coupon Bond
- Swap

Additionally, Marlowe offers subsection "Examples" with links to different GitHub resources containing examples of contracts. However, the official documentation for Plutus provides only one example - "Plutus Script for an auction smart contract" - along with several links to "How-to" guides and additional GitHub tutorials.

[6] `https://github.com/algorand-devrel/beaker/tree/master/examples`

7.3.3 Educational Material and Tutorials

Often, official documentation is insufficient to grasp concepts and understand how more complex use-cases could be implemented on some platforms. Therefore, educational materials and additional tutorials, where users publicly publish their code base or example of a particular topic (e.g., crowdfunding application) can immensely help developers.

For this purpose, **Ethereum** community has written and collected numerous tutorials on various topics, like different standards ERC-20, ERC-721, DAO, Uniswap, Chainlink, and many more. Tutorials can be filtered based on tags. We listed some of the most popular tags below, including some of the tutorials:

- Solidity - contains 53 tutorials, e.g.:
  - Decentralized Staking App
  - Aave Flash Loan Tutorial
  - Create your own ERC-20 Blockchain Token
  - Solidity and Truffle CI setup
- Security - contains 7 tutorials, e.g.:
  - A guide to smart contract security tools
  - Smart contract security checklist
  - How to use Slither to find smart contract bugs
- Some other popular tags include: Alchemy, Ethersjs, ERC-20, ERC-721, Testing

Apart from collected tutorials on the official Ethereum website, the Ethereum community's presence on both EthereumStackExchange and GitHub makes it easy for beginners and experienced developers to access numerous tutorials and educational resources that provide reliable assistance.[17]

Neo also provides a couple of tutorials on their official website besides the aforementioned examples. The official list of tutorials includes the following:

- Getting familiar with NEO-CLI commands
- Learn how to write a NEP-17 contract
- The basics of Neo's native Oracle

Nevertheless, Neo also features a multitude of tutorials that encompass an array of topics, which are included as part of the Neo Global Development GitHub repository. It is worth noting, however, that this particular repository appears to function as an augmentation to the official documentation, rather than as a standalone tutorial source. In its current state, this repository comprehensively covers the following areas:

- Introduction to Neo
- Wallet
- Transactions
- Blocks
- Network
- Persistence
- Consensus
- NVM
- Smart Contract

Despite its number of tutorial resources, online repositories containing concrete use-case implementations, building blocks, and real-world examples for Neo are comparatively sparse.

As we said in the previous section, there is a lack of practical examples and tutorials on how to implement and utilize some of the basic concepts that are described in the documentation (e.g., key-value map). Nevertheless, **EOSIO** offers couple of Webinars as learning resources. Following webinars date from 2020:

- Build a Full-stack Web Application Using EOSIO
- Build Your First Smart Contract on EOSIO
- Learn about Blockchain & EOSIO

Furthermore, there are also official training and certifications offered by EOSIO in Spanish and English. Although all courses are free, one must enroll to participate. These courses include Smart Contracts 101, Smart Contracts 201, Blockchain Foundations, EOSIO Blockchain Security, and many more. In addition to the official training and certifications, EOSIO has a couple of tutorials on its official website. There are four tutorials in total, and they cover the following topics:

- Blockchain setup, loading system contracts, and adding multiple block producers
- Running a tic-tac-toe game on the blockchain using single node blockchain and testnet; in general, building, deploying, and using smart contract
- Building a game-alike EOSIO full-stack application

There is a large number of tutorials on **Algorand's** page that can be filtered by languages, tools, levels and time to complete. While some tutorials are authored by the Algorand team, the majority are contributed by the community of developers. These tutorials offer step-by-step guides but focus on various specific topics, including but not limited to:

- Deploying a bet dapp powered by Beaker
- Crowdfunding Dapp with Reach
- Run Algorand Indexer on Azure with Azure CLI
- Sending Rewards to ICO Investors Using Batch Transactions in Python

A different category of tutorials, called Solutions, provides code repositories for various implementations based on community contributions. Some of them include the following:

- Build an employee loan arrangement dApp
- Artificial Intelligence on Algorand
- Minting NFTs on Algorand using IPFS
- Algorand QR Code Generator
- and many more ...

It is worth mentioning that anyone can contribute by creating tutorials or providing a code repository.

As noted in the previous subsection, **Cardano** offers a few tutorials within its subsections pertaining to different programming languages. Plutus provides several courses that can be completed such as:

- Plutus Project-Based Learning - a hands-on course from Gimbalabs
- Plutus Pioneer Program - core principles of Haskell and Plutus

Furthermore, it offers some basic tutorials like "Writing basic validator scripts", "Writing basic minting policies", "Property-based testing of Plutus contracts", and similar. Unfortunately, while there are some official tutorials on Cardano's documentation page, the available documentation, implementation examples, and unofficial tutorials are scarce and difficult to find.

7.4 Platform Development

7.4.1 Tools

Tools play a critical role in programming smart contracts due to their ability to simplify and streamline the development process. Smart contracts are inherently complex, and programming them can be a daunting task for even the most experienced developers. Utilizing appropriate tools, such as integrated development environments (IDEs), testing frameworks, and deployment platforms, can significantly enhance a developer's efficiency, accuracy, and overall productivity. By leveraging these tools, developers can more easily create, test, and deploy their smart contracts, which ultimately helps to ensure that the contracts are secure, reliable, and functional. Additionally, utilizing the right tools can minimize the risk of errors, reduce development time, and improve the overall quality of the code.

When setting up a local environment and developing on Ethereum, **Ethereum's** frameworks and pre-made stacks are recommended starting points. This is because they cover a lot of out-of-the-box functionality like spinning up a local blockchain instance, utilities to compile and test smart contracts, or integration with storage options like IPFS, among others. In addition, many of these frameworks offer a complete development environment like Truffle, Hardhat, and Embark. These are all focused on Javascript/Typescript and Solidity. However, there are also some that offer support and environments for Java Virtual Machine (Epirus), Rust (Foundry), and Python (Brownie).

Neo also has several pre-made stacks for developers. Their dedicated section, "Tooling," categorized these tools into Nodes, Explorer, Smart Contract Compilers, Smart Contract Dev and Debug, SDK, and dApp Integration. As we primarily focus on smart contract compiling, development, and debugging, we shall explore these two categories. Neo has furnished various compilers to enable support for multiple programming languages. NeoGo is a comprehensive and well-documented alternative implementation of the Neo stack written in Go. Additionally, the Neo-devpack-dotnet provides support for C#, whereas Boa is designed specifically for Python developers. Moreover, the Neow3j compiler facilitates the conversion of Java contracts into NEF and manifest files, enabling their deployment to the N3 network.
For smart contract developing and deploying Neo has multiple out-of-the-box solutions. Neo Blockchain Toolkit is a tool integrated with Visual Studio Code where developers can easily deploy private networks, compile, deploy and invoke smart contracts. Another similar tool is NEO-ONE - an end-to-end development framework for Neo applications created with TypeScript or JavaScript.
On the contrary, the Neo Compiler Eco platform empowers developers to code and compile smart contracts through a web interface, eliminating the need to install other development tools. Furthermore, the platform facilitates the deployment of the compiled contracts to a TestNet. Designed to streamline the onboarding process for developers, the Neo Playground platform offers an effortless means to spawn temporary development

environments for a variety of supported languages, including Java, C#, and Go. This platform enables developers to write, deploy, and test smart contracts hassle-free.

As highlighted in a previous section, **EOSIO** has developed a Quickstart Web IDE, which streamlines the process for developers to begin building on the EOSIO platform. This tool simplifies the initial setup and configuration, allowing developers to dive into the development process quickly. Similarly, EOS Network Foundation promotes DUNE (Docker Utilities for Node Execution), a client tool designed to aid blockchain developers and node operators in performing essential tasks related to smart contract development and node management. [10]
Other tools serve different purposes, e.g., eosio-explorer provides Web GUI to communicate with EOSIO blockchain in a local development environment; eosio-helm streamlines the deployment process of EOSIO nodes via Kubernetes through Docker Desktop for local deployments or via different cloud providers (Amazon Web Service, Google Cloud Platform, etc.) for cloud deployments; demux-js, a backend infrastructure pattern for sourcing blockchain events; history-tools is a suite of performance-optimized search tools, coded in C++, enabling efficient and scalable data sifting through terabytes of information from the complete history of EOSIO blockchain; eosio-toppings represents a monorepo with various packages making a web-based tool for easier application creation on EOSIO blockchain. Furthermore, the Community Developer Tools section lists a couple of development, testing, and deploying frameworks for EOSIO, like Azakazam, Lamington, EOSFactory, Hydra, and Tank.

As a robust and mature blockchain platform, **Algorand** offers a wide array of tools that simplify the development of decentralized applications (dApps). On their dedicated page for tools and ecosystems, it is possible to choose among tags, some of which are *smart contracts, IDEs, teal, wallets, frameworks.* Nevertheless, with a plethora of tools available in the Algorand ecosystem, it would be impractical to list them all. Therefore, for the purpose of this paper, our focus will be on tools that streamline the development process, including those for smart contract development, testing, debugging, and deployment, similar to the platforms mentioned earlier. Two of the leading frameworks, as they appear in the *Get Started* page for developing Algo applications, are:

1. AlgoKit - the primary tool used by the Algorand community to develop, test, and deploy Algorand smart contracts.

2. Beaker - a frameowrk for building smart contracts using PyTeal. It is also designed to make developing, testing and deploying easier.

Other useful tools include two Algorand IDEs - algoDEA and Algodesk.io and some smart contract utilities like PyTeal (Python language), Reach (Javascript-like language), Tealang (C-like high-level language).In addition to the tools mentioned, it's worth noting that Algorand also provides a range of software development kits (SDKs), including

.NET, Dart, PHP, Rust, Swift, Vertices, and Unity, that enable seamless integration with the Algorand blockchain.

Additional information and tools in the ecosystem can be found on the official Algorand page. [7]

In addition to explaining its programming languages and their specific applications, **Cardano** also offers a selection of valuable tools to assist developers in beginning their development journey. For Marlowe, the language for financial contracts, there are several tools listed on the official Cardano documentation page such as:

- Marlowe Playground - an online environment for writing, simulating, and analyzing Marlowe contracts. It supports both the textual (Haskell and JavaScript) and visual (Blockly) representations of Marlowe code.
- Marlowe Runtime - an application backend for managing contracts.
- Marlowe CLI - a command-line interface for running contracts.
- Marlowe Lambda - a Marlowe Runtime client for AWS Lambda services

Similarly, for Plutus developers, there is a tool called Demeter. Demeter is an online development platform that offers a comprehensive set of tools for constructing Cardano dApps from start to finish. It includes starter kits for various project types, such as Plutus.

Furthermore, Cardano features a specialized section known as "Builder Tools," which offers a variety of tools that, while not directly associated with smart contract development, are still valuable resources. These include wallet SDKs, bridge interfaces, and libraries. Some of the tools are Aiken, Blockfrost, cardanocli-js, cardano-wallet-js, Ogmios, and many more. [5]

7.4.2 Standards

Standards play important role in smart contract platforms for variety of reasons. They help to ensure interoperability between different smart contract platforms, allowing them to seamlessly communicate and interact with one another. They also promote security and reliability by establishing best practices for development. Furthermore, by specifying guidelines for contract design, code quality and security measures, the risk of vulnerabilities and exploits is reduced.

With the same objective in mind, the **Ethereum** community has implemented a range of standards to ensure the composability of smart contracts and dapps.n general, the

[7]`https://developer.algorand.org/ecosystem-projects/`

introduction of standards within the Ethereum community follows a standardized process, wherein proposals for these standards are presented as Ethereum Improvement Proposals (EIPs). These EIPs are then subject to discussion and analysis by members of the community. [8] These standards are divided into following categories:

1. Standard Track: refers to any modification that has an impact on the majority, if not all, of the Ethereum implementations. It is also further divided into 4 categories: *Core, Networking, Interface, ERC*
2. Meta Track: processes surrounding Ethereum
3. Informational Track: pertains to an Ethereum-related matter that requires attention or offers general advice or insights to the Ethereum community.

Moreover, Ethereum defines several token standards, some of which are:

- ERC-20 - a standard interface for fungible (interchangeable) tokens, like voting tokens, staking tokens or virtual currencies.
- ERC-721 - a standard interface for non-fungible tokens. Additional standards like ERC-2309, ERC-4400, and ERC-4907 containing some more specific functions with regard to non-fungible token have also been developed.
- ERC-1155 - a token standard which can contain both fungible and non-fungible assets.
- ERC-4626 - a tokenized vault standard with aim to optimize the technical parameters of yield-bearing results. [16]

On the contrary, **Neo** does not explicitly lists its standards. The only two standards mentioned on the Neo's official website are NEP-17 and NEP-11, i.e., a standard for fungible and non-fungible token, respectively. NEO Enhancement Proposals (NEPs) outline guidelines for the Neo platform that cover various aspects such as core protocol specifications, client APIs, and contract standards. Their GitHub profile provides a repository that contains both active and accepted proposals for various standards. The accepted proposals include:

- NEP-2 - Passphrase-protected private key
- NEP-6 - Wallet Standard
- NEP-9 - URI Scheme
- NEP-14 - NeoContract ABI

[8] https://ethereum.org/en/eips/

- NEP-15 - NeoContract Manifest
- NEP-16 - NEO Executable Format (NEF)
- NEP-18 - Neo Address Resolution
- NEP-19 - Debug Info Specifications
- NEP-20 - Authentication Scheme
- NEP-21 - Dapi for N3
- NEP-22 - Contract Basic Method Guideline

In the same section mentioned earlier, the Community Developer Tools, **EOSIO** provides several standards for different purposes. These are listed and briefly explained below:

- dGoods - a freely available and open-source protocol for managing virtual representations of both digital and physical items
- Marble - a lightweight, highly customizable digital item standard
- Atomic Assets - RAM efficiency and usability focused NFT standard
- Simple Assets - a standard for digital assets on EOSIO, whether fungible, non-fungible, or non-transferable.

Algorand provides a built-in standard for different kinds of assets called Algorand Standard Asset (ASA). ASAs are a type of digital asset that can be created, issued, and transferred on the Algorand blockchain. They are tokens representing ownership of a digital or physical asset, such as cryptocurrencies, real estate, intellectual property, or even loyalty points. Furthermore, they are designed to be highly scalable, fast, and secure, leveraging the Algorand blockchain's features, including its pure proof-of-stake consensus algorithm, fast transaction finality, and low transaction fees. They can be used for various use-cases and customized to meet specific requirements. ASAs have different properties, such as fungible or non-fungible, and can be configured with various parameters, including total supply, decimals, and freezing, which allow for flexible tokenization and management of digital assets. ASAs also support standard Algorand transaction features, such as atomic transfers, which enable complex and secure multi-party transactions. [1]

Like NEO, **Cardano** does not explicitly specify standards on its documentation page except for Cardano Improvement Proposals (CIPs). Cardano Improvement Proposals are formal documents proposing changes, improvements, or new features for the Cardano ecosystem. CIPs help create a standardized process for community-driven development and decision-making. Nevertheless, upon reviewing CIPs, numerous standards have already been put into effect, while several others remain open for discussion. Therefore, we have Wallet Key Generation Standard, BECH32 address format, which enhances error detection and ensures better human readability, URI scheme, and more.

CHAPTER 8

Discussion

In Chapter 6, we collected findings regarding the technical implementation and evaluated them based on our comparison metric. In the previous chapter, we examined the attributes of selected platforms using the criteria outlined in the established catalog. This chapter integrates the insights from both chapters and expands upon the strengths and weaknesses of the platforms. In addition, we clarify the distinctions between the platforms and discuss which aspects of the catalog hold relevance for developers. We consolidate our findings for each platform, including reflections and experiences from the implementation of different use-cases.

8.1 Key characteristics, strengths and weaknesses

Ethereum is a mature platform with a large and active developer community. Its maturity may be its most considerable strength, considering the number of decentralized applications developed and deployed on it covering a variety of use-cases. Ethereum's Turing-complete programming language, Solidity, allows developers to create complex applications encompassing custom logic within the smart contracts. By having multiple test networks and an easy-to-set-up local blockchain instance, Ethereum enables extensive testing opportunities before any application is deployed to production. On the official page, Ethereum offers numerous development and testing tools and frameworks. Furthermore, it also provides many tutorials and code samples categorized in different groups. Despite the learning curve associated with new programming languages such as Solidity and Vyper, numerous official and unofficial resources are available to aid in quickly understanding it. For fungible and non-fungible token use-cases, Ethereum was an easy-to-use platform, as many examples and clearly defined ERC standards are provided as mentioned before. However, some platforms required less knowledge and made the process more straightforward. On the contrary, Ethereum has a more complicated structure and multiple configurations that one needs to know, such as migrations configurations, deployment scripts, and similar, which may be challenging for new developers. Additionally, the main disadvantage of the platform is its high fees, i.e., a direct relationship between the price of the native token and the level of network usage.

Neo is a smart contract platform with short transaction times and low transaction fees. Unlike Ethereum, Neo utilizes a dual token system consisting of NEO and GAS tokens. The GAS token serves as the payment method for transaction fees and smart contract execution costs. Decoupling these costs from the NEO token helps ensure a stable value for NEO, as its price is not directly influenced by network usage and vice versa. By offering support for widely-used programming languages such as C#, Java, and Python, Neo eliminates the need for additional learning efforts, potentially reducing costs when initiating a new project. Another strength of Neo worth considering when choosing this platform is the ability to update a contract without changing its transaction hash. Speaking of documentation, although Neo provides a developer portal where documentation and tutorials can be found, only basic examples are available. The implementation of token use-cases did not require much effort due to the clear explanations of the standards. However, finding more advanced examples was challenging as they are scarce in both the official documentation and online resources. One of the technical limitations is that developers cannot utilize the full range of C# features due to the disparity between the Neo Virtual Machine (NeoVM) and the .NET Intermediate Language (IL). To ensure compatibility and security within the Neo blockchain ecosystem, the Neo development team has created a limited version of the C# programming language called NeoContract, which is tailored for smart contract development. NeoContract restricts certain features of the C# language and provides its own set of rules, conventions, and optimizations. Furthermore, it should be noted that the community was unresponsive when inquiries were made.

EOS is designed to provide low latency and to handle a high volume of transactions by utilizing a delegated proof-of-stake (DPoS) consensus mechanism. This enables faster transaction processing and higher throughput compared to some other blockchain platforms. Furthermore, in technical terms, EOS offers developers the ability to explicitly manage CPU and RAM resources, providing a high level of efficiency in resource management and performance optimization and thus ensuring predictable and consistent performance for smart contracts. EOS supports C++ as its primary language for smart contract development, which is a widely adopted and powerful language known for its performance, efficiency, and extensive developer community. The developer portal provides a good theoretical background, however, it lacks practical examples. During the process of searching and exploring EOS documentation, it was unclear whether EOS.IO is still actively maintained and if EOS itself is being actively developed. As a result, it may be advisable to follow the official page of the EOS Network Foundation for the most up-to-date information. Moreover, developing selected use-cases required the highest effort, as there was no official standard for non-fungible tokens, but rather several implementations from multiple companies. Additionally, challenges arose during the development of the supply chain management solution. For instance, certain features were inaccessible during testing on a local network and required additional activation. This information was only revealed when attempting to initiate a genesis node. Another aspect worth noting is that, despite the use of C++, EOS does not provide access to all standard data structures. This means that adapting a solution to work with the available resources in EOS may require additional time and effort.

Cardano is another smart contract platform utilizing Proof-of-Stake (PoS) consensus mechanism, called Ouroboros. This algorithm provides security and scalability while minimizing energy consumption. Considering the token use-cases Cardano differs from other mentioned platforms in the way that tokens are not built with smart contracts but are rather considered *native assets* and live on the ledger. In Cardano, native assets are generated through Cardano-CLI by defining policies and various parameters. While this token generation approach may initially appear more straightforward compared to creating smart contracts, it required significant effort to create tokens, particularly non-fungible ones. This was primarily due to a lack of structure and detailed explanations in the Cardano documentation. Moreover, since several parts of the process require manual entry, there are numerous stages during implementation that are susceptible to errors. On the other side, Cardano offers several languages to write smart contracts. One of Cardano's languages, Marlowe, is a domain-specific language designed specifically for financial contracts. It is not Turing-complete, making it less suitable for complex use-cases beyond the financial domain. However, it offers distinct advantages such as enhanced security, certainty, termination guarantees, and behavior correctness. Its second language, Plutus, is based on the functional programming language Haskell. This association reduces the complexity of smart contracts and enhances their readability. Nevertheless, it should be noted that if developers are not already familiar with Haskell, it may require additional effort and incur higher training costs during the learning period. Due to the same reason, we could not fully implement our third use-case, supply chain, in a given time frame. Overall, the main disadvantage of the Cardano platform is the utilization of specific languages and approaches, for which there is not enough resources or community support outside of the Cardano developer portal.

Algorand is a blockchain platform with several key advantages. It utilizes a Pure Proof-of-Stake (PPoS) consensus mechanism, ensuring decentralization, security, and scalability. By employing PPoS, Algorand achieves fast block confirmation times, enabling a high volume of transactions per second, resulting in high throughput. The platform's commitment to supporting developers is evident through its provision of a well-structured and extensive range of resources. From the theoretical perspective, Algorand offers detailed documentation, whitepapers, and research materials explaining the basics and more advanced concepts. In addition to theoretical content, Algorand offers many practical examples and use cases. Developers can access sample code, tutorials, and hands-on guides that demonstrate how to implement various features and functionalities on the Algorand blockchain. Furthermore, Algorand sets an excellent example of how official and unofficial documentation should complement each other, which stands as one

of the platform's most significant advantages. While some details may not be thoroughly explained in the official documentation, the Algorand community demonstrates high responsiveness in its communication channels such as Discord and the Algorand Developer Forum. As a result, any doubts or issues encountered by developers can be resolved within a matter of hours.
The smart contract language for Algorand, Python, along with the PyTeal library, may initially appear challenging due to the absence of standard data structures. However, the Beaker Framework offers a straightforward and user-friendly entry point for developers, making the development process simpler and more accessible. Despite the need for an adjustment and learning period, the learning curve during the implementation process was gradual, primarily thanks to the factors mentioned earlier. These factors contribute to making Algorand a highly competitive platform, positioning it as one of the top choices after Ethereum.

#	Platform	Advantages	Disadvantages
1	Ethereum	mature, large community, rich and structured documentation	seems complex and challenging for new developers, new language, high fees
2	Algorand	detailed documentation, responsive community, transaction finality	limited language support, lack of common data structures
3	Neo	dual token system, language support, low transaction fees, ability to update a contract	limited documentation, unresponsive community, C# limitations
4	Cardano	tokens as native assets, multiple languages, scalability	lack of structured documentation, new languages, resource scarcity, small community
5	EOS	high transactions throughput, flexibility through C++	no NFT standard, no support for Windows OS, scarce documentation, lack of common data structures

Table 8.1: Ranking of Smart Contract Platforms: Advantages and Disadvantages

The table above summarizes the advantages and disadvantages of various platforms that can be considered during the decision-making process. The platforms listed in the table are also ranked based on our experience during the implementation process, with the first being the most comprehensive and the last being the most challenging.

8.2 Implications for Developers

Although the previous table provides a ranking, it's clear that there is no definitive 'best' platform. As with any field, selecting one option out of many involves making trade-offs. Indeed, the concept of a 'best' platform is highly dependent on specific factors that vary significantly from one scenario to another.

Even though the longevity of a platform may seem like a safe choice, our analysis has shown that this is not always the case. However, one characteristic that can definitely be marked as critically important for a platform is its adoption, evidenced by high development activity. Furthermore, having high development activity suggests an active, responsive community that can offer substantial support. Such a diverse and dynamic environment increases the likelihood that many different use-cases have been implemented, or that common issues, such as those related to syntax, standards, and compilers, have already been addressed and solved. An example of such an environment is Algorand. Although founded in 2019, it surpasses many more mature platforms like Neo, Eos or Cardano. However, it's worth noting that all the factors in our catalogue are interrelated. For example, a platform might experience high development activity because it supports widely used languages like Java, C#, or Python, thereby appealing to a broader audience.

Other factors that may influence the choice of platforms can be grouped into three categories:

1. **Requirements of the use-case**: The characteristics of the project at hand play a critical role in guiding the platform choice. Certain platforms may be more suitable for particular types of projects. For example, if the project is small-scale or represents a proof-of-concept for fungible or non-fungible tokens, it might be more appealing to choose Algorand or Cardano. These platforms enable the creation of tokens as standard assets, thereby avoiding the additional effort of writing smart contracts. On the other hand, if the project is constrained by a limited amount of hardware resources or requires additional optimization, then EOS or Neo might be good choices. Furthermore, Ethereum offers multiple test networks, a substantial set of development tools, and various standards that could be helpful in defining, developing, and simulating more complex real-world use-cases.
2. **Team Competence**: The team's familiarity with and knowledge of the different platforms can profoundly impact the selection process. Developers with extensive experience in a particular programming language may favor platforms that support

that language. For instance, Neo, which supports multiple languages including Java and C#, may be a more appealing choice to developers proficient in these languages. Furthermore, a team with previous experience on a specific platform may prefer to continue using it due to their understanding of its features and best practices.

3. **Time constraints**: The availability of time is a crucial factor in the platform selection process. Some platforms, with comprehensive documentation, robust development tools, and supportive communities, can significantly shorten the learning curve and speed up the development process. On the other hand, platforms with steep learning curves might offer greater long-term benefits but could be unsuitable for projects with tight deadlines.

8.3 Limitations and Future Work

This research paper examines three specific use-cases: fungible tokens, non-fungible tokens, and supply chain management. As blockchain technology continues to evolve, it proves increasingly applicable and beneficial across a range of sectors, such as real estate, financial services, and identity verification, to name a few. These sectors present a variety of use-cases that fall outside the scope of our implementation. Moreover, our study focused on five specific smart contract platforms. However, as mentioned before, the field of blockchain technology is vast and continually expanding, with many other platforms available that were not included in our research. Some of these alternatives could be more suitable for a particular use-case, offering unique features, benefits, or capabilities that align more closely with the specific requirements of a given application. As such, while our research provides valuable insights into Algorand, Ethereum, Neo, EOS, and Cardano capabilities, it's essential to recognize that these findings may not necessarily generalize to other platforms or scenarios.
Furthermore, we restricted our investigation to a developer's perspective. While this focus allows for an in-depth analysis of these platforms' technical and community aspects, it does not provide a holistic view that includes end-user experiences or considers broader economic and regulatory impacts.

Future research could extend its scope to consider a broader range of use-cases and include diverse blockchain platforms. This approach would provide a more comprehensive understanding of the strengths and limitations inherent in various smart contract environments.

For example, Artha et al. explore the implementation of smart contracts for e-certificates as non-fungible tokens using the Solana network [29]. They argue that Solana is attracting new users to the Web 3.0 ecosystem by reducing transaction costs and accelerating transaction speeds. On the other hand, with the rise of cross-chain interoperability solutions, studying the possibility of implementing these use-cases across multiple platforms could be considered. Morháč et al. suggest a system for fungible asset sharing across blockchains, explicitly designed to enable asset transfers among the distinct Parachains within the Polkadot paraverse ecosystem. [51]

CHAPTER 9

Conclusion

The primary objective of this thesis is to compare various smart contract platforms, with a particular focus on the development aspect. These platforms include Ethereum, Neo, Cardano, Algorand, and EOS. The aim is to highlight their strengths and weaknesses and to elaborate on their differences.

To enhance our understanding of the fundamental distinctions in their core protocols and properties, we conducted a brief review of the technical components of the selected platforms with the help of a technical catalogue of criteria.

Furthermore, to facilitate a structured comparison, we defined a new catalogue of criteria with a focus on development aspects. Additionally, we identified three use-cases to implement on each platform and established corresponding implementation metrics that are applied to these use-cases. In doing so, we contributed to the comparison from both theoretical and practical perspective. Our work confirms that it is challenging to select one platform that is suitable for all solutions; instead, there are trade-offs depending on various requirements.

In Chapter 8, we examined the differences between the platforms, evaluating their strengths and weaknesses. Moreover, we provided a ranking table based on their advantages and disadvantages and considered implications for developers. To shortly summarize our general findings: Ethereum is the most mature platform with a large community and high development activity. Neo offers fast transaction times and multi-language support, while Cardano is designed for enhanced security and scalability. Algorand emerged as a significant alternative to Ethereum due to its simplicity and active community, and EOS benefits from high throughputs and affords greater flexibility through its usage of the C++ language.

Our evaluation of three use-cases revealed that the least amount of code and time was required for Ethereum, whereas EOS proved to be the most demanding, largely due to poor documentation and a less engaged community. While Cardano demonstrated simplicity in token implementations, the third use-case could not be accomplished within the allotted 30-hour time frame. Overcoming challenges was most straightforward with Ethereum and Algorand, largely attributed to their responsive communities and comprehensive documentation, while EOS, along with Cardano, again proved to be the most difficult. However, the least experience was needed to implement use-cases with Algorand, and to a certain degree with Cardano, specifically for token use-cases.

As mentioned in previous sections, the longstanding existence of a platform might appear to be a reliable choice, but we have seen that it isn't always so. Our research paper highlights the critical importance of a platform's adoption, signifying the role of an active and responsive community in providing significant support. To conclude our thesis, we posit that the selection of the platform is primarily influenced by a range of use-case requirements (whether the use-case is a novel application or a standard solution), team competence (as experienced teams require less support and can tackle more complex applications), and time constraints (such as project duration and go-live deadlines).

APPENDIX A

Implementation notes

A.1 Ethereum

Question asked on the Ethereum StackExchange: "NFT implementation of OpenZeppelin ERC721 ownerOf always returns "ERC721: owner query for nonexistent token";

Positive remarks:

- Ethereum provides a most structured way of programming with a clear documentation, examples and community
- Easy to implement and follow any examplary use-case
- Plenty of threads on different forums and definitely the most widespread platform
- Multiple test networks for different purposes

Negative remarks / Problems encountered:

- More complicated structure
- Solidity as a new language
- Multiple configurations
- Example: What is "migrations.ts", etc.

A.2 Algorand

Blockchain consensus: Pure Proof of Stake (solving the blockchain trilemma – not having to sacrifice scalability, security nor decentralization)

Question asked on Discord: "Is it possible to fund a generated account in code from a sandbox account list (./sandbox goal account list) and if yes, how?"

Positive remarks:

- Algorand Developer Forum
  - well organized/structured
  - get to choose categories, tags, latest or hot topics
  - active community
  - multiple topics answered one day ago (at that time)
- Plenty of tutorials
- Most serious project after Ethereum, offering constant support
- Every problem we came across there was an answer on Algorand's developer forum
- Didn't know what can be used instead of maps -> we got an answer on discord almost instantly where to look and which examples might be helpful

Negative remarks / Problems encountered:

- Lack of data structure to manipulate the date
- No maps, structs, objects
- It takes time to adjust thought process
- Not able to find in the official documentation:
  - How to fund an account from private network account
  - How to get the mnemonic and create a secret key from a private network account

A.3 Neo

Question asked on Discord: "Could not implement NFT contract from the official documentation page – asked multiple questions on discord if someone can confirm if the contract is valid or not. After several days reached out to one of the group members asking similar question and discussed the problem privately."

Positive remarks:

- Usage of C#
- Possibility to use multiple widespread languages

Negative remarks / Problems encountered:

- Not very well documented
- No quality resources for help, only basic ones available
- Not standard C# can be used, due to the difference between NeoVM and .NET IL
- Supply chain management: Not that many examples with more complicated contracts available
- Discord invite link expired
- No guidelines on how to deal with exceptions
- Unresponsive community

A.4 Cardano

Positive remarks:

- Cardano supports creation of fungible and non-fungible token without using smart contracts, but rather native tokens

Negative remarks / Problems encountered:

- Cardano node installation guide for Windows still in progress
- Setting up the node was the most time-consuming setup
- Can't fund two accounts without getting too many requests error
- Need to wait 24h to be able to fund another account

- A complex process of minting native tokens
- Too many variables to handle (many places prone to error)
- No explanations of parameters (–change address=???)
- Marlowe: hard to work with strings
- Plutus:
  - Hard to understand
  - Lack of resources and tutorials
  - Lack of support

A.5 EOS

Question asked on StackExchange: "Cannot deploy contract containing KV Map API"

Negative remarks / Problems encountered:

- Only supports Linux and MacOS operating systems
- Bad error message logs (cannot tell what exactly is wrong)
- No official NFT standard
- Contract code/variables/members are retained on every action call
- No state, variables need to be saved into a map or a table
- Lack of data structures
- Documentation poorly written
- Some features were not available (need to be imported but that is only mentioned when starting a genesis node)
- No community support
- No adoption
- GitHub repository archived
- Hard to find syntax rules
- Not everything is written in a C++ manner (functions declarations)

List of Figures

List of Tables

Bibliography

[1] Algorand developer docs. `https://developer.algorand.org/docs/`. Accessed: 2023-04-15.

[2] Algorand governance documentation. `https://developer.algorand.org/docs/get-started/basics/why_algorand/`. Accessed: 2023-06-25.

[3] Algorand sets a new benchmark in performance. `https://blockzeit.com/faster-than-ever-algorands-protocol-upgrade-sets-a-new-benchmark-in-performa`
Accessed: 2023-01-06.

[4] Algorand state proofs. `https://algorand.com/resources/algorand-announcements/powering-blockchain-interoperability-and-post-quantum`
Accessed: 2023-06-27.

[5] Cardano developer docs. `https://docs.cardano.org/cardano-testnet/getting-started`. Accessed: 2023-04-29.

[6] Cardano governance docs. `https://cardano.org/governance/`. Accessed: 2023-06-25.

[7] Earn eos while learning about the eosio protocol and eos. `https://www.coinbase.com/blog/earn-eos-while-learning-about-the-eosio-protocol-and-eos`. Accessed: 2023-04-07.

[8] Eos governance documentation. `https://eosnetwork.com/governance/`. Accessed: 2023-06-25.

[9] Eos inter-blockchain communication. `https://ibc-docs.uxnetwork.io/overview`. Accessed: 2023-06-27.

[10] Eos network documentation. `https://docs.eosnetwork.com/docs/latest/smart-contracts/getting-started/dune-guide/`. Accessed: 2023-04-01.

[11] Eos reference smart contracts. `https://github.com/EOSIO/eosio.contracts`. Accessed: 2023-03-29.

[12] Eos whitepaper. https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md. Accessed: 2023-07-01.

[13] Eosio documentation. https://developers.eos.io/manuals/eos/v2.0/nodeos/usage/development-environment/index. Accessed: 2023-03-07.

[14] Eosio official documentation. https://developers.eos.io/welcome/latest/manuals/index. Accessed: 2023-03-27.

[15] Eosio public blockchain documentation. https://eos.io/eos-public-blockchain/. Accessed: 2023-03-07.

[16] Ethereum development documentation. https://ethereum.org/en/developers/docs/. Accessed: 2023-01-06.

[17] Ethereum development tutorials. https://ethereum.org/en/developers/tutorials/. Accessed: 2023-02-24.

[18] Ethereum whitepaper. https://ethereum.org/en/whitepaper/. Accessed: 2023-06-24.

[19] Implementing hydra heads: the first step towards the full hydra vision. https://iohk.io/en/blog/posts/2022/02/03/implementing-hydra-heads-the-first-step-towards-the-full-hydra-vision/. Accessed: 2023-07-02.

[20] Jump into web3 with algorand. https://www.algorand.foundation/news/jump-into-web3-with-algorand. Accessed: 2023-07-01.

[21] Neo development documentation. https://docs.neo.org/docs/en-us/develop/network/testnet.html. Accessed: 2023-01-06.

[22] Neo development examples. https://neo.org/dev#examples. Accessed: 2023-02-23.

[23] Neo-express documentation. https://github.com/neo-project/neo-express. Accessed: 2023-01-16.

[24] Neo governance docs. https://neo.org/gov. Accessed: 2023-04-29.

[25] Neo whitepaper. https://docs.neo.org/v2/docs/en-us/basic/whitepaper.html. Accessed: 2023-01-06.

[26] Indian government set to ban cryptocurrencies, Nov 2021. URL: https://www.bbc.com/news/technology-59402310.

[27] Aftab Ahmed and Nupur Anand. India to propose cryptocurrency ban, penalising miners, traders - source, Mar 2021. URL: https://www.reuters.com/article/uk-india-cryptocurrency-ban-idUSKBN2B60QP.

[28] Darcy WE Allen and Chris Berg. Blockchain governance: What we can learn from the economics of corporate governance. *Allen, DWE and Berg, C (Forthcoming)'Blockchain Governance: What can we Learn from the Economics of Corporate Governance*, 2020.

[29] Kevin Aditya Ramadhan Artha, Sarah Noor Zain, Azhar Ashhafa Alkautsar, and Mochammad Haldi Widianto. Implementation of smart contracts for e-certificate as non-fungible token using solana network. In *2022 IEEE 7th International Conference on Information Technology and Digital Applications (ICITDA)*, pages 1–6, 2022. doi:10.1109/ICITDA55840.2022.9971423.

[30] Marco Bareis. *Comparison of Ethereum and NEO as smart contract platforms*. PhD thesis, Wien, 2019.

[31] Marco Bareis, Monika Di Angelo, and Gernot Salzer. Functional differences of Neo and Ethereum as smart contract platforms. In *2nd Int. Congress on Blockchain and Applications (ICBA)*. Springer, 2020. doi:10.1007/978-3-030-52535-4_2.

[32] Shuchih E. Chang and Yichian Chen. When blockchain meets supply chain: A systematic literature review on current development and potential applications. *IEEE Access*, 8:62478–62494, 2020. doi:10.1109/ACCESS.2020.2983601.

[33] D.L. Chaum. *Computer Systems Established, Maintained and Trusted by Mutually Suspicious Groups*. University of California, Berkeley, 1979.

[34] Yan Chen. Blockchain tokens and the potential democratization of entrepreneurship and innovation. *Business Horizons*, 61(4):567–575, 2018. URL: https://www.sciencedirect.com/science/article/pii/S0007681318300375, doi:https://doi.org/10.1016/j.bushor.2018.03.006.

[35] Raeesah Chohan and Jeannette Paschen. Nft marketing: How marketers can use non-fungible tokens in their campaigns. *Business Horizons*, 2021. URL: https://www.sciencedirect.com/science/article/pii/S0007681321002202, doi:https://doi.org/10.1016/j.bushor.2021.12.004.

[36] Fabrizio Dabbene, Paolo Gay, and Cristina Tortia. Traceability issues in food supply chain management: A review. *Biosystems Engineering*, 120:65–80, 2014. Operations Management in Bio-production Systems. URL: https://www.sciencedirect.com/science/article/pii/S1537511013001554, doi:https://doi.org/10.1016/j.biosystemseng.2013.09.006.

[37] Iman Dernayka and Ali Chehab. Blockchain development platforms: Performance comparison. In *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–6, 2021. doi:10.1109/NTMS49979.2021.9432669.

[38] Lisia S. Dias and Marianthi G. Ierapetritou. From process control to supply chain management: An overview of integrated decision making strategies. *Computers and Chemical Engineering*, 106:826–835, 2017. ESCAPE-26. URL: `https://www.sciencedirect.com/science/article/pii/S0098135417300625`, `doi:https://doi.org/10.1016/j.compchemeng.2017.02.006`.

[39] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[40] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68, New York, NY, USA, 2017. Association for Computing Machinery. `doi:10.1145/3132747.3132757`.

[41] Marc Jansen, Farouk Hdhili, Ramy Gouiaa, and Ziyaad Qasem. Do smart contract languages need to be turing complete? In Javier Prieto, Ashok Kumar Das, Stefano Ferretti, António Pinto, and Juan Manuel Corchado, editors, *Blockchain and Applications*, pages 19–26, Cham, 2020. Springer International Publishing.

[42] Yongpu Jia, Changqiao Xu, Zhonghui Wu, Zichen Feng, Yaxin Chen, and Shujie Yang. Measuring decentralization in emerging public blockchains. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pages 137–141, 2022. `doi:10.1109/IWCMC55113.2022.9825341`.

[43] T. Koens and E. Poll. Assessing interoperability solutions for distributed ledgers. *Pervasive and Mobile Computing*, 59:101079, 2019. URL: `https://www.sciencedirect.com/science/article/pii/S1574119218306266`, `doi:https://doi.org/10.1016/j.pmcj.2019.101079`.

[44] Roy Lai and David LEE Kuo Chuen. Chapter 7 - blockchain – from public to private. In David Lee Kuo Chuen and Robert Deng, editors, *Handbook of Blockchain, Digital Finance, and Inclusion, Volume 2*, pages 145–177. Academic Press, 2018. URL: `https://www.sciencedirect.com/science/article/pii/B9780128122822000073`, `doi:https://doi.org/10.1016/B978-0-12-812282-2.00007-3`.

[45] R Massey, D Dalal, and A Dakshinamoorthy. Initial coin offering: A new paradigm. *https://www2.deloitte.com/us/en/pages/consulting/articles/initial-coinoffering-a-new-paradigm. html*, 2017.

[46] Manuel Mazzara. An overview and current status of blockchains performance.

[47] Silvio Micali. Algorand co-chains, 2020.

[48] Francesco Mogavero, Ivan Visconti, Andrea Vitaletti, and Marco Zecchini. The blockchain quadrilemma: When also computational effectiveness matters. In *2021 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2021. `doi:10.1109/ISCC53001.2021.9631511`.

[49] Alaa Hamid Mohammed, Alaa Amjed Abdulateef, and Ihsan Amjad Abdulateef. Hyperledger, ethereum and blockchain technology: A short overview. In *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–6, 2021. `doi:10.1109/HORA52670.2021.9461294`.

[50] Monika and Rajesh Bhatia. Interoperability solutions for blockchain. In *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, pages 381–385, 2020. `doi:10.1109/ICSTCEE49637.2020.9277054`.

[51] Dušan Morháč, Viktor Valaštín, and Kristián Koštál. Sharing fungible assets across polkadot paraverse. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–7, 2022. `doi:10.1109/ICECET55527.2022.9872938`.

[52] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[53] Henrik Niss and Sebastian Owuya. *Performance and Scalability of Emerging Blockchain Technologies*. PhD thesis, 2022. URL: `http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-320189`.

[54] Person and Samuel Shen Alun John. China's top regulators ban crypto trading and mining, sending bitcoin tumbling, Sep 2021. URL: `https://www.reuters.com/world/china/china-central-bank-vows-crackdown-cryptocurrency-trading-2021-09-24/`.

[55] Abderahman Rejeb, John G. Keogh, and Horst Treiblmaier. Leveraging the internet of things and blockchain technology in supply chain management. *Future Internet*, 11(7), 2019. URL: `https://www.mdpi.com/1999-5903/11/7/161`, `doi:10.3390/fi11070161`.

[56] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis, and Lejia Shen. Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research*, 57(7):2117–2135, 2019. `arXiv:https://doi.org/10.1080/00207543.2018.1533261`, `doi:10.1080/00207543.2018.1533261`.

[57] Wanshui Song, Wenyin Zhang, Linbo Zhai, Luanqi Liu, Jiuru Wang, Shanyun Huang, and Bei Li. Eos. io blockchain data analysis. *The Journal of Supercomputing*, pages 1–32, 2021.

[58] Wanshui Song, Wenyin Zhang, Linbo Zhai, Luanqi Liu, Jiuru Wang, Shanyun Huang, and Bei Li. Eos.io blockchain data analysis. *The Journal of Supercomputing*, 78(4):5974–6005, 2021. doi:10.1007/s11227-021-04090-y.

[59] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2), 1996.

[60] Martin Valenta and Philipp Sandner. Comparison of ethereum, hyperledger fabric and corda. *ebook] Frankfurt School, Blockchain Center*, 2017.

[61] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[62] Brent Xu, Dhruv Luthra, Zak Cole, and Nate Blakely. Eos: An architectural, performance, and economic analysis. *Retrieved June*, 11:2019, 2018.

[63] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 47(10):2084–2106, 2021. doi:10.1109/TSE.2019.2942301.