Università degli Studi di Cagliari

DOTTORATO DI RICERCA

Matematica e Informatica

Ciclo XXXIV

SOFTWARE ENGINEERING PRACTICES APPLIED TO BLOCKCHAIN TECHNOLOGY AND DECENTRALIZED APPLICATIONS

Settore scientifico-disciplinare di afferenza

INF/01 - INFORMATICA

Presentata da:	Lodovica Marchesi
Tutor:	Professore Roberto Tonelli
Co-Tutor:	Dottor Giuseppe Destefanis

Esame finale anno accademico 2020/2021

Tesi discussa nella sessione d'esame aprile 2022

Università degli Studi di Cagliari

Ph.D. DEGREE

Mathematics and Computer Science

Cycle XXXIV

SOFTWARE ENGINEERING PRACTICES APPLIED TO BLOCKCHAIN TECHNOLOGY AND DECENTRALIZED APPLICATIONS

Scientific Disciplinary Sector

INF/01 – COMPUTER SCIENCE

Ph.D Student:	Lodovica Marchesi
Supervisor:	Professore Roberto Tonelli
Co-supervisor:	Dottor Giuseppe Destefanis

Final exam. Academic Year 2020/2021

Thesis defence: April 2022 Session

Author’s email.: lodovica.marchesi@unica.it

Author’s address:

Dipartimento di Matematica e Informatica
Università degli Studi di Cagliari
Via Porcel, 4
09123 Cagliari
Italia

Acknowledgments - Ringraziamenti

Mio padre una volta mi ha detto che essere un ricercatore è come essere un esploratore. Sempre in cerca di nuovi territori da scoprire, analizzare e sperimentare. Alla fine di questo percorso, sono soddisfatta ed entusiasta ma anche determinata a continuare a fare ricerca, ad essere un'esploratrice.

Quindi inizio col ringraziare te papà, senza il quale non sarei quella che sono oggi sotto tanti punti di vista, la tua passione per la conoscenza mi stimola ogni giorno. Sei la mia fonte inesauribile di insegnamenti, consigli e supporto.

Grazie a te Roberto, per avermi sostenuta sin da prima del Dottorato, e per tutti i preziosi consigli. Seguirti nelle varie sfide scientifiche e progetti di questi anni è stato sfidante ed arricchente.

Grazie a Giuseppe, per la tua ospitalità, gentilezza e lungimiranza umana ed accademica. Mi hai dato e continui a dare fiducia, e questo mi sprona a crescere e migliorare.

Un grazie speciale a Nicola, con cui ho condiviso questo percorso e a voi colleghi, Ilaria, Marco, Andrea, Gavina, Katiuscia, nonostante la pandemia ci abbia tenuti lontani dall'ufficio per la maggior parte del tempo, siete sempre pronti a condividere e confrontarsi. Se questi anni sono stati un'esperienza positiva lo devo a tutti voi.

Grazie alla mia Famiglia, per la vostra continua presenza nonostante la lontananza fisica, per ciò che mi avete insegnato e continuate ad insegnarmi, per l'amore che mi date e lo stimolo a rendervi orgogliosi di me.

Grazie a Simone, che ogni giorno mi sopporti, per aver condiviso in prima linea ogni successo e delusione di questo percorso. La tua determinazione e attitudine al duro lavoro sono per me fonte d'ispirazione.

Grazie alle mie amiche, che pur non essendo molto avvezze di tecnologie mi avete sempre seguita ed ascoltata, e per tutti i momenti insieme, siete preziose.

I would also like to thank the Brunel Software Engineering Lab (BSEL) Group for hosting me at their department and helping me with my research during my period as visiting academic guest at the Brunel University of London, UK.

Finally, I would like to express my gratitude to everyone that helped me during the writing of this thesis and the years of my PhD. In particular, I thank Prof. Steve Counsell of the Computer Science Department of the Brunel University of London (UK) and Prof. Mohamad Kassab of the Engineering Division of the Pennsylvania State University (USA) for their helpful reviews of this thesis.

Sommario

List of Figures

List of Tables

List of pubblications

In International Journal

PEERJ CS – “Forecasting Bitcoin closing price series using linear regression and neural networks models”, N. Uras, L. Marchesi, M. Marchesi, R. Tonelli. In PeerJ Computer Science, Volume 6, pp. e279, PeerJ Inc., 2020.

BRCA – “Abcde–agile block chain dapp engineering”, L. Marchesi, M. Marchesi, R. Tonelli. In Blockchain: Research and Applications, Volume 1, pp. 100002, Elsevier, 2020.

IEEE Access – “Assessing the Risk of Software Development in Agile Methodologies Using Simulation”, Maria Ilaria Lunesu, Roberto Tonelli, Lodovica Marchesi, Michele Marchesi. In IEEE Access, Volume 9, pp. 134240-134258, IEEE, 2021.

In International Conferences

“The ICO phenomenon and its relationships with ethereum smart contract environment”, G. Fenu, L. Marchesi, M. Marchesi, R. Tonelli. In Proceeding of the 2018 International Workshop on Blockchain Oriented Software Engineering, pp. 26-32, IEEE, 2018.

“An agile software engineering method to design blockchain applications”, M. Marchesi, L. Marchesi and R. Tonelli. In Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia, pp. 1-8, 2018.

“Design patterns for gas optimization in Ethereum”, L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, D. Tigano. In Proceeding of the 2020 International Workshop on Blockchain Oriented Software Engineering, pp. 9-15, IEEE, 2020.

“Can the Blockchain Facilitate the Development of an Interport Community?”, P. Serra, G. Fancello, R. Tonelli, L. Marchesi. In proceeding of the International Conference on Computational Science and Its Applications, pp. 240-251, Springer Cham, 2021.

“Automatic Generation of Blockchain Agri-food Traceability Systems”, L. Marchesi, K. Mannaro, R. Porcu. In Proceeding of the 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2021, pp. 41–48, 9474795, 2021.

“Design Patterns for Smart Contract in Ethereum”, G. Destefanis, L. Marchesi. Published in 2021 IEEE Second Workshop on Blockchain-based Architectures (BlockArch), keynote speech, 2021.

“Software Engineering Practices applied to Blockchain Technology and Decentralized Applications”, L. Marchesi. In proceeding of the Third Workshop on Blockchain-based Architectures (BlockArch), keynote speech, 2022.

Chapter in Book

Chapter 12 - Blockchain technologies and IoT in What Every Engineer Should Know About the Internet of Things, Taylor & Francis, 2021.

Submitted papers:

“A Blockchain Architecture for Industrial Applications”, Lodovica Marchesi, Michele Marchesi, Roberto Tonelli, Maria Ilaria Lunesu. Submitted to Blockchain: Research and Applications, Elsevier, 2021.

“Automatic generation of Ethereum-based Smart Contracts for Agri-Food Traceability System”, L. Marchesi, K. Mannaro, M. Marchesi and R. Tonelli. Submitted to IEEE Access, 2022.

“Application prospects of Blockchain technology to support the development of interport communities”, P. Serra, G. Fancello, R. Tonelli and L. Marchesi. Submitted to Computers, MDPI, 2022.

Arxiv:

“Security checklists for Ethereum smart contract development: patterns and best practices”, L. Marchesi, M. Marchesi, L. Pompianu, R. Tonelli. arXiv preprint arXiv:2008.04761

Conferences

Attended the 27th IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER2020), February 19-21 2020, London, Ontario, Canada.

Presented "Design Patterns for Gas Optimization in Ethereum" at 2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), February 18 2020, London, Ontario, Canada.

Presented "ABCDE - Agile BlockChain dApp Engineering" in "Blockchain Permissioned, DEFI e loro applicazioni" Seminar of DMI, February 24 2021, University of Cagliari, Cagliari, Italy.

Presented "Automatic Generation of Blockchain Agri-food Traceability Systems" in 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), May 2021, online.

Attended the 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), May 17 2021 to June 4 2021, online.

Attended and student volunteer at 22nd International Conference on Agile Software Development (XP2021), June 14-18 2021, online.

Presented “Can the Blockchain Facilitate the Development of an Interport Community?” at the 21st International Conference on Computational Science and Its Applications (ICCSA), September 16 2021, online.

Invited as Keynote Speecher to present “Software Engineering Practices applied to Blockchain Technology and Decentralized Applications” at the Third Workshop on Blockchain-based Architectures (BlockArch), March 12 2022, online.

School and Seminaries:

LASER 2019 Summer School in “Artificial Intelligence, Machine Learning and Software Engineering”, June 1-9 2019, Isola d’Elba, Italy.

BSEL 2019 Summer School in "Empirical Software Engineering", June 25-28 2019, Brunel University, London, UK.

SILICIO: la storia e il futuro della tecnologia fra Robot e Esseri Viventi, Federico Faggin, October 2 2019, University of Cagliari, Cagliari, Italy.

CRIPTOVALUTE E BLOCKCHAIN: VERSO LE APPLICAZIONI REALI, October 2 2019, University of Cagliari, Cagliari, Italy.

COMPLEXITY AND TOKEN ECONOMY, February 12 2020, University College London, London, UK.

Python for Machine Learning Research course, M. Marras, December 2020, online.

Professional Agile Leadership - Essentials Training (PAL-E), Scrum.org, December 16-17 2021, Paris, France.

0 Thesis Introduction

A few years after the introduction of Bitcoin in 2009, developers and businessmen realized that a blockchain can be also used to run a decentralized computer. Ever since blockchain software development is becoming more and more important for any modern software developer and IT startup.

Due to the advantages that the implementation of such a system could provide, there has been a huge development in the field of blockchain technology applied to various sectors. Many companies and startups are already adopting, and working on blockchain technology, trying to exploit the many advantages it promises, so we are experiencing a strong growth of ideas and applications. Nonetheless, blockchain software production still lacks a disciplined, organized, and mature development process, as demonstrated by the many and (in)famous failures and frauds that occurred in recent years. The goal of my research is to study innovative software engineering techniques applicable to the development of blockchain applications.

I decided to focus on the use of agile practices because these are suitable for developing systems whose requirements are not fully understood from the beginning, or tend to change, as is the case with most blockchain-based applications.

In particular, I contributed to the proposal of a method to guide software development, called ABCDE, meaning Agile BlockChain Dapp Engineering. The method takes into account the substantial difference between developing traditional software and developing smart contracts and separates the two activities. It considers the software integration among the blockchain components - smart contracts, libraries, data structures - and the off-chain components, such as web or mobile applications, which altogether constitute a complete dApp system. The proposed method also addresses specific activities for both security assessment and gas optimization, two of the main issues of dApp development, through the systematic use of patterns and checklists.

Agile methodologies aim to reduce software development risk using short iterations, feature-driven development, continuous integration, testing automation, and other practices. However, the risk of project failure or time and budget overruns is still a relevant problem. Always with a view to developing innovative software engineering techniques, I studied and developed a new approach to model some key risk factors in agile development, using software process simulation modeling (SPSM). The approach includes modeling the agile process, gathering data from the tool used for project management, and performing Monte Carlo simulations of the process, to get insights into the expected time and effort to complete the project, and their distributions. While the simulator hasn't been specifically applied to blockchain projects yet, it has all the features to be able to do so; this will in fact be one of the next objectives of my research.

In the context of the study of blockchain applications, I also proposed an evaluation framework to compare public and permissioned blockchains, specifically suited for industrial applications. Then, I presented a complete solution based on Ethereum to implement a decentralized application, putting together in an original way components and patterns already used and proved. This solution is characterized by a set of validator nodes running the blockchain using Proof-of-Authority or similar efficient consensus algorithms, by the use of an Explorer enabling users to check the blockchain state, and the source code of the Smart Contracts running on it. From time to time, the hash digest of the last mined block is written into a public blockchain to guarantee immutability. The right to send transactions is granted by validator nodes to users by endowing them with the Ethers mined locally. Overall, the proposed approach has the same transparency and immutability of a public blockchain, largely reducing its drawbacks.

The key reason to use a blockchain is trust. There is a growing demand for transparency across the agri-food supply chain from customers and governments. The adoption of blockchain technology to enable secure traceability for the agri-food supply chain management, provide information such as the provenance of a food product, and prevent food fraud, is rapidly emerging, due to the inherent trust and inalterability provided by this technology. However, developing correct smart contracts for these use cases is still more of a challenge than it is for those executed in other fields. Numerous agri-food supply chain management systems based on blockchain technology and smart contracts have been proposed, all however ad-hoc for a specific product or production process and difficult to generalize. For this reason, my research also focused on defining a novel approach for easily customizing and composing general Ethereum-based smart contracts designed for the agri-food industrial domain, to be able to reuse the code and modules and automate the process to shorten the time of development, keeping it secure and trusted. Starting from the definition of the real production process, I aimed to automatically generate both the smart contracts to manage the system and the user interfaces to interact with them, thus producing a working system in a semi-automated way.

Another kind of supply chain in which blockchain technology can be applied with potential advantages is shipping logistics. Here, blockchain can be the solution to the problem of distrust among players, as it does not rely on commercial third parties. It is also believed to have the potential to positively affect maritime processes and accelerate the physical flow of goods. Hence, with the support of SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis, I explored the application prospects and the practical impacts, benefits, pros and cons, and economic and technical barriers related to the implementation of Blockchain technology to support the creation of an interport community.

Finally, I decided to include in this thesis, albeit marginally, a research not focused on software engineering, but which still concerns the blockchain phenomenon, in particular cryptocurrencies. This is an important topic also considering the context of blockchain application funding. The work deals with the study of techniques for the forecasting of time series, in particular, to forecast daily closing price series of Bitcoin, Litecoin, and Ethereum cryptocurrencies, using data on prices and volumes of prior days. Cryptocurrencies' price behavior presents new opportunities for researchers and economists to highlight similarities and differences with standard financial prices. I followed different approaches in parallel, implementing both statistical techniques and machine learning algorithms, comparing my results with different benchmarks. The models perform well also in terms of time complexity and provide overall results better than those obtained in the benchmark studies, improving the state-of-the-art.

0.1 Main contributions

The main contributions of this thesis can be summarized by the following list:

1. The development of the ABCDE method which, to my knowledge, is the first attempt to develop a systematic process for dApp development, from requirement gathering to design, coding, security assurance, and deployment. The proposed method is presently focused on the Ethereum blockchain and its Solidity language, which are at the current time the most used to develop dApps. However, it can be adapted to other environments.
2. The provision of security assessment checklists for the design, coding, and testing phases that can be easily used for the development of SCs, in the realm of Ethereum and Solidity.
3. The provision of patterns and advice to solve or mitigate the problems arising from the gas mechanism. To ease their usage, the patterns have been organized into categories based on their characteristics.
4. The development of a flexible and extensible ASD simulator, able to simulate virtually every Agile development process, based on incremental development. This simulator models most aspects of ASD - team members, USs, activities, events - and can be easily extended and customized, due to the full object-oriented approach used for its development.
5. The development of a risk management model, considering team factors, requirement correctness, and effort estimation. The input data, parameters, and events, as well as the relevant output distributions, are analyzed and discussed.
6. The proposal of a new framework for choosing the blockchain architecture most suited to a specific application, and its use to justify and propose an architecture for managing consortium blockchains, which retains all the positive characteristics of public blockchains, but largely reduces their drawbacks regarding scalability, privacy, cost, and efficiency.
7. A formalization of the typical permissioned architecture, explaining the characteristics it must have to have a transparency and strength almost equal to that of a public blockchain. Precisely, the permissioned blockchain must be periodically anchored to a public blockchain (which is a tool already used, especially in distributed data storage solutions), and at the same time, an explorer able to explore the blockchain independently from the provided apps must be provided.
8. For blockchains based on the gas mechanism, a further contribution is to use gas (Ether or another cryptocurrency of that specific blockchain) to enable writing only for authorized actors. The idea is: instead of giving permissions depending on your login authorization, you are enabled because you have gas available. This is useful because since gas is limited, it also allows to dynamically manage the write permissions.
9. The development of a system to automatically develop custom dApps for the agri-food supply chain, by building configurable SCs to be assembled together. This helps the developers in creating higher-quality SCs because the SCs which are configured for a specific system are already proven and debugged. Secondly, it can help reducing development time, because systems are generated by compiling a description of the system given as tables of data. Thirdly, this approach makes food safety compliance

easy and significantly cuts down on paperwork for the actors in the agri-food supply chain.

10. The exploration of the practical implications and impacts of the application of Blockchain technology for the establishment of an interport community.
11. The study and development of algorithms to forecast the daily closing price series of Bitcoin, Litecoin, and Ethereum cryptocurrencies, using data on prices and volumes of prior days. The models perform well also in terms of time complexity and provide overall results better than those obtained in the benchmark studies, improving the state-of-the-art.

0.2 Outline

To ease the reading of this thesis I divided it into four parts, in particular, the thesis is organized as follows.

Part I (pg. 5) describes blockchain technology, clarifying its key concept and its evolution over time into the ideal environment to run Smart Contracts. It also provides the essential notions to understand the functioning of a dApp, meaning a software system that uses DLT, in this case a blockchain, as a central hub to store and exchange information. These concepts are essential to understand the work carried out in this thesis.

Part II (pg. 25) focuses on the study and development of innovative software engineering techniques for blockchain applications. In particular, it first presents the ABCDE method I developed together with my research group. Particular emphasis was given to the sections concerning security assessment and gas saving. Then, it presents and discusses a way for assessing the risk of software development in agile methodologies using simulation.

Part III (pg. 102) focuses on blockchain applications. Firstly, it formalizes an analysis of the characteristics of a permissioned blockchain, in order to have the same benefits of a public blockchain reducing its disadvantages, providing an evaluation framework. Then, it presents two possible applications of blockchain technology, in the agri-food supply chain domain and in an interport community network. The former includes a novel approach for easily customizing and composing general Ethereum-based smart contracts designed for the agri-food industrial domain, to be able to reuse the code and modules and automate the process to shorten the time of development, keeping it secure and trusted. The latter focuses on the study of the application prospects and practical implications of the application of Blockchain technology for the establishment of an interport community, through a SWOT analysis.

Part IV (pg. 162) presents a work to forecast the daily closing price series of Bitcoin, Litecoin and Ethereum cryptocurrencies, using data on prices and volumes of prior days. In particular, it follows different approaches in parallel, implementing both statistical techniques: the Simple Linear Regression (*SLR*) model for uni-variate series forecast using only closing prices, and the Multiple Linear Regression (*MLR*) model for multivariate series using both price and volume data; and machine learning algorithms: Multilayer Perceptron (*MLP*) and Long short-term memory (*LSTM*).

Part I

Blockchain technologies: definitions and characteristics

1 Introduction

The purpose of Part I is the definition and characterization of blockchain technology in its general aspects. To this end, the section begins with a detailed exposition of the principles and operating technology of the first blockchain, that of Bitcoin. Even if this blockchain does not have many "industrial" applications, it is in it that most of the operating principles of this technology have been introduced. For this reason, I believe that knowing the first and paradigmatic blockchain is important for understanding the subsequent evolutions of this technology, which affects industrial applications in practice.

I will also present the concepts of Smart Contracts and Decentralized Applications, which are essential to understanding the work carried out in this thesis. To do that, I focus on the Ethereum blockchain, which is the first and presently still the most used blockchain to develop smart contracts on public blockchains.

1.1 Cryptocurrencies phenomenon

The first cryptocurrency, Bitcoin (BTC), was introduced in 2009 and, after a couple of years of quiet operation, made a successful outcome. Initially, the phenomenon was surrounded by an aura of secrecy and illegality due to the high level of anonymity guaranteed. Subsequently, Bitcoin was characterized as a means of making payments quickly and cheaply, and as a possible safe haven currency: it is called *digital gold* because it is issued in limited quantities and therefore cannot be inflated.

Unlike any other form of digital money, such as online transfers or debit cards, the Bitcoin system is not based on a central intermediary, but is totally decentralized on the Internet, and therefore on a global network. Bitcoin transactions are activated anonymously, because Bitcoin holders are identified by totally anonymous addresses. However, the flow of Bitcoin between one address and another is totally open and traceable, because the database of all transactions, the Blockchain, is public. We are therefore in the presence of a totally new approach compared to the traditional ones, both from the point of view of finance and technology.

The price of a Bitcoin, initially zero, reached a speculative peak at around 1200 US $ at the end of 2013, and then declined, stabilizing in a range between 200 and 300 US $ for most of 2015. Starting from November 2015, the price recorded a slow but gradual increase until 2017. On 17 December 2017, a new record was recorded, reaching the value of US $ 20,000 per Bitcoin. The value then plummeted rapidly, dropping below $ 8,000 in February 2018 and stabilizing at around $ 6,000 for the rest of 2018. Since 2019, the value has fluctuated from a low of around $ 3,500 in January to over $ 40,000 in December 2020. On February 8, 2021, after Tesla's purchase of $ 1.5b in Bitcoin, it exceeded the token value of $ 50,000. As of today, April 2022, it is hovering around US $ 46,000. With these prices, the total capitalization of Bitcoins exceeds US $ 1 trillion, and is therefore sufficient to support strong exchange flows and to guarantee the sustainability of the system even in the long term.

In 2015, Ethereum also established itself as a second-generation cryptocurrency, oriented not so much to provide a store of value, but a tool for carrying out *Smart Contracts*: contracts executed through software programs and guaranteed not by central authorities, but by cryptography and by Blockchain technology. The cryptocurrency of Ethereum, called Ether (ETH) had a stable value of around 10 US $ until 2017, the year in which it had a very strong increase in value, hitting a peak of US $ 1,261 in January 2018. Then, it fell again, with lows on April 2018 (around US $ 700), June 2019 (around US $ 300), February 2020 (around US $ 200) and December 2020 (around US $ 600). Rising in 2021, Ethereum soared to new heights in November 2021, reaching over 4,800 U.S. dollars. As of today, April 2022, its capitalization has reached around US $ 400 billion.

The last few years have seen a huge interest in the cryptocurrency phenomenon by companies, financial institutions and public bodies. The business opportunities linked to the financial phenomenon of cryptocurrencies are of interest, but above all are of interest the opportunities linked to the technology of the Blockchain, seen as an innovative way of managing certifications and contracts automatically, quickly and without intermediaries. In the world, many conferences have been held on the applications of cryptocurrencies and Blockchain technology to the financial sector, public administration, the Internet of Things, healthcare and many other sectors.

The number of daily Bitcoin transactions fluctuates between 300,000 and 400,000, which corresponds to almost the current limit of the network, which can validate at most about 4-5 transactions per second. Initiatives are underway to overcome this limit, in particular by executing minor transactions, which are the vast majority, outside the Blockchain. Today there are more and more companies worldwide that accept payments in Bitcoin, through services such as Coinbase and BitPay.

Finally, there are numerous investments by "Venture capital" in startups operating on Bitcoin and Blockchain, in 2013 investments were made for 93.8 million US $, in 2014 they reached 315 million US $, and in 2015 490 million US $. During 2017, a new peak was recorded, mainly triggered by the phenomenon of Initial Coin Offers (ICO), a fundraising mechanism using blockchain technology, which allows to accept cryptocurrencies in exchange for a token that can be sold in future on the secondary market or used to purchase services or products. The enthusiasm for this new idea, the constantly rising prices and profits and the FOMO (Fear Of Missing Out) have meant that many billions of dollars were poured into tokens, even exceeding venture capital investments in high-tech initiatives in the same period. In early 2018, the bubble deflated, with the global capitalization of digital currencies going from more than $ 800 billion on January 7, 2018 to about $ 115 billion in February 2019. Over the next few years, however, the renewed interest in digital currencies has pushed their global market capitalization to ever-increasing levels, as noted above, to well over $ 2 trillion during 2021.

1.2 The enabling technologies

Cryptocurrencies are primarily based on asymmetric cryptography and hash fingerprints. The former guarantees that the possession of the private key of a key pair is unique and cannot be falsified. In this way, possession of the currency, or the authenticity of a contractual counterparty, is guaranteed. The latter guarantees the inalterability of the information, in this case of the register of all transactions. In both cases, these are “strong” technologies, which guarantee the reliability of the information on a mathematical basis. The probability of being able to violate these technologies is not zero but it is an extremely low number, of the order of one divided by a number of about 80 decimal digits, and therefore practically zero.

Another technology on which cryptocurrencies are based is the Internet and peer-to-peer (P2P) computing between nodes. In practice, they are based on a network of tens of thousands of interconnected nodes which allow the system to function. Anyone can download the open-source software, connect to the network and contribute to its functioning. This system works without the presence of one or more nodes that control the others. In this way, there are no central authorities and single points of failure of the system. Everything works fine as long as the number of connected nodes exceeds a certain threshold (roughly equal to a few hundred). The incentive to keep the nodes connected and to carry out the necessary calculations for the functioning of the system is given by the activity of mining, which allows you to earn new Bitcoins, or in general new cryptocurrency, in exchange.

1.2.1 Cryptography

The term Cryptography refers to the techniques used to encode and decode a document, so as to be able to transmit the encoding on an "insecure" channel, that is subject to interception, without the person intercepting the document being able to decode its content.

Suppose you want to encode a file D, made up of a sequence of bits, then a number. This is modified using a key K, or another sequence of bits, typically 256, and a known algorithm C (f, k). The result of the processing is another file E = C (D, K), which contains the information of file D, but encoded in such a way as to be unreadable.

Starting from E it is possible to regenerate file D, using a decoding key H, consisting of another sequence of bits derived from K, and a decoding algorithm C^{-1} (f, k). In this way we will have D = C^{-1} (E, H).

1.2.1.1 Asymmetric Encryption

The main cryptographic tool used in the Blockchain is asymmetric cryptography, which aims to uniquely and robustly identify the holder of the authorization to operate on a public address associated with a currency or a contract. Only the user who owns the secret private key can in fact operate on that address.

Asymmetric cryptography was created with the aim of encoding a document with the public key K, sending the encryption even through an insecure channel, and regenerating the original document with the private key H. Since H cannot be traced back from K, the decryption cannot be done even knowing the original document, K, and the encoded document. However, with asymmetric cryptography it is also possible to reverse the use of keys, encrypting a document with the private key H, and decrypting it with the public key K. Since only the person who generated H can know this key, decryption with the public key of a well-known document encoded with H, guarantees that the “signer” of the document is the one who knows H. This is also the mechanism of the digital signature.

In elliptical asymmetric cryptography, the two keys K and H have the following properties:

- Known H (private key), one can easily generate K (public key) from it, but not vice versa: known K, one can find H only by trial and error, trying all possible combinations of H.
- When E is known, D cannot be decoded unless we know H.
- To find H, you have to try again by trial and error, trying all possible combinations.

In particular, Bitcoin and other cryptocurrencies use the ECDSA (Elliptic Curve Digital Signature Algorithm) with the secp256k1 curve, which uses private keys of 256 bits (32 Bytes) of random data.

Figure 1 schematises the process of transmitting encrypted files between two or more users, through the method of asymmetric encryption.

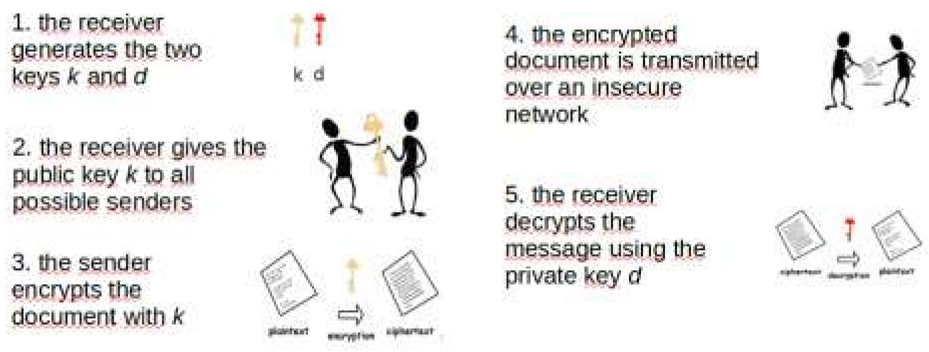

1. the receiver generates the two keys *k* and *d*
k d
2. the receiver gives the public key *k* to all possible senders
3. the sender encrypts the document with *k*
plaintext
encryption
ciphertext
4. the encrypted document is transmitted over an insecure network
5. the receiver decrypts the message using the private key *d*
ciphertext
decryption
plaintext

Figure 1. Typical use of asymmetric cryptography

1.2.2 Hash Footprints

Another fundamental technology used in the context of cryptocurrencies is the hash imprint. In mathematical and computer language, a hash function H (n) is a non-injective, and therefore non-invertible, function that maps a string of arbitrary length, n, into a string of predefined length.

The algorithm processes any amount of bits and meets the following requirements:

- H (x) is very different from H (y), even if x and y differ very little (even one bit). In practice, having a document hash guarantees that it has not been modified (otherwise it would produce another hash)
- If x is a document, it is virtually impossible to alter it so that the altered document has the same hash value as the original
- Once x is known, it is quick and easy to calculate H (x), but once H (x) is known, x can only be found by trial and error (trying all possible combinations)

Example: Suppose we want to encode the two strings "ELLIPTICAL" and "ELLIPTICAM", using the 256-bit hash algorithm SHA-2 (Secure Hash Algorithm 2), standard NIST (National

Institute of Standards and Technology). It is observed that although the two strings differ from each other by only one bit, their respective hashes are totally different.

• echo ELLIPTICAL | sha256sum

1b3fa0958b29fc66f2a0047b0b6d686baf15367388e2529fa5107d8da5274dfd

• echo ELLIPTICAM | sha256sum

48406940107944f8c45b252dc3016fa66b1a878fbfa8b522b739079cd5097326

1.2.3 The digital signature

Modern asymmetric cryptography was born to securely transmit information over computer networks, while hash functions were born for file management. The set of the two techniques is used to implement the digital signature (FD).

Three needs that the digital signature aims to satisfy are:

- guarantees that the signer of a digital document (a file) is the owner of the FD (authentication)
- guarantees that the document has not been altered after signing (integrity)
- prevents repudiation of the signature by the signatory (non-repudiation)

The use of the digital signature requires that there is a guarantor and custodian of the signature. The two fundamental elements of a signature scheme created with the dual-key cryptography system are the signature algorithm and the verification algorithm. The signature algorithm creates an electronic signature that depends on the user's key and the content of the document to which it must be attached. A pair (document, signature) represents a signed document, that is, a document to which a signature has been attached. Blockchain technology allows both to prove date and not modification of a document (by writing the hash imprint on the blockchain), and to prove the identity of whoever sent a transaction to the blockchain (through possession of the private key associated with the address of sending, which in turn can be associated with an identity).

1.3 The IT bases of cryptocurrencies

Cryptocurrencies can only exist thanks to the Internet. In fact, they require the real-time transmission of transactions on the network, and also require the connection of many hundreds / thousands of nodes that contain the transaction validation register, the Blockchain. These nodes are connected to each other in peer-to-peer mode, without any centralized control: it can be said that the network is the cryptocurrency itself.

The nodes run open-source software developed by a community (or company), which is typically very sophisticated. It includes:

- specific data structures for the efficient storage and retrieval of information;

- data exchange protocols on the peer-to-peer network;
- the ability to carry out transactions, verifying their consistency with the aforementioned cryptographic techniques;
- the ability to validate blocks of transactions, in competition with the other nodes of the network (mining);
- the ability to execute a language associated with transactions, the basis for smart contracts.

1.4 Characteristics of the Blockchain technology

The fundamental enabling technology of Bitcoin and cryptocurrencies is the "Blockchain".

The Blockchain is a distributed, shared database whose past history is unalterable. A public BC is based on the P2P system outlined above and is open to anyone who wants to contribute to its operation by installing a copy of it on their PC. It is equipped with mechanisms to reach consensus among all nodes on the information to be stored, so that no one can take control of it, not even generating a mass of fictitious nodes. This is achieved at the expense of a high computational cost, which is paid for by mining. The security of the BC technology is such that, even if only one copy of the BC remained, it could start again from that copy; furthermore, if one wanted to alter the past history of this single copy, this would still require a prohibitive number of calculations.

The main features of this technology are:

1. **Distribution**: information is stored on multiple nodes, giving resilience and security.

2. **Disintermediation**: transactions are managed without intermediaries and without a central management authority.

3. **Transparency**: all transactions are stored unalterably and in clear text. For each transaction you know the amount in Bitcoins and the addresses from which they are withdrawn and to which the transfer goes. In this way, a complete tracking of Bitcoin transfer flows from one address to another is possible.

4. **Anonymity**: in the Bitcoin system, the holders of the funds are identified only by an anonymous address, to which the private key is associated. Thus, while the flow of Bitcoin from one address to another is completely transparent, the holders of the funds remain anonymous.

5. **Double spending protection**: if you try to spend the same funds more than once, only the first verified transaction is accepted, and the others are rejected.

6**. Immutability / Non-repudiation**: once a transaction has been sent and accepted, it cannot be canceled for any reason.

7. **Security**: the transaction can only be activated by knowing the private key relating to the withdrawal address of the funds. If this key is lost, the related funds are lost forever.

8. **Programmability**: complex actions can be programmed, called Smart Contracts (SC), which are also fully verifiable and can be activated by multiple parties involved (with their respective private keys). Ethereum is the currency that offers the most functionality, but SCs are also possible with Bitcoin.

9. **"Notarial" storage**: it is possible to use special transactions to store information on the Blockchain. This information (usually limited to a few tens of Bytes per transaction) is used to certify the existence and integrity of a document, or set of documents, at a certain date. There are initiatives such as Hyperledger (in which Accenture, Fujitsu, Hitachi, IBM, Intel, JP Morgan participate among others), Microsoft's Blockchain-as-a-Service, Factom inc., Which aim to create even complex document certification systems, based on the Blockchain.

1.5 The fundamentals

In this section I will present the fundamental concepts of blockchain technology: addresses, blocks, transaction and consensus. In particular, I will refer to the Bitcoin blockchain since it is the first and best known, but these concepts are also similar for all other blockchains.

1.5.1 Addresses

A Bitcoin address is a number associated with a private-public key pair. It is the public part of all Bitcoin transfers. The Bitcoin Blockchain stores all Bitcoin transfers to and from various addresses in clear text.

An address is generated from the public key so that:

- it corresponds to a single public key;
- given the address, it is not possible to know the public key; this is because the explicit knowledge of this key could in the future make the system more vulnerable, perhaps in the presence of quantum computers capable of attacking the currently used cryptography;
- it contains checks in order to recognize any typing errors (as with the tax code).

When transferring the Bitcoins associated with one address to another address, the public key of the first must be revealed. The procedure for generating a new Bitcoin address is as follows:

1. a random number d (private key) of 256 bits is generated;
2. the public key B is generated starting from d and from the standard elliptic curve secp256k1;
3. starting from B, the Bitcoin address A is generated with a complex procedure, described later;
4. a Bitcoin address has about 195 bits, and is represented with 30-34 alphanumeric characters (BASE58 representation) always starting with a "1".

Given an address A, only the owner of the key *d* can generate B from it, and from B generate A.

The detailed steps to generate a Bitcoin address A from the public key B are:

1. Calculation of the RIPEMD160 hash of the SHA256 hash of B:

   Hash := RIPEMD160 (SHA256 (B))

2. Add a byte (address version) at the head of the hash:

   Ha := b.hash with b equal to:

   0x05 (00000101) for the main net (Bitcoin)

   0xc4 (11000100) for the net test

3. Calculation of the SHA256 hash of the SHA256 hash of ha:

   Hh := SHA256 (SHA256 (ha))

4. Extract the first 4 bytes of hh, to be used as checksum (cs):

   Cs := First4Bytes (hh)

5. We concatenate b, hash and cs (8 + 160 + 32 = 200 bits):

   A := BASE58 (b.hash.cs)

A BASE58 string is generated by encoding a number (a sequence of bits) in base 58, using the following 58 alphanumeric characters as digits from 0 to 57:

123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz

Note that the "ambiguous" characters are missing: 0, O, I, l

Examples of Bitcoin addresses:

1NdNaNG1K9eDW5jfMDmT1biyih5micPsGE

1yQ3KkhfhQCMABZkZxTdjAC2nGFUxoDcR

Figure 2 graphically shows the generation of a Bitcoin address. The "paranoid" procedure of calculating A starting from B with 4 successive hashings is used to:

- prevent key B from being made public because it could be attacked with quantum computers in the future;
- insert a "checksum" that verifies whether the address has been entered incorrectly: in this case, the software does not make payments to this address;

When the Bitcoins associated with an address are spent, the public key B is made known in the BC, and thus it becomes more vulnerable. For this reason, some recommend changing the address at each transaction, if it has to pay a "change" to an address of the person who activated it.

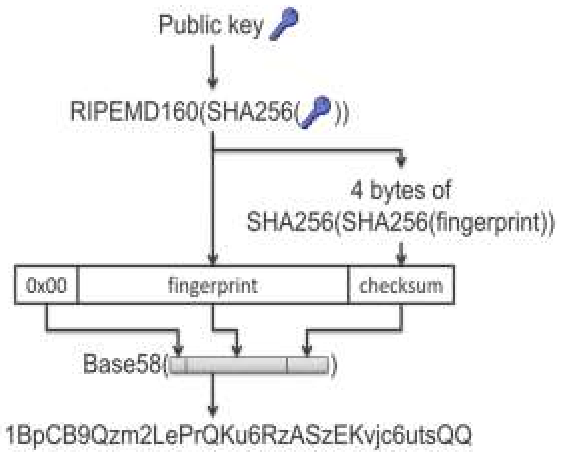


Figure 2 Generation of a bitcoin address

1.5.2 Blocks

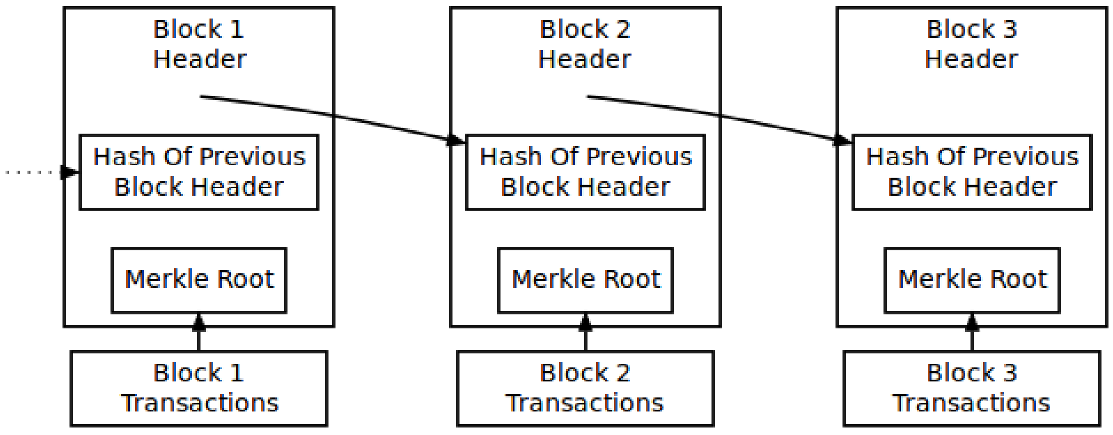
The Blockchain is made up of an ordered sequence of blocks. These blocks contain the validated transactions, which in turn contain the Bitcoin transfers from one address to another, as well as the payment of the block validation reward to the address of the miner who validated it.

A block is made up of a header and all its transactions. A block header contains the following data:

- Date and time
- Number of transactions contained in the block
- Nonce (a 32-bit integer)
- Block hash
- Hash of the previous block
- Merkle tree hash containing the transactions

A Merkle tree is an efficient data structure to verify by hash that the transactions were not altered.

The Blockchain is a sequence of blocks, each with a hash to ensure its inalterability. Each block also incorporates the hash of the previous block, thus "hooking" the blocks together and creating a "chain". If a block of the Blockchain were altered, the hash would also be altered, and therefore to preserve the integrity of the Blockchain it would also be necessary to modify all subsequent blocks. Since the computation of the hash of a block is very expensive computationally, such a "forgery" is practically impossible, and this is one of the main advantages of the Blockchain. Figure 3 shows a simplified representation of the Bitcoin Blockchain.



Simplified Bitcoin Block Chain

Figure 3 Simplified schematic of the Bitcoin Blockchain.

1.5.3 Transactions

The purpose of a transaction is to register the transfer of Bitcoins from one or more addresses to one or more other addresses in the Blockchain. Each transaction has a hash that uniquely identifies it, also called its "id". A transaction has one or more inputs, and one or more outputs, although there are exceptions to this rule. The inputs are the outputs of previous transactions not yet spent (UTXO, Unspent Transaction Output). Each input contains an address, the hash (id) of the transaction that transferred the Bitcoins to the address, and the amount transferred in that transaction (in Satoshi, that is 100 millionth of a Bitcoin). All this information is verifiable because it is already contained in the Blockchain. In output there are one or more addresses, with the amount (in Satoshi) to be transferred for each address. Typically, the sum of all Satoshis in input to the transaction is greater than the sum of those in output. The difference between input and output goes to the miner who validates the block as a "fee”, and which is therefore added to the fixed fee in Bitcoin given to miners for validation. If the commission for the miners is zero, or too low, the transaction, while valid, may not be accepted, or its validation may be delayed a long time.

Transactions, once entered in the BC, are irrevocable. The Bitcoin protocol does not provide for any way to cancel a verified and recorded transaction. For a transaction to be valid, it is necessary that:

- All its inputs must be "signed" by the private keys associated with their addresses.
- None of the inputs must have already appeared as inputs in another transaction (“double spending” is not possible).
- The addresses of the outputs must be valid.
- The sum of the input amounts must be greater than or equal to that of the output amounts.

The inputs of a transaction are unspent outputs (UTXO) of a previous transaction. These inputs must all be spent 100% within the transaction.

If you want to transfer only part of the input to the output, you need to add a further output that transfers the "change" to an address controlled by whoever carries out the transaction itself. This address can be the one (or one of those) of input, but usually a new address is generated, for security and privacy reasons. Figure 4 schematically shows the relationships between inputs and outputs of a group of related transactions.

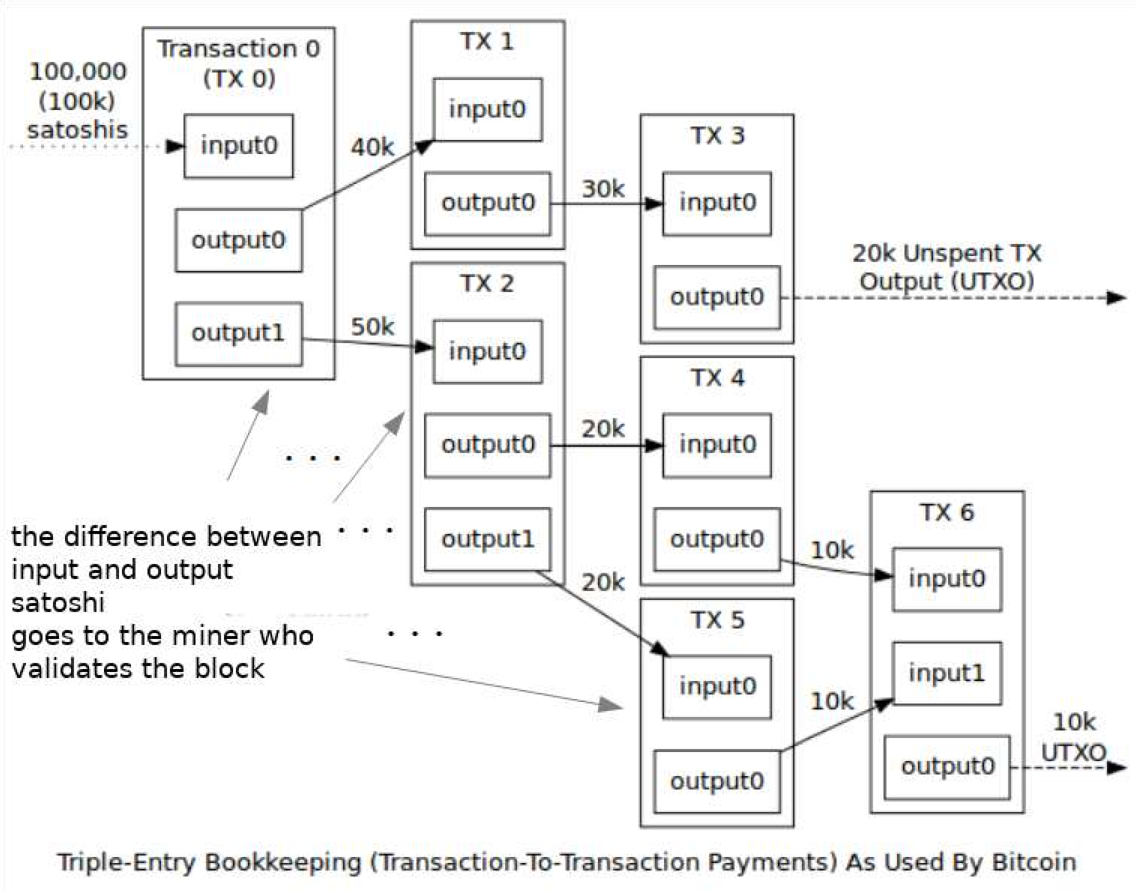


Figure 4 A sequence of linked Bitcoin transactions.

In each block, the first transaction called Generation Transaction, but often called itself "Coinbase") has a "dummy" input, called "Coinbase". Its output is the address of the miner who validated the block and is therefore a transaction entered by the miner itself.

Coinbase transfers a fixed amount of Bitcoin to the output as a "reward" for validation, plus the commissions (fees) coming from the transactions of the block. This fixed amount, initially 50 Bitcoins, halves every four years. As of December 2021, it is 6.25 Bitcoins.

The Bitcoin protocol also provides for more complex transactions than simply transferring currency. These transactions allow the execution of scripts whose success is linked to the validation of the entire transaction; we will see them in more detail in the section on Smart Contracts. Among these, the simplest transaction is the “OP_RETURN” type transaction. It has no output, and sends all Bitcoin inputs to the miner. It is used to record information (equal to 80 bytes) within the transaction itself.

The most obvious form of attack on the Bitcoin system is "Double Spending". It is an attempt to spend two or more times the output of a transaction, thus "multiplying" one's Bitcoins.

The protocol protects itself from this attack by considering valid only the first transaction that spends the output and that is recorded in the Blockchain. In doing so, the problem is solved in a decentralized way, without the need for a central authority to decide which transaction is the valid one.

1.5.4 Consensus

One of the fundamental points for the proper functioning of the Blockchain and to obtain the necessary trust from users is that of the consent mechanism used to decide whether or not to insert information in the Blockchain.

Since the Blockchain is based on a distributed approach, without any central authority responsible for making decisions, a voting mechanism is needed by the nodes to make the decisions, and therefore to arrive at consensus. The Bitcoin Blockchain uses this mechanism to decide which blocks to insert into it. Other Blockchains can make decisions at different levels of granularity, for example to decide which transactions to enter.

In general, the consent mechanism must be:

- **Agreement Seeking**: all honest participants must agree on the solution.
- **Collaborative**: all participants work together on the result, placing the interest of the group before the individual one.
- **Egalitarian**: all votes have the same weight.
- **Inclusive**: as many participants as possible should be involved in the process.
- **Participatory**: anyone who intends to do so must be able to participate in the process of reaching consensus.

There are various possible ways to arrive at a consensus among the nodes of which block or information to insert into the Blockchain. All are designed to prevent a malicious party from taking control of the validation, thus destroying the trust and ultimately the usefulness of the Blockchain. The main ones are listed and briefly described below.

Proof of Work is the mechanism of Bitcoin and most cryptocurrencies. In it, nodes compete to solve a computationally complex problem, such as computing a hash that starts with a given number of zeros, varying the nonce, and other irrelevant information in the block. The first node that manages to solve the problem communicates it to the others, and consensus is reached. This method has proved very robust, and prevents Sybil Attack, which would be too expensive. A Sybil Attack consists of adding a large number of nodes controlled by the attacker to the network, thus taking control of it. Its main disadvantage is the very high consumption of hardware resources and electricity.

Proof of Stake In this approach, block validation is assigned to nodes in proportion to the amount of cryptocurrency owned by them, in a probabilistic way. Thus, the computational cost is negligible. The idea is that those who have an interest (stake) to protect due to their involvement in the network will not carry out destructive actions against it. The main disadvantage is that any reward for validating the block tends to go to those who already have a lot of wealth, triggering “rich-get-richer” phenomena.

Proof of Importance similar to Proof of Stake, but in addition to the amount of cryptocurrency, the number of transactions made and received by the validator is also taken into consideration, and therefore the importance and contribution given to the network by it.

Proof of Burn with this method, block validation rights are obtained by "burning" the cryptocurrency owned. This is typically achieved by transferring it to an address generated so as not to know its private key. This cost clearly makes Sybil Attack unlikely. The method is not widely used. The Counterparty network used it as a “boostrap” mechanism to assign participants the initial amount of cryptocurrency generated.

Deposit based consensus in this approach, currency must be deposited to participate in the validation. If a transaction validated by a node is rejected by the others, the node loses its "deposit".

Byzantine fault tolerant this approach applies to private Blockchains, in which participants are invited, and it is reasonable to think that only a minority of them can become hostile. Consensus is achieved by a transaction voting mechanism. When a given number of votes among participants is exceeded, the relative decision to accept or reject the transaction is made.

Federated Byzantine Agreement also in this approach the participants are invited. Each node keeps a list of other nodes it trusts. A node votes if it is certain of the decision, or if the majority of trusted nodes make a decision. The Stellar system uses this form of consent.

1.6 Ethereum and Smart Contracts

One of the fundamental properties of the Blockchain is that it can be the basis for Smart Contracts. They were introduced by Ethereum in 2015, introducing a Blockchain and a low-level language, along with various high-level and effectively usable languages, among them I mention Solidity, Vyper and Yul, to code and execute Smart Contracts.

Smart Contracts are automated contracts, in which participants prove their identity and approval of the contract with their private key. Many of the major cryptocurrencies are able to carry out Smart Contracts, with different levels of sophistication. The concept of Smart Contract is older than the blockchain, because its definition was proposed by Nick Szabo in 1994. By the way, Szabo is one of the main suspects of being Satoshi Nakamoto, the anonymous inventor of Bitcoin. According to Szabo’s definition:

“A Smart Contract is a computerized protocol for executing transactions, which executes the terms of a contract. The general objectives of a Smart Contract are such as the usual contractual conditions (such as payment terms, rights, confidentiality, and even enforcement), to minimize both intentional and accidental exceptions, and to minimize the need for trusted

intermediaries is minimal. Other economic objectives include the reduction of costs due to fraud, arbitration and enforcement costs, and other transaction costs".

A Smart Contract is basically a program, which runs on a secure medium (trusted) and is in turn secure. It takes digital signatures of participants, and other information, from secure sources as input. In output, it transfers cryptocurrency imports, can actively activate other SCs, record information or signal changes to external systems. SCs also often have a user interface that emulates the jargon and logic of contract clauses. Since the execution of a software program is deterministic and immutable, with the same input and state of the program, the code can be considered a contract. Once the contractual clauses are inserted into the code of a SC, and this is accepted by the contractors, the effects are no longer linked to their will or to the action of intermediaries. Obviously, precise guarantees are needed:

- the code must have no errors, must be executed correctly and must not be modified;
- the inputs to the code must have secure and identifying sources;
- the code outputs must achieve the desired effects.

In other words, there must be a mechanism that guarantees the trust that the contractors place in the smart contract. At this point the Blockchain comes into play: it provides all the aforementioned guarantees, without the need for a central authority. Since the first versions of the implementation of the Bitcoin Blockchain, it was equipped with a real programming language, albeit at the assembler level.

The Bitcoin transaction, which involves the transfer of cryptocurrency from one or more inputs to one or more outputs, is in fact carried out by executing code written in this language, which takes care to verify the authenticity of the private keys in possession of who carries out the transaction. However, the language of Bitcoin is limited, does not involve decision points or cycles, and is therefore not Turing-complete.

Figure 5 schematically shows in the lower part the conditions and the preparation of a Smart Contract, and in the upper part its execution on a Blockchain.

A Smart Contract is a program that is created with a specific transaction; it knows the address from which the creation starts, has its own address that serves to identify it and has public functions, which can be called from the outside that implement the functions.

The creation and modification of the data of a SC cost "GAS", and therefore Ethers. If public functions of the SC are "read only", they cost nothing and the call is not registered; if they can modify the BC, the call is made with a transaction, which costs GAS, which is given to the miners.

The use of the gas mechanism in Ethereum, which I will detail in the next subsection, ensures that a contract always terminates, even if the corresponding program enters an infinite loop due to an error. In this way, you are also protected from Denial of Service (DOS) attacks, which would cost the attacker too much.

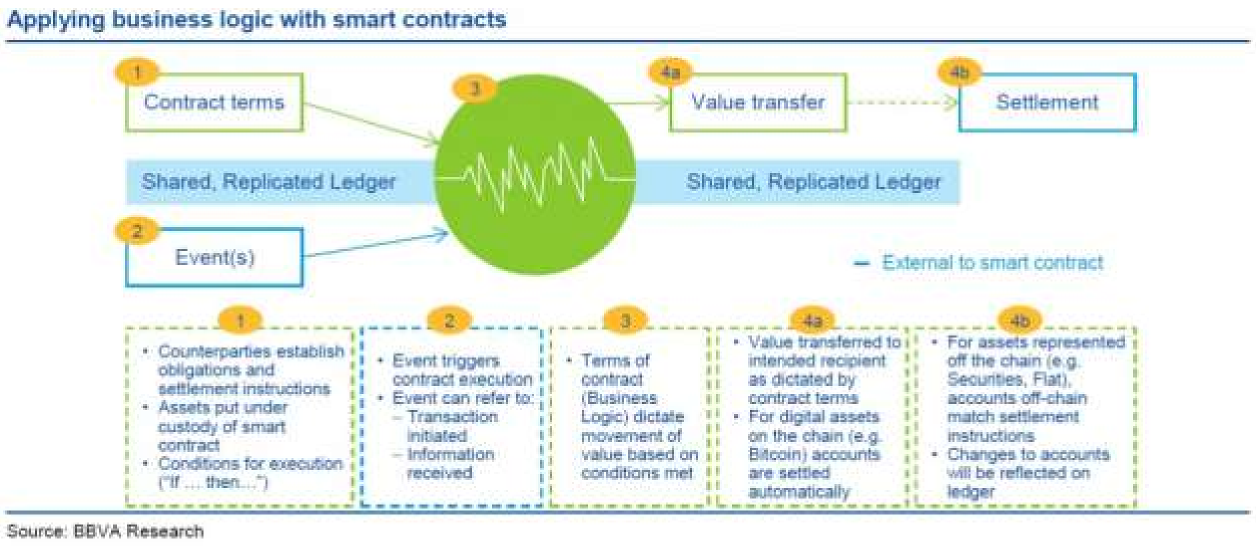

Applying business logic with smart contracts
1
Contract terms
3
4a
Value transfer
4b
Settlement
Shared, Replicated Ledger
Shared, Replicated Ledger
2
Event(s)
External to smart contract
1
• Counterparties establish obligations and settlement instructions
• Assets put under custody of smart contract
• Conditions for execution ("If .... then...")
2
• Event triggers contract execution
• Event can refer to:
– Transaction initiated
– Information received
3
• Terms of contract (Business Logic) dictate movement of value based on conditions met
4a
• Value transferred to intended recipient as dictated by contract terms
• For digital assets on the chain (e.g. Bitcoin) accounts are settled automatically
4b
• For assets represented off the chain (e.g. Securities, Fiat), accounts off-chain match settlement instructions
• Changes to accounts will be reflected on ledger
Source: BBVA Research

Figure 5 The preparation and execution of a Smart Contract on a Blockchain.

1.7 Smart Contract Characteristics

A Smart Contract should be:

- **deterministic**: given the same input it must always provide the same output. To this purpose, it must not call non-deterministic functions, and it must not use non-deterministic data resources;
- **terminable**: by definition the SC must be able to finish within a certain time limit. To ensure this, there are several methods: use a timer, so if the execution lasts more than a given time limit it is externally interrupted; use incomplete Turing blockchains, which are not allowed to enter infinite cycles, like Bitcoin blockchain [1] [2] [3]; use the step meter method, in which a program is interrupted once a certain number of steps have been completed; or charge a cost to each operation and interrupt the execution if the prepaid commission has been reached. The last one is the case of Ethereum;
- **isolated**: each contract must be kept isolated to avoid corrupting the entire system in the event of a virus or bug;
- **immutable**: once deployed on the blockchain, a SC cannot be changed. However, it can be disabled forever. This property, together with the ability to publish the source code of the SC, guarantees the highest level of transparency and trust.

1.8 Ethereum Virtual Machine and Gas Mechanism

The Ethereum Virtual Machine (EVM) is the virtual machine on which all SCs work in Ethereum. It is based on 256 bits words, and is Turing Complete. It is simple but powerful. All SCs are sets of bytecode instructions, which are executed in sequence. However, the bytecode allows jumps, thus enabling a Turing complete behaviour.

In Solidity, when a SC is compiled, it is converted into a sequence of “operation codes”, also known as opcodes. These are identified by abbreviations, for example ADD for addition, MUL for multiplication, etc. All the opcodes and their description are available in the so-called Ethereum yellow paper [4], the document which first described this system. Each opcode has a predetermined amount of gas assigned to it, which is a measure of the computational effort required to perform that particular operation. Bytecodes are similar to opcodes but are represented by hexadecimal numbers. The EVM executes bytecodes.

Table 1 Gas costs in Ethereum.

Operation	**Gas**	**Description**
ADD/SUB	3	Arithmetic operation
JUMP	8	Unconditional Jump
SSTORE	5000/20000	Storage operation
BALANCE	400	Get balance of an account
CALL	25000	Create a new account
CREATE	32000	Create a new account

Table 1 [4] [5] shows the amount of fee (gas) due for the various operations in Ethereum; in the yellow paper, specifically in the appendix g, a complete table is available. For example, applying the blockhash operation requires 20 gas units, while the ADD operation requires 3 gas units. Is worth noting that a SC consists of numerous operations, and some of them consume much more than a simple arithmetic operation.

The gas is therefore a fee for the execution of a computation, paid by who sends the transaction that triggers the computation.

By default, the minimum amount of gas for an operation that affects the state of the EVM, is 21000 gas. For example, this is the amount required to send Ethers from one account to another. To execute a function of a SC this amount will be 21000 gas plus the gas needed to perform each of the required opcodes. An exception to this behavior is when the called function is read-only and simple. Such a function is called a view function, and its execution is free and immediate, because it does not change the state of the EVM. So, calling a _view_ function within a Call in a local node does not cost gas, while calling the same function from a deployed SC within a Transaction costs gas.

Before performing an operation, the user sets a gas limit, which corresponds to the maximum amount of gas that he/she wants to pay. If the gas actually required by the operations overcomes the gas limit, that operation will be aborted, each change will be rolled-back, but all the gas will be spent and therefore lost. If the user sets a gas limit higher than the one required, the operation will be carried out, and only the used gas will be spent. It is almost impossible for a user to know in advance how much gas a transaction will exactly require. However, it is not wise to set a gas limit too large, because in the case of a bug or error in the SC, there is a risk of exceeding this limit and losing all the gas.

The total fee actually paid, is equal to the total gas used multiplied by the "gas price". The gas price is not fixed but set by the user. Miners prefer transactions with higher fees; transactions whose fees are too low may never be included in the blockchain. On the contrary, setting a very high gas fee guarantees that the transaction will be executed quickly. At ethstats.net website [6] you can find the suggested gas price in real-time. At the time of writing this article the suggested gas price is 8 gwei, which corresponds to 0.000000008 Ether. Note that this number is constantly changing.

There is a gas limit also for every block. It corresponds to the maximum amount of gas that all the transactions included in the block can consume. This number also determines the maximum number of transactions to include in the block. Even the "block gas limit" is not fixed but is determined by the miners. The higher this limit is, the more the miners will earn in terms of transaction fees; however, the computational load to compute the block transactions will also increase.

Gas mechanism serves as an incentive system both for miners, to spend hardware and electricity costs to validate transactions, and against attackers, who would spend money to perform an attack. The gas mechanism has the advantage of providing a good incentive for the miners and a disincentive for the attackers. However, it has the disadvantage that operations can become very expensive.

1.9 Memory Usage in Ethereum

When developing a SC, different types of storage are available [4]:

- **Stack** (volatile stack access): it contains small local variables. It is almost free to use but can hold a limited amount of data. All operations are performed on the stack. It can be accessed with different instructions, such as PUSH, POP, COPY, SWAP, …
- **Memory** (volatile memory access): it contains temporary values, generated during the execution. It is erased at the beginning of every function call. It can be accessed with three instructions: MLOAD, MSTORE, MSTORE8.
- **Storage** (persistent memory): it contains all the SC state variables. Every contract has its own persistent storage. A contract can strictly read or write only its own storage. The opcodes to operate with it are: SLOAD and STORE.
- **Calldata**: special data location that contains the function arguments, only available for external function call parameters.
- **Event Log**: a special memory in the blockchain were data related to the Events raised by SCs are stored. These data cannot be accessed by SCs, but only by external applications.

All operations used to manipulate memory cost gas. The most expensive ones are those affecting the Storage. This because the data are permanently stored in a database replicated across tens of thousands of nodes.

1.10 Decentralized Applications

A dApp is defined as a software system that uses distributed ledger technology (DLT), typically a blockchain, as a central hub to store and exchange information, through smart contracts. Note that it is not a blockchain software able to manage a new cryptocurrency or other applications - that is, software enabling blockchain nodes, which needs different kinds of development practices, not the subject of this work.

A blockchain is an append-only, distributed data structure, managed by a set of connected nodes, each holding a copy of the blockchain, and able to execute SCs, programs residing in the blockchain itself.

The blockchain state is changed through sending *transactions* to the network - in *public blockchains*, everyone can send a transaction, but only valid ones are processed. The valid transactions are recorded in sequentially ordered blocks - hence the name "blockchain" - whose creation is managed by a consensus algorithm among the nodes. All transactions are sent from a single address, which is in turn associated to a private key. Only the owner of the private key can sign the transactions coming from an address, using asymmetric cryptography. A transaction can transfer digital currency between addresses, can create a SC, or execute one of its public functions; in this case, the function is executed by all nodes, when the transaction is evaluated. Most present real applications of dApps and smart contracts are intended for the management of digital currencies or tokens, which have a true monetary value.

The use of dApps has been introduced also for other purposes, like notarization of information, identity management, voting, games and betting, goods provenance certification, and many others [7].

In this thesis, I will use as a reference Ethereum, which is presently the most used blockchain to develop smart contracts on public blockchains [8] [9].

Data on dApps running on permissioned blockchains are more difficult to find, because they refer to projects which are not publicly accessible, but Ethereum is very popular also for dApps running on permissioned blockchains. Open source DLTs such as Hyperledger and Corda are also widely used.

The Ethereum Virtual Machine (EVM), able to execute SC bytecode, runs on all nodes of the Ethereum blockchain [10]. In practice, the SCs are written in high-level languages (HLL). Nowadays, the most popular HLL for Ethereum is called Solidity. As already mentioned, SCs run in an isolated environment. The results of their execution must be the same whatever node they are executed on; consequently, they cannot get information from the external world (which mutates with time) and cannot initiate a computation autonomously (for instance at given times). SCs can only access and change their state, and send messages to other SCs.

The state of a SC is permanently stored in the blockchain, using *storage* variables. Moreover, once a SC is deployed, it is in the blockchain forever - it cannot be modified or erased, though it can be forever disabled. Creating a SC and changing its state costs units of "gas", which must be paid in Ether (the digital currency of the Ethereum Blockchain). Each SC has a unique Ethereum address. In Solidity, a SC can inherit from other SCs; it has a public interface, that is a set of functions that can be called through a transaction. The call of a public function of a SC is called a "message". Sending a message to a SC can be performed either by posting a transaction coming from an address, or by executing code of the same, or of another, SC. In the former case, the transaction must be accepted by the network, and it will take time (at least 10-15 secs.), and a greater amount of gas. In the latter case, the transaction is executed immediately. A SC can receive and send Ethers, from and to another SC, or an address. A function which returns a value without changing the state of its SC is executed immediately by the EVM and costs nothing. This kind of function is called "*view*".

A typical dApp architecture is shown in Figure 6. Here, the Ethereum blockchain is shown on the left, highlighting that a node is composed of its enabling software, of the EVM, and of the SC bytecode and its storage, recorded in the blockchain. The software system running on mobile devices and/or on servers, possibly on the Cloud, exchanging information with users and external devices, which we call "App System", is shown on the right. Its User Interface (UI) typically runs on a web browser. The server component stores data that cannot be stored in the blockchain, and performs business computations. In a non-trivial system, a dApp is typically composed of various SCs deployed on the blockchain.

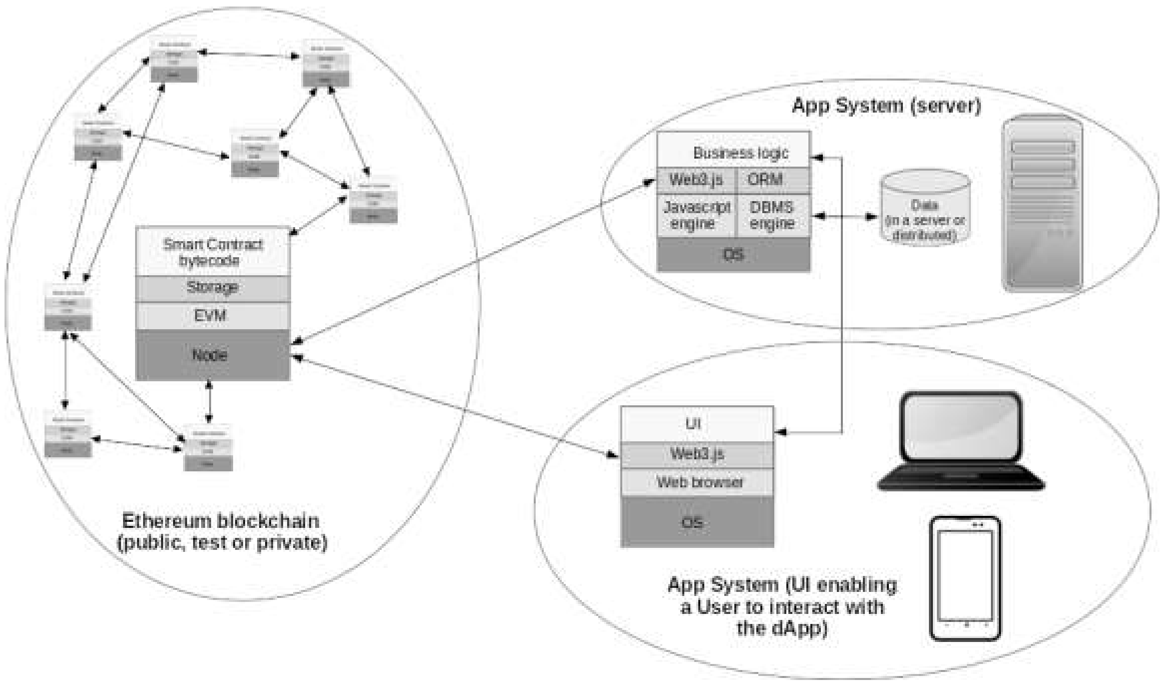

Smart Contract bytecode
Storage
EVM
Node
Ethereum blockchain (public, test or private)
App System (server)
Business logic
Web3.js
ORM
Javascript engine
DBMS engine
OS
Data (in a server or distributed)
UI
Web3.js
Web browser
OS
App System (UI enabling a User to interact with the dApp)

Figure 6 A typical architecture of an Ethereum dApp application. The blockchain with its SCs is shown on the left, the App System on the right.

Part II

Agile BlockChain Dapp Engineering

2 Introduction

Part II of this thesis focuses on the study and development of innovative software engineering practices for blockchain applications.

Blockchain software development is becoming more and more important for any modern software developer and IT startup. Nonetheless, blockchain software production still lacks a disciplined, organized and mature development process, as demonstrated by the many and (in)famous failures and frauds occurred in recent years.

In particular, section 2.1 investigates the potentiality of agile in the context of dApp development. Section 2.3 explores the related work. In section 2.4, I present a complete method addressing blockchain software development, called ABCDE, meaning Agile BlockChain Dapp Engineering. The method considers the software integration among the blockchain components - smart contracts, libraries, data structures - and the off-chain components, such as web or mobile applications, which all together constitute a complete dApp system.

I advocate for ABCDE the use of agile practices, because these are suited to develop systems whose requirements are not completely understood since the beginning, or tend to change, as it is the case of most blockchain-based applications. ABCDE is based on Scrum, and is therefore iterative and incremental. From Scrum, the method keeps the requirement gathering with user stories, the iterative-incremental approach, the key roles, and the cerimonies. The main difference with Scrum is the separation of development activities in two flows - one for smart contracts and the other for off-chain software interacting with the blockchain - each performed iteratively, with integration activities every 2-3 iterations.

ABCDE makes explicit the activities that must be performed to design, develop, test and integrate smart contracts and off-chain software, and documents the smart contracts using formal diagrams to help development, security assessment, and maintenance. A diagram derived from UML class diagram helps to effectively model the data structure of smart contracts, whereas the exchange of messages between the entities of the system is modeled using a modified UML sequence diagram.

The proposed method has also specific activities for security assessment and gas optimization, through systematic use of patterns and checklists. These will be detailed in section 2.4.4 Security assessment for Smart Contracts and 2.4.5 Gas optimization. The method is described in detail and an example is given in section 2.5 to show how to concretely implement the various development steps.

Agile methodologies aim to reduce software development risk using short iterations, feature-driven development, continuous integration, testing automation, and other practices. However, the risk of project failure or time and budget overruns is still a relevant problem. Always with a view to developing innovative software engineering techniques, in section 3 I present and discuss a new approach to model some key risk factors in agile development, using software process simulation modeling (SPSM). The approach includes modeling the agile process, gathering data from the tool used for project management, and performing Monte Carlo simulations of the process, to get insights about the expected time and effort to complete the project, and about their distributions. To validate the simulator, and to demonstrate how the method can be used, I analyzed three open-source projects, gathering their data from JIRA repositories. In the future I plan to apply it to blockchain project repositories as well.

The researches presented in the followings were partially published in:

- “Abcde–agile block chain dapp engineering”, L. Marchesi, M. Marchesi, R. Tonelli. In Blockchain: Research and Applications, Volume 1, pp. 100002, Elsevier, 2020. [11]
- “Design patterns for gas optimization in Ethereum”, L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino, D. Tigano. In Proceeding of the 2020 International Workshop on Blockchain Oriented Software Engineering, pp. 9-15, IEEE, 2020. [12]
- “Security checklists for Ethereum smart contract development: patterns and best practices”, L. Marchesi, M. Marchesi, L. Pompianu, R. Tonelli. arXiv preprint arXiv:2008.04761 [13]
- “Assessing the Risk of Software Development in Agile Methodologies Using Simulation”, Maria Ilaria Lunesu, Roberto Tonelli, Lodovica Marchesi, Michele Marchesi. In IEEE Access, Volume 9, pp. 134240-134258, IEEE, 2021. [14]

2.1 Agility and dApp development

Nowadays, the developments of dApps worldwide share some common characteristics. Several teams involved are typically working on ICO projects, which gathered money through tokens and are about applications of blockchain technology [15]. Other teams are working on DeFi (Decentralized Finance) projects, which gathered money through venture capital and/or token minting. Other kinds of projects are promoted by startups trying to take advantage of the novelty of dApps to develop disruptive solutions, or to get a niche where to thrive. In all cases, they are typically small, self-organizing co-located teams, where experts of system requirements are highly available.

Other characteristics of dApp development is that dApps typically are not life-critical applications, though several among them can be mission-critical. However, the time-to-market and the ability to get early feedback from the users and the stakeholders are essential, because often the requirements of the dApp initially are only vaguely defined and are subject to change. All these features make dApp development an ideal candidate for the use of Agile Methods (AMs). In fact, AMs are suited for small, self-organizing teams, possibly co-located, working on projects whose requirements can change [16]. AMs are considered to be able to deliver quickly and often, as needed by dApp projects.

The most used AM is presently Scrum, which is iterative and incremental, with short iterations (1-4 weeks) [17]. Scrum does not prescribe specific software development practices, but is focused on the process. In short, Scrum, as most other AMs, typically performs requirement elicitation through user stories (USs), that are short descriptions of how the system answers to inputs from users, or from external devices [18].

USs are mostly gathered at the beginning of the development, but can be modified and augmented at any time. The project advances iteratively implementing a subset of the USs at each iteration. The person in charge of choosing the goal of each iteration, and explaining it to the team is called "Product Owner", to reach goal a subset of USs is selected and implemented.

As described below in section 2.4.1, the proposed ABCDE method is based on Scrum, and is specifically focused on the software development process, and not on specific design or programming practices.

Besides Scrum process, there are other agile practices that are well suited to dApp development. In the followings, I list and shortly describe agile practices suggested in ABCDE process, and how they can be applied to smart contract development.

- *Test Driven Development (TDD)*: this practice prescribes writing the tests before the code [19], using an automated test suite that can be run whenever needed. For the App System, one can use one of the many existing testing frameworks. For SCs written in Solidity, at the moment the most popular testing environment is Truffle [20]. Note that performing automated testing on SCs is not trivial, because tests need to be run on the blockchain, which is a separate entity from the testing environment itself. This is similar to testing the interface of a database. Also, in order to test the software interacting with the SCs, we need a "mock object" able to simulate the blockchain, if the required SCs are not yet implemented, and/or if we need to improve the speed of testing.
- *Continuous Integration*: merging all developers working copies to a shared mainline, even several times a day. Developing dApps, this practice is critical, and it should be practiced both on the App System and the SCs, checking at each merging also how the two systems interact through transactions. This practice, and the following one, requires the use of a version control system helping integration and versioning, as well as of an automated test suite, to assess the absence of undesirable side-effects.
- *Collective code ownership*: every developer is allowed to intervene on whatever code s/he considers appropriate to modify. With small, dynamic teams as typically happens with dApp development, this practice should clearly be applied. As regards smart contract development, team members modifying code written by other members should be very careful not to infringe security and gas optimization provisions. Note that often the team members expert in SC development differ from those expert in App System, so their spheres of influence should remain separate.
- *Refactoring*: this practice too needs to have an automated test suite, that can be run when the refactoring is made, to assess the absence of unwanted side effects [21]. This is especially needed with the complex architecture of dApps, whose components interacts through transactions.
- *Information Radiators (Cards, Boards, Burndown charts)*: making visible the status of a project using boards that can be observed by everyone and updated in real time can obviously greatly benefit dApp development and its use is therefore strongly encouraged in ABCDE.
- *Coding Standards*: the dynamicity of the teams and the push to quickly develop applications make necessary that the project manager (or the Scrum Master) ensures that this practice is strictly followed. This would greatly facilitate code understanding, and ease subsequent maintenance activities.

- *Pair Programming (PP)*: using ABCDE, I suggest to use PP in the case the software to be developed is critical, is not yet well understood, or there are new team members to train on the job.

2.2 Security assessment

Many dApps deal with direct digital currency or token usage, that is with entities that have a direct, real monetary value. In other cases, they may deal with contractual issues, again with strong economic implications, as in the case of document certification, supply chain management, voting systems. Therefore, in many cases dApps are business-critical, and very strict security requirements should be assured. Code inspection, security patterns, and thorough tests must be applied to get a reasonable security level. ABCDE proposed security assessment will be described in section 2.4.4.

2.3 Related work

2.3.1 Software Engineering for Dapp development

Software Engineering (SE) for dApp development, sometime called Blockchain-Oriented Software Engineering (BOSE) is still in its infancy. The first call for BOSE was made in 2017 by Porru et al. [22]. They highlight "*the need for new professional roles, enhanced security and reliability, novel modeling languages, and specialized metrics*", and propose "*new directions for blockchain-oriented software engineering, focusing on collaboration among large teams, testing activities, and specialized tools for the creation of smart contracts*". They also suggest the adaptation of existing design notations, such as UML, the Unified Modelling Language [23] to unambiguously specify and document dApps.

The book by Xu et al. is perhaps the most complete overview of the engineering aspects of blockchains to date [24]. Among others, it deals with some SE issues, such as the evaluation of the suitability to use a dApp or not, the selection and configuration of the proper blockchain solution (public, permissioned, private), a collection of patterns for the design of blockchain-based applications, and even model-driven generation of SC code. Some of the topics of the book were introduced previously in [25].

Wessling et al. propose a method to find how the architecture of an application could benefit from blockchain technology. They identify the actors involved and how they trust each other's to derive a high-level hybrid architecture of a blockchain-based application [26].

Fridgen et al. propose an approach for eliciting use cases in the context of blockchain-based applications, applying action design research method. Their method is evaluated in four distinct case studies regarding banking, insurance, automotive and construction [27].

Jurgelaitis et al. propose a method based on Model Driven Architecture, which could be used for describing blockchain-based systems using a general language in order to facilitate blockchain development process [28].

A paper by Beller and Hejderup [29] is worth mentioning, though it does not really advocate to use SE practices to develop blockchain-based applications. Instead, it is about "*how blockchain technology could solve two core SE problems: Continuous Integration (CI) Services such as Travis CI and Package Managers such as apt-get*". The use of SCs to manage agile development, including the automated compensation of developers when their software passes acceptance tests was also proposed by Lenarduzzi et al. [30].

Chakraborty et al. used an online survey to get answers from 156 active blockchain software (BCS) developers, finding that "*standard software engineering methods including testing and security best practices need to be adapted with more seriousness to address unique characteristics of blockchain and mitigate potential threats*" [31].

The same authors published an extended version of the same research, further highlighting that there is a need for "*an array of new or improved tools, such as: customized IDE for BCS development tasks, debuggers for smart-contracts, testing support, easily deployable simulators, and BCS domain specific design notations*" [32]. They found that most BCS developers feel that BCS development is different from traditional one, due to the strict and non-conventional security and reliability requirements, and to other unique characteristics of the dApp development domain (e.g., immutability, difficulty in upgrading the software, operations on a complex, secured, distributed and decentralized network). As anticipated in the Introduction, these findings confirm the expedience to devise a software engineering process such as ABCDE for BCS development.

2.3.2 Security for dApps

Regarding dApp security, many publicly available documents, and scientific papers have been already published. Among the most recent ones, the survey of Praitheeshan et al. analyzes the literature about Ethereum smart contract security, summarizing the main security attacks against SCs, their key vulnerabilities, the security analysis methods and tools [33]. They classify analysis methods in static analysis, dynamic analysis, and formal verification, and discuss the relative pros and cons of these classes, also providing a large bibliography with 160 references.

Huang et al. deal with SC security in a broader way, considering also Hyperledger security, and performing a survey from a software lifecycle perspective [34]. After a classification of security issues in SCs, both in Ethereum and Hyperledger Fabric, they consider the security activities according to the various phases of dApp development (design, implementation, testing before deployment, and runtime monitoring), quoting several references and giving practical advice. These two papers together include references to virtually all the work which have been published about SC security to date.

The works on SC security consider in depth the various kinds of attacks and vulnerabilities of dApps, and how to find and mitigate them. However, they typically do not take an overall approach to secure software development life cycle. This is a relatively recent field, whose forefront representatives are Microsoft's Security Development Life cycle (SDL), OWASP's Comprehensive, Lightweight Application Security Process (CLASP) and McGraw' Touchpoints [35].

Though secure software development mostly prefers waterfall-like methodologies, it can be performed also with agile processes [36].

2.3.3 Domain-specific UML additions

Various papers have been published to suggest upgrades of Unified Modeling Language [23] notation to enable it to better represent specific application fields.

Baumeister et al. described an extension of UML for Hypermedia design, through the addition of a new Navigational Structure Model and new stereotypes [37].

Baresi et al. [38] extend and customize UML with web design concepts borrowed from the Hypermedia Design Model. Hypermedia elements are described through appropriate UML stereotypes.

Rocha and Ducasse [39] study SC design and compare three complementary software engineering models - Entity-Relationship diagrams, UML and BPMN. To better represent SC concepts, they propose a simple addition to UML Class Diagrams, that is a small "chain" icon in the UML class representing a contract as a notation to more easily identify it as a blockchain artifact.

2.4 Proposed Method for dApp Development

2.4.1 Rationale and motivation

This approach, ABCDE, takes into account the substantial difference between developing traditional software (the App System) and developing smart contracts, and separates the two activities.

For both developments, ABCDE takes advantage of an agile approach, because agile methods are suited to develop systems whose requirements are not completely understood since the beginning, or tend to change, as it is the case of dApps. This ruled out the use of plan-driven methods such as waterfall, and iterative-incremental methods relying on longer iterations.

ABCDE is an agile method based on Scrum [17], due to Scrum's simplicity and popularity - Scrum is by far the most used software development method [40]. In Scrum, a subset of USs is implemented at each iteration.

Also a Lean-Kanban approach would be feasible, implementing the USs in a continuous flow, with the *work in progress* controlled by the Kanban board [41]. In this case, the board should show, in different "lanes", the USs of both the SC system and the App System. However, because many dApp development projects are new, and thus being built from scratch and with a dedicated team, Scrum is more suited than Kanban - which is instead very suited for teams working concurrently on multiple projects.

From Scrum, ABCDE keeps the requirement gathering with user stories, the iterative-incremental approach, the key roles, and the meetings (sprint planning, daily Scrum, sprint review, and sprint retrospective). The main differences with Scrum are:

- the separation of development activities in two flows, each performed iteratively, with integration activities every 2-3 iterations;
- clarification of the activities that must be performed to design, develop, test and integrate smart contracts and dApp system - this is not included in Scrum;
- emphasis on documenting the smart contracts using formal diagrams, to help development, security assessment, and maintenance - these diagrams are not intended to be exhaustive, but are not required in Scrum;
- specific activities related to security assessment and gas optimization.

To document in a structured way the smart contracts, I found very useful some UML diagrams, properly modified, which are described in section 2.4.3. I used UML because it is by far the most used modeling language in software engineering, and is provided of the "stereotype" construct which enables to add easily the required features to the diagrams. UML provides standard diagrams to effectively model both the data structure of smart contracts (class diagram), and the exchange of messages between the entities of the dApp system (sequence diagram).

Eventually, my and my team’s experience in software quality assessment, made us appreciate the systematic use of patterns and checklists. These tools greatly help developers to proceed in a structured and systematic way. For this reason, we used this approach for security assessment and gas optimization, starting from an accurate literature investigation on the subject. Sections 2.4.4 and 2.4.5 describe in detail these components of ABCDE.

2.4.2 The process

As written before, ABCDE is based on Scrum. The key roles of Scrum, and consequently of ABCDE, are Scrum Master (which might be called *ABCDE Master*), Product Owner and Team. These roles are well known, so I will not describe them in detail.

The steps of the proposed ABCDE design method, which is currently focused on Ethereum blockchain and Solidity language, are shown in Figure 7. Note that most steps are in fact performed many times, because the approach is iterative and incremental.

In the figure, the pink circles represent sprint planning meetings (SPM) held at the beginning of each sprint (iteration), and sprint review meetings (SRM) held at the end of sprints. Daily scrums (stand-up meeting held each day) and retrospective meetings are not reported.

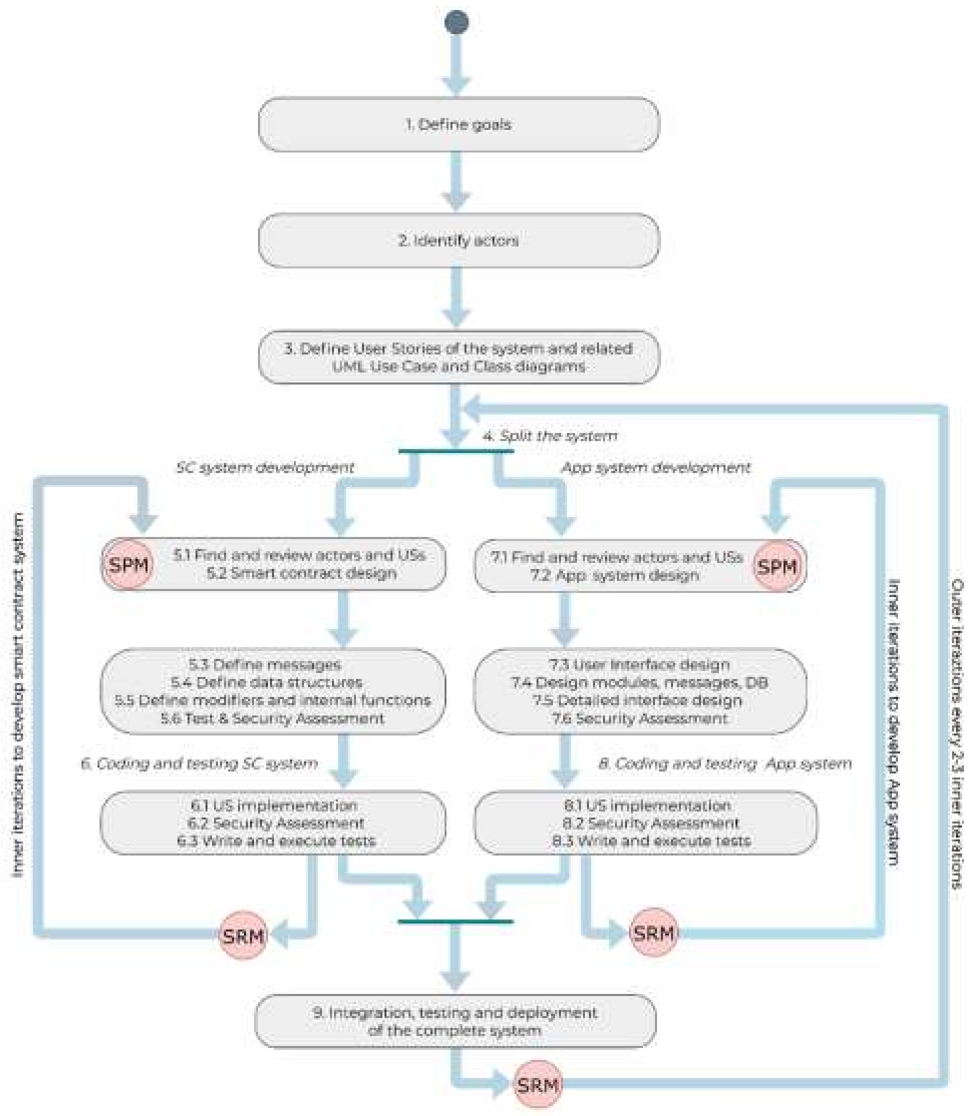

1. Define goals
2. Identify actors
3. Define User Stories of the system and related UML Use Case and Class diagrams
4. Split the system
SC system development
App system development
SPM
5.1 Find and review actors and USs
5.2 Smart contract design
7.1 Find and review actors and USs
7.2 App system design
SPM
5.3 Define messages
5.4 Define data structures
5.5 Define modifiers and internal functions
5.6 Test & Security Assessment
7.3 User Interface design
7.4 Design modules, messages, DB
7.5 Detailed interface design
7.6 Security Assessment
6. Coding and testing SC system
8. Coding and testing App system
6.1 US implementation
6.2 Security Assessment
6.3 Write and execute tests
8.1 US implementation
8.2 Security Assessment
8.3 Write and execute tests
Inner iterations to develop smart contract system
Inner iterations to develop App system
Outer iterations every 2-3 inner iterations
SRM
SRM
9. Integration, testing and deployment of the complete system
SRM

Figure 7 The proposed ABCDE process; the circles represent the Scrum meetings.

In deeper detail, the proposed development process is the following:

1. **Goal of the system**. Write 10-30 words summing up the goal, and display them in a place that is visible to the whole team. This is a practice that, as far as I know, was introduced by Coad and Yourdon in their 1991 book on object-oriented analysis [42], and that I always found useful. It has some similarities with the "Sprint Goal" that Scrum method prescribes to find and make visible to the team, at the beginning of each iteration [17], but here the goal is for the whole system.

2. **Find the actors**. Identify the actors who will interact with the dApp System. The actors are human roles, and external systems or devices that exchange information with the dApp to build.

3. **User Stories**. The system requirements are expressed as user stories (USs) [18], to be able to follow the classical agile approach for project management, used in Scrum [17].

In this step, the dApp System under development should be considered in full. The decision to develop it using a blockchain, a set of servers, possibly in the cloud, or another architecture, is not important here. At this point, I found useful, though not mandatory, to use a UML Use Case Diagram to graphically show the relationships among the actors and the USs. If the decision is taken to implement the system using a blockchain, for instance by applying the decision framework proposed by Scriber [43], the following steps are taken.

4. **Divide the system in two subsystems**.

   - The smart contracts running on the blockchain (Steps 5-6).
   - The App System, that is the external system that interacts with the blockchain, creating and sending transactions, and monitoring the Events that may happen when a smart contract executes a function (Steps 7-8).

   At this point, an architecture of the whole system should be drafted, highlighting what data should be put on-chain and what should be placed off-chain. The guideline is that SCs should manage the data and processing that need to be transparent and immutable for the dApp to be trusted by its actors. This includes the management of actors' identity. All other data, processing and user interfaces should be managed off-chain. In the case of data which must be trusted, but cannot be stored in the blockchain due to its transparency, data privacy can be achieved using the "Off-Chain Data Storage" pattern [24]. Leakage of transaction volume and parties involved might still be possible, and can be avoided by further obfuscating techniques.

5. **Design of the smart contracts**. This step is about designing the SCs, using in this case the Solidity language. This activity has very peculiar characteristics with respect to standard software design, as highlighted by [31]. The activity is performed through iterations that include coding and delivering increments of SCs, which are the USs chosen for each iteration. It is divided in sub-steps, which we explicitly consider and list following a logical sequencing (but which should not necessarily be performed in a "waterfall" sequence). These sub-steps, as well all the sub-steps of the following main steps, derive from our experience in smart contract and dApp development, and from discussions had with many dApp developers. They are the following:

   5.1 Replay Steps 2 and 3 (finding Actors and USs) by focusing only on actors directly interacting with the SCs. If external SCs are used by the SCs of the system under development, they should be included among the actors. For each user story defined in this step, define also the related acceptance test(s).

   5.2 Define broadly the SCs composing the SC subsystem. For each SC, state its responsibilities to store information and to perform computations, and the related collaborations with other SCs. For non-trivial systems, you will typically need various interacting SCs. Also consider the use of inheritance for abstracting common features of SCs. Describe in detail the collaborations with external SCs, including libraries. UML class diagrams with proper additions will be used, as shown later in section 2.4.3.

5.3 Define the flow of messages and Ether transfers among SCs, external SCs and the App System. Use augmented UML sequence diagrams to document these interactions, if they are non-trivial (see section 2.4.3). If needed, define the state changes of SCs using UML statecharts.

5.4 Define in detail the data structure of each SC, its external interface (Application Binary Interface, ABI) and the relevant events that can be raised by it.

5.5 Define the internal, private functions and the modifiers - special functions that usually test the preconditions needed before a function can be safely executed

5.6 Define the tests and perform the security assessment practices. This is a very important step because, as already explained above, most SCs are very critical and deal with money. Sec. 2.4.4 will describe in deeper detail the security assessment used for Ethereum SCs.

6. **Coding and testing the SC system**. Following the agile approach, the SC system is built and tested incrementally. The coding and testing activities are:

6.1 Incrementally write and test the SCs. Owing to the strict security requirements, typically this activity cannot be performed in a strict incremental way, just implementing one user story after another. Instead, starting from the data structure and interfaces of SCs, the overall kernel SC architecture is implemented and tested first. This can be accomplished by using special "user stories" which are not the description of the interaction with users, but are about the implementation of the architecture of the system. Then, complementary USs can be added

6.2 Perform the security assessment and gas optimization of the code written for the increment (see Tables reported in [44] and in section 2.4.5).

6.3 Write automated Unit Tests (UTs) and Acceptance Tests (ATs) for the SCs and USs implemented, respectively. Add the new tests to the test suite. The most used testing environments for Solidity is Truffle [20]. Run the whole test suite to make sure that the additions did not break the system.

7. **Design of the external interaction subsystem (App System)**. This step is about designing the App System, which interacts with the users and devices, send messages to the blockchain, and can manage its own repositories (data bases and/or documents). This activity is very similar to designing a standard web application. It just adds another actor - the blockchain - which can receive (but cannot send) messages, and can raise events. Note that also in this case we must be very careful about security aspects. In fact, often the hacks of dApps systems are made exploiting App System weaknesses, rather that SCs' ones.

7.1 Redefine the actors and the USs for the App System, starting from those gathered in Steps 2 and 3, adding the new actors represented by the SCs that interact with the App System. Define the acceptance tests of the App System.

7.2 Design the high-level architecture of the App System, including server and client tiers, and detail the way it accesses the blockchain, setting up and running one or more nodes, through an external provider, or using a standard wallet.

7.3 Define the UI of the App System, typically with a responsive approach, so that it can run on both mobile terminals and PCs. Having a fancy UI is of paramount importance to achieve the market success of the application. I suggest to perform UI design using a well-known standard approach, such as Usage-Centered Design [45] or Interaction Design [46].

7.4 Define how the App System is decomposed in modules, their interfaces and the flow of messages between them. Define, if needed, the state diagrams of the modules, and the actions they take when events are raised by SCs. Define the structure and memorization of permanent data. Select which data are anchored to the blockchain, by notarization of their hash digest through the "Off-Chain Data Storage" pattern [24]. Define the structure of the data or classes of the App System, including the flow of data and control between modules. The interactions with the SCs must be consistent with the analysis of Step 5.3. This design activity is not performed up-front, but through iterations that include coding and delivering increments of the App System, implementing USs chosen for the iteration. Due to the strict security requirements, this design phase must be quite detailed, and made consistently with the corresponding activities of SCs design. UML class and sequence diagrams can help to design and document also this system.

7.5 Perform a security assessment of the external system, as described below in section 2.4.4.

8. **Coding and testing the App System**. In parallel to the SCs system, the App System is built and tested, using the same approach of SCs development (Scrum or Lean-Kanban). If the developments of SCs and App System are made iteratively, every two or three iterations the results of the two branches must be integrated, as shown in Figure 7. If a continuous-flow, Lean-Kanban approach is performed, the integration should happen at the completion of a given set of USs, in both branches; it will be activated by a specific user story put on the Kanban board. The activities happening in parallel are:

    8.1 Incrementally implement the USs of App System. This step belongs to the "right flow" of ABCDE (see Figure 7), and does not differ from the implementation of a web application.

    8.2 Perform the security assessment of the code written for the increment.

    8.3 Write automated Unit Tests (UTs) and Acceptance Tests (ATs) for the USs implemented. Add the new tests to the test suite. Run the whole test suite to make sure that the additions did not break the system.

9. **Integrate, test and deploy the dApp System**. To integrate SCs and App System, the overall systems built up to that moment must be deployed into a local or a testnet blockchain, and integration tests must be run to check whether all the components

interact together as expected (e.g., events raised by SCs are collected by the App System, messages sent by the App System activate blockchain transactions that are validated and correctly executed, and so on).

2.4.3 UML diagrams for modeling SCs

As written before, the most popular blockchain for dApp development is presently Ethereum, and the most used language is Solidity [47]. This language is object-oriented (OOPL) because smart contracts are defined similarly to classes - they have internal variables, and public and private functions able to access these variables. However, Solidity has no *true* classes, but only smart contracts. Each SC can inherit from one or more other SCs. With respect to a standard OOPL, Solidity adds specific concepts like events and modifiers, and exhibits strong limitations in the types available for the SC data structure, and in the management of collections of data - the only collections available so far are the array and the mapping. In the followings, I will describe an adaptation of UML diagrams specific for Solidity 0.8. Possible modifications and extensions for other SC languages will be discussed in the section about future developments.

When designing and documenting SCs, graphic diagrams can be very useful to highlight the connections and the exchange of messages. To this purpose, I advocate the use of a subset of UML diagrams, being UML the universal standard for software design diagrams. Note that some specific concepts must be introduced to account for peculiar SC features. Luckily, UML has an extensibility mechanism called *stereotype*, which can be used to introduce new concepts, through tagging.

The UML diagrams considered to model SCs are Class diagrams and Sequence diagrams. Also, UML Statecharts can be used to graphically represent the various states of a SC, or of a App System module and its transitions. Statecharts, however, do not need any specific stereotype. I already suggested to use also the Use Case diagrams to graphically show actors and related USs (in place of Use Cases).

The Class diagram enables to represent the structure and relationships of SCs. Table 2 shows the stereotypes introduced in UML class diagrams in order to tag the SC specificities, and their description.

A special kind of transaction is used to create a SC, after its source code has been compiled to bytecode. The other two kinds of transactions are the transfer of Ethers, and the invocation of a function on an existing SC (message).

Table 2 Additions to UML class diagram (stereotypes).

Stereotype	**Position**	**Description**
<<contract>>	Class symbol - upper compartment	Denotes a SC. May also be <<abstract contract>>
<< interface>>	ditto	A kind of contract holding only function declarations

<< library contract>>	ditto	A contract taken from a standard library
<< enum>>	ditto	A list of possible values, assigned to some variable. The values are listed in the middle compartment. There is no bottom compartment (holding operations).
<< struct>>	ditto	A record, able to hold heterogeneous data. The fields are listed in the middle compartment. There is no bottom compartment.
<<event>>	Class symbol - middle compartment	An event that can be raised by a SC's function, signaling something relevant to external observers.
<< modifier>>	Class symbol – bottom compartment	A particular kind of guard function, called before another function.
<< array>>	Class symbol, middle compartment, or role of an association	The multiple variable, or the 1:n relationship, is implemented using an array.
<< mapping>>	ditto	The multiple variable, or the 1:n relationship, is implemented using a generic mapping.
<<mapping [*address*]>>	ditto	A multiple variable, or the 1:n relationship, which is implemented using a mapping from an Ethereum address to the value.
<<mapping [*uint*]>>	ditto	A multiple variable, or the 1:n relationship which is implemented using a mapping from a unsigned integer to the value.

To address the need to manage complex data, Solidity has the "struct" construct. The relationships among SCs and/or structs can be effectively captured by a UML class diagram:

- the multiple inheritance among SCs is the same as with classes;
- when a SC sends a message to another SCs, they can be linked using an association (if they are logically associated), or a dependence;
- item structs and enums can be included in the data structure of a SC, and this relationship is modelled using a composition.

A specific concept of Solidity is that of *events*, raised when something relevant happens, which can be caught by external observer programs. Remember that SCs cannot directly invoke functions of external systems, and thus events are a mean for SCs to communicate with the external world.

Another peculiar concept of Solidity are the *modifiers*. These are Boolean functions called before a function is executed. They are able to check constraints, and possibly to stop the function execution.

The last four stereotypes of Table 2 are about Solidity collections. Owing to the limitations of blockchain storage, Solidity allows only two kinds of collections - the array and the mapping. These stereotypes denote the kind of collection used for multiple variables of a data structure (middle compartment of UML class symbol), or for implementing an association, aggregation or composition. The array is an ordered set of values, indexed by their position, as in most computer languages. The corresponding stereotype is "<<array>>".

The mapping is able to store key-value pairs - the keys being stored as hash values of the actual keys. Given a key, a mapping can efficiently retrieve the value, but it is unable to iterate on its elements, both keys and values. Given the importance of the mapping in Solidity, I introduced three stereotypes to represent a mapping, denoted by the homonymous keyword. The first is the generic mapping; the second is the mapping having an Ethereum address as key, which is very used. The third refers to a common Solidity pattern - using as keys positive, sequential integers, so that it is possible to iterate over them.

Another UML diagram very useful to represent the interactions among SCs and external actors is the Sequence Diagram, used in UML to model messaging. In a blockchain, the relevant messages are related to the transactions, which are sent from external actors, or from SCs to other SCs. Remember that messages are synonyms of "calls of public functions".

A characteristic of Ethereum main net is that messages sent to a SC through a transaction take time (typically 15-20 seconds or more) to be answered. However, if a message is sent to another SC during the execution of a function of a given SC (Contract Internal Transaction), the time delay is negligible. This happens because the EVM, during the execution of the calling function, is able to locate in the blockchain and call any other SC. To explicitly show this difference, which can be very important for response time, security and gas consumption, I introduced the stereotypes <<trans-msg>> and <<internal-msg>> tagging the message calls sent through a transaction, and directly by a SC, respectively.

Another peculiarity of Ethereum is that a SC function which does not change the Blockchain is called a "view" function and can be called immediately and at no cost. Again, this is because the EVM can locate the SC in the blockchain, verify that the function is "view" and call it very quickly, using a negligible amount of resources. All other messages are executed only if proper gas is paid.

Another kind of message that can be sent is the transfer of Ethers from an address to another. To represent this transfer, I use the Return Message of UML (a dashed arrow), tagged with the stereotype <<ethers>>.

Finally, the <<fallback>> stereotype tags the homonymous special function of each SC, which is called whenever a message is not matched, or an Ether transfer fails. This function implements recovery procedures and is particularly critical for security.

The Sequence Diagrams represent the message exchange among external actors and SCs, all called *participants*, in each scenario. The messages between external actors follow the usual UML notation. An external actor, however, can also send Ethers to another. This notation allows the use of standard frames and fragments, such as "alt", "opt" for condition testing, "loop" for loops and "par" for fragments running in parallel. Table 3 reports the stereotypes introduced in UML Sequence diagrams to identify the participants sending messages from their unique address, and the kinds of messages they exchange.

Table 3 The stereotypes added to UML Sequence diagrams.

Stereotype	Position	Description
<<person>>	Participant box	A human role who posts transactions using a wallet or an application.
<<system>>	ditto	An external software system, able to send transactions to the Blockchain.
<<device>>	ditto	An IoT device, able to send transactions to the Blockchain.
<<contract>>	ditto	A SC belonging to the system.
<<external contract>>	ditto	A SC external to the system.
<<oracle>>	ditto	A particular type of SC, which holds information coming from the external world, provided by a trusted provider.
<<account>>	ditto	An Ethereum address, just holding Ethers. It can only receive or send Ethers, when its owner activates the transfer.
<<wallet>>	ditto	An Ethereum wallet, holding the private keys to access addresses, able to send transactions and to interface with its owner.
<<trans-msg>>	Message	The message is sent using an Ethereum transaction.
<<internal-msg>>	Message	The message is sent by a SC, so it is executed immediately.
<<view>> or <<pure>>	Message	The function called is of type "view" or "pure", so it costs no gas.
<<fallback>>	Message	Call to the fallback function. Only called by a SC on itself.
<<ethers>>	Return Message	The dashed arrow represents a transfer of Ethers, and is can be drawn also as a stand-alone message.

2.4.4 Security assessment for Smart Contracts

Assessing and defining patterns of good programming practice for SCs for granting security in dApps is a difficult task and is an ongoing area of research. Nevertheless, based on the programmers' experience and on recent exploited weaknesses - very (in)famous and critical also for the amount of real money involved-, some major suggestions for security assessment in SCs have been identified and discussed among the Solidity developer community. In fact, Ethereum and Blockchain ecosystem are highly new and still somewhat experimental. In addition, SCs are often designed to handle and transfer significant amount of money (in cryptocurrency, but easily exchangeable to real money). Therefore, it is necessary that they correctly achieve their purposes, but it is also crucial that their execution is secure against attacks.

A sound method for dApp development cannot overlook security. While various research papers discuss smart contract design patterns [48] [49] [50] or apply them to specific domains [51] [52], the goal is to provide users with a guide that helps them during the smart contract development process, allowing developers to easily verify if they applied all the relevant security patterns and best practices to their smart contracts.

Following a secure software development lifecycle approach, ABCDE does not limit security assessment to testing, or to a specific phase performed after development, but it introduces security assurance practices in all three phases of design, coding and testing. Moreover, ABCDE stresses that the first and foremost concept in security management is to have a security mindset. The development team(s), and the whole organization, must be fully aware of the importance of security and protection from attacks.

Since ABCDE is an agile process, it is based on principles and practices such as: maximize communication, short iterations, refactoring, continuous testing, simplicity, intention-revealing code, use of simple tools. All these practices surely help security. However, Agile means also incremental development, where USs are continuously completed, added to the current working system and tested. This greatly helps productivity, but might be at the expense of security, because there is the risk that these continuous additions may introduce unwanted side effects, and even security breaches.

In this section I present the security pattern collection and the three security checklists to be performed during the different phases of the development process. Section 2.4.4.1 defines the critical issues regarding the safety of a dApp and provides general guidelines specifically related to SC security. Sections 2.4.4.2 and 2.4.4.3 present the methodology for collecting patterns. Section 2.4.4.4 discusses the abstract security patterns. The remainder of the section presents the three checklists for the design (section 2.4.4.5), coding (section 2.4.4.6), and test (section 2.4.4.7) phases. Finally, section 2.4.4.8 presents other design patterns discussed in literature.

In order to both keep the checklists updated and provide them as a tool that can be easily used by developers, I also released the checklists in a spreadsheet file available at the following link: http://tiny.cc/security_checklist [13]. These checklists can be customized, removing the not relevant parts, on a project-by-project basis.

2.4.4.1 General concepts of dApp security

The critical issues regarding the safety of a dApp can be divided in three areas:

- *Issues related to Blockchain itself*: the blockchain itself could be attacked. It is known, for instance, that blockchains using proof-of-work for block generation are subject, at least theoretically, to the so-called "51% attack". Those based on proof-of-stake are vulnerable to other types of attack, for example to "fake stake attack". Using Ethereum technology, the use of the main net lowers the probability of a "51% attack", given the number and the computing power fielded by the miners. Instead, using Ethereum Classic blockchain, a fork derived from Ethereum in 2016, the probability is higher because its miners' computer power is much lower. Using a permissioned blockchain, for instance Ethereum Parity "proof-of-authority", the blockchain security depends on the honesty and reliability of the validating members, and on their control over their respective IT services. Clearly, this kind of attacks are more a problem of design choice of the technology to be used than of proper dApp design, so their prevention goes beyond the scope of this thesis.
- *Issues related to SCs*: the most critical part of a dApp are the SCs, whose bytecode is publicly available, and exposed to all possible exploits. Moreover, developers often lack

a full knowledge about implementation and usage of SCs, because this technology is in its early stage, it is evolving fast and is different from traditional development. In literature there are several analyses of possible vulnerabilities related to both Ethereum virtual machine and Solidity language [33] [34] [53]. These are a good starting point for providing a checklist of patterns to verify the SCs under development.

- *Issues related to the App System*: The App System is composed of the server and client side of the dApp, interacting with the SCs on one side, and with human actors, IoT devices and other systems on the other side. It must be designed and implemented with care, but it is somewhat less critical, provided that all best practices related to the security of Web applications are used; a special emphasis must be made to safeguard the access to the private keys of the various actors.

In this section I tackle the issues related to SCs development which are the most critical, common and interesting for blockchain software developers. A good starting point to focus on security are the Top 10 Proactive Controls of OWASP organization [54]. Among them I identified those most relevant for dApp security, often neglected in SCs development, and report them ordered by importance:

- C1: **Define Security Requirements**. This looks straightforward, but it is often underestimated. You must explicitly define the security requirements needed for your system. The requirements can be written as User Stories, or as non-functional features, and should have acceptance tests in the form of test cases to confirm these requirements have been implemented.
- C2: **Leverage Security Frameworks and Libraries**. Don't write everything from scratch, but reuse software that is security-hardened, is coming from trusted sources and is maintained up to date.
- C5: **Validate All Inputs**. This should be performed for user inputs on server-side, because client-side validation can be bypassed. Also, let the SC itself perform validation of key data sent to it through messages.
- C6: **Implement Digital Identity**. In a dApp environment, digital identities are guaranteed by addresses and by the ownership of the relative private key, so this control is quite straightforward. Nevertheless, specific checks for address ownership must be implemented to grant SCs security from unauthorized uses.
- C7: **Enforce Access Controls**. SC can check access levels of addresses through a mapping, and act accordingly.
- C8: **Protect Data Everywhere**. Be aware that data stored in a SC are always accessible to read, independently of their visibility.
- C10: **Handle All Errors and Exceptions**. It is known that even small mistakes in error handling or forgetting to handle errors can lead to catastrophic failures in distributed systems. This is particularly true for SCs.

The general guidelines reported by the Solidity best practices [55], section "General Philosophy", which complement OWASP ones, are specifically related to SC security:

1. **Prepare for failure**. Be able to respond to errors, also in the context of SCs, which cannot be changed once deployed.
2. **Rollout carefully**. Try your best to catch and fix the bugs before the SC is fully released. Test contracts thoroughly and add tests whenever new attack vectors are discovered.
3. **Keep SCs simple**. Complexity increases the risk of errors, so ensure that SCs and functions are small and modular, reuse SCs that are proven, prefer clarity to performance.
4. **Keep up to date**. Keep track of new security developments and upgrade to the latest version of any tool or library as quickly as possible. It can be hard to enforce this security pattern once the SC has been deployed, since the code cannot be directly updated. Thus, special care must be devoted to this pattern before deploying.
5. **Be aware of blockchain properties**. While your previous programming experience is also applicable to SC programming, there are several pitfalls to be aware of. An example can be the well-known Parity Wallet “Hack” occurred in November 2017, where the (apparently incidental) use of a self-destruct function by a user who took ownership of the library frozen all Parity multisig wallets and all the cryptocurrencies kept in there.

Although the literature provides various lists of guidelines and best practices to improve security, it is not trivial to apply them to the SC development process. For this reason, the focus is to provide users with checklists that can be easily adopted during the development activity. In the following, I focus on security assurance practices regarding SC design, coding, and testing. Indeed, issues related to SCs are the most critical and less studied among the three categories of issues cited above. Specifically, performing this analysis, I considered both the Proactive Controls of OWASP, and the general guidelines reported above.

2.4.4.2 Security: patterns and best practices

The aim is to provide the reader with an easy way to verify that all the known security issues are managed. Several related works provide various practices to mitigate these issues. They often refer to all these practices as *patterns*. Indeed, most of these items are very simple and not structured to be properly called patterns.

Accordingly, I divide them into two different categories: *abstract security patterns* and *best practices*. The first category includes patterns that are not related to a specific step of the development process. The second category includes those items related to a specific phase of the smart contract development process. Moreover, from the best practices, it is easy to extract checklists to be used to perform security assurance during each development phase. The complete checklists is provided online at http://tiny.cc/security_checklist [13]. These checklists can be customized based on the project requirements. These practices should complement a broader software development process for producing dApps, such as an agile one.

Table 4 collects the abstract security patterns. For each pattern, I provide a brief description of both the problem addressed and its solution, along with a list of references to the papers and articles discussing it. A short discussion is also presented in section 2.4.4.4. Tables Table 5, Table 6 and Table 7 present the security assurance practices I propose. They describe the checks to be performed (column *Name*), a short description of the vulnerabilities and how to avoid them, one or more references to learn more about the problem and a reference to the related security patterns shown in Table 4.

Table 4 Abstract security patterns.

ID	Name	Description	Ref.
CEI	*Check Effect Interaction*	When performing a function in a SC: first, check all the preconditions, then apply the effects to the contract's state, and finally interact with other contracts. Never alter this sequence.	[49]
PD	*Proxy Delegate / Decorator*	Proxy patterns are a set of SCs working together to facilitate upgrading of SCs, despite their intrinsic immutability. A Proxy is used to refer to another SC, whose address can be changed. This approach also ensures that blockchain resources are used sparingly, thus saving GAS.	[56] [57] [51] [52] [12]
AU	*Authorization*	Restrict the execution of critical methods to specific users. This is accomplished using mappings of addresses and is typically checked using modifiers.	[48]
OW	*Ownership*	Specify the contract owner, which is responsible for contract management and has special permissions, e.g., it is the only address authorized to call critical methods. This pattern can be seen as a special instance of the authorization pattern.	[48]
OR	*Oracle*	An oracle is a SC providing data from outside the blockchain, which are in turn fed to the oracle by a trusted source. Here the security risk lies in how the source can be actually trusted.	[48] [50]
RO	*Reverse Oracle*	A reverse oracle is a SC providing data to be read by off-chain components for checking specific conditions.	[50]
RL	*Rate Limit*	Regulate how often a task can be executed within a period of time, to limit the number of messages sent to a SC, and thus its computational load.	[49]
BL	*Balance Limit*	Limit the maximum amount of funds held within a SC.	[49]
GC	*Guard Check*	Ensure that all requirements on a SC state and on function inputs are met.	[57]
TC	*Time Constraint*	A time constraint specifies when an action is permitted, depending on the time registered in the block holding the transaction. It could be also used in Speed Bump and Rate Limit patterns.	[48]
TE	*Termination*	Used when the life of a SC has come to an end. This can be done by inserting ad-hoc code in the contract or calling the *selfdestruct* function. Usually, only the contract owner is authorized to terminate a contract.	[48]
MH	*Math*	A logic which computes some critical operations, protecting from overflows, underflows or other undesired characteristics of finite arithmetic.	[48]
PR	*Privacy*	Encrypt on-chain critical data improving confidentiality and meeting legal requirements, such as the European GDPR	[50]
REU	*Reusability*	Use contract libraries and templates as a factory for creating multiple instances.	[50]
MU	*Mutex*	A mutex is a mechanism to restrict concurrent access to a resource. Utilize it to hinder an external call from re-entering its caller function again.	[49]

SB	*Speed Bump*	Slow down contract sensitive tasks, so when malicious actions occur, the damage is limited and more time to counteract is available. For instance, limit the amount of money a user can withdraw per day, or impose a delay before withdrawals.	[49]

2.4.4.3 Collecting design patterns

The methodology for collecting design patterns and building the best practices checklists is described in the following steps:

1. I queried on June 1st, 2020 the Web, searching for design patterns and best practices for smart contracts.
2. I then manually inspected the results, collecting several data sources gathering different lists of patterns. Most of the sources are scientific papers, but I also found some forum articles.
3. I built a first list of 54 items potentially relevant to this work, by merging all the lists collected in the previous step.
4. I filtered out the list, by excluding 6 items that, although are useful patterns, are not directly related to security aspects. For the sake of completeness, I decide to include those items in Table 8 and discuss them in section 2.4.4.8.
5. I analyzed all the remaining items and split them into two main categories: 16 *abstract security patterns* and 32 *best practices*.
6. I split the best practices into three groups according to the related phase in the development process: design, coding, testing and deployment.

Summing up, I identified 16 abstract security patterns, and 32 best security practices: 8 for designing, 18 for coding, and 6 for testing and deployment of SCs. The other design patterns not related to security are 6.

2.4.4.4 Abstract security patterns

Patterns are schemes of a standard solution to a recurring problem, with a certain structure. However, the security patterns in the dApps scenario, do not always have a structure and uniformity comparable to traditional design patterns [58]. Some of the presented patterns are variants of well-known traditional design patterns, applied in the blockchain and smart contracts scenario, whereas others are specifically designed to address the peculiarities of Ethereum and Solidity context.

I describe below some of the patterns I deem to be the most significant. A complete description of all the patterns is available in the previously reported Table 4.

An example of traditional design pattern which is useful also in the blockchain context is the *Proxy* pattern. It introduces the possibility of upgrading a contract, which is by nature immutable, without breaking any dependency. The idea is to divide the contract into modules and to use a Proxy contract to delegate calls to specific modules. In this way, indeed, it is possible to modify a contract by implementing a new version and replacing its address with the previous one in the archive stored in the Proxy.

An example of pattern specific of the blockchain context is the *Oracle* pattern, which is introduced to respond to the need to acquire information from the outside world. In fact, a smart contract is by nature isolated, and cannot acquire information directly. This is because network nodes must agree on the state of transactions. To accomplish this, nodes should evaluate only *static* data. On the contrary, the outside world, for instance a website API, could provide different responses to the same query performed by different network nodes, breaking blockchain consensus. An oracle is a smart contract handled by a trusted authority for uploading outside data to the blockchain. This allows network nodes to query the static blockchain data instead of the outside world, overcoming the limits described above.

Another pattern particularly important in a decentralized context is the *Authorization* pattern. Since SCs are publicly accessible to all blockchain participants, it is critical to restrict authorizations to perform specific tasks. Specifically, for each contract method, developers must specify the subset of participants who can call it. Contracts usually define at least one *contract owner*, which is the only entity authorized to call critical methods. Table 6 shows various techniques to handle authorizations properly during the coding phase.

A *Time constraint* defines when the related action is allowed to be performed. Different blockchain nodes could process transactions at different timestamps, due to network latency or further causes. Consequently, part of the network could consider an action as executed beyond the time constraint, and the corresponding transaction could be rejected by the whole network. Accordingly, developers must set time constraints carefully, for instance by ensuring that there is enough time between two consecutive time constraints.

2.4.4.5 Security in the design phase

In the design phase, developers must be aware of, and use security patterns, as reported in references [49] [48] [56], which I refer to. During smart contract design, you must think strategically, and apply patterns and checks regarding the architecture and general modeling of the SCs. Here, you must decide if and how to apply decoupling and fail-safe patterns, such as Proxy [52] and Check-Effect-Interactions [49]. Minimization of dependencies, and careful planning of reuse through inheritance and external libraries is another activity typically performed in the design phase [55]. You should also decide how to manage authorizations to the use of the system, and how to avoid race conditions due to wrong assumptions on system time and transaction ordering [48]. You should carefully plan Ether management, if your SCs must hold, receive and deliver Ethers. To this purpose, it is wise to limit amounts and frequency of Ether withdrawals and use a "pull" approach to it [55].

In the remainder of this section, I describe some of the design best practices. For the sake of brevity, I describe a few of them, while Table 5 shows the full list.

Table 5 Security assurance checklist for the design phase.

Name	**Description**	**Ref.**	**Related Pattern**
Include fail-safe mechanisms	It is important to have some way to update the contract in the case some bugs will be discovered. Incorporate an emergency stop functionality into the SC that can be triggered by an authenticated party to disable sensitive functions. The fail-safe mechanism, if implemented using the *Proxy Delegate* could be also exploited for forwarding calls and data to another contract, which is an updated version of the current one (for instance, a version where the bug has been fixed).	[49]	SB, RL, TE, PD, OW
Never assume that a contract has zero balance	Be aware of coding an invariant that strictly checks the balance of a contract. An attacker can forcibly send ether to any account, and this cannot be prevented.	[55]	CEI, MH, GC
State Channel/Off-chain Support	In some contexts, transactions either have too high fee compared to their value or must have low latency. In these cases, rather than performing each blockchain transaction, it is possible to firstly perform the operations outside the blockchain, and then register all the results batching the requests in a unique blockchain transaction.	[50] [59]	RL
Limit the amount of ether	If the code, the compiler or the platform has a bug, the funds stored in your smart contract may be lost, so limit the maximum amount. Check that all money transfers are performed through explicit withdrawals made by the beneficiary.	[60] [60]	RL, BL, AU
Beware of transaction ordering	Miners have the power to alter the order of transactions arriving in short times. Inconsistent transactions' orders, with respect to the time of invocations, can cause race conditions.	[33]	TC
Be careful with multiple inheritance	Solidity uses the "C3 linearization". This means that when a contract is deployed, the compiler will linearize the inheritance from right to left. Multiple overrides of a function in complex inheritance hierarchies could potentially interact in tricky ways.	[55]	PD, REU
Use trustworthy dependencies	Use audited and trustworthy dependencies to existing SCs and ensure that newly written code is minimized by using libraries.	[55]	REU
Withdrawal from Contracts / Pull over Push	When you need to send Ethers or tokens to an address, don't send them directly. Instead, authorize the address' owner to withdraw the funds, and let s/he perform the job.	[47] [57]	CEI

A *fail-safe mechanism* is a function that allows contract owners to disable specific SC methods. Developers should always design mechanisms to either update or terminate contracts because, due to the immutability of blockchains, SCs cannot be removed once published. The fail-safe best practice can be exploited to accomplish several security patterns. For instance, it can be used for terminating a contract (*Termination* pattern), and optionally for enabling a new version of the contract (*Proxy delegate*). Moreover, this mechanism could be used for slowing down sensitive tasks (*Speed Bump*, *Rate Limit*). Usually, this mechanism is enabled by the contract owner (*Ownership*).

2.4.4.6 Security in the coding phase

Below, I describe some of the most representative coding best practices. The full checklist for security assessment in the coding phase is reported in Table 6.

During coding, one major class of problems derives from *external calls*, namely from functions which recur to others' SC code for completing their execution. In fact, a SC can call another SC, exploiting the execution of code contained in the latter contract. The pattern can be recursive, so the called SC can in turn perform an external call, and so on. Therefore, external calls must be treated like calls to 'untrusted' software. They should be avoided or minimized, because some malicious code could be introduced somewhere in a SC belonging to this path, and any external call represents a security risk. It is true that all external SCs are already present in the blockchain, and thus are immutable. However:

- if their code is not thoroughly checked by a competent professional, a SC might not work as intended;
- if the called SC makes use of the *Proxy* pattern, it can be changed by its author;
- in complex dApps, to avoid rewriting of the whole system in the case of a change, mechanisms to dynamically change the address of the called SC are typically used;
- another typical risk of such contract interaction is *reentrancy*, namely the called contract can call back the calling function before the overall function execution has been completed. This pattern has been performed in the DAO attack.

When it is not possible to avoid external calls, label all the potentially unsafe variables, functions and contracts interfaces as "untrusted".

To prevent these issues, check accurately all preconditions and restrict concurrent accesses to resources, as defined by the *Check-effect-interaction* and *Mutex* patterns.

Another important best practice for SC security and error handling is to *validate inputs* by using *assert()*, *require()* and *revert()* guard functions. They are a very powerful security tool, and are the subject of security pattern *Guard Check* presented in Table 4. In general, use *assert()* to check for invariants, to validate state after making changes, to prevent wrong conditions; if an *assert()* statement fails, something very wrong happened and you need to fix the code. Use *require()* when you want to validate user inputs, state conditions preceding an execution, or the response of an external call. Use *revert()* to handle the same type of cases as *require()*, but with more complex logic [55].

Table 6 Security assurance checklist for the coding phase.

Name	**Description**	**Ref.**	**Related Patterns**
Be careful with external calls	If possible, avoid them. When using low-level call functions make sure to handle the possibility that the call will fail, by checking the return value. Also, avoid combining multiple ether transfers in a single transaction. Mark untrusted interactions: name the variables, methods, and contract interfaces of the	[55]	CEI, MU, GC

	functions that call external contracts, in a way that makes it clear that interacting with them is potentially unsafe.		
Beware of re-entrancy	Never write functions that could be called recursively, before the first invocations is finished. This may cause destructive consequences. Ensure state committed before an external call.	[60] [55]	CEI, MU
Embed addresses to grant permissions	Make sure that critical methods can be invoked only by a specific set of addresses, which belong to privileged users. For instance, each contract has an owner and only this address can invoke certain methods, like the method for updating the address of the owner of the contract.	[50]	AU, OW
Use hash secrets to grant permissions	Sometimes you need to provide authorizations to some authorities whose addresses are not known yet in the developing phase (for instance, they are unknown authorities). Although the *Embed permissions* pattern cannot be applied, hash secrets help providing user permissions without specifying any address. First, generate a secret key and, in the contract, provide permissions by requiring its hash. Then, send (off-chain) the secret key to the authorities you want to grant permissions.	[52] [50]	AU
Use multi-signature	Define a set of entities (or addresses) that can authorize an action and require that only a subset of them is required to authorize the action.	[52] [50]	AU, OW
Avoid using tx.origin for authorizations	*tx.origin* is a global variable that returns the address of the message sender. Do not use it as an authorization mechanism.	[61]	AU
Encrypt on-chain data	Encrypt blockchain data for improving confidentiality and privacy. This is particularly important when actors are in competition.	[50]	PR
Hash objects for tracking off-chain data	Large objects (such as videos) should not be embedded in the blockchain, their hashes can be easily uploaded instead. Hashing objects can be also applied to hide sensitive data in order to meet specific legal requirements, such as the European GDPR.	[50]	PR
Use platform related standards	Use platform related standards, like the ERC (Ethereum Request for Comment) standards, which are application-level blueprints and conventions in the Ethereum ecosystem.	[50]	REU
Prevent overflow and underflow	If a balance reaches the maximum uint value it will circle back to zero; similarly, if a uint is made to be less than zero, it will cause an underflow and get set to its maximum value. One simple solution to this issue is to use a library like *SafeMath.sol* by OpenZeppelin. This issue has been partially solved in Solidity 8.0.	[33]	MH, GC, REU, BL
Beware of rounding errors	All integer divisions round down to the nearest integer. Check that truncation does not produce unexpected behavior (locked funds, incorrect results).	[55]	MH, GC, REU
Validate inputs to external and public functions	Make sure the requirements are verified and check for arguments. Use properly *assert()*, *require()* and *revert()* to check user inputs, SC state, invariants.	[33] [57]	GC
Prevent unbounded loops	When executing loops, the gas consumed increases with each iteration until it hits the block's gasLimit, stopping the execution. Accordingly, plan the number of iterations you need to perform and establish a maximum number. If you still need more iterations, divide computation among distinct transactions.	[57] [12]	RL, BL, TC, TE

Provide fallback functions	The "fallback function" is called whenever a contract receive a message which does not match any of the available functions, or whenever it receives Ethers without any other data associated with the transaction. Remember to mark it as *payable*, be sure it does not have any arguments, has external visibility and does not return anything. Moreover, keep it simple and if the fallback function is intended to be used only for the purpose of logging received Ether, check that the data is empty (i.e. require(msg.data.length == 0)).	[55] [33]	CEI, MU, GC
Check if built-in variables or functions were overridden	It is possible to override built-in globals in Solidity. This allows SCs to override the functionality of built-ins such as *msg* and *revert()*. Although this is intended, it can mislead users of a SC, so the whole code of every SC called from the SC you are writing must be checked.	[55]	GC
Use interface type instead of the address for type safety	When a function takes a contract address as an argument, it is better to pass an interface or contract type rather than a raw *address*. If the function is called elsewhere within the source code, the compiler will provide additional type safety guarantees.	[55]	GC
Be careful with randomness	Random number generation in a deterministic system is very difficult. Do not rely on pseudo-randomness for important mechanisms. Current best solutions include hash-commit-reveal schemes (ie. one party generates a number, publishes its hash to "commit" to the value, and then reveals the value later), querying oracles, and RANDAO.	[48] [34]	OR, REU
Be careful with Timestamp	Be aware that the timestamp of a block can be manipulated by a miner; all direct and indirect uses of timestamp should be analyzed and verified. If the scale of your time-dependent event can vary by 30 seconds and maintain integrity, it is safe to use a timestamp. This includes things like ending of auctions, registration periods, etc. Do not use the *block.number* property as a timestamp.	[33]	TC

2.4.4.7 Security in the testing and deployment phases

In this subsection I focus on the testing and deployment steps and describe some of the best practices shown in Table 7.

As in traditional software engineering, also in the smart contract context *developing unit testing* is important. Currently, there are several techniques for testing smart contracts. ABCDE does not prescribe the use of specific testing practices, such as Test-Driven Design, but I highlight the importance of testing. One way is to use a browser-based real-time compiler and runtime environment for Solidity, such as Remix.

Another way is to *use frameworks for testing*. Presently, among the most popular testing frameworks for Ethereum dApps there are Truffle [20], Embark [62] and Etherlime [63].

Moreover, the Ethereum community operates multiple *test networks*. These are used by developers to test applications under different conditions before deploying them on the main Ethereum network. The most famous ones are Ropsten [64] , Rinkeby [65] and Goerli [66]. Finally, it is possible to set up a local Ethereum blockchain which can be used to run tests, execute commands, and inspect state while controlling how the chain operates. An example of such a software is Ganache [67], from the Truffle suite.

Table 7 Security assurance checklist for the testing and deployment phases.

Name	Description	Ref.
Fix compiler warnings	Take warnings seriously and fix them. Always use the latest version of the compiler to be notified about all recently introduced warnings.	[47]
Lock programs to specific compiler version	Contracts should be deployed with the same compiler version and flags that they have been tested with, so locking the version helps avoid the risk of undiscovered bugs.	[55]
Enforce invariants with assert	An assert guard triggers when an assertion fails - for instance an invariant property changing. You can verify it with a call to *assert()*. Assert guards should be combined with other techniques, such as pausing the contract and allowing upgrades. (Otherwise, you may end up stuck, with an assertion that is always failing.)	[55]
Develop unit testing	Be sure to have a 100% text coverage and cover all critical edge cases with unit tests. Do not deploy recently written code, especially if it was written under tight deadline.	[55]
Use frameworks for testing	When approaching smart contract testing, do not start from scratch but use existing framework for contract testing.	[55]
Use test networks	Before deploying the smart contract in the main network, try it in a public test network or use a software for configuring a private local network.	[55]

2.4.4.8 Other patterns

As described in the methodology (2.4.4.3), 6 more patterns emerged from the literature. Those patterns are not strictly related to contract security but are useful for a good SC design. Therefore, I decided to include them in this work for completeness and briefly describe some of them in the followings. Table 8 shows the full list of those patterns.

The *Publisher-Subscriber* pattern is a well-known design pattern studied in traditional (i.e., not blockchain-based) software engineering. According to this pattern, when a module needs to receive some messages from other software modules (e.g., a smart contract), developers should implement a messaging infrastructure that allows each module to be easily notified when a new message is generated.

The *Tokenisation* pattern suggests using tokens for representing good and services in the blockchain. The Ethereum ecosystem provides standards to handle several types of tokens, such as the ERC20 and ERC721.

Table 8 Other design patterns.

Name	Description	Ref.
Publisher-Subscriber	When a state change must trigger a computation in a different object, implement a messaging infrastructure where the contracts that produce messages (called *publishers*) can generate messages and the other contracts (called *subscribers*) receive them. The pattern, also known as *Observer*, reduces the overhead of constant information filtering.	[51]

Tokenization	Use tokens for transferring digital or physical services. Use standards, such as ERC20 and ERC721.	[50]
X-confirmation	To ensure that a transaction is confirmed (i.e., there is a low probability that a fork happens), wait for new blocks to be added to the blockchain. The number of blocks depends on the adopted blockchain.	[50]
Contract Registry	Often a smart contract needs to interact with other contracts which can be updated over time. A contract registry maps each smart contract to the address of its latest version. Accordingly, when invoking a smart contract, the correct address should be retrieved from the registry.	[50]
Eternal storage / Data Contract	Contract data and logic should be stored into separate contracts. In this way, when the logic needs to be updated (by using a new smart contract), there is no need to migrate old data.	[50]
Abstract factory	Sometimes, systems need to work with groups of related contracts, for instance with contracts which represent various level user account. To keep the system independent from the different contracts, define an abstract contract for creating all the related contracts.	[51] [50] [52]

2.4.5 Gas optimization

Besides security, another important factor of SCs that must be carefully designed since the beginning is their cost. Creating SCs and writing permanent data in a public blockchain can be very costly, so it is important to keep them to a minimum, and to limit the transactions that write or modify these data. Also, the messages exchanged among the App System and the SCs, and among SCs, must be properly designed and well documented.

In this subsection I present a collection of patterns for the design and development of Smart Contracts, with the aim of saving gas. So far, only a few efforts have been made to collect and classify patterns related to this issue, mainly published on blogs or discussion lists [68] [69] [70] [71].

As already discussed in 1.8, during the development and execution of an Ethereum SC there are costs, calculated in gas/gwei; these costs can be divided in fixed costs related to the creation of the SC or to the sending of the transaction; costs related to the permanent storage of the SC state, in storage; costs relating to the storage of temporary variables, necessary for the execution of functions, in memory; costs related to the execution of the operations.

Accordingly to the literature, each pattern is described by:

• a **Name**: it must be simple, focused on the issue and easy to remember. It allows you to unambiguously refer to the pattern;

• the **Problem**: it describes the issue that the pattern resolves. It may include a list of conditions that must be met for the solution to be valid.

• the proposed **Solution**: it clarifies what to do in order to overcome the Problem. It does not describe a specific concrete implementation, but an abstract and precise enough solution, that can be immediately applied, and reused multiple times.

I collected 24 gas saving patterns, which I assigned to 5 categories. The categories resulted from analyzing the collected patterns and observing in which context they are used – for instance, limiting the storage (which is ludicrously costly), or limiting the computations made by function execution. Some patterns might belong to more than one category. I assign it to the most appropriate one.

Applying patterns to an application can better align it with the unique properties provided by the blockchain, overcoming its limitations. The presented patterns are based on multiple sources, such as the Solidity documentation, the study of blogs and discussion forums on Ethereum on the Web, and the examination of existing SCs.

Moreover, in Ethereum the maximum size of the bytecode of a SC is restricted to 24576 Bytes by the standard EIP 170 (see section 13.4.2 of [24]). For complex SCs, that size limit can be hit easily, so many of the gas saving patterns are useful also to make a SC viable.

2.4.5.1 External Transactions

This category, detailed in Table 9, includes patterns related to the creation of contracts and the sending of transactions from external addresses, including JavaScript applications, using Web3.js standard library.

Table 9 External transactions gas-saving patterns.

Name	**Problem**	**Solution**
Proxy	SCs are immutable. If a SC must be changed due to a bug or a needed extension, you must deploy a new contract, and update all SCs making direct calls to the old SC, thus deploying also new versions of these. This can be very expensive.	Use Proxy delegate pattern. Proxy patterns are a set of SCs working together to facilitate upgrading of SCs, despite their intrinsic immutability. A Proxy holds the addresses of referred SCs, in its state variables, which can be changed. In this way, only the references to the new SC must be updated.
Data Contract	When a SC holding a significant amount of data must be updated, also all its data must be copied to the newly deployed SC, consuming a lot of gas.	Keep the data in a separate SC, accessed by one or more SC, using the data and holding the processing logic. If this logic must be updated, the data remain in the Data Contract. This pattern usually is included also in the implementations of the Proxy pattern.
Event Log	Often events maintain important information about the system, which must be later used by the external system interacting with the blockchain. Storing this information in the blockchain can be very expensive, if the number of events is high.	If past events data are needed by the external system, but not by SCs, let the external system directly access the Event Log in the blockchain. Note that this Log is not accessible by SCs, and that if the event happened far in time, the time to retrieve it may be long.

2.4.5.2 Storage

This category, detailed in Table 10, includes patterns related to the usage of Storage for storing permanent data.

Table 10 Storage gas-saving patterns.

Name	Problem	Solution
Limit Storage	Storage is by far the most expensive kind of memory, so its usage should be minimized.	Limit data stored in the blockchain, always use memory for non-permanent data. Also, limit changes in storage: when executing functions, save the intermediate results in memory or stack and update the storage only at the end of all computations.
Packing Variables	In Ethereum, the minimum unit of memory is a slot of 256 bits. You pay for an integer number of slots even if they are not full.	Pack the variables. When declaring storage variables, the packable ones, with the same data type, should be declared consecutively. In this way, the packing is done automatically by the Solidity compiler. (Note that this pattern does not work for Memory and Calldata memories, whose variables cannot be packed.)
Packing Booleans	In Solidity, Boolean variables are stored as uint8 (unsigned integer of 8 bits). However, only 1 bit would be enough to store them. If you need up to 32 Booleans together, you can just follow the Packing Variables pattern. If you need more, you will use more slots than needed.	Pack Booleans in a single uint256 variable. To this purpose, create functions that pack and unpack the Booleans into and from a single variable. The cost of running these functions is cheaper than the cost of extra Storage.

2.4.5.3 Saving Space

This category, detailed in Table 11, includes patterns related to saving space both in Memory and Storage.

Table 11 Memory and Storage gas-saving patterns.

Name	Problem	Solution
Uint vs Uint256*	The EVM run on 256 bits at a time, thus using an uint* (unsigned integers smaller than 256 bits), it will first be converted to uint256 and it costs extra gas.	Use unsigned integers smaller or equal than 128 bits when packing more variables in one slot (see Variables Packing pattern). If not, it is better to use uint256 variables.
Mapping vs Array	Solidity provides only two data types to represents list of data: arrays and maps. Mappings are cheaper, while arrays are packable and iterable.	To save gas, it is recommended to use mappings to manage lists of data, unless there is a need to iterate, or it is possible to pack data types. This is useful both for Storage and Memory. You can manage an ordered list with a mapping using an integer index as a key.
Fixed Size	In Solidity, any fixed size variable is cheaper than variable size.	Whenever it is possible to set an upper bound on the size of an array, use a fixed size array instead of a dynamic one.
Default Value	It is good software engineering practice to initialize all variables when they are created. However, this costs gas in Ethereum.	In Solidity, all variables are set to zeroes by default. So, do not explicitly initialize a variable with its default value if it is zero.

Minimize on-chain data	The gas costs of Storage are very high, and much higher than the cost of Memory.	Minimize on-chain data. The less data you put on-chain in Storage variables, the less your gas costs. Store on-chain only critical data for the SC and keep all possible data off-chain.
Explicitly mark external function	The input parameters of public functions are copied to memory automatically, and this costs gas.	The input parameters of external functions are read right from *Calldata* memory. Therefore, explicitly mark as external functions called only externally.

2.4.5.4 Operations

This category, detailed in Table 12, includes patterns related to the gas used for the operations performed within SC functions.

Table 12 Operations gas-saving patterns.

Name	**Problem**	**Solution**
Limit External Calls	Every call to an external SC is rather expensive, and even potentially unsafe.	Limit external calls. In Solidity, differently from other programming languages, it is better to call a single, multi-purpose function with many parameters and get back the requested results, rather than making different calls for each data.
Internal Function Calls	Calling *public* functions is more expensive than calling *internal* functions, because in the former case all the parameters are copied into Memory.	Whenever possible, prefer *internal* function calls, where the parameters are passed as references.
Fewer functions	Implementing a function in an Ethereum SC costs gas.	In general, keep in mind that implementing a SC with many small functions is expensive. However, having too big functions complicates the testing and potentially compromises the security. So, try to have fewer functions, but not too few, balancing the function number with their complexity.
Use Libraries	If a SC tends to perform all its tasks by its own code, it will grow and be very expensive.	Use libraries. The bytecode of external libraries is not part of your SC, thus saving gas. However, calling them is costly and has security issues. Use libraries in a balanced way, for complex tasks.
Short Circuit	Every single operation cost gas.	When using the logical operators, order the expressions to reduce the probability of evaluating the second expression. Remember that in the logical disjunction (OR, ||), if the first expression resolves to true, the second one will not be executed; or that in the logical disjunction (AND, &&), if the first expression is evaluated as false, the next one will not be evaluated.
Short Constant Strings	Storing strings is costly.	Keep constant strings short. Be sure that constant strings fit 32 bytes. For example, it is possible to clarify an error using a string; these messages, however, are included in the bytecode, so they must be kept short to avoid wasting memory.
Limit Modifiers	The code of modifiers is inlined inside the modified function,	Limit the modifiers. Internal functions are not inlined, but called as separate functions. They are

	thus adding up size and costing gas.	slightly more expensive at run time, but save a lot of redundant bytecode in deployment, if used more than once.
Avoid redundant operations	Every single operation cost gas.	Avoid redundant operations. For instance, avoid double checks; the use of SafeMath library prevents underflow and overflow, so there is no need to check for them.
Single Line Swap	Each assignment and defining variables costs gas.	Solidity allows to swap the values of two variables in one instruction. So, instead of the classical swap using an auxiliary variable, use: (a, b) = (b, a)
Write Values	Every single operation cost gas.	Write values instead of computing them. If you already know the value of some data at compile time, write directly these values. Do not use Solidity functions to derive the value of the data during their initialization. Doing so, might lead to a less clear code, but it saves gas.

2.4.5.5 Miscellaneous

This category, detailed in Table 13, includes patterns that cannot be included in the previous ones.

Table 13 Miscellaneous gas-saving patterns.

Name	**Problem**	**Solution**
Freeing storage	Sometimes, Storage variables are no longer used. Is there a way to take advantage of this?	To help keeping the size of the blockchain smaller, you get a gas refund every time you free the Storage. Therefore, it is convenient to delete the variables on the Storage, using the keyword delete, as soon as they are no longer necessary.
Optimizer	Optimizing Solidity code to save gas in exhaustive way is difficult.	Always turn on the Solidity Optimizer. It is an option of all Solidity compilers, which performs all the optimizations that can be made by the compiler. However, it does not substitute the usage of the presented patterns, most of which need information that is not available to the compiler.

2.5 Experimental Validation

The development process which later was named ABCDE was first devised in 2018 [72], and since then it has been used in several projects carried on in my University’s group, and in firms we are consulting.

Among the projects which were developed, or which are in development, I may quote a system to manage temporary job contracts; a couple of systems to trace the provenance of foods (one of which developed using Hyperledger technology) [73]; two voting systems, one managing voting in firm shareholders' and board of directors meetings, the other for anonymous voting; a system to manage energy exchange in local networks of electricity producers and consumers; a system to automate agile software development [30]; a system to notarize and to manage incentives for check-up visits.

Table 14 Survey on ABCDE Usage.

Nr.	Question	Mean	St. Dev.	Min	Max
1	Years of sw. development experience	8	9.21	2	35
2	Years of agile sw. development experience	4.5	4.54	0	15
3	Years of dApp development experience	2.6	1.70	7 months	5
4	Number of completed dApp projects	2.5	2.44	1	9
5	Number of ongoing dApp projects	1.3	0.91	0	3
6	Number of dApp projects performed using ABCDE	1.9	1.14	1	5
7	ABCDE is overall useful	4.3	0.61	3	5
8	ABCDE is useful for requirements elicitation	4.4	0.65	3	5
9	ABCDE is useful in system design using UML diagrams	4.5	0.66	3	5
10	ABCDE is useful for its iterative/incremental approach	4.0	0.68	3	5
11	ABCDE is useful in security analysis	3.8	0.70	3	5
12	ABCDE is useful in optimization of gas consumption	3.5	0.85	2	5
13	ABCDE is useful in the testing phase	3.8	0.97	3	5
14	ABCDE is useful for integrating SC and dApp systems	3.7	0.95	2	5
15	ABCDE is easy to use	4.4	0.50	4	5

The feedback of dApp developers using ABCDE method was generally positive and was used to improve the method - especially concerning security and gas optimization practices.

In Table 14 I report the results of a survey conducted among 14 developers and graduate students who used ABCDE on at least one dApp project. The scores regarding the features of ABCDE method (questions 7-15) span from 1 (not useful at all) to 5 (very useful); a neutral opinion corresponds to a score of 3. As you can see, the experience in software and dApp development varies greatly. The average number of dApp projects performed by respondents is fairly high (almost 4 projects each, of which 2.5 closed and 1.3 ongoing). The overall satisfaction for ABCDE method is quite high, as well as its ease of use. The strengths of the method are especially in the analysis and design phases, whereas it is less appreciated (but still above sufficiency) in gas optimization, and final integration. The respondents gave also several suggestions on possible improvements to ABCDE, some of which are reported in the final section of this part 2.7.

2.5.1 Building an example dApp

Here I present, as an example of ABCDE usage, a simplified version of a dApp application aiming to implement a decentralized exchange (DEX) for tokens managed on Ethereum blockchain. A DEX is a system enabling the exchange of different tokens between two holders, who interact directly, without intermediaries. I started from the well-known 0x protocol project, the subject of a successful ICO held in 2017. The specification of the DEX can be found in the 0x Whitepaper [74]. I present a simplified version of the whole system. In particular, I dropped the part related to the protocol token (Section 4 of the Whitepaper), and the signing of the offers by traders. In this example, a trading offer is simply posted to the DEX, and who wish to accept it simply sends a transaction to the DEX. The guarantees against frauds are the transparency of the underlying SC, and the hash signature of each offer, which guarantees against fraudulent changes of the offer after it is accepted.

2.5.2 The first steps of ABCDE

For the sake of brevity, I will not present the App System coding phase, and the system integration phase (phases 8 and 9), but I stop at the end the design phases (phases 5 and 7). The steps of ABCDE are presented below.

1. **Goal of the system**. *To manage a decentralized exchange, able to enable pairs of ERC20 token holders to exchange their tokens at an agreed rate on the Ethereum blockchain*.
2. **Actors**. The system has the following actors:
    - **Trader**: owner of tokens, wishing to post an offer, or to accept a posted offer.
    - **Maker**: a trader who posts an offer to sell a given amount of her/his tokens, in exchange to tokens of another type, at a given exchange rate.
    - **Taker**: a trader who accepts the offer of a Maker.
    - **Relayer**: a web system which facilitates signaling between market participants by hosting and propagating an order book of the offers.
    - **DEX**: smart contract(s) on the Ethereum blockchain which accept orders signed by both a Maker and Taker and activate the exchange of tokens.
    - **Token**: a SC on the Ethereum blockchain, managing a given token according to the ERC20 protocol.
3. **User Stories**. Figure 8 shows the actors and the user stories they are involved in, using a UML Use Case diagram, where the use cases are in fact USs. Note that these USs just specify the DEX, and do not depend on the specific technology used to implement it, except for the Ethereum blockchain, which the DEX necessarily must interact with. Given the simplicity of the example, I will not show the USs in detail, since they are self-explaining. In Figure 9 I show the UML class diagram derived by an analysis of the given USs. This diagram is not bound to a specific implementation of the relayer system, but just shows schematically the entities, the data structures and the operations emerging from the USs shown in Figure 8. In short, the system just deals with orders, posted by makers and later accepted by takers, both of who are kinds of traders. The Relayer is the service which publishes the offers and makes possible the exchange of tokens. Both traders and relayer access the smart contracts implementing the tokens in the blockchain.
4. **Divide the system into SC and App subsystems**. In this case the subdivision is trivial, because the Relayer system is a typical web application, whereas the DEX and the Tokens are smart contracts by design. The USs of the external app subsystem are the same of those reported in Figure 8, except those related with direct interaction with the

blockchain, tagged "SC" in the diagram of Figure 8. Also, the USs of the blockchain subsystem are the same of those reported in Figure 8, but that tagged "AS".

5. **Design of the SC subsystem**. The SC system is quite simple, and mainly involves the "DEX" SC, which interacts with the SCs managing the supported tokens to exchange. The data managed by the DEX are the trading fees, the list of supported tokens, and the list of the offers. To hold these lists, we use the "Eternal Storage" pattern, consisting in storing them in an external SC, so that possible changes to the DEX can be managed with no need to store again all these data [75]. Conversely, the use of the Proxy pattern [56] is excluded, because the guarantee that the DEX works properly is given by inspecting its source code. Giving the DEX's owner the ability to change the DEX, leaving the Proxy unchanged, would void this guarantee. I report in Figure 10 the UML class diagrams showing the SCs of the system. This diagram shows some of the specific stereotypes used to document an SC system, as described in section 2.4.3.

The entities shown in the top of the diagram are standard library contract "Ownable", library "SafeMath" and "ERC20" token interface, used in most contracts dealing with tokens. "ERC20" contract represent at least two token contracts active on Ethereum blockchain and managed by the DEX. The DEXStorage contract holds zero or more "Offer" records and is linked to the related DEX contracts. Modifiers and events are shown in the corresponding contracts.

Figure 11 shows a UML sequence diagram representing the interactions among most Actors of the systems, when a Taker accepts - through her/his wallet - an order seen in the Relayer's book, and sends it to the DEX for execution, including the messages exchanged among the SCs. Basically, the Taker approves, the transfer to the DEX of the tokens to give to the Maker, plus the DEX fee; after that, the DEX cashes the tokens from both Maker and Taker, keeps the fees and gives back the proper tokens to both Traders' wallets.

6. **Coding of the SC subsystem**. The developed SCs make use of existing library SCs, namely "OnlyOwner" to manage the ownership of a SC, and "SafeMath" to avoid over- and under-flow errors. They also refer to SCs already deployed on the blockchain and implementing the ERC20 standard interface for managing tokens. Note that, since Solidity version 8.0, the SafeMath library is not needed anymore, because over- and under-flow controls are made by the EVM.

The contract "DEXstorage" holds and manages the mapping "tokens" having as key the supported token symbol (4 characters) and as value the token address, and the mapping “offers” having as key the Maker's address and as value the details of the offer (symbols and quantities of the tokens to sell and buy, and their hash digest of these data). The owner of DEXstorage is its creator, who is able to change the address of the DEX, stored in "dexLatestVersion" variable. All other DEXstorage operations can be performed only by the DEX. This is ensured by using "onlyLatestVersion" modifier. A part of DEXstorage contract follows, with the Offer definition:

```
pragma solidity ^0.7.0;
import "Ownable.sol";

contract DEXstorage is Ownable {
    address dexLatestVersion;  // Address of DEX contract using this storage.
    struct Offer {
        bytes32 offerHash;    // Hash of offer's parameters.
        bytes4 tokenToSell;  // Token to sell symbol.
        bytes4 tokenToBuy;   // Token to buy symbol.
        uint256 amountToSell; // Amount to sell.
        uint256 amountToBuy;  // Amount to buy.
    }
    mapping (bytes4 => ERC20) public tokens;
    mapping (address => Offer) public offers;

    modifier onlyLatestVersion() {  // Only  DEX contract using this storage.
       require(msg.sender == dexLatestVersion);  _; }

    function setCurrentDEXVersion(address _dex) external onlyOwner {
        dexLatestVersion = _dex; }

    function addOffer(address _maker, bytes4 _symbolToSell, bytes4 _symbolToBuy,
                        uint256 _toSell, uint256 _toBuy)
                                                 external onlyLatestVersion {
        offers[_maker] = Offer(
            keccak256(abi.encodePacked(_symbolToSell, _symbolToBuy, _toSell,
_toBuy)),
            _symbolToSell, _symbolToBuy, _toSell, _toBuy); }

    function getOffer(address _maker) external view
                         returns(bytes32, bytes32, bytes32, uint256, uint256) {
        Offer memory _makerOffer = offers[_maker];
        return (_makerOffer.offerHash, _makerOffer.tokenToSell,
_makerOffer.tokenToBuy,
                _makerOffer.amountToSell, _makerOffer.amountToBuy);
    }

    function removeOffer(address _maker) external onlyLatestVersion {
        Offer memory nullOffer;
        offers[_maker] = nullOffer;
    }
}
```

The contract "DEX" implements the decentralized exchange. It allows its owner to add the supported tokens, and then the Maker to add offers. Each Maker can have just one offer active at a given time. Before posting the offer, the Maker must approve the DEX address to withdraw from the SC managing the token to sell the offered amount, plus the selling fee. The function "addOffer" is shown below:

```
pragma solidity ^0.7.0;
import "SafeMath.sol";
import "ERC20.sol";
import "DEXstorage.sol";

contract DEX is Ownable {
    using SafeMath for uint256;
    uint256 public sellFee;         // Constant amount of fee paid by makers.
    uint256 public buyFee;          // Constant amount of fee paid by takers.
    DEXstorage private dexStorage;  // DEXstorage implementation reference.
...
    function addOffer(bytes32 _symbolToSell,  bytes32 _symbolToBuy,
                    uint256 _amountToSell, uint256 _amountToBuy) external {
        require(address(dexStorage.tokens(_symbolToSell)) != address(0));
        require(address(dexStorage.tokens(_symbolToBuy)) != address(0));
        require(dexStorage.tokens(_symbolToSell).allowance(msg.sender,
                address(this)) >= (_amountToSell.add(sellFee)),
                                         "Amount to sell exceeds allowance.");
        dexStorage.addOffer(msg.sender, _symbolToSell, _symbolToBuy,
                                    _amountToSell, _amountToBuy);
    }
...
}
```

A Taker can accept the Maker's offer. Of course, also the Taker must previously approve the DEX address to withdraw from the SC managing the token to buy the offered amount, plus the buying fee. The "acceptOffer" code is shown below:

```
function acceptOffer(address _maker, bytes32 _offerHash) external {
    bytes32 offerHash;
    bytes4 tokenToSell;    bytes4 tokenToBuy;
    uint256 amountToSell;  uint256 amountToBuy;
    (offerHash, tokenToSell, tokenToBuy, amountToSell, amountToBuy) =
                               dexStorage.getOffer(_maker);
    require(offerHash == _offerHash, "The offer hash doesn't match.");
    require(dexStorage.tokens(tokenToSell).transferFrom(_maker, address(this),
                amountToSell + sellFee),"Transfer from Maker to DEX
unsuccessful.");
    require(dexStorage.tokens(tokenToBuy).transferFrom(msg.sender,
address(this),
                amountToBuy + buyFee), "Transfer from Taker to DEX
unsuccessful.");
    require(dexStorage.tokens(tokenToBuy).transfer(_maker, amountToBuy),
                                "Transfer from DEX to Maker unsuccessful");
    require(dexStorage.tokens(tokenToSell).transfer(msg.sender, amountToSell),
                                "Transfer from DEX to Taker unsuccessful");
    dexStorage.removeOffer(_maker); }
```

As in the "addOffer" function, most actions are performed inside a "require" clause, meaning that if the action aborts, the whole computation aborts, the blockchain state does not change, and the remaining gas is sent back to the caller.

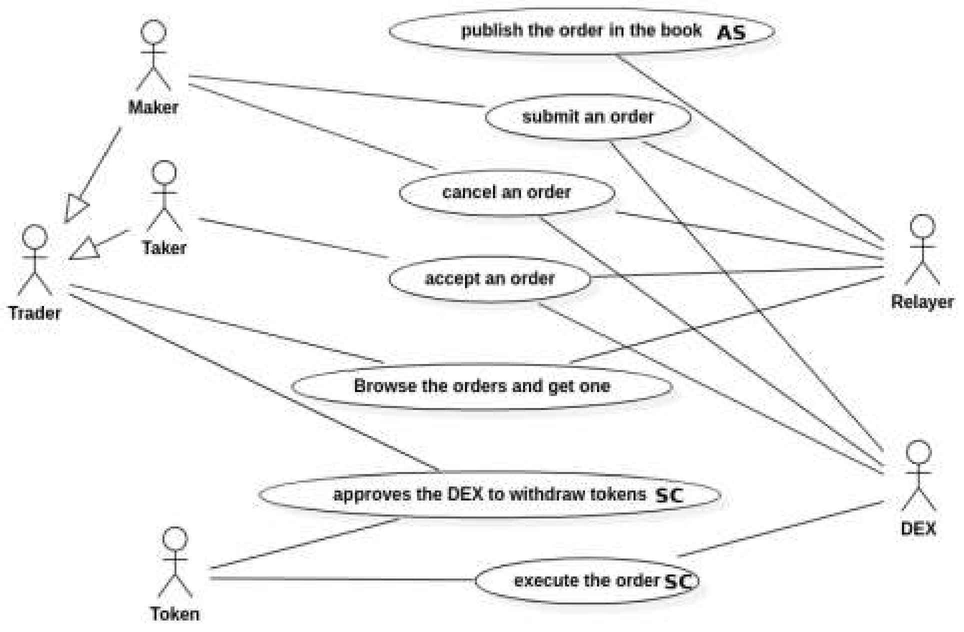

publish the order in the book AS
submit an order
cancel an order
accept an order
Browse the orders and get one
approves the DEX to withdraw tokens SC
execute the order SC
Maker
Taker
Trader
Token
Relayer
DEX

Figure 8 The User Stories of the DEX system specification.

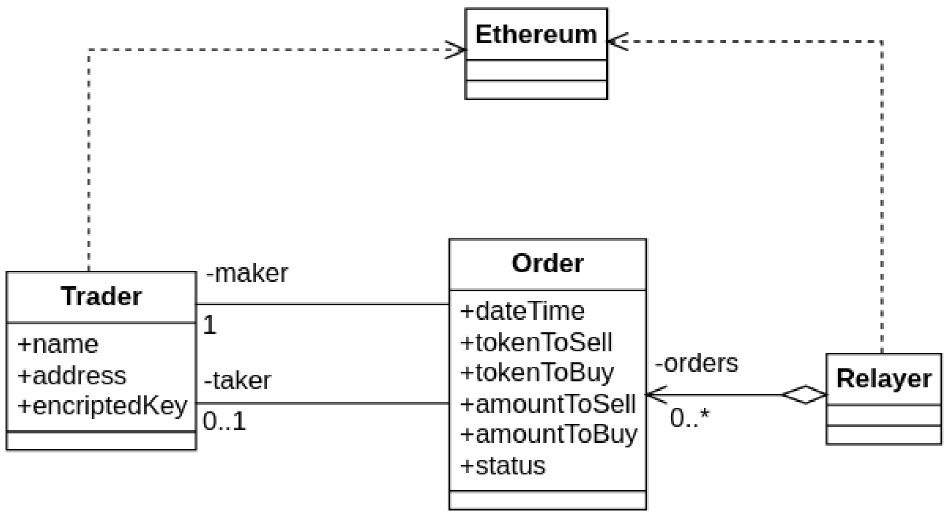


Figure 9 The standard UML class diagram derived from the USs.

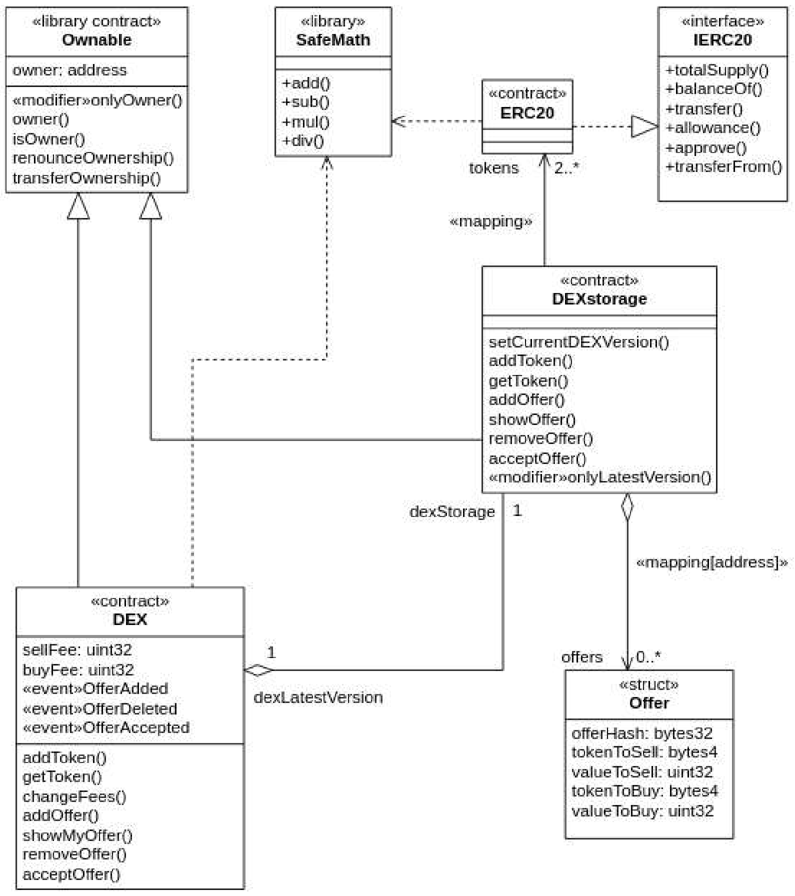


Figure 10 The modified UML class diagram, showing the structure of the required smart contracts of the DEX system.

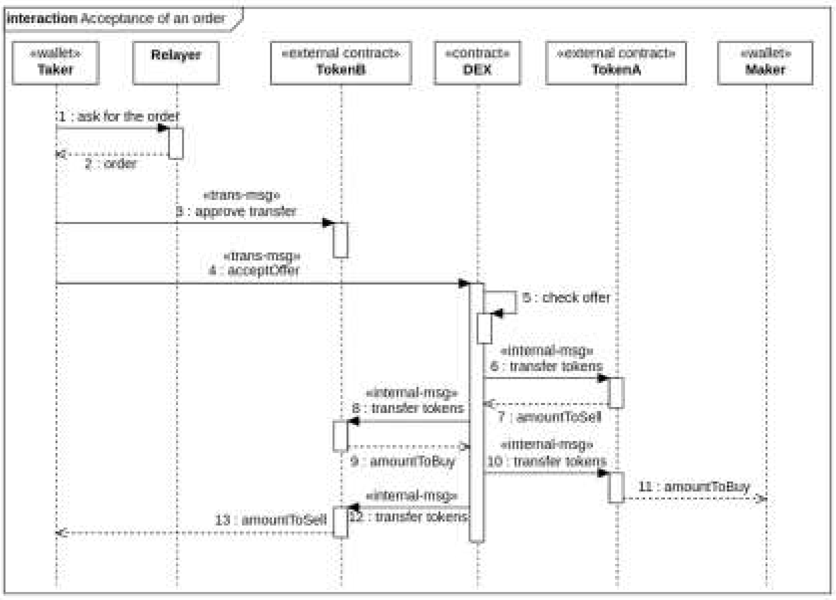


Figure 11 The UML sequence diagram showing a Taker accepting an offer and sending it to the DEX for execution.

2.5.3 Security assessment

Although the DEX system is quite simple, it has strict security requirements, because it manages tokens, which can usually be exchanged with real money. In the followings, I follow the security checklist to be applied in design phase (8 items) and coding phase (18 items), as reported in section 2.4.4. For the sake of brevity, I will not report 18 checks which are not relevant for this case study.

- **Limit the amount of ether**: I check that all money transfers are performed through explicit withdrawals made by the beneficiary address.
- **Transaction Ordering**: I added the hashing check of orders because, lacking it, a fraudulent Maker could post a very favorable offer; then the Maker might detect a Taker's transaction to accept the offer, and quickly post a faster transaction (with much higher gas value) changing the offer in an unfavorable way. The hash signature of the offer solves the problem.
- **Use trustworthy dependencies**: I use only standard and very proven libraries, such as "SafeMath.sol" and "ERC20.sol".
- **Beware of re-entrancy**: using the *Check-effect-interaction*, all relevant actions are preceded by checks. No call to other SCs is made which might trigger reentrancy issues.
- **Embed addresses to grant permissions**: most functions able to modify the SC storage can be called only by the owner, or by the DEX in the case of DEXstorage. The only functions callable by external addresses are those used by the Maker to post an offer, and by Taker to accept an offer.
- **Use platform related standards**: the only external SC called are ERC20 tokens, which are a proven Ethereum standard.
- **Prevent overflow and underflow** and **Beware of rounding errors**: the (few) relevant computations are performed using "SafeMath" library, which protects from these kinds of errors.
- **Validate inputs to external and public functions**: not only all relevant actions are preceded by checks, but all actions are performed including them inside a "require" clause. If a guard is not satisfied, the whole action aborts.

2.5.4 Gas optimization

The minimization of gas consumption might be considered part of performance optimization of any software solution. With dApps, the blockchain itself is the main performance bottleneck, because external transactions take time to be processed and accepted, even in a permissioned blockchain. With Ethereum, optimizing the performance of smart contracts corresponds to minimize gas consumption, which is linked also to the number of bytecode instructions executed.

The presented code follows the gas optimization patterns, as presented in section 2.4.5. In particular, the following patterns are relevant:

- **Proxy Delegate**: I did not use Proxy Delegate pattern for the reason explained in Step 5 of section 2.4.2.
- **Eternal Storage**: this pattern corresponds to the use of "DEXStorage" contract, which would allow to save a lot of gas in the case the DEX contract needs to be upgraded. In fact, if the new DEX version uses the same DEXstorage of the previous one, all tokens and offers can be re-used. In this case, however, the Makers must be warned to change the DEX address in their approval to withdraw tokens.
- **Pack variables**: I packed the token symbol byte arrays is in the "Offer" structure.
- **Do not initialize variables**: I did not initialize variables with default values.
- **Use Mappings**: all the needed collections in contracts are implemented using mappings.
- **Execution paths**: functions do not perform heavy computations. However, I checked all possible execution paths to make sure they are minimized.
- **Limit external calls**: functions are typically declared as "external", which are cheaper than "public" functions.
- **Limit modifiers**: at most one modifier is used per function.

2.5.5 Writing automated tests

Each non-trivial function of the contracts written so far must be provided of Unit and Acceptance Tests. Truffle framework [20] makes available two methods for testing Ethereum smart contracts: Solidity test and JavaScript test. For the sake of brevity, here I report just a fragment of the Solidity unit test written to verify "addToken()" and "addOffer()" functions of "DEXstorage" contract:

```
pragma solidity ^0.7.0;
import "truffle/Assert.sol";
import "truffle/DeployedAddresses.sol";
import "DEXstorage.sol";

contract TestDEXstorage {
    DEXstorage public dexStore;
    address public tok1;
    address public tok2;
    address public address1;
    address public address2;
    bytes4 symbTok1;
    bytes4 symbTok2;

// Run before every test run
    function beforeAll() public {
    tok1 = 0xdf9c2dD7397F7f010E20e2c3b600372c8866Beb4;
    tok2 = 0x11261dB2A31C1b02ac1D8Dbb083c917123De160F;
    symbTok1 = "TOKA";
    symbTok2 = "TOKB";
    address1 = 0x9ef914c91980995913e8b846fbFD25814708bc8A;
    address2 = 0xac0C4bBd7481e6299082a5c1CD2D2e1Bf6291De2;
    }

// Run before every test function
    function beforeEach() public {
        dexStore = new DEXstorage();
        dexStore.addToken(symbTok1, tok1);
        dexStore.addToken(symbTok2, tok2);
    }

// Test that it adds tokens correctly
    function testAddToken() public {
        Assert.equal(dexStore.getToken(symbTok1),tok1,"Address of Token A not
correct");
        Assert.equal(dexStore.getToken(symbTok2),tok2,"Address of Token B not
correct");
    }

// Test that it adds an offer correctly
    function testAddOffer() public {
        dexStore.addOffer(address1,symbTok1,100,symbTok2,200);
        DEXstorage.Offer offer;
        offer = dexStore.offers[address1];
        Assert.equal(offer.tokenToSell,symbTok1, "Token to sell not correct");
        Assert.equal(offer.tokenToBuy,symbTok2, "Token to buy not correct");
        Assert.equal(offer.amountToSell,100, "Amount to sell not correct");
        Assert.equal(offer.amountToBuy,200, "AMount to buy not correct");
    }
...
}
```

The automated tests follow the standard pattern of resetting the test environment and building the "test fixture" - that is the data needed for testing - before each test call. This is accomplished by the functions "beforeAll()" and "beforeEach()". In this way, the test results do not depend on test ordering. The tests are run by creating a "TestDEXstorage" contract, and sending the proper test messages to it, which is automated by Truffle.

2.5.6 Design and coding of App System

This subsection covers steps 7 and 8 of ABCDE process. The App System is composed of the software able to present the current offers of tokens posted by the takers, and of the software used by takers and makers, respectively to post, modify or delete offers, and to accept offers. The latter software must be provided of a wallet able to store Ethers and send transactions to Ethereum blockchain.

The design of this subsystem includes that of its user interfaces. The system is fairly complex, and the wallets must be designed and implemented using strong security practices. I will not dig further into this subsystem because, except for the wallet, it is a standard, web-based system.

2.6 Threats to Validity

In software engineering, there are three main types of validity that contribute to the overall validity of a research, i.e., internal, construct, and external validity [76]. This research regards the proposal of a new software development method for dApps, so its validity assessment is quite different from that of empirical researches.

Internal validity concerns causal relationship between independent and dependent variables, and if there is only one explanation for the research results. In this case, I do not have empirical results on the validity of the method because the application field is relatively new, ABCDE is the first proposed structured method to develop dApps, and some teams are just starting to use it. For these reasons, I believe that internal validity assessment is out of the scope of this threat analysis.

Construct validity concerns the correspondence between the research and the theory that underlies the research itself. From this point of view, a research is valid if one may exclude alternative explanations of the results. In our case, a threat to construct validity might regard the possibility that other approaches might be better than ABCDE for dApp development. For instance, one might argue that a waterfall process combined to a secure software development methodology might better address security concerns of smart contracts. My team's experiences with agile methods, which began in the late 90s, and on dApp development make us pretty confident that ABCDE is well balanced between the need to proceed quickly in the presence of uncertain requirements - as it is almost always the case for dApps - without compromising security. Moreover, as reported in section 2.5, a survey among 14 early developers who used ABCDE gave favorable results. As always, improvements might be made to ABCDE method, and the release of better methods cannot be ruled out. In this case, ABCDE will still be a yardstick for comparison for other dApp development methods.

Threats to external validity are related to generalization of this approach: ABCDE was developed specifically for Ethereum dApps. Is it equally valid for dApps intended to run on Ethereum main-net, and on Ethereum permissioned blockchains? How about the applicability to languages different from Solidity, and to other blockchains? Regarding the first issue, ABCDE was developed considering both kinds of dApps - for public and permissioned blockchains. The former has typically much stricter security and gas consumption requirements, whereas the latter tend to make use of more complex smart contracts. The approach was balanced considering both issues, addressing security and gas optimization with specific steps, and contract complexity with design diagrams and iterative and incremental development. Therefore, I believe that the validity threat of ABCDE being poorly suited to either public or private blockchain dApps is overcome.

The applicability to other languages and blockchains is a bigger threat, which will be mitigated only by extending ABCDE to cover these subjects. This is work in progress.

2.7 Conclusions and Future Work

Despite the substantial inflow of money and the strong efforts made in the development of blockchain-based applications, the application of sound software engineering processes and practices is still quite low. Moreover, dApp development has peculiar characteristics that must be addressed with specific tools and guidelines, and research on these issues has just started, being this field still in its infancy.

Applying a sound software engineering approach might greatly help to overcome many of the issues of dApp development. These include specific architectural design issues, security issues related to how a blockchain works, the need to spare gas, testing plans and strategies in the blockchain environment, corrective and evolutionary maintenance issues, also related to blockchain immutability.

To my knowledge, the work presented in this research is the first attempt to develop an organic process for dApp development, from requirement gathering to design, coding, security assurance, and deployment. The proposed method is presently focused on the Ethereum blockchain and its Solidity language, which are at the current time the most used to develop dApps. However, it can be adapted to other environments. For instance, it was used to model a dApp developed using Hyperledger Fabric [73].

ABCDE takes advantage of agile practices, because dApp development usually deals with rapid implementation of systems whose requirements are not fully understood at the beginning, and tend to change over time. However, given the specificity of blockchains, ABCDE complements the incremental and iterative development through boxed iterations, typical of agility, with more formal tools. These tools include a full modeling of interactions among traditional software and blockchain environment, using UML class diagrams, UML Use Case diagrams (in fact, representing user stories), UML sequence diagrams - all specialized for blockchain-based application development using stereotypes.

ABCDE also provides valuable practices, patterns and checklists to promote and evaluate the security of a dApp written in Solidity language, and also to reduce its gas consumption.

In this research, I also present an example of application of ABCDE for the development of a simplified Distributed Exchange system to enable trading between pairs of Ethereum tokens. This example is a useful step by step tutorial on the application of guidelines and patterns discussed.

I believe that ABCDE method can be really valuable to blockchain firms and ICO startups, which might develop a competitive advantage using it since the beginning of their development projects.

Future work will address the improvement suggestions gathered by the survey reported in section 2.5, which include: (i) extending the method to other dApp development environments, such as Hyperledger Fabric; (ii) adding best practices/guidelines for dApp maintenance; (iii) being more specific on the App System development, and on the integration and testing of SC and App systems; (iv) providing teaching materials and more practical examples. I also plan to develop tools such as automated compilers of ABCDE class diagrams into smart contract data structure.

3 Assessing the Risk of Software Development in Agile Methodologies Using Simulation

As already stressed in previous sections, agile methodologies aim to reduce software development risk using short iterations, feature-driven development, continuous integration, testing automation, and other practices. However, the risk of project failure or time and budget overruns is still a relevant problem.

In this section I present and discuss a new approach to model some key risk factors in agile development, using software process simulation modeling (SPSM), which can complement other approaches, and whose usage is particularly suited for agile development. I introduce a new approach to model some key risk factors - namely project duration, number of implemented issues, and key statistics of issue completion time - using a simulator of agile development, which we developed for this purpose. The approach includes modeling the agile process, gathering data from the tool used for project management, and performing Monte Carlo simulations of the process, to get insights about the expected time and effort to complete the project, and about their distributions. The model's parameters that can cause risk are errors in effort estimation of the features to develop, variations in developers' assignment to these features, impediments related to developers' availability and work completion. To validate the simulator, and to demonstrate how the method can be used, I analyzed three open-source projects, gathering their data from JIRA repositories. I ran Monte Carlo simulations of these projects, showing that the simulator can well approximate the progress of the real project, then varying the identified risk factors and statistically evaluating their effects on the risk parameters.

The proposed approach is relevant for project managers, being able to quantitatively evaluate the risks, provided that the process and the project's data are properly modeled and gathered.

As for the future, I am working to improve the risk assessment method, evaluating it on more case studies, scaling the model from a single team to multiple teams involved in one or more projects. Moreover, I plan to apply it also to the study of dApp projects, developed following the ABCDE method presented in the previous chapter 2.4. Being an agile development process, ABCDE is suitable to be modeled through this simulator.

3.1 Introduction

Software technical risk is hard to define and there is no agreement on a common definition. Risk is somehow defined as a measure of the probability and severity of adverse effects [77], inherent in the development of software that does not meet its intended functions and performance requirements [78].

The Software Engineering Institute (SEI) defines risk as the possibility of suffering loss [79]. In such context, the loss may describe any impact to the project, which could be in the form of diminished quality of the end product, increased costs, delayed completion, loss of market share, failure, and so on.

A broader definition is given by PMBoK (Project Management Body of Knowledge) book. Here, risk is defined as an "*event or an uncertain condition that, if it occurs, can result in positive (opportunities) or negative impacts (threats) in one or more project objectives, such as scope, time, cost, and quality*" [80].

Wallace et al. [81] identified and studied six dimensions of software project risk, i.e. risks related to: Organization environment, User, Requirements, Project complexity, Planning and control, Team.

Traditional software development methodologies deal with risk by performing a detailed up-front analysis to lower risks related to poor requirements specification and enforcing strict control planning, with frequent risk re-evaluations. However, they tend to integrate the various software modules late in the project, leaving room for integration risks.

A newer approach is adopted by Agile Methodologies, which were introduced in the late 90's precisely to lower risks due to changing requirements –- a characteristic common to most Internet-age software projects –- but related also to other issues, such as user involvement, team cooperation, late integration, insufficient feedback [82] [83] [84] [85].

3.1.1 Problem statement

Software risk management is crucial for successful project management, but it is often an aspect not properly taken into account in real projects. One of the main reasons is that project managers often do not have effective tools and practices for software risk management.

In their work [86] Chadli and collaborators presented a research focused on discovering and classifying the various tools mentioned in the literature which support Global software development (GSD) project managers, and on identifying in which way they support group interaction. In this paper, the authors state that "*Decision management, risk management, and measurement processes are not adequately supported by tools when compared to the other Software Process Management processes*".

Only recently, a few frameworks and tools have been introduced for better addressing risk management in Agile Software Development (ASD). The results of a survey conducted using a qualitative approach to analyze how risk management is carried out in Scrum software projects were presented in [87]. De Souza Lopes et al. in [88] established a framework called RIsk Management PRoduct Owner (RIMPRO), which intends to support project teams to systematically manage risks related to PO activities that may arise during the project definition proposed by PMBoK.

Risk is an "event or an uncertain condition that, if it occurs, can result in positive (opportunities) or negative impacts (threats) in one or more project objectives, such as scope, time, cost, and quality" (PMI, 2017), in [89] intelligent agents are used to manage risk. They do not simulate the process but only the stochastic impact of the variation [90] [91].

In such a scenario, Software Process Simulation Modeling (SPSM) has emerged as a promising approach to address a variety of issues related to software engineering, including risk management [92] [93]. Clearly, SPSM cannot address risk dimensions such as organization environment stability and management support, lack of user involvement, team motivation, and communication issues. SPSM can be useful to establish the impact of risks on specific topics, such as requirement estimation errors, rework due to software that does not meet its intended functions and performance requirements (software technical risk), project complexity, planning and control practices [94]. However, how and to which extent SPSM can support software risk management is a topic that still needs to be better clarified, especially in the case of ASD.

The goal of this work is to present a viable approach to risk management using SPSM when using an agile development process, that is a process based on implementing requirements or change requests specified as atomic "features" or "issues". In terms of research questions, this goal can be specified as:

- **RQ1**: To what extent it is possible to automatically import into the simulator data of real projects, from issue management systems?
- **RQ2**: How accurate can the simulator be in predicting project completion times?
- **RQ3**: Can the simulator be useful to estimate project risk (induced by errors in efforts estimation, and random developer issues assignment) with a Monte Carlo approach?

3.1.2 Contribution

In this research, I present a system to perform risk assessment of ASD projects using SPSM and a Monte Carlo stochastic approach. Regarding Wallace's risk dimensions [81], this system mainly considers requirement correctness and estimation. This approach can improve the planning and control of the project, and the management of its complexity. Also, team factors can be taken into account, by explicitly modeling developers' skills, impediments, and turnover. The dimensions of the organization environment and of user are out of the scope of the proposed approach.

The basic tool of the system is an event-based simulator, able to simulate ASD based on the implementation of User Stories (USs), or features, kept as independent as possible from one another. The agile process may be iterative, as in Scrum, or using a continuous flow approach, as in the Lean-Kanban approach.

The basic inputs to the simulator are the team composition and skills, and the USs themselves. These inputs can be fed to the simulator from the Issue Management tools. So far, the interface to the popular JIRA tool [95] was implemented. Other inputs are parameters driving the random variation of teamwork, and of USs estimation and actual cost, as well as of possible events able to influence the project.

The outputs of the simulation are key risk parameters, such as the distribution of times to complete the project or the iteration, and of the forecast project cost. The simulations are performed hundreds, or thousands, of times, properly varying the inputs, to get the distribution of the required outputs. The resulting distributions can provide hints on the expected outputs and of their variances, including proper risk percentiles, useful to monitor development and to control risk. In fact, if risk assessment is not done in the initial stage, effort estimation will be strongly affected and the overall project may risk failure [96].

The main contributions of this work are:

- The development of a flexible and extensible ASD simulator, able to simulate virtually every Agile development process, based on incremental development. This simulator models most aspects of ASD - team members, USs, activities, events - and can be easily extended and customized, due to the full object-oriented approach used for its development.
- The development of a risk management model, considering team factors, requirement correctness, and effort estimation. The input data, parameters, and events, as well as the relevant output distributions, are analyzed and discussed.
- The ability to feed project data from popular issue management tools, namely JIRA, with the ability to extend the simulator taking into account also other tools.
- A risk management analysis performed on three medium-sized Open-Source projects, comparing the simulation results of different choices of the input parameters. The results highlight both the prediction accuracy of our tool and how it can be used to manage risk.

3.1.3 Outline

The remainder of this section is structured as follows. Section 3.2 reports the main related works on software risk management and on the application of SPSM to it. Section 3.3 presents the proposed risk-assessment method. Section 3.4 describes the simulation model. Section 3.5 presents how the risk-assessment method is applied. Section 3.6 presents the case studies. Section 3.7 presents the results of the case studies, and how they can be generalized and applied to other projects. Section 3.8 presents the threats to validity. Section 3.9 eventually concludes this chapter and discusses future work.

3.2 Related work

3.2.1 Risk Management in Software Projects

It is well known that several software projects suffer from various kinds of problems, such as cost overruns, missing delivery deadlines, and poor product quality. One of the factors that cause these problems is the fact that the risks are not handled, as shown by Charette [97].

Risk management in software projects is a key component of the success of a project. Software engineering researchers and professionals have proposed several systematic approaches and techniques for effective risk management as reported by Boehm [98].

A study conducted by the Project Management Institute has shown that risk management is an activity not practiced among all the disciplines of project management in the IT industry [99]. In real software projects, risks are often managed using the insights of the project manager, and the entire process of risk management is rarely followed [100]. One of the main reasons for this is that project managers lack practical techniques and tools to effectively manage the risks.

The paper by de Oliveira Barros et al. [101] presents an approach to develop, retrieve, and reuse management knowledge and experience concerned with software development risks, using scenarios to model risk impact and resolution strategies efficacy.

Shahzad and Al-Mudimigh [102] propose a model that takes care of the most frequently occurring risks and provides a way for handling all of them. The sequence of activities for mitigating all risk factors is presented in a comprehensive model, which is useful for improving management and avoidance of software risk.

Xiaosong et al. [103] use a classical matrix approach, defining 82 software project risk factors, aggregated in 14 kinds of risk, each one with an impact level on the overall project, from Negligible to Critical. Once experts evaluate each factor, the risks are evaluated using the risk matrix. Thereafter, high-level risks should be immediately addressed, whereas medium and low risks will be monitored and kept under control.

Roy et al. identified key risk factors and risk types for each of the development phases of SDLC (Software Development Life Cycle) [104], including services and maintenance of software products.

Thieme et al. in their paper [105] present a study about the process for the analysis of functional software failures, their propagation, and incorporation of the results in traditional risk analysis methods (fault trees and event trees). A functional view on software is taken by allowing integration of software failure modes into risk analysis of the events and effects. The proposed process can be applied during system development and operation to analyze the risk level and identify measures for system improvement.

3.2.2 Risk Management in ASD

Recently, agile software development methods have been introduced, to be able to manage requirement changes throughout project development. ASD might be considered also as a risk management approach because errors and changes in requirements are one of the main risk factors in software development. Among specific studies and approaches to manage project risk in ASD processes we already quoted the RIMPRO framework by de Souza Lopes et al. [88]. I also recall the work of Tavares et al. [106] who propose *Rm4Am*, a tool that identified 5 components and 48 sub-components important in ASD and conducted an experiment with the supervision and participation of Agile expert.

3.2.3 Simulation of Software Process

Software Process Simulation Modeling (SPSM) is presented as a promising approach suitable to address various kinds of issues in software engineering [107]. The results of the review conducted by Zhang et al. [108] showed that risk management is one of the several purposes for SPSM. Liu et. al. performed a systematic review on this topic, concluding that the number of SPSM studies on software risk management has been gradually increasing in recent years and that discrete-event simulation and system dynamics are the two most popular simulation paradigms, while hybrid simulation methods are more and more widely used [109].

Practical examples of the use of SPSM are the works by Baum et al. [110], who compares pre-commit reviews and post-commit reviews using process simulation through a parametric discrete event simulation model of a given development process, and by Zhao et al. [111] who present a fine-grained dynamic micro-simulation system based on an agent model at the project level for Open Source Software Development.

Discrete-event simulation of agile development practices was first introduced by Melis et al. [112]. Anderson et al. [113] proposed an event-driven, agent-based simulation model for the Lean-Kanban process, extensible to other Agile software processes, and used it to demonstrate the effectiveness of a WIP-limited approach and to optimize the WIP limits in the various process activities. In a subsequent work, Anderson et al. [114] used an extended version of their simulator to compare Lean-Kanban with traditional and Scrum approaches on the data collected from a Microsoft maintenance project, showing that the Lean-Kanban approach is superior to the others.

Turner et al. worked on modeling and simulation of Kanban processes in Systems Engineering [115] [116]. These two works, though quite preliminary, propose the use of a mixed approach, merging Discrete Event and Agent-based simulation approaches. In particular, Discrete Event simulation is used to simulate the flow of high-level tasks and the accumulation of value, whereas Agent-based simulation is used to model the workflow at a lower level, including working teams, Kanban boards, work items, and activities.

Wang [117] introduces an agent-based simulation tool to compare solo and pair-programming, and other agile practices, in the context of Scrum development. The simulation tool can simulate all types of Scrum context and team composition to test designed strategies under various what-if assumptions in agent-based modeling.

3.2.4 Automated Approaches for Software Risk Management

Another field of interest related to software risk management is the application of Machine Learning, and similar automated techniques. In this field, Fenton et al. [118] developed a complex causal model based on Bayesian networks for predicting the number of residual defects of a system. Their approach accounts for various phases of software development, among which requirement analysis, design, and programming, testing, and rework. Each phase is modeled by a Bayesian net, whose probability values and functions are set by project managers. The model allows managers to perform various types of what-if-analyses and trade-offs, focusing on minimizing the number of defects, which is a key factor in risk management.

Tinjacá Rodríguez et al. [119] present a model based on the A.I. technique called "Rough Set" for selecting and prioritizing, in environments of uncertainty, a set of critical threats in a software development project, to minimize risks. Their model, named "Apollonian", generated 20 rules that work as a reference for any software development project, to assess the main threats to the project.

Raymond Joseph [120] proposes a machine-learning algorithm to generate risk prompts, based on software project characteristics and other factors. His approach uses multiple multilabel artificial neural networks to label software projects according to the risks to which they are most exposed.

Han [121] trained a multi-layer-perceptron to assess the risk level of a software project, using as a learning set the OMRON dataset of 40 projects, each described by 24 risk-related parameters. The results look better than a more traditional logistic regression.

Min et al. [122] applied fuzzy comprehensive evaluation to estimate a project's risk level. The approach is somewhat similar to, though much simpler than, that of ref. Fenton, but instead of Bayesian nets fuzzy algebra is used to find the probability of risks, given estimated risk factors.

Alencar et al. [89] propose a proactive and automated approach based on agent technology to assist the software project manager in the execution of the Risk Management processes.

Abioye et al. [123] present a life-cycle approach to ontology-based risk management framework for software projects using a dataset gathered from literature, domain experts, and practitioners. The risks are conceptualized, modeled, and developed using Protégé. The framework was adopted in real-life software projects.

Asif et al. [124] identify the relationship between risk factors and mitigation actions automatically by using an intelligent Decision Support System. The DSS is rule-based and identifies the rules using the Equivalence Class Clustering and bottom-up Lattice Traversal (ECLAT) algorithm, starting from expert knowledge and literature. 26 highly cited risk factors and 57 risk mitigations were identified from the literature and associated through the rules of the DSS, to help project managers to mitigate the risks.

3.2.5 SPSM for Risk Management

The literature about SPSM applied to risk management is still quite limited. Ghane [125] observes that ASD delivers workable software in short cycles, and this helps with collecting more heuristic data as compared to traditional waterfall methodologies. Such data can be used as quantitative metrics for time and effort estimation. He then introduces a risk management model that uses project simulation to produce risk metrics used to help with risk avoidance and mitigation. These metrics are used to adjust project factors such as time, cost, and scope during the lifespan of a project.

Singh et al. [126] represent the PERT graph of a software project, where nodes are the states, and arcs represent activities. Each arc bears the mean and standard deviation of its cost. A Monte Carlo simulator stimulates the model with thousands of executions, to find the cost distribution and the critical paths.

3.3 Risk Assessment through Simulation

3.3.1 The Risk-assessment methodology

The starting points in Risk assessment are the Six dimensions of risk, as defined by Wallace et al. [81]. They are:

1. **Organizational Environment Risk**, including change in organizational management during the project, corporate politics with a negative effect on the project, unstable organizational environment, organization undergoing restructuring during the project.
2. **User Risk**, including users resistant to change, the conflict between users, users with negative attitudes toward the project, users not committed to the project, lack of cooperation from users.
3. **Requirements Risk**, that is continually changing system requirements, system requirements not adequately identified or incorrect, system requirements not properly defined or understood.
4. **Project Complexity Risk**, encompassing a high level of technical complexity, the use of new or immature technology.
5. **Planning & Control Risk**, including the setting of unrealistic schedules and budget, lack of an effective project management methodology, project progress not monitored closely enough, inadequate estimation of required resources, project milestones not clearly defined, inexperienced project manager.
6. **Team Risk**, including inadequately trained and/or inexperienced team members, team member turnover, ineffective team communication.

I recall that risk management involves the following three steps:

1. **Risk identification**: where possible risks related to a project are enumerated and discussed, typically preempting what might go wrong in a proactive way.
2. **Risk analysis**: where identified risks are quantitatively and qualitatively evaluated, to ascertain the probability and critical level of their impact.
3. **Risk mitigation**: actions to both lower the probability that the adverse event occurs and reduce its impact on the project before it happens.

SPSM can be applied to risk analysis, greatly helping the quantitative assessment of some risk dimensions. This approach is intended to work together with other risk management frameworks, and not as a stand-alone method.

The use of the SPSM approach addresses mainly dimensions 3, 4, and 5, and specifically inadequate estimations of requirements, project complexity in terms of number and complexity of requirements (including sequence constraints), and poor quality of software artifacts, due again to requirements not properly understood, or to issues in project management and planning. Team risk can also be modeled by setting proper developers' skills, by representing developers' turnover, task switching, and absences due to various causes.

The Risk-assessment methodology I propose is performed in subsequent steps:

1. The development (or maintenance) process is modeled (activities, team, process, issues, constraints), and the simulator is configured to simulate it.

2. Key quantitative risk factors are identified; in our case, they are estimation errors in efforts to complete features or resolve issues, percentage of rework, variations in the skills of team members, probability of events that stall the development of single features, or block one or more developers, and so on.
3. Probability distributions are given to these risk factors, for instance, the probability distribution of actual effort needed to fix an issue, or the probability that a developer is blocked, together with the probability distribution of the time length of this block.
4. Key process outputs are identified, such as project total time, throughput, average and 95% percentile of lead and cycle times to resolve issues.
5. Hundreds or thousands of Monte Carlo simulations of the project are made varying the risk factors accordingly to their probability distributions and recording the process outputs.
6. The resulting distributions are analyzed, assessing for instance the most likely duration and cost of the project, the average time – or the distribution of times – to implement an issue or to fix a bug, the probability that a given percentage of issues is implemented within a given time.

This Monte Carlo assessment can be performed also on an ongoing project, by simulating the continuous flow of new requirements or maintenance requests, or just the remaining features to be implemented.

The proposed approach was inspired by the ever-increasing use of Issue Tracking Systems (ITS) that allow developers to easily gather all data related to change requests and defect correction, as well as to schedule and track the project flow. I started by creating a connection with the JIRA system [95], one of the most popular ITS worldwide, throughout REST calls, using JIRA APIs. Once the connection is established, it is possible to download project data and all related information in a file in JSON or CSV formats. In particular, the simulator can collect detailed data related to developers, issues, process activities, and others, as shown in Figure 12.

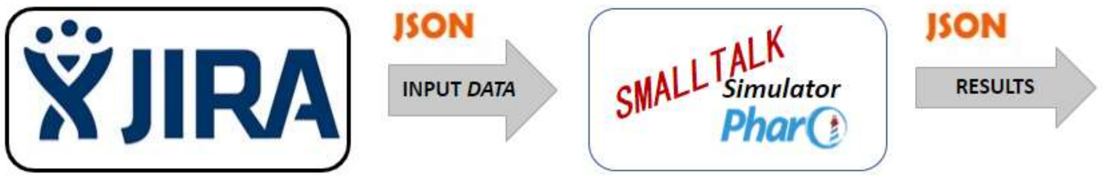


Figure 12 Interaction between JIRA Issue tracking System and the simulator.

3.4 The Simulation Model

This SPSM model uses an approach that is both event-driven and agent-based. The operations of the system are represented by a sequence of events in chronological order. Each event occurs at an instant of the simulation time and causes a change in the state of the system.

The simulator is also based on agents (the team members), who exhibit autonomous behavior. The agent's actions are not fixed but depend on the simulated environment.

3.4.1 Basic Components

The basic components of the SPSM model are the issues, the activities, the team members, and the events:

- **Issues** are the atomic work items in a development or maintenance project. They correspond to the *features* described in the Lean-Kanban approach and are like Scrum USs. Each unit of work is characterized by a unique identifier, a collection of effort values (in man-days) expressing the amount of work needed in each activity to complete the issue, a priority (an integer in a given range, expressing the importance of the issue), and information of the actual effort spent in each activity. When the last activity is completed, the issue becomes closed.
- **Activities** represent the kinds of work that must be done on the issues to complete them. Typically, they are Planning, Analysis, Coding, and Testing, but they can be configured on the specific development process chosen by the team.
- **Team members (Developers)** hold the information on the actual developers, which includes the skills in the various activities. If the skill is equal to one, it means that the team member will perform work in that activity according to the declared effort. For instance, if the effort is one man-day, the member will complete that effort in one man-day. If the skill is lower than one, for instance 0.8, it means that one-day effort will be completed in 1/0.8 = 1.25 days. A skill lower than one can represent an actual impairment of members or the fact that they have also other duties and are not able to work full time on the issues. If the skill for an activity is zero, the member will not work in that activity.
- **Events** represent what happens at a given time, which is relevant to the development. Each event has a time and is executed by the simulator when its time arrives. Three kinds of events are managed:

  i. the creation of an issue;

  ii. the start of work on an issue in a given activity;

  iii. the end of the work on the issue in a given activity.

3.4.2 The simulation process

The modeled development process proceeds through a sequence of steps. Preliminarily, one must define the ASD process, that is what are the activities of the process, their sequence, and their average relative weight on the total effort to complete an issue. Secondly, the development team members must be created, each having different skills in the various activities. Then, the simulator is started, executing steps with the following characteristics:

1. The simulation starts at time zero: t=0. Time t is expressed in working days. Each day involves 8 hours of work. At the beginning, the system may already hold an initial backlog of issues to be worked out.
2. The simulation proceeds by time steps of one day, until a predefined end, or until the event queue is empty.
3. Issues are entered at given days, drawn from a random distribution, or given as input to the system. Each issue is endowed with a collection of effort values for each activity (in days of work), to be performed to complete the issue. These values can be drawn from a distribution or obtained from real data.
4. Each issue passes through the activities. Each activity takes a percentage of the whole effort to process the issue. The sum of the percentages of all the activities must be 100%. When an issue enters an activity, the actual effort (in man-days) needed to complete the activity is equal to the total effort of the issue multiplied by the proper percentage.
5. At the beginning of each day, the developers choose the issue (and the activity) they will work on in the day. This choice is driven by the developer's skills, by the issue priority, and by the preference to keep working on the same issue of the preceding day, if possible. Whenever an issue is processed for the first time, or in the case of developer switching from another issue, a multiplicative penalty factor p, with $p \geq 1$, is applied to compute the time effort, to model time waste due to task switching (extra time needed to study the issue, which is proportional to the size of the task). The task switching problem is well known in software engineering, as reported also recently by Abad et al. [127].
6. When the work on an issue in a given activity ends, the issue is pulled to the next activity, where it can be chosen by a developer, or is closed in the case of the last activity. The developer who ended the work will choose another issue to work on, which might be the same issue, in the subsequent activity, or not.

The presented simulation model can represent development, or maintenance processes, and can be customized to cater to the specific process of an organization. From this generic model, we can derive various specific models. For instance, one might introduce WIP limits, as suggested by the Lean-Kanban approach. For a WIP-limited process, the model must be complemented by adding limits to the maximum number of issues that can be processed at any given time inside an activity.

If a Scrum-like development must be modeled, this can be accomplished by defining the length of the iteration (Sprint) and managing the number and total effort of new issues entering at the beginning of each Sprint.

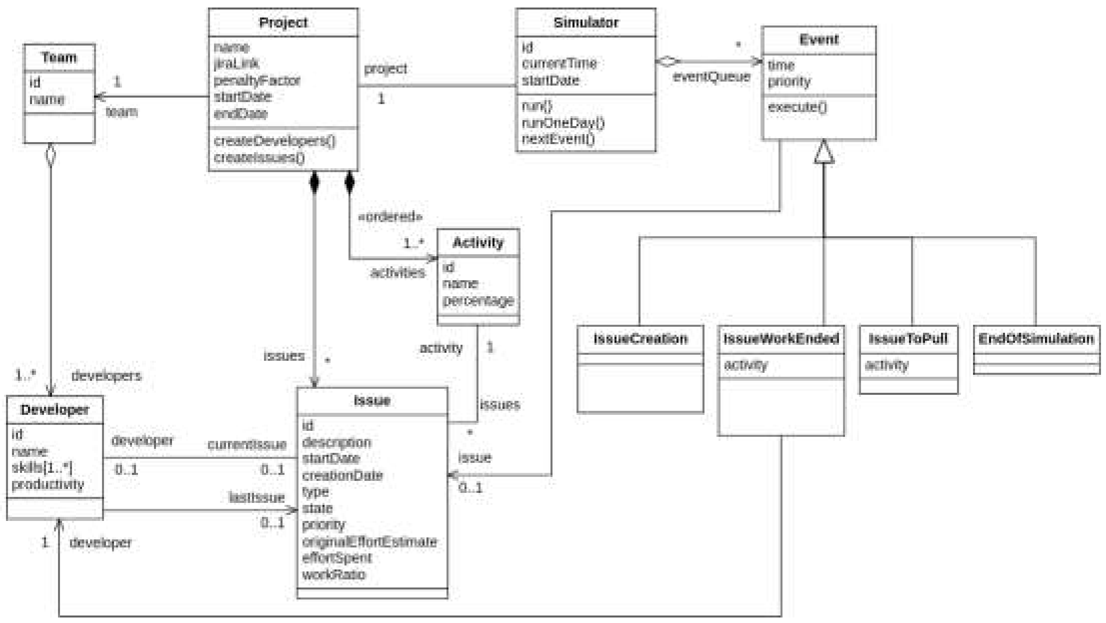


Figure 13 UML simulator class diagram.

3.4.3 Simulator Design

The simulator is implemented using Smalltalk, a language very suited to event-driven simulation and very flexible to accommodate any kind of changes and upgrades to the model. The simulator design is fully object-oriented, allowing to easily add new features if needed.

The simulation model records all the significant events related to issues, developers, and activities during the simulation, to be able to compute any sort of statistics and to draw various diagrams.

In the simulator, software requirements are decomposed into issues that can be independently implemented. The implementation is accomplished through a continuous flow across different activities, from being in Backlog to being Closed. The work is performed by a team of developers, able to work on one or more activities, depending on their background.

The different entities of the simulator are represented in the class diagram shown in Figure 13. Here you can find the four basic classes outlined in section 3.4.1, plus other key classes. The Simulator is a *singleton* (a class with just one instance) managing the simulation through the event queue. A Project defines its activities and holds the pertaining issues. The present version of the simulator allows simulating a single project, with one team working on the issues.

The team is composed of a given number of developers. Each i-th developer is characterized by a skill array (one skill for each Activity), and a productivity factor at time t, $q_i(t)$, obtained as the ratio between the number of closed issues of i-th developer at time t, $C_i(t)$ and the number of project days elapsed:

$$q_i(\mathrm{t}) = \frac{c_i(\mathrm{t})}{t} \qquad (1)$$

Each developer works on an available issue until the end of the day, or until the issue has been completed. When the state of the system changes, for instance, because a new issue has been introduced, an issue is pulled to an activity, the work on an issue ends and, in any case, at the beginning of a new day, the system looks for idle developers and tries to assign them to the issues available to be worked on in the activities they belong to.

As reported in section 3.4.2, the developer's productivity may be affected by the penalty factor p used to compute issues time effort in case of developer switching.

The penalty factor p is equal to one (no penalty) if the same developer, at the beginning of a day, works on the same issue s/he worked the day before. If the developer starts a new issue or changes issue at the beginning of the day, it is assumed that s/he will have to devote extra time to understand how to work on the issue. In this case, the value of p is greater than one and the required effort to complete the issue is the original effort, multiplied by p.

3.4.3.1 Issues

The issues that make up a project are categorized into various types: features, tasks, epics, bugs, user stories, which are given as inputs to the simulator. In the present model, the issues are atomic - meaning that they can be implemented independently from other issues - and are explicitly linked to a specific project.

New issues can be created as time proceeds. Each issue is characterized by an id, a state, the original effort estimate, the effort actually spent to date. All efforts are in man-days. The possible states of the issue are shown in Figure 14. At the beginning, an issue is *in backlog*. Then, it can be chosen for development (*To Do*), and subsequently is pulled into the first activity, where it waits to be assigned to a developer (*Waiting to be assigned*). When a developer subscribes to it, its state becomes *Under work*. When work in the current activity is finished, its state becomes *Work done*, and the issue waits to be pulled into the next activity. If the activity where the work was done is the last one of the process, the status of the issue becomes *Released*, which is the final state.

3.4.3.2 Activities

Each project holds a list of all activities defining the ASD process followed. These represent the specific work to be done on the issues; each of them covers a given percentage of the total effort needed to complete the issue.

Activities can be freely configured according to the process steps. Each activity is characterized by its name, and by the typical percentage of the total estimated time effort of an issue that pertains to the activity. For instance, if an issue has an overall estimate of 10 days, and the *Testing* activity has a percentage of 15%, in this phase the feature will be estimated to last 1.5 days. The sum of the percentages of all the project activities must be one.

The first and the last activities are *Backlog* and *Live* (Released). They are special activities because no work is performed on them - so their effort percentage is zero. The former is a placeholder, containing all issues put into processing, for instance at the beginning of each Sprint in Scrum. The latter is where the completed issues are kept.

When work starts on one issue within a given activity, the actual effort of development of the issue in the activity is computed. This is accomplished by randomly increasing or decreasing it of a percentage drawn from a given distribution. Of course, the average effort of all issues must be equal to the average of their initial estimates. A way to obtain this behavior is to multiply the estimated effort by a random number r drawn from a log-normal distribution with a mean equal to 1 and standard deviation depending on the wished perturbation.

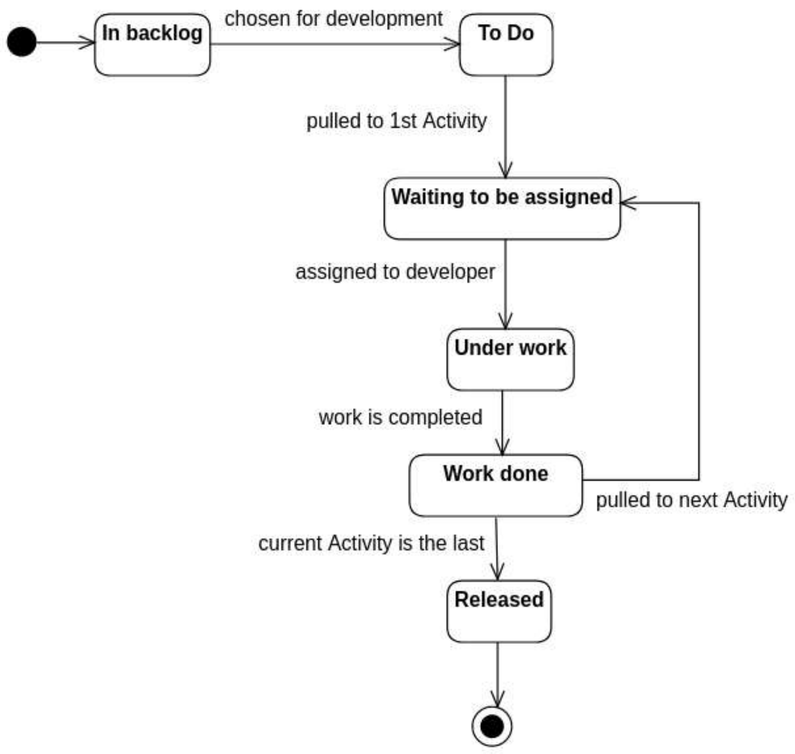


Figure 14 The possible states of an Issue.

3.4.3.3 Events

The simulator is event-driven, meaning that the simulation proceeds by executing events, according to their time ordering and priority. When an event is executed, the time of the simulation is set to the time of the event. The simulator holds an event queue, where events to be executed are stored sorted by time and priority. When an event is executed, it changes the state of the system, and it can generate new events, with times equal to, or greater than, the current time, inserting them into the event queue. The simulation ends when the event queue is empty, or possibly if a maximum time is reached, marked by an "*End of simulation*" event. Figure 15 shows an activity diagram explaining the simulator's workflow. After an initial configuration, which includes inserting into the event queue the issue creation events, and possibly the end of simulation event, the simulator main cycle starts and is shown in the diagram.

The time is recorded in nominal working days, from the start of the simulation. A day can be considered to have 8 nominal working hours, but developers can work more or less than 8 hours. To consider calendar days, it is always possible to convert from nominal working days to them. The supported events are described in Table 15.

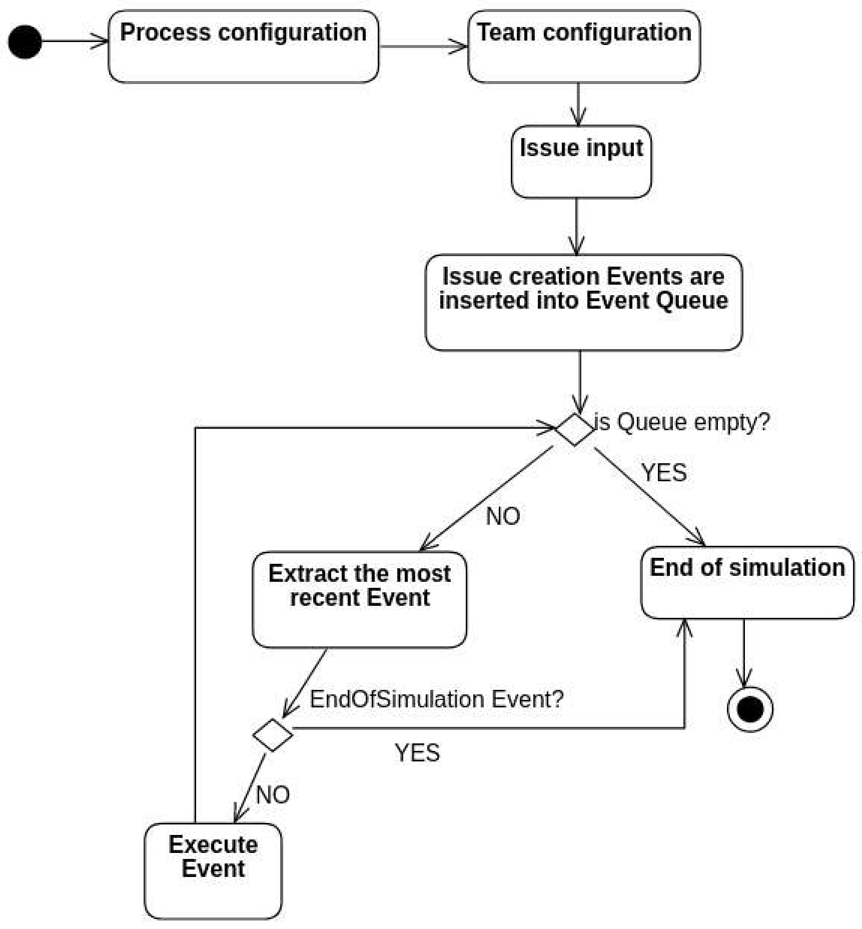


Figure 15 The UML activity diagram showing how the event-driven simulator works.

Table 15 Simulation Event Description

Event	**Description**
IssueCreation	It is raised to create a new Issue. This event refers only to issues introduced after the start of the simulation, and not to the issues already included in the initial queue. Its execution creates and inserts into the event queue, on the same time, an *IssueToPull* to activate pulling the issue from the *Backlog* activity.
IssueWorkEnded	It is raised when the work of a developer on an issue, within a given activity, ends. This may happen when the work on the issue in the activity is completed, or at the end of the working day. In the former case, the state of the issue is changed, and an event *IssueToPull* is generated for the next activity, at the same time.
IssueToPull	It is raised to request to pull an issue from one activity to the next. If the activity to which the issue should be pulled to is the last one (*Live*), the issue is forthwith moved to it, and an event *IssueWorkEnded* is created for the issue, and inserted into the event queue at the same time.
EndOfSimulation	When raised, the simulation is ended, and its results are written into a file.

The mentioned events are enough to manage the whole simulation. The simulation is started by creating the initial issues, putting them in the *Backlog* activity, and generating a number of *IssueToPull* events for the first activity equal to the number of issues ready to be pulled in the next activity, and then generating as many *IssueCreation* events for future times as required. The simulator is then asked to run, using the event queue created in this way. When the work on all issues has been completed in all activities, no more issues can be pulled and no more work can start, so the event queue becomes empty, and the simulation ends.

3.4.3.4 Component Validation

The proposed SPSM model reflects standard concepts of ASD: features (here called issues), development activities, team members, events (which are instrumental to simulate an iterative, o continuous-flow development). Most of its parameters are directly taken from real project data through an interface to an issue management system (see next Section 3.4.4).

A few parameters, however, must be properly estimated and validated. The penalty factor p described above was set to p=1.15, meaning a 15% increment in the work to be done when a developer changes the issue s/he is working on. This figure is consistent with the data by Tregubov et al. [128] who estimate that "*developers who work on 2 or 3 projects spend on average 17% of their effort on context switching between tasks from different projects*". In this case, the switching can be also between different issues of the same project, and the penalty is applied only to the time to work on that issue for the present day.

Another key parameter to be estimated, which may differ from project to project, is the percentage of work to assign to the various activities of a project. Following the same approach reported in [129], we assigned most of the effort (70%) to the "*in progress*" activity - representing the actual development made on an issue - and a residual effort to testing (15%) and deploying (15%) activities.

This choice was reviewed and approved by six expert project managers (more than 10 years of experience in managing medium-to-large size Java and Python projects) of three firms we work with, and by one expert consultant in Python and Javascript development. Their experience is mainly in Web applications (including apps for mobile devices), ERP platforms, business intelligence applications. Using a Delphi online panel, we got seven answers (with percentages obtained by rounding to five percent), made a refinement round, and adopted the median result.

3.4.4 JIRA Interface

The simulator reads data directly from JIRA through its APIs. JIRA REST APIs allow users to use several kinds of REST calls to get information about projects, issues, developers, and more. The system asks all data about a project with the specific query:

```
--data '{"jql":"project =  GROOV",
"startAt":0, "maxResults":100},
"http://localhost:8099/rest/api/2/search"
```

JIRA responds by providing a JSON file that includes the following fields: project name, project starting date, project workflows, developers, a backlog of issues, issues arrival dates, issue types, issues estimate, times, issues original times, issues times spent. Then the simulator parses the file and organizes the data to be processed. In particular, it builds the initial queue (backlog) of issues according to their priority. For each issue, the simulator sets all attributes (key, effort spent, original effort estimate, issue type, developer, issue state, issue description, …), then creates the collection of developers, the collection of activities, and finally sets up the project's starting date.

3.5 Research Design

The overall research objective is to better understand how to use SPSM to perform risk analysis on ASD projects. The inputs to the model, taken from an ITS such as JIRA and related to an ASD project, are information on team members involved, and information on all issues managed at a given time. The latter kind of information includes the date an issue entered the system, its original effort estimate, the actual time spent on it, and an estimate of the remaining effort. Information on the ASD process used must also be collected from the team. It includes:

- the sequence of activities performed by the team; at least they include: *Backlog*, *In progress*, *Done*;
- the duration of a Sprint, and the way Story Points are computed to decide how many USs to include in each Sprint if Scrum is used;
- the maximum number of USs allowed in each activity if the Lean-Kanban approach is used.

Starting from this information, it is possible to compute the outputs of SPSM, which typically are:

- The project duration if no more issues are entered;
- The number and effort of issued forecast to be closed on a given date;
- An estimate of average, standard deviation, and median of the issue cycle time, that is the time needed from the start of item processing to its completion.

Figure 16 shows the basic inputs and outputs of our model, based on JIRA issue data. Table 16 shows the parameters used in the simulation, including the class they belong to, their type, and whether they were read from JIRA, estimated from JIRA data, set after expert consultation, or dynamically computed during the simulation.

To perform risk analysis using SPSM, first we need to assess the suitability of the simulator to effectively model the software development process. This can be accomplished by running the simulator on real data and verifying its ability to mimic the real development.

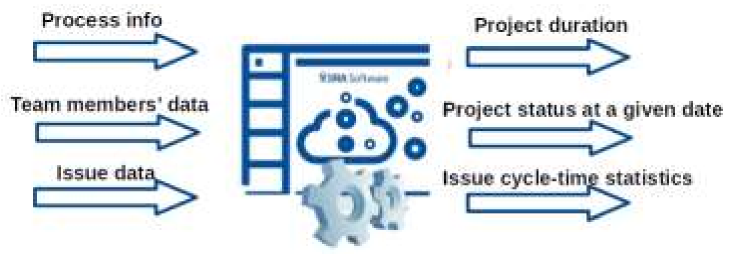


Figure 16 The Inputs and Outputs to JIRA simulation model.

Subsequently, we need to define what are the causes of risk, which can affect the desired outcomes of the project - in this case duration and cost. The main risk factors identified are erroneous effort estimates of the issues, suboptimal allocation of the issues to developers, and blocks and impediments to the work of developers. Once the statistics of these factors are precisely defined, a Monte Carlo simulation can be used to assess the risks quantitatively.

Table 16 The simulator inputs, highlighting the inputs imported from JIRA. "PART." means "partially".

Class	**Parameter**	**Type**	**Description**	**From JIRA**	**Notes**
Project	ProjectName	String	Name of the project	YES	
Project	ProjectStart	Date	Start date of the project	YES	Corrected to the date of the first completed issue
Project	NoOfActivities	Integer	Number of activities	YES	
Activity	ActivityID	String	Unique short identifier	NO	Generated by the system
Activity	ActivityName	String	Name of the activity	YES	Chosen through expert consultation
Activity	ActivityEffort	Percentage	Effort perc. of the activity	NO	
Team	NoOfDevs	Integer	No. of developers	YES	
Developer	DevID	String	Unique short identifier	NO	Generated by the system
Developer	DevName	String	Name of the developer	YES	
Developer	Skills	Float[ ]	Array of the dev. skill for each activity	PART.	Estimated from JIRA data
Developer	Productivity	Float	Productivity factor	PART.	Estimated from JIRA data
Issue	IssueID	String	Unique short identifier	YES	
Issue	IssueDescr	String	Issue description	NO	Not used
Issue	IssueStart	Date	Start date of issue development	YES	
Issue	IssueCreation	Date	Creation date of the issue	YES	
Issue	IssueType	String	Type of the issue	YES	
Issue	IssueState	String	State of the issue	NO	Dynamic value
Issue	IssuePriority	String	Priority of the issue	YES	
Issue	IssueOrig-Estimate	Float	Original estimated effort	YES	

Issue	IssueEffortSpent	Float	Original remaining effort	YES	
Issue	IssueWorkRatio	Percentage	Actual percentage of work performed	NO	Dynamic value
Issue	Assignee	String	Name of developer in charge	YES	Converted to DevID of the developer
Simulator	PenaltyFactor	Percentage	Penalty factor p for issue switchng	NO	See section 843.4.3.4
Simulator	StDvPertErr	Float	Standard dev. of the perturbation on issue estimate	NO	Typically set to 20%

3.6 Experimental Data

I analyzed three open-source projects which comply with the following preconditions:

1. they are tracked on the JIRA System,
2. they have *CreationDate*, *OriginalTimeEstimate*, *TimeSpent* and *Assignee* fields filled out for each issue; the last two are reported in man-hours;
3. they are medium size (meaning a number of issues above a few hundreds, and below one thousand).

The medium-sized requirement allows us to perform a deeper analysis of the project and the correspondent outcomes when two-month intervals are examined, which would be too time consuming for large size projects. The size is also kept limited to be able to perform several Monte Carlo runs (≥ 100).

The selected projects have a duration of around 15-20 months and an average period of 330 days. The number of team members varies between 15 and 60, but in the latter case team members do not work simultaneously, so on average, there are 15-20 developers active.

These projects, all tracked in JIRA, are built for different domains and purposes, so they present differences in topic, duration, team size and composition, workflow. The simulator can quite faithfully reproduce each of them.

3.6.1 Project: Test Engineering

The first development project is TE (Test Engineering), carried on by edX (www.edx.org), a consortium of more than 160 universities offering courses and programs through an e-learning platform completely open-source. TE is an internal project to perform testing and continuous integration of the software developed for edX and by edX partners. TE is an ongoing project, which I analyzed for a total of 570 working days, including 675 issues classified as bug, epic, story, and task. The team is composed of 13 developers with different skills and productivity, inferred by analyzing the number and effort of issues completed by each developer in the considered time interval. Issues effort estimation follows a very skewed, fat-tail distribution, which is fairly approximated by a Pareto distribution:

$$p(X) = \frac{b}{X^{b+1}} \quad (2)$$

where X is expressed in man-days, and the shape value is b≈1.35.

The basic statistics for the TE project are shown in Table 17.

Table 17 Projects' Statistics. Effort statistics in man days.

Project name	**Days**	**Issues**	**Developers**	**Mean**	**Std Dev**	**Median**
Test Engineering (TE)	570	675	15-20	2.9	4.2	1.33
Platform	622	853	14-15	4.4	6.4	1.1
CORD	192	523	13	4.3	6.4	1.0

3.6.2 Project: Platform

The second project is also carried on by edX and regards the e-learning platform of the edX consortium. In fact, it is called "Platform". Platform is an ongoing project, which I analyzed for a total of 622 working days, including 853 issues classified as: subtask, bug, story, and epic.

The team size is 65 people. At a first glance, it might seem too big, but after a careful check, I found that there was a high turnover. Most team members worked only for a limited amount of time so that the average team size is about 14-15 people, similar to the other projects.

Issues effort estimation approximately follows a Pareto distribution with shape value b=1.38. The basic statistics for the Platform project are shown in Table 17.

3.6.3 Project: CORD

The last project is called CORD (Central Office Re-architected as a Datacenter). It is a project of the Open Networking Foundation (ONF), a non-profit operator-led consortium driving transformation of network infrastructure and carrier business models. CORD is a platform leveraging leading edge SDN, NFV and Cloud technologies to build agile in-line datacenters at the edge of operator networks.

I analyzed the CORD project for a total of 192 working days, including 523 issues classified as: subtask, feature, bug, story, and epic. Issues effort estimation approximately also follows a Pareto distribution with shape value b=1.51. The basic statistics for the CORD project are shown in Table 17.

3.6.4 Simulator Assessment

The first step to assess the validity of the simulator was to check whether it can produce time duration for completion of all issues similar to those of the real projects.

I performed simulations giving as input all issues which resulted in *Closed* state, or with a remaining effort equal to zero, among all issues read from JIRA repositories. The work on each issue was performed by the same team member of JIRA data, using the developer's productivity as estimated from the same data.

For performing the simulations, I shortened their duration to an interval of about 70% of the total, starting from the date of the first completed issue. This is because almost all issues created afterward were not yet completed.

Two of the selected projects have a simulation duration of about 14 months (408 and 445 days). CORD was tested for a shorter period of 138 days, or 4.5 months. The number of team members is around fifteen people in all projects, remembering that this is an average team size, obtained by putting to work only the developers who are active in the proper time interval, as inferred by JIRA data.

These projects are built for different domains and purposes, so they present differences in topic, duration, team size and composition, workflow. The simulator can faithfully reproduce each of them. All projects are decomposed into several issues. Each issue has a total effort estimation given in man-days. Different kinds of issues belonging to a project may have a different workflow. This is modeled according to the real schema including all main activities (states).

The results for the three projects are shown in Table 18, which shows the actual duration with the average duration of the simulations. Standard deviations are shown within round brackets after the means.

Table 18 Total project duration. Mean (st. dev.) over 100 simulations.

		N. days to close project	
Project name	**N. Issues**	**Real case**	**Simulated case (s.d.)**
Test Engineering (TE)	285	408	433 (31)
Platform	321	445	415 (26)
CORD	205	138	154 (12)

Besides running the simulator for the whole development time, I also refined the effectiveness tests by considering shorter time intervals. I considered time intervals of 60 days, being two months the shortest time suitable to get significant results. Trying shorter duration did not work, both because some issues require at least 60 days to be completed, and because the number of issues to test in each interval becomes too small.

I divided the project duration into intervals of 60 days, considering for each interval the issues actually created, and the team members actually working in the real project. The issues not yet completed at the end of the interval are left to the next one for the remaining part and are considered to be completed only when their status becomes *Closed*. In these tests, the key parameter is not the duration of the project, but the number of issues completed in each interval, and the total number of issues completed at the end of the last interval.

Table 19 shows statistics and results of the simulations for the three examined projects. Note that overall time intervals considered are longer than those of the preceding simulations, and run as long as possible, compatibly with the constraint to be multiple of 60 days. The reported mean and standard deviations are made on real issues, over the 60 days time intervals. The simulated issues are averaged over 100 simulations for each project.

To give further insight on these simulations, Table 20 reports the number of issues closed in every 60-day interval for project Test Engineering. The number referring to simulated issues is again the average of over 100 simulations. The last columns report the real values. Similar behavior can be found in the issue management of the other projects. As you can see, the number of closed issues varies a great deal from two months to two months. The simulation can reproduce quite well the number of closed issues in every 60-day interval. I deemed these results good enough to consider the simulator suitable for its use in risk analysis.

Table 19 Results of simulations with 60 days intervals.

Project name	**Days**	**Issues**	**Mean**	**Std Dev**	**Simul. Issues**	**Diff.**
Test Engineering (TE)	540	369	41	24.5	370	1
Platform	600	431	43	26.9	426	-5
CORD	180	265	83.3	62.9	267	2

Table 20 Issue Management in 60-day intervals.

TE (Test Engineering) Project		
Days	**Closed Issues (Mean)**	**Real Closed Issues**
60	56	54
120	10	9
180	39	36
240	58	60
300	44	46
360	84	85
420	25	26
480	10	9
540	44	44

3.7 Risk Assessment Through Simulation

I applied the Risk assessment methodology outlined in Section 3.3.1 on the reported cases, to test its effectiveness. The key risk factors identified are variations in efforts estimated by developers to complete USs, and random developer issues assignment - to be compared with assignment according to real data. In this preliminary study, I did not consider variations in the skills of team members, and events stalling the development of single features or blocking one or more developers, though the simulator could account also for these factors.

The effort estimation error of each issue is modeled using random variations. The Percentage differences between estimated and actual times to close an issue in the three projects are very close to zero, and show a standard deviation between 0.17 and 0.26. Averaging on all closed issues of the three projects, the standard deviation is 0.22, which I approximate to 0.2.

So, the effort estimation error is obtained by multiplying the original issue effort value by a number that follows a log-normal distribution with an average equal to one, and a standard deviation equal to 0.2. In this way, the efforts averaged on all issues remain equal to the average of original issues, and the standard deviation of errors is very similar.

The key process outputs whose variations are checked are the project total time and statistics on cycle times to implement a feature. The time from the project start to the time when the last feature is released is inversely proportional to the throughput and is an excellent indicator of the good performance of the project.

The statistics on cycle times, measuring time to give value to the customer, are:

- average time to implement a feature;
- standard deviation of the time;
- median time, measuring the most likely time to complete a feature;
- 95% percentile of times, measuring the limit of the worst 5% times;
- 5%percentile of times, measuring the limit of the best 5% times;
- maximum time, measuring the worst case.

I performed a given number of Monte Carlo simulations - typically one hundred, but they might be more - for each choice of the tested risk factors, varying random perturbation, or developers' assignments to issues, recording the key outputs.

From these outputs, it is then possible to compute the desired statistics. On each of these values, it is possible to set risk thresholds that, if reached or overcome, trigger proper mitigation actions.

3.7.1 Real Cases Risk Analysis

I used the three open-source projects described in Section 3.6, considering the time interval from the first issue project creation date until the end of the projects. I started by analyzing the time estimation of project duration when only time estimation variations are considered as risk factors, but issue resolutions are made by the same developers of the real case. For a better interpretation of the results, I divided the simulations into time intervals of 60 days, as made before.

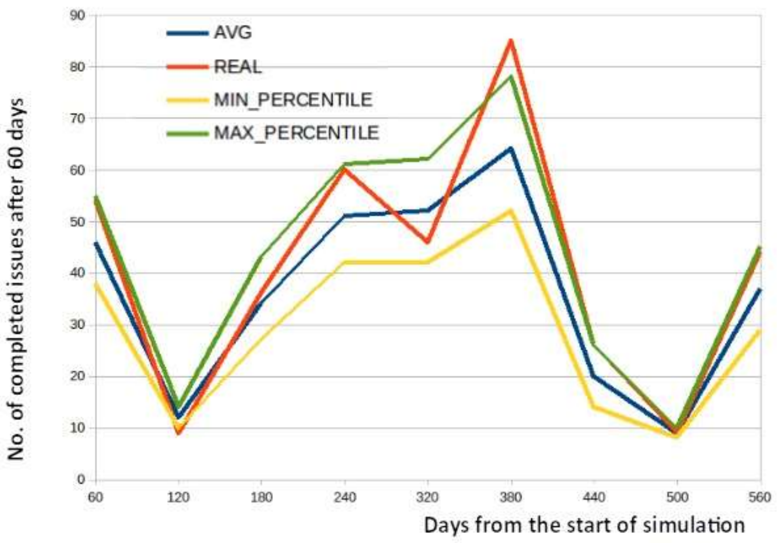


Figure 17 TE Project: N. of completed issues vs. time. Averages and percentiles over 100 simulations.

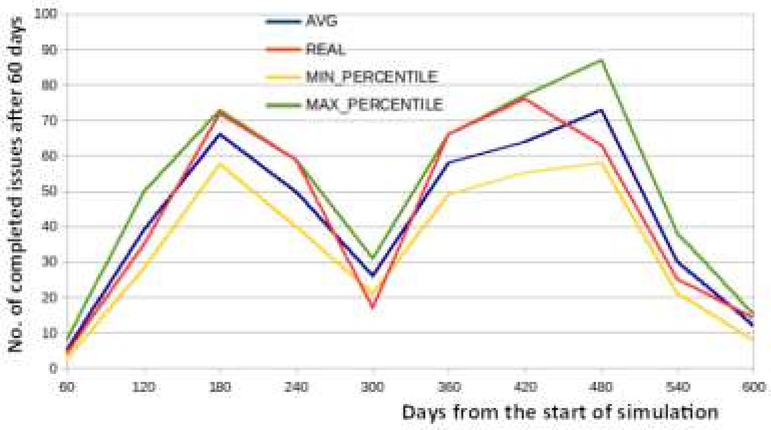


Figure 18 Platform Project: N. of completed issues vs. time. Averages and percentiles over 100 simulations.

Figure 17, Figure 18 and Figure 19 show the results for projects TE, Platform and CORD, respectively. Here, for each time interval, the values of the real number of issues completed in that interval, of the average on 100 simulations varying the issue estimates as described in Section 3.7, and of the 5% and 95% percentiles of the simulated completed issues are shown. I do not show the medians of completed issues because they are very similar to the averages, and would not add significant information to the figures.

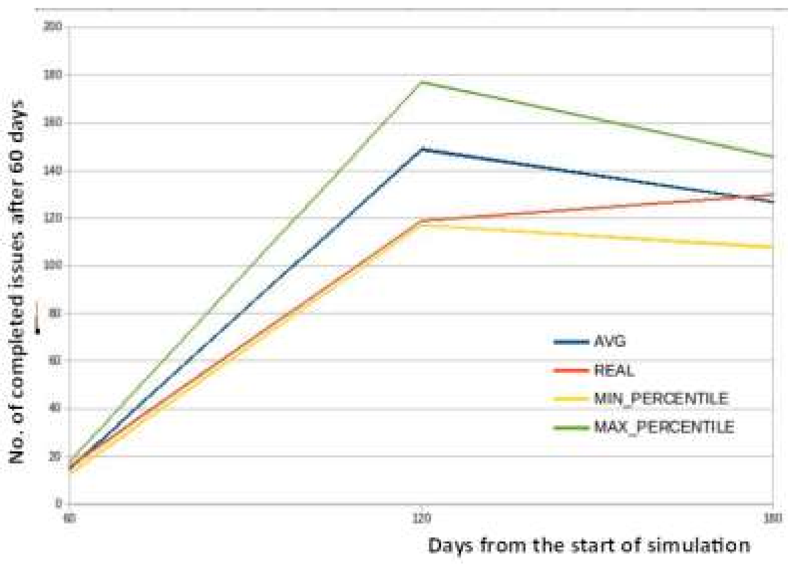


Figure 19 CORD Project: N. of completed issues vs. time. Averages and percentiles over 100 simulations.

In all projects, the number of completed issues every 60 days greatly varies. This variability is due to different reasons, such as the different commitments of the developers in different months, the tendency of issues to be completed in "batches" rather than in a continuous flow, and the low number of active developers, which makes the project output more susceptible to random variations.

Regarding the TE project, the real number of completed issues and the average of simulated numbers are quite close to each other, being the maximum percentage error equal to 33%. The real number of completed issues is almost always contained between 5% and 95% percentiles, but in the second and sixth intervals. Only in the latter case, the discrepancy looks significant. In these two cases, the risk analysis would have triggered corrective actions.

Note, however, that the average estimation of all completed issues is 354, with a percentage error of 4%, being the real number equal to 369. Considering the cumulative number of completed issues over time, the error peaks at 180 days with a value of 17%, and then decreases. On day 360 and thereafter, the error is always under 4%.

In the Platform project, whose results are shown in Figure 18, the differences between real and simulated cases look slightly lower than in the TE project. In time intervals with very few issues completed, obviously, the percentage errors are quite high, but the overall percentage difference between the total number of completed issues (423) and the average number of simulated ones over 100 simulations (416) is about 2%.

Considering the cumulative number of completed issues over time, the error is always under 6%, but after the first 120 days, where it is 25% after 60 days, and 13% after 120 days.

In Platform project, 5% and 95% percentiles over 100 simulations include the real number of completed issues, but for the result at 300 days, where the real number is lower than the 5% percentile.

The results of the CORD project are shown in Figure 19. Here there are only three simulated intervals of 60 days, with very few completed issues after the first one. For this reason, the differences between real and average numbers over 100 simulated cases of the completed issues are higher, being 25% after the second time interval.

The average of total closed issues after 180 days is 291, versus the real value of 265 (the difference is 10%). The real number of completed issues is always contained between 5% and 95% percentiles of the simulated cases.

3.7.2 Risk Analysis with Random Developer Allocation

In this section, I consider the effect of the application of both risk factors in the simulations, namely random estimation errors and random allocation of issues among developers. The estimation errors are introduced using the same approach of the previous section, multiplying the original estimates by a random number following a log-normal distribution with an average equal to one, and a standard deviation of 0.2.

The allocation of developers to issues is not equal to the real case, but developers simply "decide" what issue to work on depending on a random choice of available issues with the highest priority. This mimics better the real situation, where of course it is not known in advance which developer will work on which issue, and a risk analysis must consider an issue allocation not known in advance.

As in the previous Section, I divided the simulations into time intervals of 60 days, and report the number of issues completed in each of them. I report the number of issues of the real case, the average and the 5% and 95% percentiles computed over 100 simulations.

Figure 20 shows the results for the TE project. As you can see, the four curves are closer than in the case without random developer allocation, as shown in Figure 17.

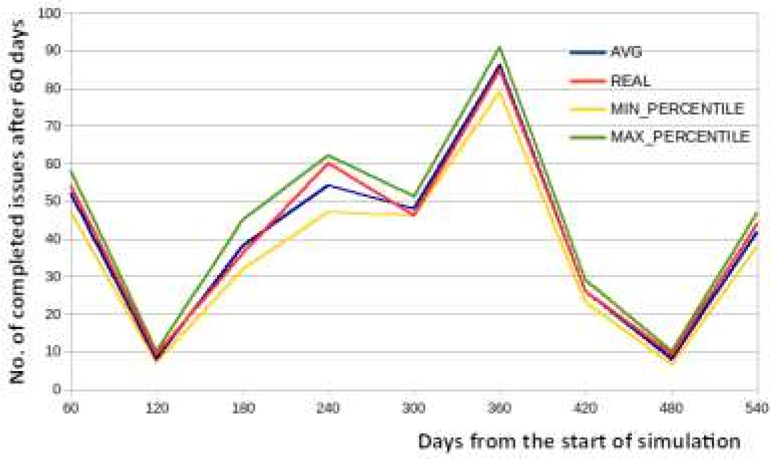


Figure 20 TE Project: N. of completed issues vs. time. Averages and percentiles over 100 simulations, with random allocation of developers.

These results are consistent with those found without risk parameters. The overall percentage difference between the total number of completed issues (369) and the average number of simulated ones over 100 simulations (362) is about 2%.

Considering the cumulative number of completed issues over time, the error is always under 5%. The 5% and 95% percentiles over 100 simulations always include the real number of completed issues.

The reproducibility of real data is well provided by the model. Having in place both risk parameters results in outputs that, on average, are closer to real data with respect to using only effort estimation perturbations.

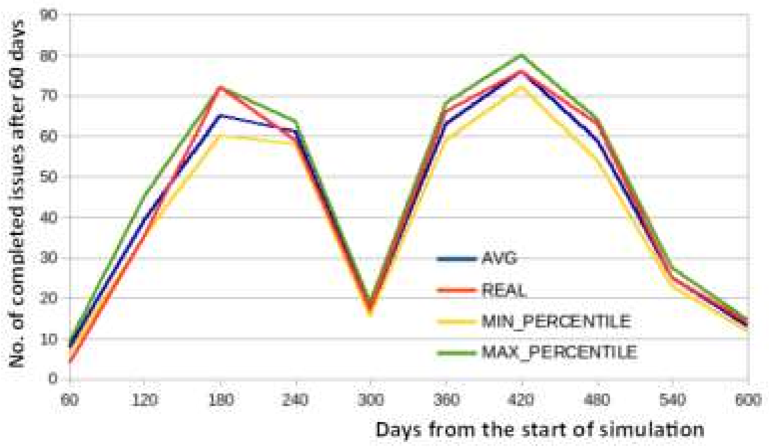


Figure 21 Platform Project: N. of completed issues vs. time. Averages and percentiles over 100 simulations, with random allocation of developers.

The data on the Platform project are reported in Figure 21. Also here, the four curves are closer than in the case without random developer allocation, as shown in Figure 18.

The overall percentage difference between the total number of completed issues (426) and the average number of simulated ones over 100 simulations (431) is about 1%.

Considering the cumulative number of completed issues over time, the error is always under 11%, but for the first time interval where it is 100%, due to the very low of completed issues (4 in the real case, 8 in the average of simulated cases). The 5% and 95% percentiles over 100 simulations include the real number of completed issues, but again at the end of the first time interval.

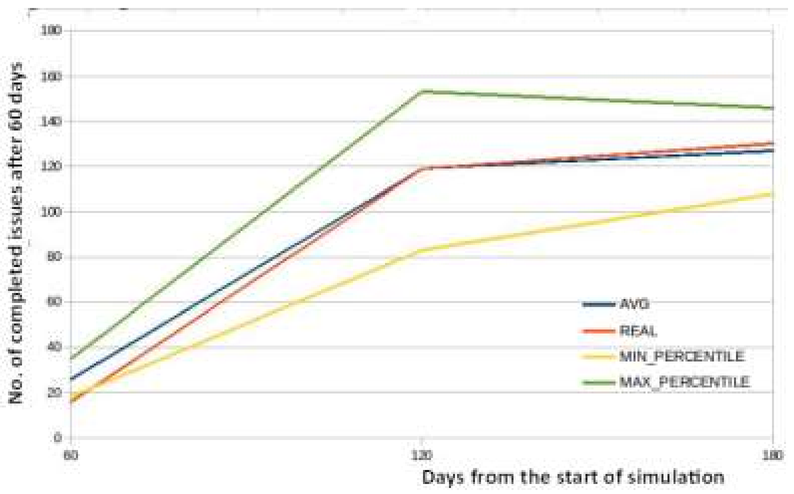


Figure 22 CORD Project: N. of completed issues vs. time. Averages and percentiles over 100 simulations, with random allocation of developers.

Eventually, Figure 22 shows the results for the CORD project. The results do not differ much from the case with variations only in issue effort estimation and are slightly better.

The average of total closed issues after 180 days is 271, versus the real value of 265 (the difference is 2%). The real number of completed issues is well contained between 5% and 95% percentiles of the simulated cases, but after the first time interval, for the usual reason of a very low number of issues.

3.7.3 Discussion

Starting from the analysis of three real projects tracked on JIRA, I made four different types of analysis with four scenarios.

The first scenario had the aim to test the simulator reliability by comparing the number of days needed to close the project in the real and simulated cases for the three projects, as shown in Table 19. The simulator reproduces the project duration in days with an error margin between 6% and 11%, considering the average of 100 simulations, with a standard deviation between 6% and 8% of the related average.

The second scenario is aimed at improving forecasts by reducing the time interval of predictions to time intervals of sixty days each. In this way, the model simulates the number of closed issues for a limited period and resets itself at the end of the interval to perform the next interval forecast, according to the supplied real data. The results show that in this way the model performs better than using the entire project lasting time.

Here the key output is not the duration of the project, but the number of issues that are completed during each time interval. In the case of issues only partially completed, they are not considered but are moved to be completed to the next time interval. The results of these simulations, obtained again by performing 100 simulations for each interval and averaging the number of issues completed after each simulation, show that the percentage error between the average and the real data is typically less than 10% in all intervals, except sometimes the first one, characterized by a low number of completed issues in the real case.

Summing up, all completed issues on all intervals yields a total number of completed issues (averaged over 100 simulations) which differ less than 1.5% with respect to the real number of completed issues.

These scenarios prove that the simulator can reproduce with enough precision the real ASD process. Referring to the Research Questions asked in the Introduction, I am able to answer to RQ1 and RQ2:

- **RQ1: To which extent it is possible to automatically import into the simulator data of real projects, from issue management systems?** Importing issue data directly from the popular system JIRA is quite straightforward. However, importing the sequence of activities actually used for a specific project, as well as estimating the skills and time commitment of developers need manual intervention. This must be done before the simulation starts, by analyzing project data. During project execution, new issues can be periodically read to update the simulation state, without further intervention.
- **RQ2: How accurate can the simulator be in predicting project completion times?** The case studies analyzed show that the simulator can be quite accurate in reproducing a real project progress - that is, the project completion time - if it is fed with accurate data about issue estimates and team composition and skills. Simulating whole projects, which lasted between six and twenty months, the error margin is of the order of 10%. Shortening the simulation intervals to two months, and then resynchronizing the simulator, yields an even lower error of the order of 1-2% averaged over 100 simulations. So, the answer to this question is that the simulator accuracy to predict project completion times is high.

The third and fourth scenarios introduce the use of the simulator for risk analysis. Basically, in both scenarios, random perturbations are applied to issue estimations, chosen from a log-normal distribution, derived from statistical analysis of the real data set. The fourth scenario adds a random choice of the developers who subscribe for resolving an issue.

I ran 100 simulations for each considered project, for both scenarios, to check the extent to which the simulated outputs differ from the real ones, in the case of random estimation errors. So, I computed not only the mean value of the forecasted number of closed issues in the time intervals but also proper percentiles, helpful to check if the real values stay within these limits.

I found that just applying random perturbations to issue estimation - which in a real project would mean that the original estimation was wrong, whereas the perturbated one is the “right” value - has limited impact on the goodness of the approximation of the simulation results with respect to the real case. This is shown in Figure 17, Figure 18 and Figure 19, and in the discussion thereof.

When random issue assignment to developers is added, the results are even better, meaning that the simulation outputs tend to be even closer to the real ones, as shown in Figure

20, Figure 21 and Figure 22, and in the discussion thereof. This result can be attributed to the fact that in the real case simulated work on the issues is performed by the developers who registered it in JIRA. So, the simulator applies a rigid assignment, which mimics the delays and unavailability of real development. When issues are randomly assigned, however, when work on an issue is needed in a given activity, all available developers are polled, and one is chosen at random. If there are available developers, work on the issue begins with no delay, and this justifies the better results in terms of completion time.

Practically, the simulator can be applied to actual risk management by applying the proper random variations to the parameters that could be the major causes of risk, performing a sufficiently high number of simulations, and evaluating the distributions of key output parameters. Moreover, in a real case, future issues can be randomly generated, using their expected effort distribution, to get a more realistic simulation of the work to be performed.

Being still an ongoing research project, the simulator has not been yet used in real development. To get feedback, I presented the results to the same experts involved in helping us to estimate the penalty factor p, as reported in Section 3.4.3.4 "Component Validation". Most experts encouraged us to continue the work, believing that assessing project risks using SPSM is a valid and promising approach, provided that other kinds of risks are considered, besides error in issue estimation. A couple of the project managers stressed that this tool might be useful, but only if provided with a user interface able to ease the tuning of the simulation parameters and to immediately highlight the risks to exceed time and costs.

This leads to the answer to the last research question:

- **RQ3: Can the simulator be useful to estimate project risk (induced by errors in efforts estimation, and random developer issues assignment) with a Monte Carlo approach?** By varying parameters - such as the variance of issue estimation errors, and developers' availability - and performing Monte Carlo simulations, a project manager can compute statistics on forecast project completion times and average time to complete issues and take proper action to control this kind of risk. Consequently, also the answer to this question is positive. Clearly, much more causes of risk might be accounted for, as described below, and we are actively working to include them in the simulator.

In this research, for the sake of simplicity, I limited to use just issue effort perturbations, and random developers' assignment. However, other causes of risk can be easily simulated, such as:

- Sequentiality constraints between issues, so that delaying or stalling the development of an issue directly influences other issues.
- Problems related to specific issues, whose development is delayed or stalled for external reasons.
- Change or deletion of existing issues.
- Different skills of developers in the process activities, and consequent preferential choices of issues to work on.

Other important causes of risk, which can also be simulated, and which embody random components and add "uncertainty" to the process are shown in the followings. The distributions of the value of these factors must be studied and defined in advance, typically by analyzing the past history of the project, or of similar projects:

- Arrival of new issues, real or simulated to account for forecasts of future work to be done; here the distribution of issue effort estimates, issue priorities, and issue arrival time must be defined. Also, change requests of existing issues not yet implemented might be considered.
- Issues which do not pass quality controls, and must be reworked, whose probability and extent of rework must, in turn, be specified.
- Changes in the availability of team members, due to various causes (vacation, illness, resignation, more important commitments to carry out). This is an important risk factor, able to substantially change the project schedule. Again, the probability and duration of developers' unavailability must be defined in advance

Clearly, in real projects, this approach should be embedded in other risk management approaches, which include risk identification and risk mitigation. This SPSM-based approach can substantially help in quantitative risk analysis but cannot cover different kinds of risk that cannot be modeled and simulated, such as Organizational Environment Risk, User Risk, and Team Risk [81]. In other words, I do not claim that the simulation technique is better than other known techniques to manage risk, but I claim that it should be used as another tool, able to complement other approaches.

For instance, in Rm4Am risk management tool for ASD [106], this tool might be used for assessing the risk of Increments (i.e., user stories/features), in the risk analysis of Product and Sprint backlogs. This should be done during the Risk weekly meeting, a subcomponent specifically added in Rm4Am to manage project risk.

3.8 Threats to Validity

The goal of this section is to discuss the threats to validity that I need to consider regarding the study of JIRA open-source projects performed to evaluate the presented risk assessment method. The typical threats to validity taken into consideration in a software process development are construct validity, internal validity, and external validity [130].

- *Internal Validity* is the approximate truth about inferences regarding cause-effect or causal relationships. Thus, it is only relevant in studies that try to establish a causal relationship and it is not relevant in most observational or descriptive studies. In this case, the analysis is focused to demonstrate the abilities to manage the risk using statistical analysis, and internal validity is not relevant to find this kind of relationship.
- *Construct validity* is focused on the relation between the theory behind the experiment and the observations. Even when it has been established that there is a casual relationship between the execution of an experiment and the observed outcome, the treatment might not correspond to the cause one thinks to have controlled and altered,

or the observed outcome might not correspond to the effect people think they are measuring. In this case, the main threat to construct validity is whether the models are accurate enough to be realistic. In other words, are issues, activities, and developers, as modeled by us, enough reliable to get plausible results? Other studies on empirical data seem to answer favorably to this question [114] [129], but more research is clearly needed. Another issue related to constructing validity is the characteristics of feature effort variations and the random issues to developers' allocation. Though these characteristics are common to many projects, the exact distribution of effort variations and the random assignment performed might be improved. Moreover, there are other possible risk factors, such as inaccurate requirements and team-related issues, such as the resignation of developers, the introduction of new developers, intra-team dependencies, and so on. What I proposed is a model able to represent some key aspects of software development, without trying to model every possible aspect.

- *External validity* is concerned with whether one can generalize the results outside the scope of the study and hold them across different experimental settings, procedures, and participants. If a study possesses external validity, its results will generalize to a larger population not considered in the experiment [114].

In this specific study, I performed hundreds of simulation runs. An important aspect to be taken into consideration is that data analyzed are real, but the analysis refers to just three simplified test cases. Although the scope of this research would not have allowed simulating more complex cases, this must be considered when considering external validity. Also, the fact that all features were available at the beginning of the simulated project, and that no more feature was added, limits the generalization of the results.

3.9 Conclusion and Future Work

Risk management is essential to software development. Agile methodologies were introduced precisely to minimize the risk inherent in traditional waterfall processes, with very high risks related to requirement changes and final integration. However, only a few studies have been devoted to explicit risk management of agile processes.

I presented a risk assessment procedure based on process modeling and simulation and tested it on three open-source projects developed using an agile approach, with requirements collected as user stories and managed as an atomic unit of work, using JIRA popular issue management tool. The process can be organized into a sequence of basic activities and can thus be modeled using an event-based simulation. The developers are in turn modeled as agents, who decide the units of work they develop and complete.

To be able to work with real data, I linked the simulator with JIRA to collect the needed information (used process, team composition, project size, number of issues, estimated effort) and show the reliability of the tool.

I have shown how it is possible to run the simulator in a Monte Carlo fashion, varying the identified risk factors and statistically evaluating their effects on the critical outputs. In this way, a risk manager will be able to analyze the expected parameters of the examined project, including the expected optimal closing time of the project, and of the various issues and features, and evaluate the percentiles of their distributions, to assess the probability of adverse and favorable variations.

I validated the simulator using three open-source medium-sized projects, whose data are available on open JIRA repositories, and considered four kinds of scenarios. The first two were used to test the reliability of the simulator; the third and fourth scenarios were used to make a risk analysis introducing the variation in the estimated effort to complete features, and when there are changes in their allocation to developers.

The proposed approach is clearly relevant for project managers, who get a tool able to quantitatively evaluate the risks, provided that the process and the project's data are properly modeled. This is made possible by the relative simplicity of Agile processes, that typically operate on atomic requirements – the features, or issues – through a sequence of activities.

Clearly, in real projects the approach should be complemented with other risk management approaches, able to cover different kinds of risk that cannot be modeled and simulated, such as Organizational Environment Risk, User Risk, and Team Risk [81].

I limited the study considering only the effort variations and the random assignment of developers to issues. Even though the obtained results are limited to the analyzed projects, the model could be customized and adapted for other projects.

In the future, I will improve the risk assessment method, evaluating it on many other case studies, and exploring the optimal parameter settings that can minimize the overall risk. An improvement I am considering is to scale the model from a single team to multiple teams, involved in one project, or even in several projects. This would greatly improve the utility of the tool for risk management in large organizations.

Part III

Blockchain Applications

4 Introduction

Blockchain and the programs running on it, called Smart Contracts, are more and more applied in all fields requiring trust and strong certifications. In the context of the study of blockchain applications, Part III firstly formalizes an analysis of the characteristics of a permissioned blockchain, in order to have the same benefits of a public blockchain, reducing its disadvantages, then it presents two possible applications of blockchain technology, in the agri-food supply chain domain and in an interport community network.

Particularly, in section 4.1 I propose a framework to compare public and permissioned blockchains, specifically suited for industrial applications. I also propose a complete solution based on Ethereum to implement a decentralized application, putting together in an original way components and patterns already used and proved. This solution is characterized by a set of validator nodes running the blockchain using Proof-of-Authority or similar efficient consensus algorithms, by the use of an Explorer enabling users to check blockchain state, and the source code of the Smart Contracts running on it. From time to time, the hash digest of the last mined block is written into a public blockchain to guarantee immutability. The right to send transactions is granted by validator nodes to users by endowing them with the Ethers mined locally. Overall, the proposed approach has the same transparency and immutability of a public blockchain, largely reducing its drawbacks.

Then, in section 5 I propose a novel approach for easily customizing and composing general Ethereum-based smart contracts designed for the agri-food industrial domain, to be able to reuse the code and modules and automate the process to shorten the time of development, keeping it secure and trusted.

Finally, section 6 explores the application prospects and practical implications of the application of Blockchain technology for the establishment of an interport community within which different ports organized as a network can exchange information and data in a secure and effective way.

The study presented in this chapter was partially published in:

- "A Blockchain Architecture for Industrial Applications", Lodovica Marchesi, Michele Marchesi, Roberto Tonelli, Maria Ilaria Lunesu. Submitted to Blockchain: Research and Applications, Elsevier, 2021.
- "Automatic Generation of Blockchain Agri-food Traceability Systems", L. Marchesi, K. Mannaro, R. Porcu. In Proceeding of the 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB 2021, pp. 41–48, 9474795, 2021.
- "Automatic generation of Ethereum-based Smart Contracts for Agri-Food Traceability System", L. Marchesi, K. Mannaro, M. Marchesi and R. Tonelli. Submitted to IEEE Transaction, 2022.
- "Can the Blockchain Facilitate the Development of an Interport Community?", P. Serra, G. Fancello, R. Tonelli, L. Marchesi. In proceeding of the International Conference on Computational Science and Its Applications, pp. 240-251, Springer Cham, 2021.

4.1 A Blockchain Architecture for Industrial Applications

As already mentioned, a few years after the introduction of Bitcoin in 2009, developers and business men realized that a blockchain can be also used to run a decentralized computer.

The main objective of this work is to clarify, discuss and add new ideas and tools to the structure and management of permissioned, or consortium, blockchains, that is blockchains whose nodes are run by selected organizations. These systems are also called "distributed ledgers" (DL or DLT, where 'T' is for "technology"). In principle, a blockchain is a DLT, but a DLT does not necessarily uses a chain of blocks to store its information, guaranteeing its immutability. In this chapter, I mainly use the term "blockchain", but most of the concepts can also be applied to DLT.

In particular, I address those based on blockchains similar to Ethereum, where you need to consume their cryptocurrency to send transactions able to modify the blockchain's state. I recall that the amount to consume is called "gas".

The main contributions of this work are the following:

i. I recap the features and qualities of blockchain-based systems, based on both public and the so-called "permissioned" or "consortium" blockchains. I propose a new framework for choosing the blockchain architecture most suited to a specific application.

ii. I use this framework to justify and propose an architecture for managing consortium blockchains, which retains all the positive characteristics of public blockchains, but largely reduces their drawbacks regarding scalability, privacy, cost and efficiency.

iii. I collect together, in a structured way, many ideas already present in the blockchain realm. The result is an architecture which is easily applicable to most consortium blockchains, being efficient, highly configurable and scalable. This architecture makes use of the Ethereum technology, but can be easily changed to support other blockchains, such as Hyperledger.

iv. I better formalize the typical permissioned architecture, explaining the characteristics it must have in order to have a transparency and strength almost equal to that of a public blockchain. Precisely, the permissioned blockchain must be periodically anchored to a public blockchain (which is a tool already used, especially in distributed data storage solutions), and at the same time an explorer able to explore the blockchain independently from the provided apps must be provided.

v. For blockchains based on the gas mechanism, a further contribution is to use gas (Ether or other cryptocurrency of that specific blockchain) to enable writing only for authorized actors. The idea is: instead of giving permissions depending on your login authorization, you are enabled because you have gas available. This is useful because since gas is limited, it also allows to dynamically manage the write permissions.

The reminder is organized as follows. Section 4.2 presents the related work; Section 4.3 discusses what are the requirements of permissioned blockchains for industrial applications, and introduces the evaluation framework; in Section 4.4 I apply the framework to choose the blockchain platform best suited to our purposes; Section 4.5 describes the proposed dApp architecture. Section 4.6 draws the conclusions.

4.2 Related Work

I aim to discuss the differences between public and permissioned blockchains, to propose a framework to choose among different kinds of blockchains and DLT, and to propose a specific solution to implement publicly accessible permissioned blockchains. Consequently, I will consider only the works on these specific subjects, and not generic works on blockchain and SC technologies and applications.

Regarding evaluation frameworks, some papers were published addressing the choice whether to use or not a blockchain to implement a specific information system. Among them, I may quote the seminal work by Peck [133], and the more recent works by Wüst and Gervais [134], and by Hassija et al. [135], though here I already assume that the choice to use a blockchain has already been made.

Once the choice to use a blockchain is made, there are many papers in literature that provide guidelines on how to choose the best suited blockchain technology for a specific application. In 2017, Koteska et al. investigated the quality requirements and solutions for blockchain implementations, starting from a literature review [136]. They analyzed the various quality issues of blockchain systems, focusing in particular on public blockchains. They gave a catalog of blockchain-specific quality criteria to provide high data integrity, security, reliability and node privacy.

In 2018, Scriber proposed a framework for determining blockchain applicability, which includes a list of 10 blockchain characteristics whose presence makes it desirable a dApp system [43]. Though Scriber's work is mainly oriented to decide whether a blockchain is suitable for a given application or not, I used many concepts and ideas of his framework to build this one, which is instead oriented to comparatively evaluate different blockchain solutions.

Maranao et al., working in a focus group promoted by the U.N. agency International Telecommunication Union (ITU) to assess criteria for distributed ledger technology platforms, proposed a DLT Assessment Framework [137]. They defined three layers: (i) Core Technology Layer; (ii) Application Layer; (iii) Operation Layer; and assigned specific criteria to each layer. They proposed one of the first DLT assessment frameworks to be standardized by an international standardization body. They then showed how the framework can be applied, evaluating the public Ethereum blockchain.

In their very detailed work, Gourisetti et al. proposed the blockchain applicability framework (BAF), specifically designed with the purpose of helping to choose not only whether a blockchain is suitable for a specific application, but also what kind of blockchain, consensus model, and features are most appropriate [138]. The BAF is divided into five domains, 18 subdomains, and about 100 controls, being comprehensive but conversely not easy to master and apply.

Colomo-Palacios et al. discussed blockchain assessment initiatives from a technology evolution viewpoint, from Blockchain 1.0 (Bitcoin and the like) to Blockchain 2.0 (Ethereum and SCs), to Blockchain 3.0 (IOTA, Cardano, Tezos, etc.), and Blockchain 4.0 (use of A.I., Blockchain as a Service, etc., still ongoing). They examined 9 papers on blockchain assessment models (including [43] and [137]), extracting technical and business oriented aspects. A total of 19 factors were found, 14 technical and five business oriented.

Garriga et al. proposed Chainmaster, a conceptual framework to aid software architects, developers, and decision makers to adopt the right blockchain technology [139]. They identified seven key architecture features of blockchain systems: (1) Cost, (2) Consistency, (3) Functionality and Functional Extensibility, (4) Performance and Scalability, (5) Security, (6) Decentralization, and (7) Privacy. They then analyzed the technological decisions in the most popular blockchains and DLT, and mapped them against the key features. Chainmaster framework was then evaluated on four real blockchain projects.

Regarding works supporting architectural design decisions on the blockchain most suited to an application, the literature is still very limited. Wessling et al. support the process of integrating decentralized elements, but they focus on lower level design patterns and do not provide true architectural guidance [26]. A very recent work by Woehrer and Zdun about architectural design decisions covers the implementation and integration of blockchain-based solutions [140]. They describe architectural design decisions and related options in terms of patterns and practices. Since most design decisions are driven by the need to offset current blockchain drawbacks - typically scalability, privacy, and usability - by using centralized elements, the authors conclude that a hybrid architecture is beneficial in many design situations.

Regarding the consensus in permissioned blockchains, their controlled environment, the need to get high performances, and the absence of necessity to directly compensate the validators, rules out the Proof-of-Work and the Proof-of-Stake approaches. In these blockchains, nodes are divided between validators and simple nodes. Simple nodes can send transactions and query the blockchain, whereas only validators can create new blocks and add them to the blockchain.

Most algorithms used in permissioned blockchains belong to the Byzantine fault-tolerant (BFT) consensus family [141]. In BFT, the consensus can tolerate a percentage of malicious validators below 1/3 of the total number of validators. In an industrial permissioned blockchain, validators belong to trusted organizations. So, the probability that one validator might become malicious and cheat is low, and that malicious ones become one-third of all validators, or more, is negligible. Of course, the total number of independent validators should be at least seven, and preferably ten or more.

Depending on the number of validators and simple nodes, the actual consensus algorithm used may vary. In the case of permissioned blockchains for industrial applications, it is difficult to forecast networks with more than a few tens of validators. In this case the preferred consensus mechanisms are variations of BFT, namely "Practical BFT" (PBFT) [142], Istanbul BFT" (IBFT) [143], QBFT [143], "Delegated BFT" (DBFT) [144], or other specific algorithms such as

"Clique" [143], "Proof of Elapsed Time" (PoET) [145], "Authority Round" (AuRa) [146], Tendermint protocol [144].

Comparative evaluations of these consensus mechanisms in practical blockchain testbeds were presented by Shapiro et al. [144], Ahmad et al. [145] and Gerrits et al. [143]. All these works evaluate the throughput of the system, measured in the maximum number of transactions per second (Tx/s), as a function of the number of nodes. Note that, due to communication overload, performances tend to decrease with the number of validators.

Shapiro et al. evaluated IBFT, DBFT, and Tendermint, showing that DBFT outperforms the other two by about one order of magnitude.

Gerrits et al. compared PBFT, IBFT, QBFT, and Clique, in the context of a use case taken from automotive industry. This study shows that the performance of the original PBFT is not adequate; IBFT and QBFT can handle up to about 450 Tx/s and Clique three times more.

Ahmad et al. compared PBFT, PoET and Clique, as well as Proof-of-Work and Proof-of-Stake. In their test, Clique outperformed PoET in terms of Tx/s, up to 50 validators. For more validators, Clique and PoET have approximately the same throughput. PoS was substantially slower, except for blockchains of more than 150 nodes. PoW and PBFT were always heavily outperformed.

The cited works cannot be directly compared, because they used very different benchmarks and contexts. However, DBFT and Clique look the best choices for permissioned blockchain consensus. Accordingly to Gerrits et al., Clique can handle a maximum of 1500 Tx/s, decreasing to 1100 with 25 validator nodes. Ahmad et al. claim 8000 T/s with 5 and 10 nodes, and almost 5000 up to 50 nodes. Afterwards, the throughput decreases to around 1000 Tx/s for 200 and 250 nodes. All things considered, Clique emerged as the best consensus mechanism for permissioned blockchains up to about 50 validator nodes, also due to its popularity and availability. In our experience, the number of independent organizations participating to the blockchain and willing to run a validator node, seldom exceeds 20-30 units. This further confirms Clique to be the consensus protocol of choice.

4.3 Uses of dApps and kinds of blockchains

The software programs using a blockchain are called "decentralized applications", or “DApps”, or "dApps" are one of the main new trends of software development. A search of scientific and technical documents made with Google Scholar in July 2021 found 36,700 results for "smart contracts" development, a number higher or much higher than the results for microservices development (20,500), global software engineering (7,670), devops development (23,500), and even than IoT "software development" (30,400). dApps include the SCs running on a blockchain, but also the software managing data outside the blockchain and the user interface to interact with it.

Initially, the primary use of SCs was to manage second-level digital currencies, called "tokens", mainly used to finance the Initial Coin Offers - crowdfunding operations gathering cryptocurrencies to finance startups [15]. Besides tokens, dApps are now being used for many applications, in the fields of data notarization, finance and insurance contracts, supply chain management [147] , [148], [149], smart and micro grid management [150], health sector (personal records, pharmaceutical product delivery, clinical trials, etc.) [151], identity management and access control systems [149], decentralized notary [149], gambling, gaming, voting [149], and many others [24] [150].

dApps and SCs can be used for automated enforcement of contractual obligations, without having to trust a central authority, and without space and time constraints.

4.3.1 Kinds of blockchains

The first blockchains were public, that is truly decentralized, censorship resistant, anonymous, and without participation and access limits. However, public blockchains are not without problems and limitations. They are typically unique systems, so they have performance and scalability issues, because the maximum number of transactions per second is low, and the size of the blockchain is ever growing. Moreover, their energy consumption is high, due to the "Proof of Work" consensus still used by the most popular ones, and there is no privacy or control of the information to be written and read.

To address these issues, and still being able to use the technology in real-world applications, permissioned blockchains were introduced. These are closed networks in which previously designated parties interact and participate in data validation and management. For this reason, they are also called "consortium blockchains".

The main feature a blockchain must exhibit is that it is an intrinsically distributed system without central authority. In permissioned blockchains, the number of nodes is much lower than in public ones, but they must still be decentralized across known participants. The reasons for decentralization may be various, for instance, no single organization might be willing to run the system for reasons related to cost, or to legal liability; or the organizations involved might not wish to let just one of them run the system; or having many, independent nodes could be a guarantee of persistence and immutability of the system.

It is also possible that a single organization runs individually a blockchain or DLT, to take advantage of its features for internal applications. Then, however, most of the reasons to use a blockchain do not hold, so I will not consider this case in the rest of this work.

Nodes in a permissioned blockchain can be *validators*, which are able to participate in the consensus mechanism to validate and add new blocks to the blockchain, or *simple nodes* holding a copy of the blockchain.

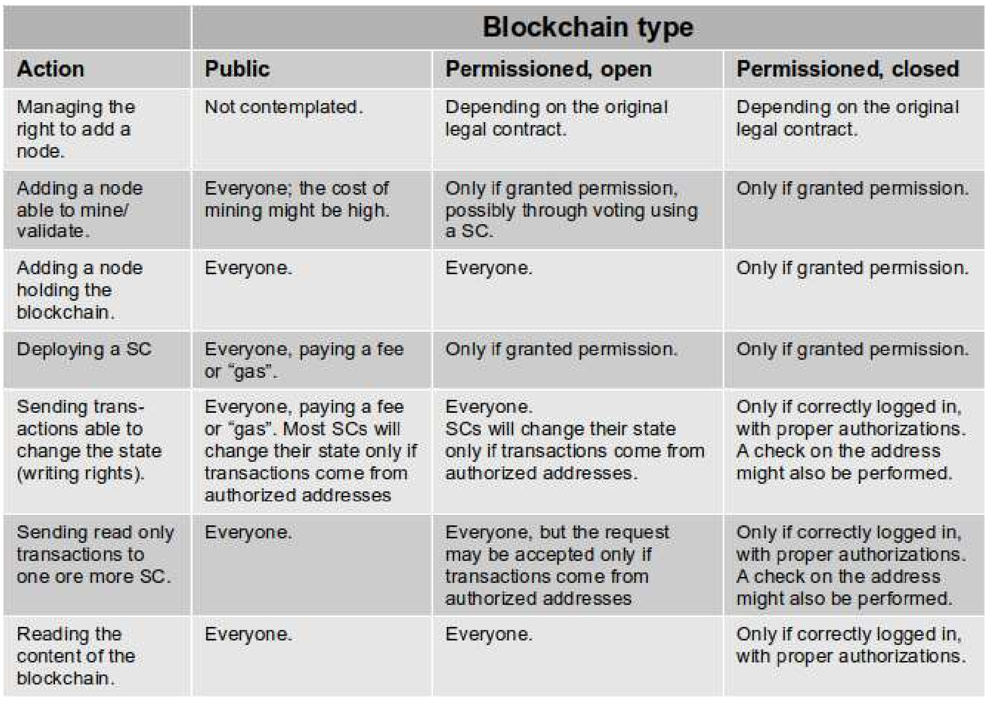
To run a consortium blockchain, there are many software systems available. Most public blockchain software is open source and suitable also to be used to manage a consortium one. A prominent consortium blockchain, aimed mainly to banking applications, is Ripple, developed by a private company and presently run by about 150 invited validators. There are also projects aimed to build consortium blockchain software. The most popular among these is Hyperledger, an open-source collaborative effort hosted by the Linux Foundation, aiming to build cross-industry blockchain technologies.

In general, blockchains can be classified according to how they grant the right to add new validators or simple nodes, and the right to read/write information. Regarding permissioned blockchains, I made a further distinction between systems intended only to be used by specific, authorized partners (*closed* systems), and systems intended to be accessed also by the general public (*open* systems). An example of the former might be a system to perform and clear money transfers among banks, which should be accessed only by the banks, and possibly by a control authority. An example of the latter might be a system to guarantee the provenance and quality of foods, which should be accessible by whoever buys the certified food. Figure 23 shows the proposed classification.

In public blockchains, everyone can add a node, and also be able to validate and add new blocks through mining, Proof of Stake or other consensus algorithms compatible with uncensored participation. The access to the functions of specific SCs, however, might be granted only to authorized addresses, implementing a control at SC level. For instance, you can change the state of a SC holding ERC20 tokens on Ethereum blockchain only if your address already owns some tokens - the change can only consist in the transfer of one's own tokens to another address.

In permissioned blockchains, the right to add a node is managed by the consortium of organizations running the blockchain, according to the original legal contract among them. Both open and closed permissioned blockchains manage validators, and simple nodes, by granting them specific permissions, at Internet connection level. This requires intervention of systems engineers, administrating the network.

In open permissioned blockchains, the addition of a node, and the downloading of the blockchain is granted to everyone. Validator nodes, however, must be approved, either automatically through a poll among existing validators, using a suitable SC, or by the consortium members. Deploying a new SC on the blockchain can be made only by participants with the permission to do so. Everyone can send transactions to the blockchain but, as in public ones, it is task of the SC receiving the transaction to decide whether to accept or not the request.



	Blockchain type		
Action	**Public**	**Permissioned, open**	**Permissioned, closed**
Managing the right to add a node.	Not contemplated.	Depending on the original legal contract.	Depending on the original legal contract.
Adding a node able to mine/ validate.	Everyone; the cost of mining might be high.	Only if granted permission, possibly through voting using a SC.	Only if granted permission.
Adding a node holding the blockchain.	Everyone.	Everyone.	Only if granted permission.
Deploying a SC	Everyone, paying a fee or "gas".	Only if granted permission.	Only if granted permission.
Sending trans-actions able to change the state (writing rights).	Everyone, paying a fee or "gas". Most SCs will change their state only if transactions come from authorized addresses	Everyone. SCs will change their state only if transactions come from authorized addresses.	Only if correctly logged in, with proper authorizations. A check on the address might also be performed.
Sending read only transactions to one ore more SC.	Everyone.	Everyone, but the request may be accepted only if transactions come from authorized addresses	Only if correctly logged in, with proper authorizations. A check on the address might also be performed.
Reading the content of the blockchain.	Everyone.	Everyone.	Only if correctly logged in, with proper authorizations.

Figure 23 Classification of blockchain types, in relation to validation and access.

Closed permissioned blockchains typically grant access permission only to clients which access the system using appropriate credentials. The specific authorizations granted also depend on these credentials, as in classic information systems.

Of course, it is also possible to have a permissioned blockchain granting open access only to specific SCs, but not to the whole blockchain. In this case, the system is classified as closed, but holding services which are in fact open.

4.3.2 An Evaluation Framework

One of the main goals of this study is to facilitate the decision of what specific blockchain architecture to choose, once made the choice to use a dApp for implementing a given application. To this purpose there are already various frameworks, among which the most relevant were reported in Section 4.2. I decided to use the work of Scriber as a starting point, noting however that this framework is focused to evaluate the suitability of blockchain technology for a given application, more than to choose among different blockchain architectures.

I kept 7 features of Scriber, removing "Workflow", "Transactions" and "Inefficiency", which are more focused on the decision whether to use a dApp to implement a specific system, or not. I added three blockchain-specific quality criteria taken from three other evaluation frameworks (described below), which complement Scriber's ones: "Privacy", "Cost" and "Scalability". I also added two more specifications of cost - the cost to add a new node (Deployment cost), and of dApp development (Development cost), both very important in the choice of the architecture and technology to adopt.

The features a dApp system must exhibit according to the chosen evaluation criteria are reported in the first three columns of the table shown in Figure 24. The remaining two columns show an evaluation of generic public and permissioned blockchains, as described in Section 4.3.1.

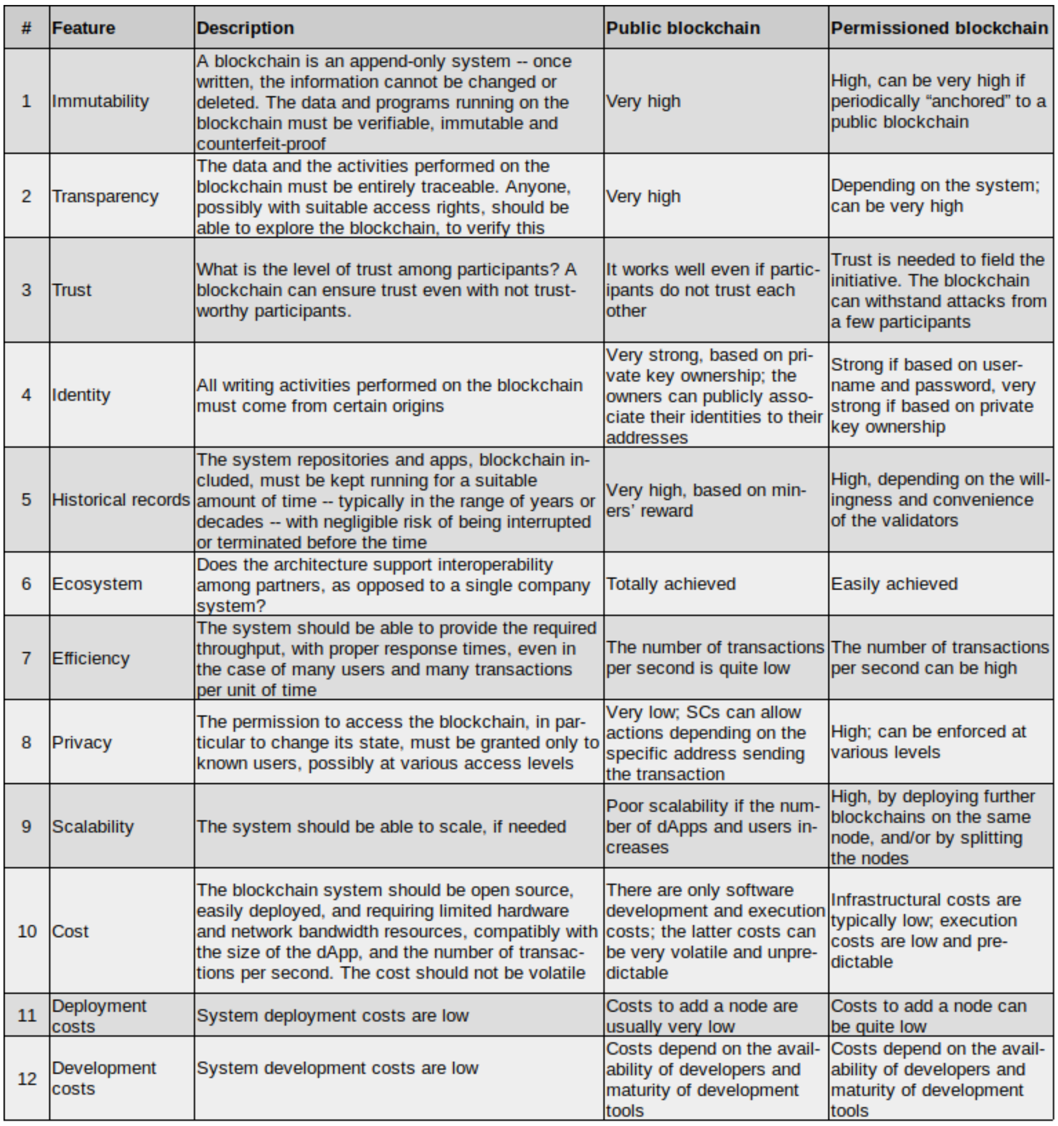
Features 1-6, taken from Scriber, are basically aimed to gain user trust without having to trust all blockchain nodes. Here a node might cease to be trustworthy also because it withdraws from the network, and not necessarily because it tries to attack the system. Features 7-12 are desirable for all software systems, but are especially difficult to obtain in public blockchains.

Developing a dApp system, the first issue to address is whether to use a public or a consortium blockchain. Public blockchains are open to everyone; the most used for implementing dApps is Ethereum, but others are available, such as EOS, Binance Smart Chain, Steem, TRON and many others. As of November 2021, 2886 dApps were running on Ethereum public blockchain, out of a total of 3799 surveyed dApps [9].

Figure 24 shows also a comparison of public and permissioned blockchains with regard to the proposed framework, with qualitative scores. Note that in the table I consider the features of the most used and proved blockchains, such as those cited above. I am aware that there are new projects aiming to overcome the limitations of public blockchains in terms of throughput, cost and scalability. However, these projects are still work in progress. For an industrial applicability perspective, the cited technologies are by far still the best in terms of reliability and easiness to find development resources - both tools and skilled people.

Public blockchains look the most stable, and easiest to start with, but lack performances and scalability. Their cost is not predictable, due to the high volatility in cryptocurrency values and in transaction validation fees. Moreover, they do not support data privacy, and thus can be non-compliant with respect to the strict guidelines of modern privacy laws, such as European GDPR.

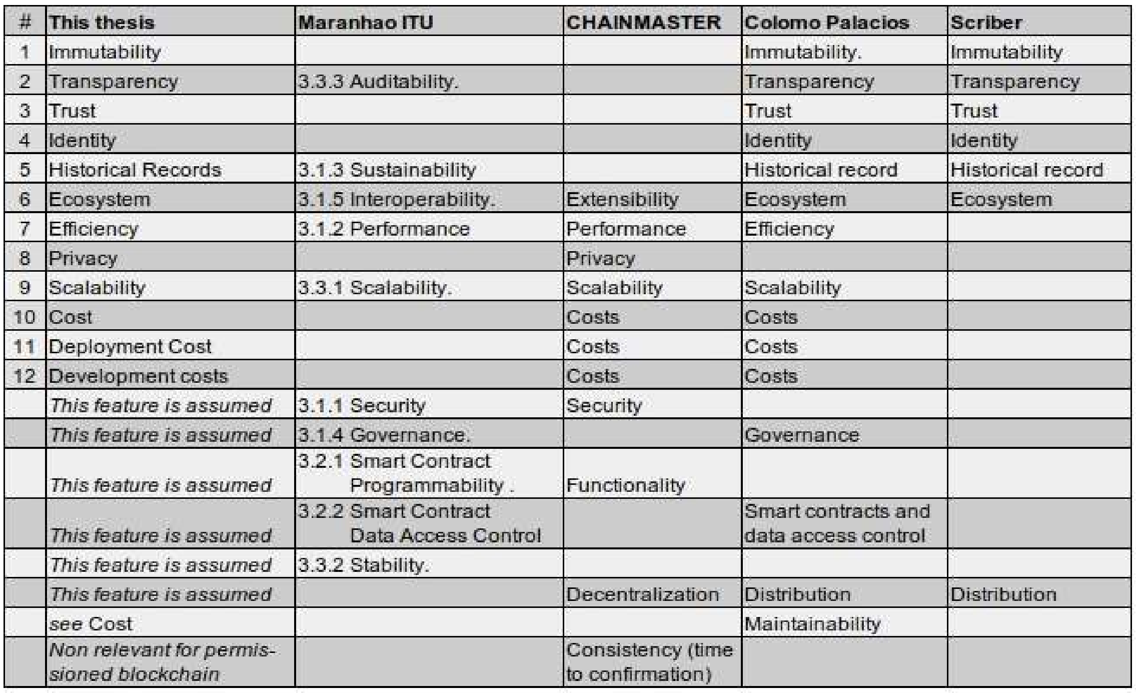
For these reasons, public blockchains are mainly used for applications managing digital money, such as the above cited tokens, and for the notarization of information. In this proposal, I focus on non-monetary, industrial applications, and thus on consortium blockchains. The proposed solution includes also the use of a public blockchain, to make the permissioned one immutable.



#	Feature	Description	Public blockchain	Permissioned blockchain
1	Immutability	A blockchain is an append-only system -- once written, the information cannot be changed or deleted. The data and programs running on the blockchain must be verifiable, immutable and counterfeit-proof	Very high	High, can be very high if periodically “anchored” to a public blockchain
2	Transparency	The data and the activities performed on the blockchain must be entirely traceable. Anyone, possibly with suitable access rights, should be able to explore the blockchain, to verify this	Very high	Depending on the system; can be very high
3	Trust	What is the level of trust among participants? A blockchain can ensure trust even with not trust-worthy participants.	It works well even if partic-ipants do not trust each other	Trust is needed to field the initiative. The blockchain can withstand attacks from a few participants
4	Identity	All writing activities performed on the blockchain must come from certain origins	Very strong, based on pri-vate key ownership; the owners can publicly asso-ciate their identities to their addresses	Strong if based on user-name and password, very strong if based on private key ownership
5	Historical records	The system repositories and apps, blockchain in-cluded, must be kept running for a suitable amount of time -- typically in the range of years or decades -- with negligible risk of being interrupted or terminated before the time	Very high, based on min-ers’ reward	High, depending on the will-ingness and convenience of the validators
6	Ecosystem	Does the architecture support interoperability among partners, as opposed to a single company system?	Totally achieved	Easily achieved
7	Efficiency	The system should be able to provide the required throughput, with proper response times, even in the case of many users and many transactions per unit of time	The number of transactions per second is quite low	The number of transactions per second can be high
8	Privacy	The permission to access the blockchain, in par-ticular to change its state, must be granted only to known users, possibly at various access levels	Very low; SCs can allow actions depending on the specific address sending the transaction	High; can be enforced at various levels
9	Scalability	The system should be able to scale, if needed	Poor scalability if the num-ber of dApps and users in-creases	High, by deploying further blockchains on the same node, and/or by splitting the nodes
10	Cost	The blockchain system should be open source, easily deployed, and requiring limited hardware and network bandwidth resources, compatibly with the size of the dApp, and the number of transac-tions per second. The cost should not be volatile	There are only software development and execution costs; the latter costs can be very volatile and unpre-dictable	Infrastructural costs are typically low; execution costs are low and pre-dictable
11	Deployment costs	System deployment costs are low	Costs to add a node are usually very low	Costs to add a node can be quite low
12	Development costs	System development costs are low	Costs depend on the avail-ability of developers and maturity of development tools	Costs depend on the avail-ability of developers and maturity of development tools

Figure 24 The features needed by a dApp system, and how public and permissioned blockchains support them.

In the table of Figure 25 I show the criteria of this framework against those of four other evaluation frameworks, that is; (i) the ten criteria of Maranao et al. [137], which will be part of ITU forthcoming standard; (ii) those of Chainmaster by Garriga et al [139]; (iii) the assessment factors of Colomo-Palacios et al. [152]; (iV) the characteristics of Scriber [43].



#	This thesis	Maranhao ITU	CHAINMASTER	Colomo Palacios	Scriber
1	Immutability			Immutability.	Immutability
2	Transparency	3.3.3 Auditability.		Transparency	Transparency
3	Trust			Trust	Trust
4	Identity			Identity	Identity
5	Historical Records	3.1.3 Sustainability		Historical record	Historical record
6	Ecosystem	3.1.5 Interoperability.	Extensibility	Ecosystem	Ecosystem
7	Efficiency	3.1.2 Performance	Performance	Efficiency	
8	Privacy		Privacy		
9	Scalability	3.3.1 Scalability.	Scalability	Scalability	
10	Cost		Costs	Costs	
11	Deployment Cost		Costs	Costs	
12	Development costs		Costs	Costs	
	This feature is assumed	3.1.1 Security	Security		
	This feature is assumed	3.1.4 Governance.		Governance	
	This feature is assumed	3.2.1 Smart Contract Programmability .	Functionality		
	This feature is assumed	3.2.2 Smart Contract Data Access Control		Smart contracts and data access control	
	This feature is assumed	3.3.2 Stability.			
	This feature is assumed		Decentralization	Distribution	Distribution
	see Cost			Maintainability	
	Non relevant for permis-sioned blockchain		Consistency (time to confirmation)		

Figure 25 A comparison of the features needed by a dApp system according to different evaluation frameworks.

Note that I considered some criteria - those tagged with "This feature is assumed" - to be fulfilled by default by blockchain technology, or to be not relevant for permissioned blockchains, so they are not considered by this framework. The last criterion shown, "Consistency" of Chainmaster framework, was deemed not relevant because it is defined as the time to confirm that a transaction is securely appended to the blockchain. This criterion is important in public PoW blockchains, but it is not relevant in permissioned blockchains, whose consensus algorithms are not subjected to forks which can cancel valid transactions.

4.4 Choosing the blockchain platform

A dApp is a software system that uses DLT, typically a blockchain, as a central hub to store and exchange information, through SCs. It is composed of SCs running on a blockchain, and of applications able to create and send transactions to them. These applications typically provide a human interaction interface, running on a PC or on a mobile device. Additional information could be stored on one or more servers, and business logic could be executed on these.

My primary goal was to design a suitable blockchain architecture for "industrial" applications, that is information systems whose goal is to manage contractual relationships between industrial customers and suppliers, including supply chain management. Clearly, the first step is to choose the underlying blockchain "engine". I evaluated the public Ethereum blockchain against what I believe are the most mature and used technologies to implement permissioned blockchains. They are: (i) Ethereum using Clique, a Proof-of-Authority (PoA) consensus mechanism discussed in Section 4.2 – let’s call "Ethereum PoA" this platform; (ii) Hyperledger Fabric.

This choice is confirmed by the recent work of Polge et al. [153], who list and compare five major private blockchain frameworks. Besides Ethereum and Hyperledger Fabric, they consider also Quorum, which is a fork of Ethereum; MultiChain, which is a fork of the Bitcoin blockchain, but in its stable version 1.0 does not allow SCs; R3 Corda, which is especially devoted to financial applications. Another platform whose popularity is increasing is Hyperledger Besu, which is compatible with Ethereum. Both Quorum and Besu can easily be used in place of Ethereum PoA, so I just evaluated the latter.

To justify the choice of the blockchain platform, it is possible to use the features of Figure 24 to define a framework to determine the best architecture, with respect to a specific application. Each feature is evaluated using an integer scale from 1 (least suited) to 5 (most suited).

The criteria are subjectively weighted by importance, being the weight values related to the specific system to implement. I chose the weights targeting a system whose goal is to manage contractual relationships between industrial customers and suppliers, including supply chains. Such a system would certify orders, provisions and shipments of raw materials, semi-finished and final products, and their processing steps. The system actors are the various supplier and customer firms, the wholesalers, the certification authorities.

#	Feature	Description	Weight	Ethereum main net	Ethereum PoA	Hyperledger Fabric
1	Immutability	Risk of forgery of data and/or SCs. 1: high risk - 5: low risk	5	5	4	4
2	Transparency	Ease to inspect the blockchain, having suitable access rights	4	5	5	4
3	Trust	The system can increase trust between participants	3	5	4	4
4	Identity	All writing activities must come from certain origins	5	3	4	4
5	Historical records	Risk that the system will not run for a suitable amount of years. 1: high risk - 5: low risk	4	4	3	3
6	Ecosystem	The architecture facilitates integration of various companies	3	4	5	5
7	Efficiency	Ability to provide high throughput and low response time	5	1	5	5
8	Privacy	Ease to manage access permissions to the blockchain	4	3	4	5
9	Scalability	The system should be able to scale, if needed	4	2	4	4
10	Cost	Writing costs are reasonable and stable	5	1	5	5
11	Deployment costs	System deployment costs are low. 1: high cost - 5: low cost	3	5	3	3
12	Development costs	System development costs are low. 1: high cost - 5: low cost	4	4	4	2
	TOTAL SCORE			164	206	198

Figure 26 The features needed by a dApp system, and how public and permissioned blockchain support them.

Figure 26 shows the 12 criteria, the weight given to each of them, and the evaluation scores of the three platforms. The total scores are the weighted sums of all 12 criteria.

To get these scores I interviewed seven blockchain experts, five from academia and two from a private company which produces dApps. The experts agreed on the weights to assign to each feature and voted independently. The median of the seven votes was adopted to rate every criterion for the three platforms, and the total score was computed for comparison.

The rationale behind these weights is the following:

- The most important criteria for a permissioned blockchain were deemed to be:
    - Immutability, because a data structure managed by a limited set of organizations might be the target of a successful attach, or even some of the participants might collude to alter the data.
    - Identity, because being certain about the identity of participants who send transactions is a key requirement in contractual relationships.
    - Efficiency, both in throughput and data storage - an obviously important requirement.
    - Cost, again for obvious reasons; note that various types of cost are taken into account in three criteria, so overall it is the most important requirement.
- The second most important criteria are transparency (this could have been even higher), stability over time, ease of management of access permissions, compliance with privacy law, and scalability. All these criteria are very important when managing contractual obligations, or guarantees of quality and provenance.
- Slightly less important, but still important, are trust, which in a permissioned blockchain is often taken for granted, and ease of integration among parties, which is partially already included in the cost criteria.

Note that there is no criterion whose weight is below 3 - in fact, all the framework criteria are important for judging the suitability of a blockchain architecture for industrial applications.

The scores regarding immutability, transparency and trust of the permissioned blockchains are very close to those of the public one. This is due to the fact that the permissioned blockchains considered are periodically “anchored” to a public blockchain, and that they are provided of an explorer enabling independent browsing of their state. These features will be described in detail in the next Section.

The “winner” is Ethereum PoA, with 206 points, whereas Hyperledger largely prevails over Ethereum main network.

The Ethereum main network was penalized mainly by the unpredictability and amount of its transaction costs, as well as by its low efficiency and scalability, which were quite highly weighted criteria. Note that the new Ethereum 2.0, or Eth2, which was recently released in December 2020 with the shipping of the Beacon Chain, will provide much higher efficiency and scalability [154]. However, the new version is still in its early, experimental phases.

Regarding Hyperledger Fabric, I know that it is one of the most used DLT systems for industrial applications [155] [156]. In my comparison, Fabric got a score almost equal to Ethereum PoA's. It was considered slightly better for Privacy, but slightly less transparent, and with higher development costs, due to the higher complexity of the Hyperledger Fabric platform with respect to Ethereum one, and to the smaller number of skilled developers available.

4.5 The proposed dApp architecture

Once chosen the platform, I derived the overall dApp architecture, starting from general considerations, but also considering the specificities of Ethereum, among which the need to consume "gas" for sending transactions is perhaps the most relevant. The proposed dApp architecture is shown in Figure 27. This is a general-purpose architecture, showing all the possible components. In specific applications, some components might not be needed, and should be removed.

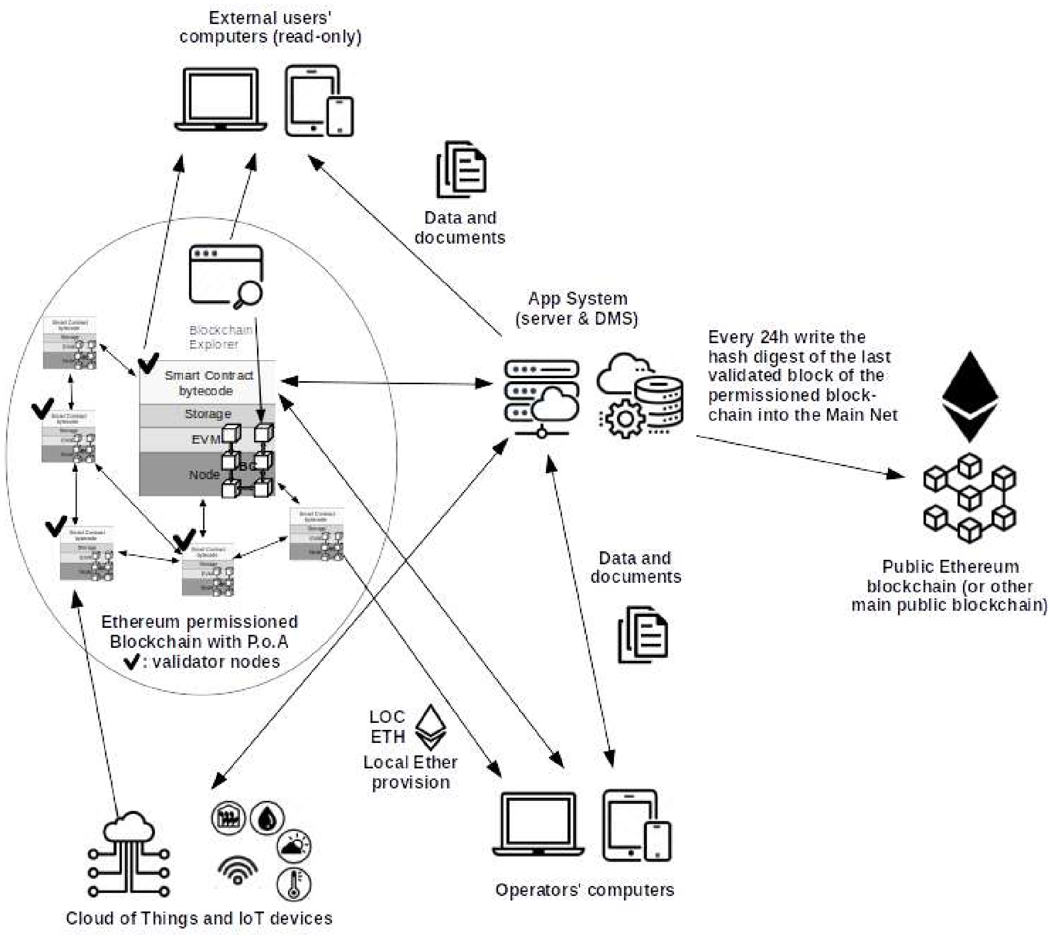

External users' computers (read-only)
Data and documents
App System (server & DMS)
Every 24h write the hash digest of the last validated block of the permissioned block-chain into the Main Net
Blockchain Explorer
Smart Contract bytecode
Storage
EVM
Node
BC
Public Ethereum blockchain (or other main public blockchain)
Data and documents
Ethereum permissioned Blockchain with P.o.A
✓: validator nodes
LOC ETH
Local Ether provision
Operators' computers
Cloud of Things and IoT devices

Figure 27 The proposed architecture of a dApp application.

The proposed architecture has four kinds of actors:

- **Validators** (shown with a bold "V"), the nodes running the system, managed by the key consortium participants. These nodes hold a copy of the blockchain, validate transactions, group them in blocks, and decide to add blocks to the blockchain using Clique consensus mechanism, or a similar one.
- **Participants**, the nodes holding a copy of the blockchain, and able to receive, validate and broadcast transactions, but not to participate in the consensus mechanism. These nodes are managed by organizations which obtained the permission to do so, but are not (yet) full members of the consortium.

- **Operators**, who are enabled to send transactions changing the blockchain state. Operators use terminals and GUI software which are part of the overall system, and belong to organizations participating to the system.
- **External users**, who can access the system nodes in read-only mode, using standard terminals and with software provided by the system.

The validators are run by the organizations of the consortium, which should be independent from each other, to avoid that a single organization might try to falsify the blockchain data, or simply decide to stop supporting the system.

It is important to assess the reasons why validators take the burden of managing the blockchain. The main reasons are either to propose the blockchain as a service to customers for a profit or being involved in the management of the dApp(s), which in turn can provide a benefit to the validators. This benefit might be direct, coming from the sale of products or services, or indirect, think for instance to a public body promoting some service linked to its mission. An assessment of validators before they are added to the system is necessary, and allows to perform risk analysis, estimating the probability validators might turn off their node within one or more years, and thus computing the minimum number of them needed to guarantee the persistence of the dApp. Clearly, in the case some validators leave, this should trigger a search for new validators, to keep the dApp stability.

Regarding external users, there are two possibilities:

- everyone can access the blockchain - in this case, the validators allow public access to the SC interfaces and to the explorer (see later);
- the access is reserved to authorized users - in this case, authentication and access control must be provided by validators, before users can access the dApp.

The components of the architecture are:

- The Ethereum PoA blockchain, shown as a network of validators and regular nodes on the left.
- An external system, called App System, holding the data and applications not residing in the blockchain; it is shown in the center.
- The terminals of operators and external users (top and bottom of the figure), running dApp software providing the user interface, and able to manage the private keys of operators.
- A system to perform identity management and access control, integrated in the App System, and possibly also using a SC.
- A link to a public blockchain (in this case Ethereum), to periodically write the hash digest of the last block locally mined.
- An Explorer running on one or more nodes holding a copy of the blockchain, to browse the actual state of the blockchain without the mediation of the user interface.

- Links to IoT devices, which send data to the blockchain, or receive commands from the system.

In the following I will describe in greater detail these components.

4.5.1 Ethereum PoA blockchain

Ethereum Foundation, and Ethereum implementation under Hyperledger project (called Besu), offer Clique as one of the preferred consensus protocols. Clique is a form of Proof of Authority (PoA), a customized form of Proof of Stake where the identity and reputation of the validator performs the role of stake instead of stake with some monetary value. As already stated in Section 4.2, Clique is one of the best and most popular consensus algorithms for permissioned blockchain, so it was chosen for this architecture.

In Clique, each validator is not allowed to validate two consecutive blocks, in order to preserve equilibrium among validators, and to minimize damage if a validator become malicious and validates a wrong block - in which case it is quickly spotted by other validators, ousted from the validators' set using a vote, and the wrong block is eliminated from the valid blockchain through a fork. Also regular nodes may be present, holding a copy of the blockchain.

The validators also create Ethers expendable in the consortium blockchain, which we call “Local Ethers” (LOCETH). LOCETHs do not have monetary value, and cannot be exchanged against other currencies. However, they must be used to send transactions able to change the status of the system, as further explained in Subsections 4.5.3 and 4.5.4.

All nodes hold the blockchain enabling software, which includes the Ethereum Virtual Machine, running SCs. The SC bytecode, endowed with its permanent data (Storage), is stored in the blockchain, and is loaded into the node memory for its execution. All the nodes execute every SC, and execution results must be the same for all nodes, hence the impossibility for SCs to access the external world. They can access only their data and other SCs stored in the blockchain, which are the same in all nodes.

4.5.2 App System

Another key dApp component is a software system running on mobile devices and/or on servers, possibly on the Cloud, which we call "App System", following the nomenclature of ABCDE method presented in Part II. It holds the information which cannot stay in the blockchain - because it is too large, or for privacy reasons. The App System exchanges information with users and external systems and devices, and performs business computations. Of course, it is also able to send transactions to the blockchain, having a direct connection with a node, and being the owner of an address and of the corresponding private key.

If the dApp must hold large amounts of information, such as documents and images, these documents are stored off-chain on one or more Document Management Systems (DMS), by the App System. The hash digest of the document and a link to retrieve can be stored in the blockchain, guaranteeing the date of the document, and its integrity. This approach is also called “Off-Chain Data Storage” pattern [50] [24]. In the case of sensitive data stored off-chain, the App

System also takes care of managing access rights to them, providing the information only to qualified users.

Saving data in this way is compatible with privacy regulations, because no actual data is stored in a transparent medium such as the blockchain. Moreover, huge amounts of data can be managed, stored and certified, despite the relatively limited room available in blockchains, most of which were never intended to substitute a DMS or a database. In fact, storing large amounts of information on a permissioned blockchain based on Ethereum is not viable for the following reasons: (i) big data means big transactions to write them into the blockchain, which in turn means overladen communications and less performance; (ii) the computation needed to assemble and communicate the block with these transactions again means reduced performance of the system; (iii) the size of the blockchain, which is an append-only repository, would quickly become huge, and impair efficient data retrieval.

To conclude this section, I stress that the App System is not necessarily a single, centralized system, nor does it have to manage a single, centralized database or DMS. The App System is a service that, if needed, can run on several physical or cloud servers. The operators who need to store a document can directly specify the URL of the DMS where to store it, and there can be many of them. For instance, each organization storing data might manage its own DMS, including granting access permission to it. What is important is that the data can be accessed by whoever is entitled to access it, and that the access permission is given by the owner of data, possibly also through the blockchain itself. Also, the database holding the system data might be a decentralized one, like IPFS. An example of medical records management using IPFS is reported in [157].

4.5.3 Terminals and apps

This component includes the applications, running on PCs and/or mobile terminals, which enable the interaction with human users. For external users, it can be a simple app, able to connect to a blockchain node or to an authentication server, and to show to the user the requested information, gathered from the blockchain and/or from the App Server. In Ethereum all users can send “view” queries, which return information from the SCs, without changing the blockchain, and which cost no gas.

For operators, the app includes a wallet, that is software able to generate and store the private key associated to the operator's blockchain address, to create transactions, sign them with the private key and send them to a node. The operator's private key is unblocked by a password, and possibly by the very ownership of a mobile phone. In this way, the identity of the sender of the transaction is guaranteed, in a way compatible with European Union eIDAS Regulation [158].

Since operators send write transactions to the blockchain, their wallet also holds LOCETHs, which are used to pay the needed gas, thus acting like a true cryptocurrency wallet. In this way, writing can be controlled by the validators, providing LOCETHs only to approved organizations, and in the proper amount. These organizations will in turn send the LOCETHs to their operators' wallets, to enable writing to the blockchain.

If an organization opts out of the system, they will have to return the residual LOCETHs, and will not be provided of more, thus effectively stopping their use of the system.

The operator's app will also facilitate the data input and control operations the operator is in charge of. Depending on the specific applications, the app is able to exchange data with the blockchain (by sending transactions), and/or with the App System.

4.5.4 Identity management and access control

In the previous section, I stressed how the apps running on mobile or PC terminals can work as a wallet, guaranteeing the association between the address and the ownership of the corresponding private key.

When operators register to the system, they generate the address, and a register managed by the App System associates the address with their identity (name, SSN, and other data). This association can be made public - for instance to identify an authorized auditor, or the organization which the operator is the legal representative of - or not.

Additionally, the system must register the access permissions of the user. This can be done in a traditional way, using access control lists or role-based access control managed by a server (which is part of the App System), or through a dedicated SC able to associate the users' addresses to their permissions.

A further access control, as cited before, can be granted by endowing operators with LOCETH, the gas enabling the sending of transactions. This functionality of the system works in the following way:

1. LOCETHs generated by validators are sent to a system wallet.
2. From this wallet, LOCETHs are sent to the wallets of the organizations which need to use the system, in proper amounts. In this way it is possible to control system usage, and to bill for it.
3. The organizations send the LOCETHs to the wallets of their operators, thus enabling them to send transactions. The amounts depend on the actual number and complexity of transactions to be sent.
4. When the LOCETH level in a given operator's wallet falls below a given threshold, a request to top up the wallet is sent to the wallet of their organization.
5. Organizations can receive LOCETHs by the system wallet upon request, or according to an agreed schedule.

4.5.5 Explorer and anchoring on a public blockchain

In the architecture presented, one or more blockchain nodes provide an Explorer, which is a software that allows its users to access the blockchain transactions, and to inspect the source code of SCs. For Ethereum, there are various open-source Explorers available for this task. Among them, I may quote BlockScout, Expedition Block Explorer, and Alethio.

In this way, the transparency of the consortium blockchain equals that of a public one, because all transactions and accounts, including those of SCs, can be independently inspected. The use of the Explorer can be granted to everyone, or only to registered users with the proper credentials, depending on the specific system.

If the Explorer guarantees a transparency similar to that of public blockchains, anchoring to a public blockchain guarantees a similar level of immutability. The idea is that, from time to time, the hash digest of the last block validated in the permissioned blockchain is written into a public blockchain. This idea has already been applied, especially in the field of distributed data storage solutions [159]. The time interval might be 12 or 24 hours, or even less.

The public blockchain used to anchor the permissioned one can be Ethereum, but also Bitcoin or others, provided they are consolidated enough and stable. The cost of each registration, at the current fee rate, are of the order of a few USD or less, so it should not be an issue for an industrial initiative. The transaction towards the public blockchain is sent by an address managed by the App System, which is published. Clearly these registrations need to manage a "true" cryptocurrency wallet, which is not related in any way to the wallets holding LOCETHs.

In this way, everyone can access the last registration on the public blockchain, using a public Explorer on it, which can access all the transactions sent by a given address. Then, it is possible to verify that the registered hash digest is equal to that of a block validated in the permissioned blockchain, at a date and time immediately prior to the public registration. The hash digest of local blocks can be browsed using the local Explorer.

The combination of the immutability of the public blockchain, and of the transparency of both public and permissioned blockchains, made possible by the respective Explorers, make the latter as immutable and transparent as the former.

4.5.6 IoT devices

The Internet of Things (IoT) is the extension of the Internet to connected physical objects that can be monitored, controlled, or interacted with, to enable ubiquitous industrial services. Examples of IoT industrial use are freight transportation, automatically registering temperatures, position, arrival times, and status of shipping containers and trucks as they move; tracking components in aircraft, automotive, or other industries, which is critical for both safety and regulatory compliance; supply chain and Digital Product Passport digitalization and control; logging of operational maintenance data, and many others.

The interaction between blockchain and IoT has been proposed since the introduction of SCs, for two main reasons. The first is because the blockchain can provide IoT devices the security and the ability to be tamper-proof. The second is the fact that a blockchain is distributed, and an IoT device can connect to any of its nodes, avoiding the bottleneck of a single access point.

An IoT sensor can be provided with an address, a private key, and a connection to the blockchain, and thus be able to send its data through a transaction, which guarantees timestamp and immutability of the registration. To this purpose, many initiatives aim to develop and field blockchains specifically suited to IoT management, such as IOTA and IoTex.

Things, however, are not so simple, because the number of IoT devices can be huge, and the rate of transactions coming from each of them can be high, stressing both the throughput and the size of the blockchain. To solve this issue, sets of IoT devices are connected to some flexible and robust cloud computing environments, able to process and manage IoT services. This solution is called "Cloud of Things" (CoT), and its integration with the blockchain (BCoT) is the subject of a large amount of research, aptly reviewed and summarized by Nguyen et al. [160].

In Figure 27 I show both single IoT devices directly connected to the blockchain, and a set of them connected to a CoT, a service running on the Cloud, gathering the IoT data, and registering them to the blockchain - thus becoming a BCoT. The IoT data are typically not entirely registered on the blockchain, but only a digest of them is written. If needed, the raw data can be stored in the Cloud, or in a server of the App System, which is drawn as connected with a line to the CoT.

4.6 Conclusions and future work

The key reason to use a blockchain is trust. If a system can be developed and deployed by an organization, and its users trust this organization, there is no reason to use a blockchain.

In the case that it is not possible to trust a single organization managing the system, which should be open to all participants - some of whom might try to attack or exploit the system - a public blockchain is the choice. In managing digital currencies and tokens, public blockchains like Bitcoin, Ethereum and many others proved to be very effective and reliable.

If the system to develop deals with contractual relationships between participants, does not directly manage digital currencies, and there is no single operator which has everybody's trust, a permissioned blockchain is the typical choice. There are many possible platforms and architectures to develop such a system, so I proposed an evaluation framework to ease the choice, which extends and blends the criteria of existing frameworks.

I also specified in detail an architecture for industrial dApp systems, which again clarifies, extends and merges existing ideas and patterns in a comprehensive approach. It is based on Ethereum software, using a Proof of Authority consensus mechanism, which is fast and energy-saving. The described architecture guarantees the same level of trust and transparency of the public blockchain it is anchored to, allowing much better performances and scalability, at low, predictable cost.

At the same time, it encompasses compliance to privacy regulations, preserving the same level of privacy granted by a private blockchain, and enables the consortium to set different access permissions for different users. The control of writing rights is also performed by means of the local Ethers produced by validator nodes, embracing the advantages of a public blockchain and those of a private one.

I used Ethereum PoA as a reference blockchain, but in principle it could be substituted by any blockchain provided with the possibility to install an explorer. The proposed architecture is already used in some industrial projects, among which I may quote Etherna, a BaaS (Blockchain as a Service) product, which allows and encourages customers to setup and run their own nodes [161]. Depending on the participants' commitment, these nodes can even be validators.

Presently, we are working with a set of Sardinian institutions and firms to start a consortium blockchain, with the aim to certify the provenance and quality of local products.

We are also working on extending the architecture, enabling multiple blockchains to communicate and exchange transactions. This functionality is obtained by defining "Edge nodes", which are validators in a blockchain, but also have the credentials and gas to send transactions to other blockchains, through the Edge nodes of the latter. This allows the approach to scale virtually without limits, adding new nodes and new blockchains.

The target application is the Digital Product Passport (DPP), which is part of the Europe Union Circular Economy Action Plan [162]. A DPP is a combination of: (1) a unique product identifier; (2) data collected by different value chain actors related to this unique identifier; and (3) a physical link (tagging) between the product and the data. Note that a final industrial product will be often an assembly of complex parts, each in turn having its DPP.

An architecture like the one proposed here is very suited to DPP management, with multiple instances aimed to manage the supply chains and to certify the quality of the various sub-products, and of the final product. The final blockchain system would be devoted to track and certify also the useful life of a product, including maintenance and repairs, and also the operations on its parts after its disposal, tracking reuse, recycling and final disposal. Each dApp instance tracking one or more parts would manage their unique identifiers, and the various dApps should be able to easily exchange data, thus providing a complete DPP of the product and its parts, in a tamper-proof and transparent way.

5 Automatic generation of Ethereum-based Smart Contracts for Agri-Food Traceability System

There is a growing demand for transparency across the agri-food supply chain from customers and governments. The adoption of blockchain technology to enable secure traceability for the agri-food supply chain management, provide information such as the provenance of a food product, and prevent food fraud, is rapidly emerging, due to the inherent trust and inalterability provided by this technology. However, developing correct smart contracts for these use cases is still more of a challenge than it is for those executed in other fields.

Numerous agri-food supply chain management systems based on blockchain technology and smart contracts have been proposed, all however ad-hoc for a specific product or production process and difficult to generalize.

In this chapter, I propose a novel approach for easily customizing and composing general Ethereum-based smart contracts designed for the agri-food industrial domain, to be able to reuse the code and modules and automate the process to shorten the time of development, keeping it secure and trusted.

Starting from the definition of the real production process, I aim to automatically generate both the smart contracts to manage the system and the user interfaces to interact with them, thus producing a working system in a semi-automated way. In addition, I describe a case study on honey production to show how the approach works. Future work will first extend the scope of the approach to other supply chains, moreover, while the current platform used is Ethereum, in the future the approach will be easily extended to other blockchain platforms.

5.1 Introduction

A decentralized application (DApp, Dapp, dapp, or more frequently dApp) is a computer application that runs on a distributed peer-to-peer (P2P) system, that is on a network of nodes, with no node acting as supervisor. A dApp is stored and executed on a blockchain, in order to be decentralized, transparent, deterministic, and redundant. It is developed by writing smart contracts (SCs), which are small script programs running on every node of the blockchain and may have a user interface (UI) that allows users and devices to interact with SCs. SCs are immutable - no one can tamper with the code of the contract - and distributed, because of their storage inside the blockchain.

In the last decade, there has been a huge development in the field of blockchain technology applied to various economic sectors, due to the advantages that the implementation of such a system could provide.

Many companies and startups are already adopting, and working on blockchain technology, trying exploit the many advantages it promises, so we are experiencing a strong growth of ideas and applications.

Several research papers, like [163] [164] [165] [166] [167], just to cite a few, have shown that the use of the blockchain can advantageously help to achieve traceability, by storing data which are non-forgeable, and with certain date. Consequently, companies are trying to adopt this technology in various sectors by harnessing in particular its ability to get transparency in scenarios where numerous untrusted actors get involved.

For these reasons, the blockchain is gaining increasing popularity as a technology to enable traceability in a certified and immutable way in the agri-food sector, helping to avoid fraud and counterfeiting by creating an auditable record of the journey from the farm to the fork behind all physical components of the food products. Today more than ever, customers are demanding transparency, especially with food. People want to feel secure and to know how a product was farmed or manufactured, and which ingredients are involved in its production. In Europe, food legislation is particularly strict and the implementation of traceability systems are mandatory, but they are unable to fully guarantee consumers against fraud. For this reason, innovative methods for traceability systems based on product identification are needed.

This has also given rise to a growing demand for blockchain developers, and today SC development experience is one of the most sought-after skills by large companies. According to [28], developers need to have a clear understanding of the capabilities and limitations of blockchain technology, and to acquire the skills needed to implement the technology by knowing how the new architecture would affect the application trustfulness.

Some companies have launched pilot or proof of concept projects to implement blockchain technology in a wide range of sectors, but at present many limitations still have to be considered and addressed. Most of the published works concerning the application of blockchain technologies, for example into the supply chain management, reported no detailed information about the technical implementation, and there are still few practical uses of blockchain technology.

Research on blockchain in many different application areas is going through an exploratory phase, and supply chain management is one of the main areas of interest to be studied in terms of the benefits that would be obtained on the traceability system, such as facilitating food safety and fraud prevention.

Clearly, there is a need for structured methods to facilitate and make more efficient the development of applications based on blockchain technology for the management of the traceability of the agri-food supply chain.

In previous research works [168] [169], my research group explored how the Internet of Things can be combined with blockchain technologies to address potential issues in the agri-food industry, and we implemented a model proposal by combining internet of things (IoT) technology - in particular radio frequency identification (RFID) sensors and near field communication (NFC) tags - with blockchain and interplanetary file system (IPFS) technology, to guarantee transparent and auditable traceability of the goods from farm to fork, providing data that demonstrate the quality of all intermediate products. I believe that, also in the light of our experience, researchers and developers could benefit from a general-purpose approach for agri-food supply-chain management.

The goal of this work is to facilitate and make more efficient the development of blockchain applications for the agri-food supply chain management, by using configurable blocks to be assembled together, so as not to start from scratch every time.

Starting from the definition of the real production process, I aim to automatically generate both the smart contracts to manage the system, and the user interfaces to interact with them, thus producing a working system in a semi-automated way.

To summarize, I propose a novel approach for customizing and composing general Ethereum-based smart contracts (SCs) designed for the agri-food industrial domain in a simple way, to be able to reuse the code and modules and automate the process to shorten the time of development, keeping its secure and trusted. As far as I know, this is the first attempt to develop a semi-automatic configurable system that supports the entire class of supply chains for the agri-food industrial domain.

Though the approach is targeted to the agri-food domain, it can be easily extended to many other kinds of supply chains, where a product, a service or a shipment is delivered by assembling and working on parts, and/or passes through different kinds of transformations and state changes.

The main contributions of this work are:

i. The study and development of a general model of food production, specifically targeted to field traceability systems using blockchain technology;

ii. The development of a set of modules, both general SCs and UI applications, able to be easily configured to generate a system for tracking real agri-food supply chains;

iii. The development of a structured way, starting from the definition of the food production process through pre-defined tables, to configure these modules and to easily generate the final system also by developers with only limited knowledge of blockchain technology;

iv. The development of a case study (honey production) to show how the approach works.

The remainder of this chapter is organized as follows. Section 5.2 deals with the background and related work. In section 5.3, I describe the proposed methodology by identifying entities and events of food production. In section 5.4 I present the design of the general software modules supporting these entities and events. Section 5.5 presents the case study of a traceability system for the honey production, explaining how this approach works. This goal has been evaluated through a case study in which the approach is demonstrated. Finally, section 5.6 discusses and draws conclusions.

5.2 Background and Related Work

In this section, I first introduce an overview of blockchain technology and SCs, then I present the state-of-the-art of modeling techniques for developing blockchain-based systems. Finally, I recall the main studies that highlight the benefits of the use of blockchain systems in the agri-food supply chain domain.

5.2.1 Modeling Proposals for Smart Contracts Development: Related Research

In the last years, smart contracts have increasingly gained ground in ICT applications, but smart contract development still remains a challenging task to many developers. This is largely due to its special design challenges, and to the differences between SC development and traditional software development.

In [170], the authors found four issues that might face developers when writing smart contracts:

1. the difficulty of writing correct contracts;
2. the inability to modify or terminate contracts;
3. the lack of support to identify under-optimized contracts, and finally;
4. the complexity of SC programming languages.

Subsequently, [171] conducted an empirical study to explore the potential challenges faced by developers during SC development, with a focus on Ethereum blockchain. The survey results revealed several major challenges. In particular, existing tools for SC development are still very basic. Programming SCs is different than programming in standard programming languages, due to the fact that the blockchain and the code residing there cannot be changed after it has been deployed.

Currently, the are different blockchain platforms (e.g., Bitcoin, Corda, Ethereum, Hyperledger Fabric, Tendermint, etc.) but not all of them support SCs. Moreover, each platform offers distinctive features, and there is no standard way to write SCs. For instance, Corda is an open-source permissioned blockchain platform - meaning a blockchain managed by a consortium of organizations, which run the validator nodes and perform the consensus mechanism - that allows transacting directly with SCs, and it is explicitly designed to account for a highly regulated environment, e.g. the financial service industry.

I focus on the difficulty of writing correct SCs, that are contracts functioning as intended by their developers. If a SC does not execute as intended, some or the whole currency managed by it would disappear, or other unintended effects might be triggered by an attacker.

To tackle this issue, [170] identified three solutions: i) to semi-automate the creation of SCs; ii) to provide developers with guidelines; iii) adoption of formal verification techniques. In my opinion, a solution to ease the process of writing SCs is to semi-automate the creation of smart contracts.

In [172] a general proposal is presented for extending existing software modeling notations to include specific blockchain concepts or integrations. According to these authors, modeling is an important part of designing software and in their preliminary work they start the discussion on specialized modeling notations for dApps. The authors show three complementary modeling approaches based on well-known software engineering models: entity relationship model (ERM), unified modeling language (UML), and business process model and notation (BPMN).

Then they apply them to an example of blockchain-oriented software (BOS) that implements part of the business logic in the blockchain by using SCs.

In the literature, there are a few proposals for standardization of software engineering of blockchain technologies with reference also to automatic generation of SCs from formal models.

According to [28], Blockchain technology and SC development lack clarity in their implementation. They propose a method based on Model Driven Architecture, which could be used for describing blockchain-based systems in a more general language to determine whether it is possible to model blockchain structure and SC logic, and which business logic should be conveyed in SCs, and which should stay off-chain.

In [173] a model-driven engineering (MDE) tool called Lorikeet for the implementation of business processes on blockchain to manage assets was presented. Model-driven engineering is a software engineering methodology that automatically creates software system code from formal models, and helps developers to manage software complexity by only focusing on building high-level models. Lorikeet can automatically create well-tested SC code from specifications that are encoded in the business process data schema.

In a subsequent work, in [174] the authors presented a model-driven blockchain application development approach for business processes and asset management. They provide templates for the customizing data schemata for both fungible and non-fungible assets registries. Moreover, they propose SC generation methods to automatically transform models into SC programming language code, namely into Solidity. The generated SCs consist of SCs for business process execution, and SCs managing standard ERC-20/ERC-721 compliant tokens. The proposed approach is implemented using their smart contract generation tool: Lorikeet.

In [175] the authors proposed a modeling approach that supports the semi-automated translation of human-readable contract representations in terms of ADICO statements - different components that include attributes, deontic, aim, conditions, Or else (consequences associated with non-conformance) - to enable the codification of laws into verifiable and enforceable computational structures in the public blockchain.

In [176] the authors proposed B-MERODE, to fulfill the need to develop new methods for the analysis and engineering of Business Processes (BPs) supported by a blockchain. This is a novel approach to generate SCs supporting cross-organizational collaborations, and relying on model-driven engineering and artifact-centric business processes. Finally, they demonstrated its feasibility by modeling the case of a rice supply chain through B-MERODE.

In [177] the authors introduced a framework for designing smart contracts in terms of finite state machines. They provide a tool with a graphical editor for defining the contract specifications as automata, and for translating them into SC code.

5.2.2 Blockchain Technology in Agri-Food Supply Chain

Blockchain, and more generally the distributed ledger technology (DLT), is a promising technology that is tamper-proof and decentralized. Self-executing and self-verifying SCs can manage transactions between mutually untrusted parties. In this context, scholarly literature on the adoption of blockchain technology in specific traceability systems for the agri-food supply chain is beginning to emerge. In particular, since 2018, a lot of research efforts have been made on the use of blockchain technology for traceability systems [163].

In [178] the authors discussed the potential of distributed systems to transform the agri-food industry. In [179] the authors performed a literature review of relevant papers about the adoption of blockchain technology for generic supply chain management, covering the literature until 2018, and found out that most of the papers were focused on the use of blockchain for traceability, but only one out of 40 papers dealt with the agricultural field. Another relevant literature review about the use of IoT technology in agriculture, which is a prerequisite for automated blockchain registrations, was performed by [180].

Other research papers, namely [165] [181] [147] [182] [183] [184], just to cite a few, presented traceability systems that use blockchain technology and SCs.

According to these researches, this technology guarantees:

- Data integrity and provenance of documents and records on the blockchain;
- Immutability and transparency of data recorded on the blockchain, resulting in traceability of agri-food products from root to retail;
- Compliance for the quantities of the products involved (grapes, wine, bottles), based on the annual production of the land and the yield in the various stages of processing. This is achieved with the system of tokens, which are associated with the products and cannot be altered, as they are managed on a blockchain;
- Ability to retrace the entire supply chain, simply by accessing the blockchain, and public servers storing relevant documents, starting from the QR code shown on the final product.

In [185] claim that by 2023 the global blockchain supply chain market will grow to $ 3,314.6 million, with an increase in annual growth rate of 87% .

Various papers presented real blockchain solutions for supply chain management, which proved to be successful. Among others, AgriDigital [186] an Australian system for managing grain supply chain released in 2017, counts at the end of 2020, more than 7000 users and a transaction value of $ 3,793 million. In [187], the authors developed AgriBlockIoT, a fully decentralized, blockchain-based traceability solution for agri-food supply chain management, able to seamlessly integrate IoT devices, using and comparing both Ethereum and Hyperledger Sawtooth blockchains. In [165], the authors studied and developed an agri-food supply chain traceability system for China based on RFID (radio frequency identification) and blockchain technology, to guarantee food safety. In [73], the authors proposed a generic agri-food supply chain traceability system based on blockchain technology implementing the “farm-to-fork” (F2F) model currently used in the European Union, which can integrate current traceability rules and processes, using Hyperledger Sawtooth, and implemented following an agile approach.

In [188], the authors proposed a product traceability system based on blockchain technology, in which all product registration and transfer histories are perpetually recorded by using SCs. An event response mechanism was designed to verify the identities of both parties of the transaction, and guarantee the validity of the transaction.

In [189], the authors proposed a monitoring framework that combines SCs and evaluation models for the automatic evaluation of the quality of fruit juice samples. SCs are executed to record production data on a blockchain, and can decide whether the production process is working correctly, or should be terminated for non-compliance. The feasibility of the system has been evaluated by implementing a prototype version of the quality monitoring system for flat peach juice production based on the Ethereum platform and executed in the Remix IDE.

Many studies in the literature focused on highlighting the benefits and value derived by blockchain implementation in the agri-food supply chain domain. In particular, the authors in [184] conducted an analysis to model a traceability system based on blockchain technology in the agriculture supply chain. They identified thirteen enablers that encourage blockchain adoption in the agriculture supply chain, such as anonymity and privacy, immutability, SCs, secured and shared database, traceability, transparency, and others. Then they established hierarchical levels and relationships between the involved actors in the supply chain through interpretive structural modeling (ISM), and decision-making trial and evaluation laboratory (DEMATEL) methodologies. The enablers were identified from existing literature and validated by experts from the field of agri-based supply chains and technology. Moreover, the authors conducted an interesting literature review revealing that BC technology offers various benefits leading to improvements in the sustainability of agricultural supply chains.

Recently, also in [190] the authors demonstrated in their work how the applicability of blockchain and SCs in the field of agriculture can ensure traceability of agricultural products. In their research, they described in detail two SCs, and showed a gas cost analysis of the operations. Specifically, a SC called Storage Contract is used in the pre-harvesting period for monitoring the storage condition connected with the system, the second SC is called Distribution Contract and is used in the post-harvesting period. Finally, they analyzed the model in terms of advantages and disadvantages.

In a similar work, the authors proposed in [191] a framework for providing complete transparency and unforgeable product information in the oil supply chain. They tried to conceptualize the process for end-to-end product tracking. They identified the role and function of every actor and described the structure of two SCs in terms of attributes, events, modifiers, and functions. The first SC, named CheckProgress, aims to monitor the product's information and keep track of it. The second SC, the OilDistribution contract, checks the authenticity of the actors.

In all these research papers, no approach was formalized for the development of SCs. Domain concepts are entirely delegated to the individual work of software developers, potentially leading to pitfalls well known in the field of software engineering, such as poor maintainability and low levels of reuse.

To the best of my knowledge, and by comparing this work to others which deal with the use of dApps to certify the origin of food and prevent food fraud, I assert that this is the first work that proposes and formalizes a general approach to develop dApps to track agri-food supply chains, useful for most kinds of food production.

5.3 Methodology and Problem Representation

According to [192], to present this approach I first describe the adopted methodology that acts as a guideline to address the problem. The relatively young field of SC development is made problematic by their current lack of formalization. So, this work is focused on defining and using a template easy to modify and customize, able to automatically generate SC code. I start by describing in detail which are the configurable SC building blocks, which represent the key entities and concepts of agri-food supply chain, and how to represent them using SCs in the blockchain. Note that, starting from this description of the system, it is possible to automatically generate also the User Interface able to interact with the SCs and other components of the system.

The entire approach was carried on taking advantage of ABCDE (Agile blockchain dapp engineering) method to design and implement dApps, presented in 2.4. The approach aims to:

- Document in a transparent and immutable way all relevant events relevant to production;
- Allow authorities, laboratories and certified experts to asseverate the production, giving proof of their identity and their certifications;
- Integrate manual registrations and automatic registrations made by Internet of Things (IoT) devices, which are increasingly widespread;
- Keep track of the quantities produced, so that these cannot be increased by introducing products of non-certified origin;
- Give evidence of all stages of production to the authorities responsible for verifying the specifications;
- Allow retailers and end consumers to learn about the history of the products purchased, from the field to the purchased product, using an app.

5.3.1 The Problem Domain in Agri-food Supply-Chain

Before building a software system, software engineers need to capture the knowledge of the problem domain, that is all information that defines the problem the software system aims to solve. The first step is to get a deep and consistent understanding of the area under analysis.

Most agri-food supply chain systems get their primary inputs from one or more primary sources (soil, herd, beehives, lake, sea, and so on.), then a primitive resource is produced (harvest, milk, raw honey, fishes, etc.), this product is transformed, possibly several times, until the final product is packed and delivered to customers. Events relevant to the process can occur in each of these phases. Most of the processes in the agri-food industry manage "batches", where a batch means a specific quantity of product that is intended to have uniform character and quality.

In general, the agri-food products are identified through a batch management system, both in the case in which the product is marketed without undergoing significant transformations, and when it is processed to obtain output products that are significantly different from input ones. The reliability of a production batch can be guaranteed through an efficient and transparent system of product and process traceability.

Certifying the origin, ingredients, and processing methods to guarantee high quality standards of an agri-food product is a problem too complex to be tackled top-down, therefore I propose a bottom-up approach to correctly identify incoming, processing, and outgoing goods.

By using analysis techniques and object-oriented design, I performed the analysis of the agri-food industrial domain from the software engineering viewpoint, to find common objects, and objects linked to specific processes. In particular, this phase aims to:

- Investigate and define the roles of major actors involved in the system;
- Determine the entities that emerge and recur in this type of system;
- Decide how to be able to represent a general-purpose system;
- Determine the relationships between these entities and the events of interest for the traceability system (events that need to be made permanent for the traceability).

5.3.1.1 Identifying the Actors

The first step in use case analysis is to identify the major actors. The agri-food supply chain domain is primarily characterized by autonomous and independent actors that in recent years are more and more interacting with globally interconnected systems. I consider the agri-food supply chain as a sequence of processes from production to the final product, that involves directly or indirectly individuals or groups of actors with different roles, at various levels and steps of the production process. For instance, there are the producer, its suppliers, people who make up the workforce, retailers, and consumers themselves. Not all processes involve all of these actors, but they are found in most agri-food processes. Note also that if an actor is an organization, like the producer firm or the analysis lab, the actions related to this actor are performed by one or more human person who have the right to represent the organization.

In the model, each actor is identified by a unique address, and is able to send transactions to the blockchain from this address. The actor owns the private key associated with the address, thus being the only person able to send messages from that address. For human actors, a mechanism which associates a human identity to the address is also needed, which is performed by the Address Catalog described in section 5.3.1.2.

The key actors of a typical agri-food production system are:

- **Administrator/Owner**: administers the software system by managing and controlling the reading and writing access to the system by other actors, and their permissions. This is a role present in most business process management systems.
- **Producer**: produces the raw materials that are the inputs of the supply chain. It is able to generate tokens associated with the produced materials. It can manage information certified in the system.

- **Supplier**: supplies materials, services, or devices needed for producing the target goods. It is able to generate tokens associated with the produced materials. It can manage information certified in the system.
- **Transformer**: transforms raw material, or already transformed material, into intermediate goods or into the final good. It is able to take ownership of tokens associated with the produced materials. A transformer can work on multiple input materials or goods, and produce multiple goods. It can manage information certified in the system.
- **Wholesaler**: buys the target good in large quantities and sells it to other wholesalers, or to retailers. It is able to take ownership of tokens associated to target good. It could manage information certified in the system.
- **Retailer**: buys the target good to sell it to End Customers. It is able to take ownership of tokens associated to target good. It could manage information certified in the system.
- **End Customer**: buys the target good from Retailers. S/he is usually not provided with an address.
- **Certification Authority**: Public or private authority in charge of controlling and certifying a given production. It is able to certify the goodness of amounts and documents, or to directly produce certificates. Usually, it manages information certified in the system. For instance, the Regional Authority, or Protection Consortium that perform inspections to verify the conformity of the products and the work of each actor of the supply chain. The inspections can be performed by viewing the data documents stored by the nodes of the blockchain.
- **Professional**: is a person with a given degree and experience, qualified to certify the goodness of amounts and documents. The list of professionals qualified in a given field is usually maintained by a Certification Authority.
- **Analysis Lab**: is a laboratory able to perform physical, chemical, and/or biological analysis of given materials, products, or goods. The list of Labs qualified in a given field is usually maintained by a Certification Authority. It can manage information certified in the system.
- **Warehouse**: receives, stores and sends goods. It is able to take ownership of the tokens associated to target good, or simply to register their storage, leaving the ownership to the original one.
- **Device**: is an IoT device connected to the Internet, able to send transactions with measurements relevant to the supply chain. It can provide for instance weights, temperatures, Ph values, RFID tracks of shipments, positions, etc.

There can be different sub-types of each actor. Some actors could bear different roles together; for instance, a transformer or a wholesaler could also be a retailer, if enabled to sell goods directly to the end customer.

5.3.1.2 Entities

The next step is to decompose the design of an agri-food supply chain into its "entities", identified as either an object or a process, understood as a concept that has an identity and is meaningful for the model. In this phase, I focus on their identification and not on the relationships between objects and processes. I identified the main entities involved in an agri-food supply chain, that have distinct identities and share common features. They are:

- **Address Catalog**: for each address, the identity and the role(s) of the owner are specified; the catalog is managed by the Administrator/Owner of the system.
- **Producer**: a farmer or a firm producing or transforming agri-food products. Its representative(s) are identified by blockchain addresses.
- **Productive Resource:** it represents something that produces the main raw agri-food products. Typically, it is a field (orchard, vineyard, wheat field, vegetable garden, greenhouse, olive grove, etc.), a group of animals (flock, herd, poultry, etc.), a set of beehives. It is owned by a Producer. The system can hold information and documents on it and can register events related to its cultivation or farming.
- **Product**: it represents a production batch of something that comes from a Productive Resource, or from the transformation of other products. For instance, grapes are produced from a vineyard, must from pressed grapes, wine from must. It is linked to a Producer which is in charge of its processing. The system can hold information and documents on it and can register events related to its processing.
- **Token**: a given quantity (a number) created and assigned to a given address. The token represents the ownership of a specific amount of material, good, or asset. It can be split and transferred to other address(es). Since the token, once created, cannot be increased, it guarantees that only the original material/good/asset is managed by the system. Many kinds of tokens can be managed by the system.
- **Notarization** of documents assessing the process (treatments, harvest, chemical analysis, quantity produced in subsequent steps, etc.) and assuring the parties that the document is authentic and can be trusted. It enables verification of the originality of a document that must be kept on a server and available to download to authorized users. The notarization must include:
    - hash of the document;
    - registration date (always available as date and time of the transaction);
    - link to the document in the server, or information on how to access it;
    - possible metadata.

5.3.1.3 Data Types

The object model represents the part of the world that is of interest to the agri-food supply chain domain. Some attributes of the model, represent groups of related data that are the same in every agri-food supply chain, such as type, name, unique identifier, owner. However, an agri-food supply system may hold many more data, specific to the particular production, and production process.

Since the goal is to develop a general-purpose system, able to be configured for every agri-food production process, I use a flexible data representation. A first set of types allowed by the model are basic types, such as int, float, string, date, text (multiline string), enum. Other data types are more structured, and represent information needed to access data on the Internet, or to notarize and check the notarization of data. The allowed types are:

- **int**: numeric input field (digits with an initial '+' or '-' sign). It may have minimum and maximum value constraints.
- **float**: numeric input field, with decimal point. It may have minimum and maximum value, and precision of decimal part constraints.
- **string**: one-line string input field. It may have maximum length. Be careful to filter out any control characters.
- **enum**: input field of a string chosen from a given list. The admissible values are given in a list.
- **text**: multi-line string input field. It may have maximum length. Be careful to filter out any control characters.
- **link**: a one-line string input field containing a URL. It checks that the URL corresponds to an existing page or file. If the operator clicks on the URL, it shows its content in a pop-up.
- **hashlink**: a one-line string input field containing a URL pointing to a file, with another read-only input field holding the hash digest of the file. It checks that the URL corresponds to an existing file. It calculates the hash of the file with the given algorithm and shows it. If the operator clicks on the URL, it shows its content in a pop-up.
- **upload**: a local file input field that allows the operator to navigate the file system, choose a file, and then activate an "upload" button. If the operator clicks on the file name, it shows its contents in a pop-up.
- **hashupload**: a local file input field as above. It calculates the hash of the file with the given algorithm and shows it, allowing the upload of the file. If the operator clicks on the file name, it shows its contents in a pop-up.
- **upload or hashupload with photo**: it allows the operator to activate the camera of a device, to take a photo, to view it, to discard it and take another, to confirm its sending as a .jpg file, as in the case of "upload". Show a second read-only input field with the file hash digest if hashupload. If the operator clicks on the file name, it shows its contents in a pop-up.

5.3.1.4 Events in Agri-Food Supply Chain

The presented concepts are general of an agri-food supply chain, and would be valid also for information systems not based on a blockchain. On the contrary, the events that I am going to describe are directly linked to a supply chain management system based on SCs executed on a blockchain. In other words, they are events registered and enabled by blockchain technology.

In the specific case of Solidity language, an event is an inheritable member of the contract, which stores the arguments in the transaction's log for notifying services outside of the blockchain. In addition, for a better understanding of an agri-food supply chain process, I have grouped all these events into two macro categories:

1. **Transformation events**, which create a product starting from one or more resources, transform one or more products into one or more others, or divide the product into more sub-products that are of the same kind, but of lower quantities;
2. **Documentation events**, which associate data related to the production process to a product (or resource), but do not transform it and do not create other products.

I identified the most common events managed by a supply chain management system that uses a blockchain:

- **Asseveration**: a given entity stored on the blockchain (hash of a document, data, etc.) is certified by a transaction sent from an address of a person/body able to certify it. Also, the compliance of a product and token creation or transformation can be asseverated.
- **Creation of tokens** associated with some products of the process, at a given date. The creation can be provided of further data about the physical product associated with the tokens, and even of the notarization of documents attesting the truthfulness of the creation.
- **Product Merging**: the act of merging products of the same kind, producing other products of the same kind. Different batches of the same material can be merged, producing one or more new batches. The tokens associated with the products must be burned, and new tokens associated with the new products are created, preserving the overall number of tokens.
- **Product Splitting**: the act of splitting one product batch into two or more batches of the same kind. Also in this case the tokens associated with the original product must be burned, and new tokens associated with the new products are created, preserving the overall number of tokens.
- **Product Transformation**: the act of merging one or more products, producing one or more products of different kinds. For instance, grapes can be transformed into must, used to produce wine, but also to marc used to produce grape pomace brandy. Also in this case the tokens associated with the original products must be burned. So, new tokens associated with the new products are created by preserving the overall balance of quantities. The transformation is often associated with asseveration events.
- **Data Registration**: a record holding specific data is stored in the blockchain by a given address, guaranteeing the date and the actor who stored it. For instance, fertilization, pesticide treatments, and pruning are recorded as events linked to a field (Productive Resource).

- **Notarization Event**: the data concerning a Notarization (see section 5.3.1.2) are registered by a specific address, ensuring the date and inalterability of the document, and the signature of the registrant.
- **Certification of data or of a notarization**: a third party certifies, with a transaction coming from its address, the correctness of a data registration, or of a notarized document.
- **Unlocking**: one or more transactions from given addresses are needed to unlock a process, that is to register another event (typically, a transformation event).
- **Payment:** a payment (in cryptocurrency) is made available to a given address.

Events are declared by using a keyword followed by the name of the event to identify it, and a parameters list to save when the event is triggered. These parameter values enable to log the information or for executing the conditional logic. Events enable communication with the smart contract from front-end or other applications.

5.4 Building a configurable dApp system for agri-food traceability

Starting from the aforementioned phase, this approach proposes configurable and modular building blocks for agri-food supply chain management systems which can be represented on a blockchain.

After having analyzed the problem domain and identified the actors and the key entities, I designed a general SC structure, able to represent the problem domain, and to be configured to support specific agri-food supply chains. The SCs run on Ethereum blockchain, typically a permissioned version of it, and are written in Solidity language. The data structures of these general SCs include general data, which all instances of the SCs should have, and configurable data, specific for each supply chain.

Figure 28 shows the UML class diagram representing the basic structure of the system of Smart Contracts. This diagram uses the ABCDE method notation, which augments UML with stereotypes specific to Solidity language. Here, the UML class notation is used to represent classes and records - denoted with stereotypes "<<class>>" and "<<struct>>", respectively. The meaning of other stereotypes I used are easily understood. The general data of contracts are shown as UML attributes. Each productive resource or product has a mapping of relevant events, which are the elements where the configurable data are stored. In fact, the data pertaining specific agri-food productions are always associated to events happening to the resources or products. These data are stored in a byte array named "parameters" of struct "AgriEvent". For each data, I store its type, name and value, packed into bytes. In this way, the SC representing a single productive resource or product, can be configured to represent virtually any possible data structure and all kinds of events.

The automatic generation of the system's SCs is done starting from JSON, or .scv, files. In this system, the SCs used are the same regardless of the use case (for instance, olive oil, wine or honey production, just to cite a few). The customization is made by specifying the actual producers involved, the roles active in the system, the name and basic data of the resources and of the kind of products managed, the supported events, and finally the actual data used (name and type).

The system creates a SC for each producer (*Producer*), and a SC for each resource (*ProductiveResource*). These SCs, once created, do not change during the supply chain management.

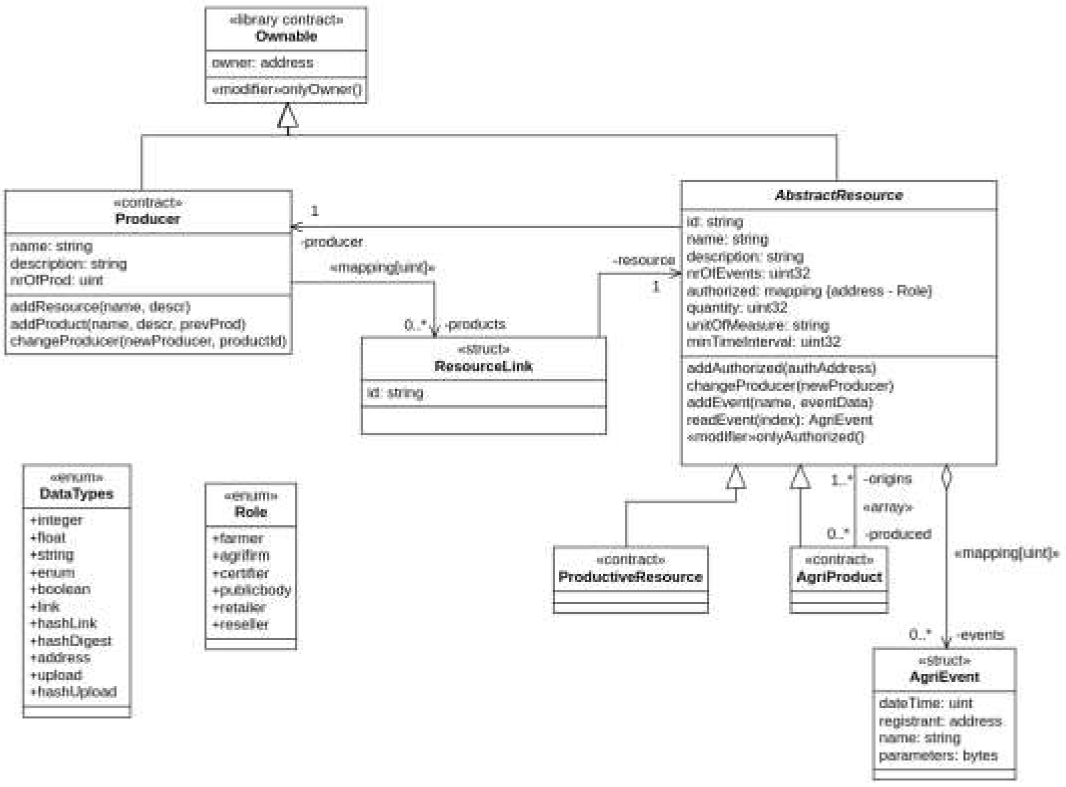


Figure 28 UML class diagram, with ABCDE method extensions, representing the Smart Contracts used in the system.

Each batch of product being grown, processed or transformed (*AgriProduct*), is associated to a SC, which takes into account the recordings of events (*AgriEvent*) on it.

Since both a *ProductiveResource* and an *AgriProduct* share several data and operations, they inherit from *AbstractResource* abstract SC.

Producer contains a mapping with the identifiers and the addresses all productive resources and products owned by, or related to it.

Products are generated or transformed starting from one or more productive resources, or from one or more existing products. Each product tracks its origin(s) through an array of addresses of the upstream resources or products. An *AbstractResource* holds a list of events (*AgriEvent*), which in turn contain a list of data related to the event (all encoded in the string of bytes named “parameters”), to flexibly attribute data to the events, as written above. If necessary, a list of generic data could also be added to the AbstractResource, encoded in a string of bytes, as in the “parameters” field of *AgriEvent*.

A QR code printed on the final product allows to find the related *AgriProduct* on the blockchain. It will directly hold the blockchain address of the SC associated with the batch of final products. From the QR code it is possible to retrieve all the events in (reverse) registration order and, for each event, all relevant data, including links to documents and their possible hash digest.

If an *AgriProduct* has one or more origins, by navigating to these further origins (*AgriProduct* or *ProductiveResource*), all upstream products/resources can be found, with the related events, and so on. At the end of this chain of products, you will also access its *ProductiveResources*, and from this its original *Producers*. Backward navigation must be thoroughly designed, to make it easier in the case of multiple origins.

The relationship between product and upstream products/resources is navigable in both directions. An *AgriProduct* contains the array “origins”, with the addresses of the contracts of type *ProductiveResource* or *AgriProduct* that created it (*ProductiveResource* has no origins). This is a list created during the creation of *AgriProduct* and cannot be modified. A generic product/resource also contains a “produced” array with the addresses of the generated *AgriProducts*, due to events of kind: *Creation*, *Transformation*, *Division*, or *Contribution*. This array can be updated, typically by appending the address of a newly created *AgriProduct*.

To optimize the code, I use OpenZeppelin [193] - a toolkit to develop, compile, update, deploy and interact with Smart Contracts. I also systematically apply gas-saving patterns (section 2.4.5).

Note, however, that the flexibility of this approach will lead to high gas costs in creating and updating SCs on the Ethereum blockchain. Representing data using arrays of bytes (see next Section 5.4.1 is way more costly than using native data, both in terms of storage and of computations needed to code and decode it. For this reason, I advise using a permissioned blockchain publicly accessible for reading, and not a public blockchain. The costs of using a permissioned, or consortium, blockchain is much lower, and above all much more predictable than the cost of a public blockchain.

5.4.1 Data Types Representation

As already mentioned, since the goal is to develop a general-purpose system, able to be configured for every agri-food production process, I use a flexible data representation. Besides the common data, such as type, name, unique identifier, and owner, each element can be provided of a list of data, each represented by a string. The string representing a data includes three sub-strings giving the name of the data, its type, chosen among the set of allowed types previously described, and its value.

For instance, the grape harvest event could have associated the list of parameters that describe how many quintals were collected (type *int*), the grape variety (type *string*), and even a photo taken during the harvest (type *upload*). The following piece of code shows how these three data might be embedded in a string:

```
1harvest#4t#3grape#nebbiolo#Bphoto#https://langhe.it/nebb-12.jpg
```

Here the first character is the data type (1: int, 3: string, B: photo upload), then there is the name of the field, the#' separator, and the encoded data. Each field ends with a '#' separator. The harvest amount is 225 quintals (*4t* is the Base 58 encoding of 225). The field (“*grape*” has “*nebbiolo”* as a value (a well-known variety of red grapes); the field, whose name is “*photo*”, holds the URL of an image related to the harvest.

5.4.2 Off-Chain Components

5.4.2.1 Off-chain Data

The costs of data storage in public blockchain are volatile and very expensive, so the blockchain is not a place suitable for containing large amounts of data. Even if a permissioned blockchain is used, the amount of data that can be stored inside a blockchain is limited, because data would be replicated in every node, unnecessarily wasting resources. Moreover, sometimes storing large amounts of data within a transaction can be downright impossible, due to the limited block size of the blockchain. For example, Ethereum has a block gas limit to limit the number, computational complexity, and size of transaction data included in any block.

In the case of large data, rather than storing the raw data directly on the blockchain, it is useful to store there a link to the data, and a short information able to identify the data. This pattern, known as Off-Chain Data Storage [24], consists in storing the hash digest of the raw data on-chain. This approach can be used to guarantee the date and integrity of such data. The hash value, recorded immutably in a blockchain transaction, guarantees that the original raw data from which the hash was derived were not changed afterwards. If the off-chain data change after the recording of their hash digest, the hash digest read from the blockchain and that computed on the changed data will differ, thus demonstrating the alteration.

The data stored off-chain have the following characteristics:

- They are accessed through a URL stored in the blockchain.
- They can be stored in different repositories, under different URL locations.
- If their immutability needs to be certified, their hash digest is stored in the blockchain, together with their URL. The date or registration and the address of the registrant are also always stored.
- The access control to the data must be performed off-chain, by the same system holding the data. In fact, the URL written in the blockchain cannot be hidden, due to the blockchain transparency.
- If the data are simple - that is a record with a few, simple fields - it is more convenient to store them directly on-chain. In agri-food management, many operations - for

instance fertilization, pesticide treatments, pruning, Ph analysis, weighing - are described by a few data.

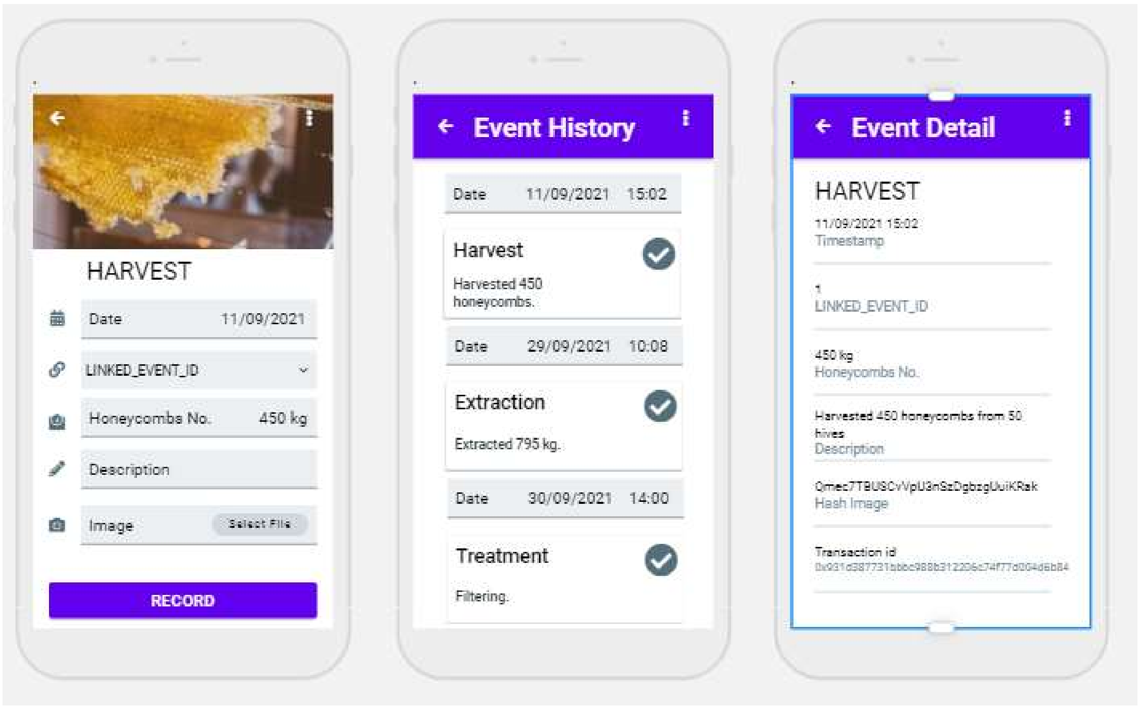
Recently, the combination of a blockchain and a distributed file systems has been used, and looks promising. According to [169], a well-established platform, Inter Planetary File System (IPFS), that is a peer-to-peer distributed file system ensuring immutability and non-reliance on a central server, could be a valid solution to store data off-chain. This solution has cost advantages, and ensures the integrity of the hash value that represents the raw data.

5.4.2.2 User Interface

The entities and the events of an agri-food supply chain, as defined in the sections above, are standard, and can cover most of production process. Consequently, also the applications enabling their input, editing (when allowed), and retrieving, will perform standard tasks.

For this reason, their user interface (UI) can be automatically generated, starting from the same description of the system used to generate the SCs. More precisely, once the producers, actors, resources, products and events of a specific production process are defined, with their data, constraints, and authorizations, it is possible to automatically generate an app able to create and edit the events defined for the production process. In this approach, the app is in fact a responsive HTML5 Web page provided with Javascript code.

The style and the appearance of the UI can be customized, but the data input, with all proper checks, does not require further programmer's intervention. Also, the navigation among the events, the products, and the productive resources can be automatically programmed, starting from a QR code written on the final product. This navigation does not require the user controls a blockchain address, and can be performed by every customer.


HARVEST

Date 11/09/2021

LINKED_EVENT_ID

Honeycombs No. 450 kg

Description

Image

Event History

Date 11/09/2021 15:02

Harvest

Harvested 450 honeycombs.

Date 29/09/2021 10:08

Extraction

Extracted 795 kg.

Date 30/09/2021 14:00

Treatment

Filtering.

Event Detail

HARVEST

11/09/2021 15:02
Timestamp

1
LINKED_EVENT_ID

450 kg
Honeycombs No.

Harvested 450 honeycombs from 50 hives
Description

Qmec7TBUSCvVpU3nSzDgbzgUuiKRak
Hash Image

Transaction id
0x931d387731abbc988b312206c74f77d004d6b84

Figure 29 Recording an event and navigating a product's event history.

Figure 29 shows some possible screenshots of the application, automatically generated, relating to the honey case study that I am going to present in Section 5.5. In this figure, you see how easily it is possible to record an event (for example *Harvest*) and the data associated with it, to view the history of events associated with a product, or to view the details of an event already recorded.

5.5 Case study: a blockchain traceability system for the honey supply chain

In this section, I show the effectiveness of the approach proposed in the previous section by providing a practical case study from the domain of the agri-food supply chain. This case study is a simplified version of a blockchain-based traceability system for certifying the origin and the quality of honey produced by members of a consortium. In particular, the system allows to trace the honey production, from the beehives to the honey harvest, to the honey potting, until the sale of jars to a wholesaler and finally to shops and supermarkets.

The choice is not accidental because honey, the main consumer product from beekeeping, has been identified as one of the most adulterated foods in the world through dilutions, substitution, or other fraudulent forms as misleading description of sources and geographical origin.

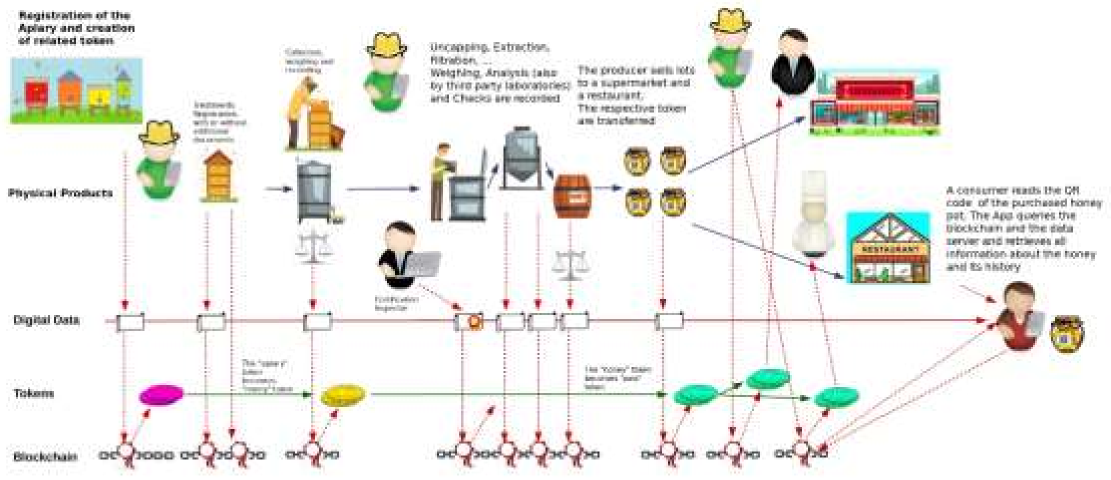

Registration of the Apiary and creation of related token
Uncapping, Extraction, Filtration, ... Weighing, Analysis (also by third party laboratories) and Checks are recorded
The producer sells lots to a supermarket and a restaurant. The respective token are transferred
Physical Products
A consumer reads the QR code of the purchased honey pot. The App queries the blockchain and the data server and retrieves all information about the honey and its history
RESTAURANT
Digital Data
Tokens
Blockchain

Figure 30 Simplified view of the honey supply chain process.

In Figure 30 I represent, in a simplified way, the process of the honey supply chain, in different layers. From top to bottom, the figure shows the actors, the layer of physical products (apiary, honey, jars), that of digital documents stored in one or more servers accessible from the Internet, the layer of "tokens", and of course the blockchain.

In the physical world, the actors perform actions, and register the related events in the underlying layers. Some events produce documents or images, which are stored off-chain in the layer of Digital Data, and record the document's link and hash digest on the blockchain, as described in section 5.4.2.1. Other events simply directly record the related information on the blockchain.

The tokens, stored in the blockchain, represent the physical quantities of productive resources and products. In this case study, the first token represents the number of beehives of the Productive Resource at the source of the represented honey production. The second token represents the amount of extracted honey expressed in Kg, which is related to the number of beehives - N beehives cannot produce more than kN Kg of honey, where k is a proper constant, depending on the specific kind of bee, beehive, and year. The third token represents the number of jars of a given weight produced, which is obviously related to the amount of honey poured into the jars. The SCs managing the transformation events from the Productive Resource (the apiary, with its beehives) to the Agri Product honey, and from the Agri Product honey to the Agri Product "batch of jars" will enforce the constraints that the amount of honey depends on the number and constant k of beehives, and that the amount of honey poured into the jars is greater or equal to the number of jars, multiplied by their capacity.

The proposed solution is designed to be highly transparent and scalable. Anyone can access the dApp website, open the Web page designed for customer access, read the QR code printed on the honey jar which includes the address of the Agri Product corresponding to the batch of the jar, and from this access the blockchain and start to navigate through the SCs holding the events describing the history of the honey in the jar.

Manufacturers and distributors, as well as retailers, can also create transactions through the mobile application, after a login giving their credentials. After the login, the system Web site will redirect the user to the proper Web page, which of course differs from that shown to generic users. A second control of the actors' ability to send transactions is also made by the SCs, which will accept transactions only from a list of accredited addresses, and further control that the specific service was invoked by an actor enabled to invoke it.

Figure 31 schematically shows some of the basic interactions that make up the system. The system consists of the blockchain with its SCs, of one or more servers that manage data on the cloud (documents, images, maps, etc.) and which might be also the websites of the consortium or of the beekeepers, and of the applications that run on a PC or on smartphones connected to the network.

Beekeepers, with their smartphones, can record the treatments done on the apiary, the honey extraction, and all the stages of its transformation. The analysis laboratory records the results on the cloud server, and certifies their hash digest on the blockchain, with a program that runs on a PC. An inspector of the Apiary Consortium examines the certified history of the honey in production and records in turn his certification. A consumer, reading the label of the honey jar she bought with a smartphone, can access the whole history of honey, certified on the blockchain.

5.5.1 Defining the actors

Accordingly to the methodology described, the honey passes through different chain actors to reach the final consumers. The first step is to identify and describe the actors involved in the system:

- **Beekeeper**: owns the apiaries and takes care of the breeding and good health of the bees. These are people engaged in beekeeping for the production of honey for sale and consumption. They sell raw honey to processors, or package and sell it directly to retailers and consumers.
- **Apiary Consortium**: a consortium of beekeepers that acts as a certifier of the quality and origin of honey. They do not participate directly in the production of honey, but promote marketing campaigns.
- **Processor**: this actor purchases crude honey from beekeepers, packages it with its brand name, and then sells the processed honey to retailers and consumers.
- **Certifier**: any independent actor who certifies the quality of the beehives and of the honey (agricultural expert, analysis laboratory, regional body, and the like);
- **Retailer**: delivers honey to consumers. This is a shop that engages in honey trading by buying honey directly from producers and sell it to final consumers.
- **Customer**: a person who, given a barrel, bucket, or jar of honey, even before buying the product, wants to verify its origin and history.

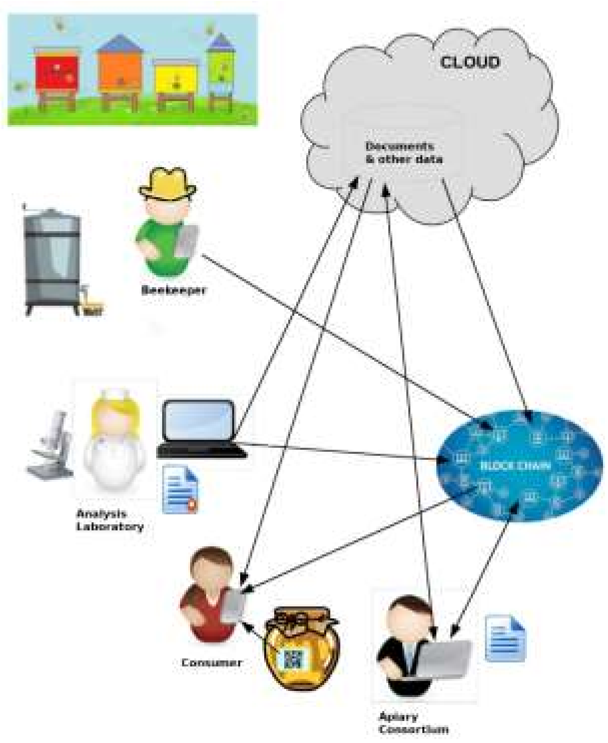


Figure 31 System interactions among actors.

5.5.2 Defining the entities

Traditionally, the best-known primary products of beekeeping are honey and wax. Pollen, propolis, royal jelly, poison, bees' larvae are also primary marketable bee products. For the sake of simplicity, I only deal with honey, and not with other products derived from the beehive.

The traceability system is therefore built starting from the entities of the honey chain and from the basic events, shown in Table 21 and Table 22, respectively, which are drawn up describing the specific production process. The primary productive resources are the apiaries, which vary little over time, and usually produce various products, on a periodic basis.

In this analysis, "honey" actually means a specific batch of product, whose processing chain is tracked by the system. A product or a resource can be transformed into another product (for example, an apiary into extracted honey, and then this into honey jars), or it can be divided into batches (which are also "products") for different processes (for example, a product "Honey" can be divided to be given to two different producers for potting and reselling). A product can derive from one or more primary resources, or from one or more upstream products.

Table 21 Resources and products managed by a honey traceability system.

Entity	**Data storage**	**Description**
Apiary	Blockchain, Producer's server	Registration of a production area, and related beehives. It can include photos, maps and other documents, which are kept on a dedicated server. The recording of this data takes place during the system initialization phase, before it enters operations. New data can be added, or data can be modified, at a later time. The blockchain holds links to the data, as well as their hash digest.
Honeycomb	Blockchain, possibly Producer's server	Registration of a specific set of honeycombs, harvested from the beehives. It is associated with events that document its harvesting, and which may include pdf files, photos, and other documents, which are kept on a dedicated server.
Honey	Blockchain, possibly Producer's or Retailer's server	Registration of a specific batch of honey, extracted from the honeycombs. It is associated with events that document its processing, and which may include pdf files, photos, and other documents, which are kept on a dedicated server.
Honey jars	Blockchain, possibly Producer's server	Registration of a batch of packaged honey, in which all jars have the same QR code on the label, allowing product tracing. Here we deal with jars, but in principle they might be also drums or buckets.

Both productive resources and agro-industrial products contain the following data, recorded in the blockchain:

- **id**: unique internal identifier, managed by the system;
- **name**: name of the product (or apiary);
- **quantity**: quantity of the product or resource;
- **unit of measure**: unit of measure of the given quantity;

- **producer**: the beekeeper, producer or retailer which produces, works on, or is in charge of the product or resource;
- **authorized** list of operators enabled to enter events on the product or resource.

The quantity serves to prevent products that are not tracked from being introduced into the process in an uncontrolled way. Data relating to a product or resource are associated with it through specific events.

Table 21 shows the system entities, the data associated with them and their description.

5.5.3 The system's events

I recall that there are two types of events: *Transformation Events (*TE) and *Documentation Events* (DE). Transformation events transform Apiary Productive resource into Honeycombs, Honeycombs into Honey, Honey into Honey jars (or buckets). The system also handles events able to transform two or more products into a single product (in this case, several batches of honey, coming from different hives, could be merged into a single batch), and to create one or more products starting from one or more products of the same kind (in this case, a batch of honey could be divided into smaller batches).

Table 22 shows the types of events managed by the system. The “Device” column shows the device with which the relative data are entered, which can be: PC, Smartphone, Tablet, or IoT device. The “Data storage” column illustrates where the data related to the event are stored: if it reports “blockchain”, the data are only in the blockchain, if it reports anything else, the data are in the indicated device and a corresponding link to the real data, and their hash digest, is registered in the blockchain. For each event, a description is provided. The “Entity” column describes the resources/products holding the data of the event.

Table 22 Events and functions of a honey traceability system: Transformation Events (TE), Documentation Events (DE).

Event/ Type	**Device**	**Data Storage**	**Entity**	**Description**
Harvest TE	Smartphone or Tablet	Blockchain, Producer's server	Apiary	This event takes place when honeycombs are harvested from beehives, at a given time of the year. It creates a Honeycomb batch. It guarantees that the transformation takes place respecting the temporal and quantitative constraints, that is the total quantity collected per year must be consistent with the number of hives in the apiary, according to a parameter defined when initializing the Apiary SC. For instance, a given apiary cannot produce more than 300 kg of honey per year.
Extraction TE	Smartphone or Tablet	Blockchain, Producer's server	Honey-comb	This event takes place when honey is extracted from honeycombs, after their harvesting. It creates a Honey batch. It guarantees that the transformation takes place respecting the constraints on the total

				quantity of honey contained inside the honeycombs, and on the percentage that can be extracted.
Merging TE	Smartphone or Tablet	Blockchain	Honey	Two or more batches of honey are merged, and transformed into a new batch. The event ensures that the total amount of honey is conserved. This event is initiated by a "pivot" honey Agri-Product, and is registered for all input honey batches. It does not include off-chain information.
Splitting TE	Smartphone or Tablet	Blockchain	Honey	A (batch of) honey is split into two or more batches. The sum of the quantities of the new products created must be equal to (or possibly less than) the quantity of the original product.
Potting TE	Smartphone or Tablet	Blockchain, Producer's server	Honey	A batch of honey is potted into a lot of jars. The event guarantees that the total amount of honey of the jars is equal (or less than) the quantity of the input honey batch. Additional information related to the event can be stored off-chain.
Conferral DE	Smartphone or Tablet	Blockchain	Honey or Honey jars	A product is conferred by the Producer in charge of it to another actor. For example, a given quantity of honey jars is sold to a wholesaler. The conferral must be accepted by the destination producer; this event, if conferral is not accepted, produces nothing and can be canceled by the simple insertion of subsequent events. Hence, it is a Documentation Event.
Acceptance DE	Smartphone or Tablet	Blockchain	Honey or Honey jars	A Producer (Processor or Retailer) accepts the conferral of a product by another Producer. The ownership of the Agri-Product is transferred to this Producer; no new product is created, so this is not a transformation event. Subsequent events will be registered by the new owner.
Description DE	PC	Blockchain, Producer's server	Apiary, Honey or Honey jars	Descriptive data associated with a product or an apiary are recorded. It can include photos, maps, and other documents, which are kept on a dedicated server. It can affect the quantities produced (e.g.: if hives are added or removed to an apiary).
Treatment DE	Smartphone or Tablet	Blockchain, possibly Producer's server	Apiary or Honey	Registration of bee breeding practices made on an apiary, or treatments made during the production process (addition of preservatives, pasteurization and so on.). It is possible to integrate it with off-chain documents, kept on the Producer's server.
Certification DE	Smartphone or Tablet	Blockchain, Producer's or Certifier's server	Apiary, Honey or Honey jars	A certifier (Apiary Consortium, or another Certifier) certifies a treatment, or the produced honey. It produces a certificate, kept on the server, whose hash digest is registered on the blockchain. This event is registered by a

				transaction sent from the address of the certifier, so its origin is certain.
Automatic Registration	IoT	Blockchain	Agri-Product	An IoT device performs a registration following an external event. For example, a refrigerator records the internal temperature every 2 hours. In this way, it will be possible to ascertain the storage conditions of fresh food. Not relevant for the honey system.
Verification	Smartphone or Tablet or PC		Honey jars	This is not a true event, but a service provided by the system. End customers, or any other actor in the process, check the integrity and history of the product. They read the product QR code, which includes the address of the SC related to the honey jars lot, and can read its history on the blockchain. If they want, they can also find the related documents (e.g.:.pdf files, images) following the links.

Figure 32 shows a simplified honey production process. Agro-industrial products are denoted with icons depicting the product. Each product can be associated with one or more *Documentation events* (DE), in the image denoted by the document with seal icon, and the dotted arrow. *Transformation events* (TE) are denoted by solid arrows with the corresponding event label, starting from the right side of products or resources, and cause them to be transformed into other products, or divided into batches. In the example shown, the honey comes from two apiaries (1 and 2). Through a single event (Harvest) it becomes honeycomb. Once extracted (Extraction), it becomes honey and is then divided (Splitting) into two batches (Honey 1 and Honey 2). These batches are then potted; the first is put in jars, the second in buckets.

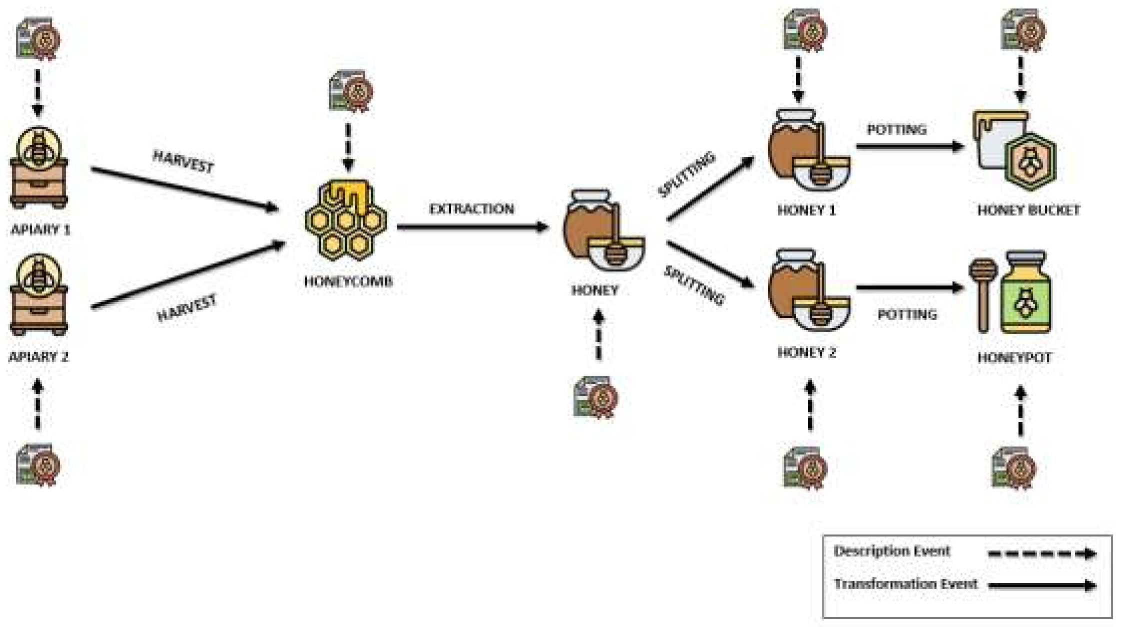
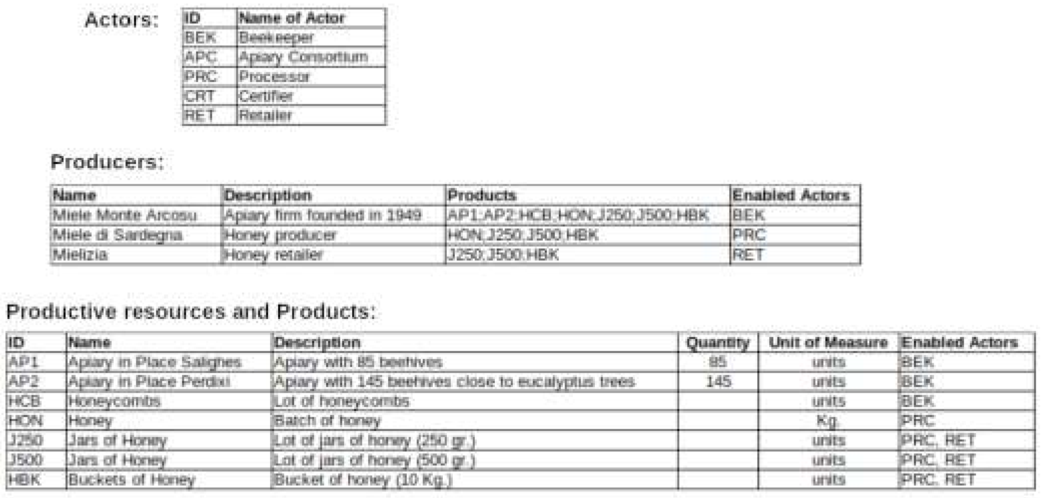


Figure 32 Honey production flow.

5.5.4 Automatic generation of the system's Smart Contracts

As already mentioned, the system's SCs are automatically generated from JSON, or .csv, files (in fact, .csv files are converted into JSON ones). In the case study system, I have five JSON files. Specifically, a JSON file for each of the following entities: actors, producers, resources/products, events, and parameters.



Actors:

ID	Name of Actor
BEK	Beekeeper
APC	Apiary Consortium
PRC	Processor
CRT	Certifier
RET	Retailer

Producers:

Name	Description	Products	Enabled Actors
Miele Monte Arcosu	Apiary firm founded in 1949	AP1;AP2;HCB;HON;J250;J500;HBK	BEK
Miele di Sardegna	Honey producer	HON;J250;J500;HBK	PRC
Mielizia	Honey retailer	J250;J500;HBK	RET

Productive resources and Products:

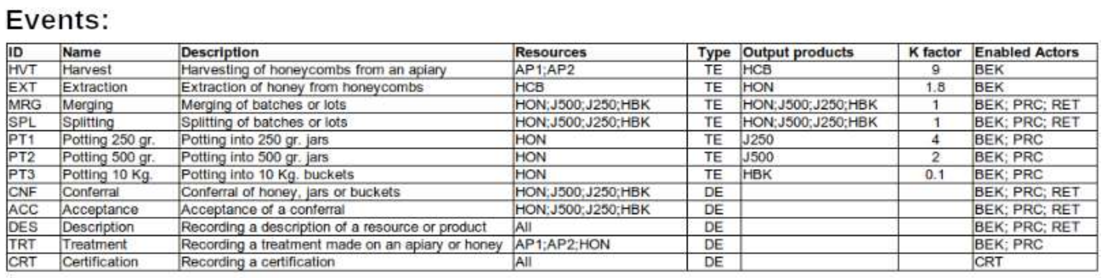
ID	Name	Description	Quantity	Unit of Measure	Enabled Actors
AP1	Apiary in Place Salighes	Apiary with 85 beehives	85	units	BEK
AP2	Apiary in Place Perdixi	Apiary with 145 beehives close to eucalyptus trees	145	units	BEK
HCB	Honeycombs	Lot of honeycombs		units	BEK
HON	Honey	Batch of honey		Kg.	PRC
J250	Jars of Honey	Lot of jars of honey (250 gr.)		units	PRC, RET
J500	Jars of Honey	Lot of jars of honey (500 gr.)		units	PRC, RET
HBK	Buckets of Honey	Bucket of honey (10 Kg.)		units	PRC, RET

Figure 33 The content of the .csv files describing Actors, Producers and Resources.

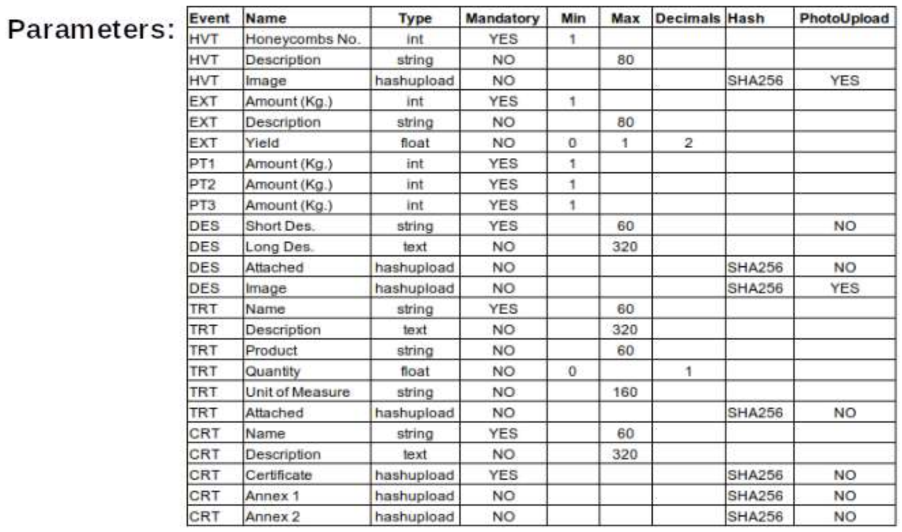
Figure 33 shows the content of these files for actors, producers and resources (Productive Resources and Products), for three producers - a beekeper (Miele Monte Arcosu), a firm which pots and sells honey (Miele di Sardegna), and a honey retailer (Mielizia).

The described resources are two apiaries managed by the beekeper, and various kinds of intermediate (honeycombs and raw honey) and final products (buckets and jars of honey). Only the apiaries have a defined quantity of beehives, which is set when the system is initialized, and can be changed using a specific function of the Product SC, called by its owner. The quantities of other products are set by the events which create them, under the constraints quoted in Section 5.3.1.4, "Transformation events".

The beekeeper also pots and sells honey, the producer just pots and sells it, whereas the retailer is only able to sell honey jars and buckets.


Events:

ID	Name	Description	Resources	Type	Output products	K factor	Enabled Actors
HVT	Harvest	Harvesting of honeycombs from an apiary	AP1;AP2	TE	HCB	9	BEK
EXT	Extraction	Extraction of honey from honeycombs	HCB	TE	HON	1.8	BEK
MRG	Merging	Merging of batches or lots	HON;J500;J250;HBK	TE	HON;J500;J250;HBK	1	BEK; PRC; RET
SPL	Splitting	Splitting of batches or lots	HON;J500;J250;HBK	TE	HON;J500;J250;HBK	1	BEK; PRC; RET
PT1	Potting 250 gr.	Potting into 250 gr. jars	HON	TE	J250	4	BEK; PRC
PT2	Potting 500 gr.	Potting into 500 gr. jars	HON	TE	J500	2	BEK; PRC
PT3	Potting 10 Kg.	Potting into 10 Kg. buckets	HON	TE	HBK	0.1	BEK; PRC
CNF	Conferral	Conferral of honey, jars or buckets	HON;J500;J250;HBK	DE			BEK; PRC; RET
ACC	Acceptance	Acceptance of a conferral	HON;J500;J250;HBK	DE			BEK; PRC; RET
DES	Description	Recording a description of a resource or product	All	DE			BEK; PRC; RET
TRT	Treatment	Recording a treatment made on an apiary or honey	AP1;AP2;HON	DE			BEK; PRC
CRT	Certification	Recording a certification	All	DE			CRT


Parameters:

Event	Name	Type	Mandatory	Min	Max	Decimals	Hash	PhotoUpload
HVT	Honeycombs No.	int	YES	1				
HVT	Description	string	NO		80			
HVT	Image	hashupload	NO				SHA256	YES
EXT	Amount (Kg.)	int	YES	1				
EXT	Description	string	NO		80			
EXT	Yield	float	NO	0	1	2		
PT1	Amount (Kg.)	int	YES	1				
PT2	Amount (Kg.)	int	YES	1				
PT3	Amount (Kg.)	int	YES	1				
DES	Short Des.	string	YES		60			NO
DES	Long Des.	text	NO		320			
DES	Attached	hashupload	NO				SHA256	NO
DES	Image	hashupload	NO				SHA256	YES
TRT	Name	string	YES		60			
TRT	Description	text	NO		320			
TRT	Product	string	NO		60			
TRT	Quantity	float	NO	0		1		
TRT	Unit of Measure	string	NO		160			
TRT	Attached	hashupload	NO				SHA256	NO
CRT	Name	string	YES		60			
CRT	Description	text	NO		320			
CRT	Certificate	hashupload	YES				SHA256	NO
CRT	Annex 1	hashupload	NO				SHA256	NO
CRT	Annex 2	hashupload	NO				SHA256	NO

Figure 34 The content of the .csv files describing Events, and Parameters, that is data values.

Figure 34 shows the content of the .csv file with the Event definition, and of the file with the definition of the data fields specific for each event.

The event definition includes the list of the resources the event can be attached to and, in the case of transformation event, the list of the products that can be generated. These events define also the factor k, which is used to determine the quantity of the generated product, given the quantity of the generating product. For instance, the unit of measure of honey is Kg, whereas the unit of measure of 500 g jars is their number. So, the conversion factor is 2 - 1 Kg of honey will produce 2 jars. Description events have nor output products, neither conversion factor.

The parameters table, shown in the same figure, reports for each event the data specific for that event. For instance, let examine the Harvest event (HVT). Remember that in this process harvest means to collect the honeycombs full of honey from the apiaries. The harvest event has three data fields:

- **Honeycombs No**: is equal to the number of harvested honeycombs, is an integer and is mandatory.
- **Description**: is a string holding a description of no more than 80 characters, optional.
- **Image**: is an optional image, obtained also using the smartphone camera, uploaded to a cloud server, and whose URL and hash digest are stored in the blockchain; it is optional.

This information is also used to automatically generate the user interface of the apps in charge of taking inputs from the operators of the system, and of the app able to navigate through the history of a final product.

5.5.4.1 System's characteristics

To summarize, the proposed system has the following characteristics:

1. Ability to manage the honey production cycle, as described following field inspections and interviews with beekeepers; later, other products can be added such as propolis, royal jelly, wax, etc.
2. Use of cloud space to store and retrieve documents, both documents produced by the operators, such as photos, and documents available in other ways, as long as they are accessible from the Internet.
3. Use of standard Android smartphones or tablets for data entry by system operators, via an easy-to-use user interface.
4. Use of a responsive website for system operations and management by the operators, and for retrieving information on honey batches by consumers.
5. Use of a permissioned blockchain with public access to ensure cost stability.
6. Need to produce and associate appropriate labels with QR code, to tag the final products (jars, buckets, honey drums).
7. Opportunity for the Apiary Consortium to become an official actor and certifier of the traced batches of honey. for this purpose, it must:
    a. Have a blockchain address, and an administrator to manage its private key and send transactions.
    b. Advertise the address on its site, to assure users about the identity behind the transactions originating from that address.
    c. Define procedures and parameters to certify the batches of honey produced by its members, like actual size and good state of health of an apiary, organoleptic qualities of the batch of honey analyzed, certification of laboratory analyses made on honey samples, etc.
8. The final consumer, pointing the smartphone at the QR code of the honey jar label, is connected to the website of point 4, and here s/he can go back to the “history” and certifications of the honey purchased.

5.6 Conclusions and Future Work

Nowadays, consumers worldwide want to be sure that the food they eat is safe and can be reliably traced back to its point of origin to give assurance that what they are buying is authentic and healthy. For this reason, they are demanding the highest standards of food safety throughout the supply chain, and they are willing to pay for the intangible attributes of secure traceability and country of origin labeling. Traceability systems are considered important to ensure the safety of a food product and prevent food fraud in the food supply chain. It is essential to improve the current traceability systems, as unscrupulous producers could exploit the gaps in the systems to their advantage and to the detriment of consumers.

Systems based on blockchain technology and smart contracts, integrated with the Internet of Things, allow to implement a traceability system where the producers can share the responsibility to contribute information to their products, and independent third parts can identify themselves and certify the correctness of the data related to products' origin and quality. In this way, the customer can be assured of the truthfulness of the reported information with a high degree of confidence.

In this context, I proposed a system enabling developers to quickly and smoothly develop traceability systems in the agri-food domain, without the need to grasp in every detail the technicalities of SC development, which is clearly different from classical software development. To this purpose, I accurately represented the problem domain, which was found suitable to such an approach, and developed a system able to automatically generate both the SCs and the UI of a tracing system.

The approach starts from the description of the supply chain to be traced in terms of actors, producers, resources and products, events and data. This description is given using a set of spreadsheet pages, which is a tool very easy to use also by people expert in the domain, but not in computer science. From these pages, converted to .csv files, the SCs are generated, as well as the HTML5 pages able to interact with them and providing the UI of the Dapp.

This methodology can be used at every node of the supply chain, and can capture critical events, which are subsequently recorded immutably. Also, the actors who registered the events can be identified with a very high degree of certainty. In this way, the certification of every step of the production process is not only made by the producer itself - as it is in traditional systems - but can be audited by trusted third parties, which give a much higher degree of trust that the information on the product is correct.

Specifically, the advantages of the management system of an agri-food supply chain via blockchain, generated in a semi-automatic way, are:

- The consumer can be sure of the origin, of the production process, and of the quality of the product purchased.
- The task of authorities in charge of the control of products, and of production processes, is facilitated, and on-site inspections can be reduced.
- The manufacturer can certify in a simple and non-falsifiable way all the steps of a production.
- Software development times and costs are reduced, while maintaining a high level of security and trust.

- Blockchain system might also manage contractual transactions and payments. In this case, the system could also be extended to allow payments, and the transfer of ownership of a product could also be associated to a cryptocurrency transfer.

To the best of my knowledge, this is the first attempt to automatically develop custom dApps for the agri-food supply chain, by building configurable SCs to be assembled together. I truly believe that the proposed approach is very innovative. Moreover, the proposed approach was actually used to develop some real tracing systems, thus confirming its capacities.

The research presented in this chapter adds value to the state of the art in several ways. Firstly, it helps the developers in creating higher-quality SCs, because the SC which are configured for a specific system are already proven and debugged. Secondly, it can help reducing development time, because systems are generated by compiling a description of the system given as tables of data. Thirdly, this approach makes food safety compliance easy, and significantly cuts down on paperwork for the actors in the agri-food supply chain.

This flexible approach has potential applications well beyond the honey industry, which I proposed as a case study, to a range of other agri-food producers. It could will be widely used for the design and development of dApps aimed to implement agri-food supply chain traceability systems. The presented methodology facilitates the communication between domain experts and developers, and then automatically transform the key concepts of the problem domain into SC code.

I am presently working on two extensions of the approach. First, to reduce the cost in gas of SC creation and execution, I am working to give the system the ability to automatically generate specific SC data structures from the data specification, instead of using the representation with byte arrays. This is not a simple task, because I also have to generate getter and setter functions, maintaining the security and reliability level of the present system.

Secondly, I am working to port the approach and the system to other Distributed Ledger Technology systems which are used for implementing permissioned blockchains, namely Hyperledger Fabric and Tendermint. I am also considering to generalize the system to support other kinds of supply chains, besides agri-food sector.

6 Can the Blockchain facilitate the development of an interport community?

Seaports are part of a complex and information-intensive maritime supply chain that includes a set of organizations and operators who need to exchange a large amount of information and data with each other. As supply chains become increasingly integrated and connected, the connectivity of stakeholders must be ensured not only within individual ports but also between ports.

This section explores the application prospects and practical implications of the application of Blockchain technology for the establishment of an interport community within which different ports organized as a network can exchange information and data in a secure and effective way.

With the support of SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis, I address several research questions concerning the practical impacts, benefits, pros and cons, economic and technical barriers related to the implementation of Blockchain technology to support the creation of an interport community.

The remainder of this chapter is organized as follows. Section 6.1 introduces the study; section 6.2 briefly describes the case study. In section 6.3 I highlight the research questions I addressed. Section 6.4 presents the SWOT analysis performed on the application prospects of using blockchain technology to manage logistics in an interport community. Finally, section 6.5 answers to the research questions and draws some conclusions.

6.1 Introduction

In the framework of the 4.0 revolution, digital transformation is of utmost importance for port and maritime logistics. Particularly, as crucial nodes in supply chains, seaports are required to constantly innovate and evolve to keep up with technological changes and remain competitive. Since the 1980s, the modernization of seaports has been shaped by digital innovation through three main stages of transformation:

- **first stage** (1980s): transformation into paperless procedures (development of the first EDI-based port community system, development of maritime industry-specific UN/EDIFACT message standards, etc.)
- **second stage** (1990s - 2000s): transformation into automated procedures (application of automatic identification systems AIS, introduction of radio-frequency identification for port operations, etc.)
- **third stage** (2010s - onwards): transformation into digital procedures to improve responsiveness and decision making (sensors, mobile technologies, cloud computing, machine learning, etc.), and to support the ongoing interaction and connection between the actors involved.

Now, as supply chains become increasingly integrated and connected, it is essential to ensure the connectivity of stakeholders not only within the single port but also between the various nodes of the transport network [194] [195]. Seaports are part of a complex and information-intensive maritime supply chain that includes a set of organizations and operators that are connected and distributed [196]. However, several challenges have so far characterized and slowed down the development of shared digital solutions in the port community: different levels of digital maturity between actors, missing standards, reluctance of operators to participate and share information, etc. [197]. The recently much debated Blockchain technology could offer interesting opportunities in this regard and is believed to have a huge impact on the future of the digitization of port and maritime logistics [198].

Whatever the sector of application, Blockchain technology allows for more secure tracking of all types of transactions (money transactions, data transactions, information transactions, etc.), reducing delays, additional costs and human errors [199]. Blockchain is designed to achieve decentralization, real-time peer-to-peer operation, anonymity, transparency, irreversibility and integrity in a widely applicable manner [200]. In the context of shipping, Blockchain technology is potentially a solution to the problem of distrust among players, as it does not rely on commercial third parties, but on a network of peers [201]. It is also believed to have the potential to positively affect maritime processes [202] and accelerate the physical flow of goods [203].

The growing interest towards Blockchain technology in the shipping sector is also evidenced by the development of the related scientific literature. To have an overview of the quantitative impact (in terms of number of published works) the topic of Blockchain in the maritime sector is having on the scientific literature, in early December 2020 a preliminary keyword search was carried out using the Scopus database. The search identified 61 studies (including only journal articles and conference proceedings), all published starting from 2017, which contained the following terms in the title, abstract or keywords: Blockchain + shipping, Blockchain + maritime, Blockchain + port(s). Most of these studies seem to focus primarily on the main trends and challenges, technical aspects, general opportunities and impacts related to state-of-the-art technologies, while the practical implications of adopting Blockchain solutions for specific port processes, as well as the actual repercussions for the different actors, seem to need further investigation, also considering that the technology is still new and immature [204] [205]. In fact, some trials and Blockchain pilot projects are already available in the maritime industry. However, most of them are linked to autonomous initiatives of industrial operators or single ports. For example, Maersk and IBM have decided to jointly build their own Blockchain solution to reduce the cost of global shipping by eliminating inefficiencies resulting from paper-based processes [206] while the ports of Rotterdam and Antwerp are developing their own Blockchains. In this regard, it should be emphasized that as long as Blockchain exists only in individual ports or in small groups of operators, its benefits will not be fully explored and exploited. In fact, in increasingly connected and integrated markets, the new era of digitization concerns not only the port's ability to become smart, but above all the ability to do so by connecting to larger networks.

Considering the above, this study intends to add to the existing literature by exploring the application prospects and practical implications of Blockchain technology for the construction of an interport community within which different ports organized as a network can exchange information and data in a secure and effective way.

6.2 Case study

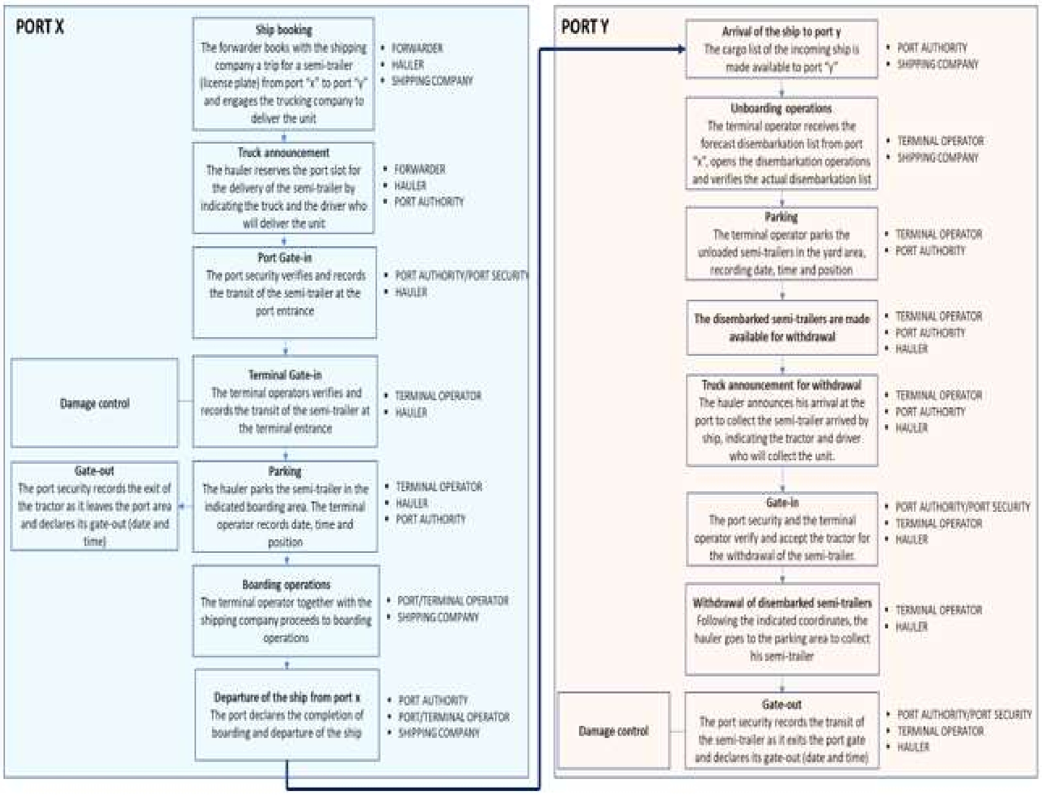
The idea for this exploratory study was born in the framework of a previous research project, the so-called EasyLog project, which designed and implemented an ICT system for the exchange and sharing of operational data between five ro-ro ports in the upper Tyrrhenian Sea [194]. The EasyLog system was designed according to a modular structure in which each port had its own customized local module for managing gate-in gate-out operations, and the five local modules communicated with each other using a shared set of rules for data exchange, formatting and availability. The EasyLog system allowed the exchange of data both within the port (between haulers, port authorities, and terminal operators) and between the five ports of the network. Despite the important innovation introduced by the EasyLog system, it did not include all the parties involved in the process (for example, shipping companies), it concerned only a limited number of operational information related to gate operations but no sensitive data, its process phases were not systematically linked between the stakeholders, thus leaving some room for improvements that Blockchain technology can potentially achieve. Figure 35 illustrates a simplified flow diagram relating to the diverse activities, and related information flows and operators involved, that could potentially benefit from the use of Blockchain for the purpose of creating an interport community between port “x” and port “y”.

With the support of SWOT (Strengths, Weaknesses, Opportunities and Threats) analysis, this work analyzes the application prospects to exploit the advantages of Blockchain to facilitate the secure exchange of information between the ports and the parties involved in an interport logistics chain.

6.3 Research Questions

It should be emphasized that, to the best of my knowledge, no available studies have been found that explore the practical implications and impacts of the application of Blockchain technology for the establishment of an interport community. Through this exploratory study, I try to provide insights on the topic by addressing the following research questions:

- Can Blockchain facilitate the development of an integrated port community?
- Which logistic activities in an interport community would be most impacted by Blockchain and in which way?
- What parties involved in a port community would benefit the most from Blockchain?
- What are the pros and cons of introducing Blockchain in a port community?



PORT X

Ship booking
The forwarder books with the shipping company a trip for a semi-trailer (license plate) from port "x" to port "y" and engages the trucking company to deliver the unit
- FORWARDER
- HAULER
- SHIPPING COMPANY

Truck announcement
The hauler reserves the port slot for the delivery of the semi-trailer by indicating the truck and the driver who will deliver the unit
- FORWARDER
- HAULER
- PORT AUTHORITY

Port Gate-in
The port security verifies and records the transit of the semi-trailer at the port entrance
- PORT AUTHORITY/PORT SECURITY
- HAULER

Damage control

Terminal Gate-in
The terminal operators verifies and records the transit of the semi-trailer at the terminal entrance
- TERMINAL OPERATOR
- HAULER

Gate-out
The port security records the exit of the tractor as it leaves the port area and declares its gate-out (date and time)

Parking
The hauler parks the semi-trailer in the indicated boarding area. The terminal operator records date, time and position
- TERMINAL OPERATOR
- HAULER
- PORT AUTHORITY

Boarding operations
The terminal operator together with the shipping company proceeds to boarding operations
- PORT/TERMINAL OPERATOR
- SHIPPING COMPANY

Departure of the ship from port x
The port declares the completion of boarding and departure of the ship
- PORT AUTHORITY
- PORT/TERMINAL OPERATOR
- SHIPPING COMPANY

PORT Y

Arrival of the ship to port y
The cargo list of the incoming ship is made available to port "y"
- PORT AUTHORITY
- SHIPPING COMPANY

Unboarding operations
The terminal operator receives the forecast disembarkation list from port "x", opens the disembarkation operations and verifies the actual disembarkation list
- TERMINAL OPERATOR
- SHIPPING COMPANY

Parking
The terminal operator parks the unloaded semi-trailers in the yard area, recording date, time and position
- TERMINAL OPERATOR
- PORT AUTHORITY

The disembarked semi-trailers are made available for withdrawal
- TERMINAL OPERATOR
- PORT AUTHORITY
- HAULER

Truck announcement for withdrawal
The hauler announces his arrival at the port to collect the semi-trailer arrived by ship, indicating the tractor and driver who will collect the unit.
- TERMINAL OPERATOR
- PORT AUTHORITY
- HAULER

Gate-in
The port security and the terminal operator verify and accept the tractor for the withdrawal of the semi-trailer.
- PORT AUTHORITY/PORT SECURITY
- TERMINAL OPERATOR
- HAULER

Withdrawal of disembarked semi-trailers
Following the indicated coordinates, the hauler goes to the parking area to collect his semi-trailer
- TERMINAL OPERATOR
- HAULER

Damage control

Gate-out
The port security records the transit of the semi-trailer as it exits the port gate and declares its gate-out (date and time)
- PORT AUTHORITY/PORT SECURITY
- TERMINAL OPERATOR
- HAULER

Figure 35 Flow diagram of processes in a ro-ro interport logistics chain.

Table 23 SWOT analysis of Blockchain technology.

Strengths	**Weaknesses**
• Trust • Security • Transparency • Tamper proof • Identity management	• Governance • Scalability • Adoption • Redundancy • Compliancy to GDPR
Opportunities	**Threats**
• Traceability • Disintermediation • Cost reduction • Supply chain flow management • Coordination	• Privacy • Legal aspects and normative • Connection to off-chain data

6.4 SWOT analysis

It should be emphasized that, to the best of my knowledge, no available studies have been found that explore the practical implications of Blockchain technologies for what concerns strengths, weaknesses, opportunities and threats in managing port logistic, in particular for the case of an interport community.

In the following, I present a SWOT analysis for this specific case examining how the Blockchain can be used to concretely contribute to enhance port logistics. Table 23 identifies the main features contributing to the four dimensions of the SWOT analysis. The individual aspects are then discussed in the following paragraphs.

6.4.1 Strengths

Trust: In ports it is important that companies in the supply chain can trust each other to share information and increase efficiency in shared processes. A major obstacle to data sharing is that a leakage in sensitive information can affect companies' business. Managing the information flow of shared data with the Blockchain on the contrary can provide a trusted and unforgeable corpus of data records that can help in clearly identify responsibilities and sources of damages, delays, misconducts, errors and other aspects.

Security: Records stored into Blockchain blocks are secure since a double encryption mechanism is in place [207]. First, transactions are validated by means of private keys that make extensive use of cryptography. Thus, transactions cannot be altered and are secure. Second, each block is cryptographically chained to the previous one so that any attempt of records manipulation in a block is mirrored in changes in all the following blocks, so that after the addition of some block at the end of the chain data can be considered secured.

Transparency: Shipping companies typically keep their data reserved and secret to keep a competitive advantage over their competitors. On the other hand, port logistics can benefit from data sharing for planning in advance, for monitoring and coordinating operations and so on. Recording data on the Blockchain directly provides transparency and renders information directly and immediately available to authorized actors [207]. This is especially useful when the information needs to be updated and quickly broadcast to all the actors. With Blockchain solutions, data is shared in real-time, and payment can be made and confirmed almost instantly.

Tamper proof: Transparency is also provided by the tamper proof of Blockchain data. In fact, once data are loaded into the blocks they cannot be manipulated or altered in any way. All the actors involved in port logistic will benefit from this certainty of information not only in real time but also for what concerns historical data for retrospective analysis for process improvement.

Identity management: Given that all writing operations on Blockchain are managed through Blockchain addresses which can operate only using the associated private keyword all operations can directly be reconducted to the address' owners providing a transparent a built-in mechanism for managing identities. Furthermore, Smart Contracts can be written so that only specific addresses are allowed to perform specific operations providing a built- in secure mechanism for managing permits and authorizations.

6.4.2 Weaknesses

Governance: The missing of a central authority can be an advantage but also a problem. In some cases, in fact there is the need for organizing some governance at different levels. In port logistic, once multiple participants are connected, questions will arise about the governance of the system, how and who is authorized to access the data (accessibility), and who owns the data (ownership) shared in the Blockchain. Furthermore, in the case of a private Blockchain solution, there is the question of who is the neutral party in charge of setting the network rules and granting access authorization.

Scalability: Blockchain technology suffers from scalability and performance problems: all nodes in the chain must process all transactions and this may present a problem with large-scale and especially global implementations [200]. In a large interport community the number of actors and operation involved can be cumbersome and peaks of activities in short time (e.g., load or download of a ferry carrier) can become critical. The adoption of the correct Blockchain solution must consider this problematic so that the proper transaction rates and throughputs must be assured to match logistic needs.

Adoption: The adoption of Blockchain technology can be difficult due to technical and functional aspects. Implementing Blockchain technology on a very large scale would require a great deal of computing power. Currently, only a few larger companies seem to be experimenting with Blockchain. Eventually, they will create a platform that SMEs could also connect to. SMEs, however, may be reluctant to participate in the platforms of these large players. Also, because different companies' databases may use different standards or languages to store data, interaction may prove difficult. Addressing this would require resources and time to align databases or invest in a shared language for the Blockchain.

Redundancy: Blockchain provides a redundant distributed ledger replicated on all the nodes. If this is an advantage for what concerns security, transparency and trust, it can be seen as a waste of resources which could be leveraged to other purposes. This may become particularly problematic when storage and computational power are critical resources for the actors involved.

Compliance to GDPR: In the framework of EU GDPR (General Data Protection Regulation) all sensitive data pertaining to private citizens or companies must be treated according to a strict regulation. In particular, citizens and companies have the right to be forgotten, so that upon a specific request all organization keeping citizens’ or companies’ data must remove them from their public databases. In the case of Blockchain all data recorded on the ledger cannot be removed or canceled anymore. This means that a specific strategy must be adopted from the very beginning to decide and plan what data must be recorded on-chain and what data must be kept off-chain.

6.4.3 Opportunities

Traceability: The Blockchain natively stores record in a temporally ordered sequence of valid transactions so that all operations related to port logistic can immediately be tracked, like load or unload of tracks and containers into shipments and other physical movements. But also, other events, like business agreements, buy and sell, signing of insurance contracts, can be easily included into the Blockchain tracking system, strongly reducing monitoring and auditing costs [207].

Disintermediation and Cost Reduction: Currently many of the port logistic transactions are carried out by e-mail or telephone calls, there are several intermediaries involved, with consequent increases in costs and possible human errors. As for the Blockchain and the smart contracts, these promise an increase in efficiency, a reduction in brokerage costs and greater neutrality in the regulation of contracts. In fact, since all the relevant events of a supply chain are recorded in a certain and immutable way, all the notified parties can immediately plan the consequent actions, including any payments [208].

Supply chain flow management: The Blockchain is the ideal technology for supply chain management, where the supply can be physical or simply information flow [18]. Instead of exchanging documentation, the stakeholders involved in the process are given permission to access the blocks where the data is stored. This leads to the creation of unique and shared information that can be accessed in real-time and with lower transaction costs. The process can be further accelerated by involving stakeholders external to the process (banks and insurance companies). The database can be further enhanced by using Internet of Things (IoT) devices and connecting them as Blockchain nodes. Furthermore, by connecting smart devices, it is also possible to fully automate the process.

Coordination: Currently, each freight operator has its own management platform and data set, with limited interconnection capacities. Operators operate with a limited view of the situation and of the impact of each element on the overall performance of the supply chain. Container supply chain information is dispersed. With Blockchain, transport operators can securely share their information with their trading partners through a reliable, multi-layered data access architecture. This would foster interoperability, efficiency and productivity. Furthermore, different Blockchain networks could interact with each other around the world [208].

6.4.4 Threats

Privacy: User privacy could be reduced because all nodes contain a complete copy of the ledger and there is no central authority to contact in the event of an obvious security breach [200]. One of the solutions is a platform that will combine a Blockchain, repurposed as an access control moderator, with an off-chain storage solution [209]. Users will not have to trust third parties and are always aware of the data that is collected about them and how it is used. Furthermore, the Blockchain recognizes users as owners of their personal data.

Legal aspects and normative: Currently there is a lack in the normative and regulation for what concerns the role of Blockchain technology as distributed ledger and on the legal recognition of Smart Contracts. This lack creates insecurity, because some aspects of smart contract technology could be adopted by the logistics market, only to be upregulated, or even to be considered illegal [210].

Connection to off-chain data: When dealing with massive documents, due to space constraints or costs, the typical pattern is to store them off-chain and to upload on-chain only their hashes and a pointer or reference to the external storage location. This does not grant that the external storage has the same properties the Blockchain has, and eventually the external storage can be dismissed over time. In this case only the metadata related to the documents can be retrieved but not the documents themselves. When planning to use off-chain storage this aspect must be considered [12].

6.5 Answer to Research Questions and Conclusions

Based on the considerations made, it can be said that Blockchain has considerable potential to facilitate the development of an integrated cross-border port community as it can increase trust between the parties, it can provide a facilitated communication platform, and it allows to guarantee the identities of the involved parties and their will to send the transaction.

The logistic activities that can be most affected by the Blockchain seem to be those relating to the notarization of contractual documents and passageways to the gates, including photographs and documents relating to the state of vehicles and goods for any insurance purposes. In addition, the management of access authorizations to the system in all ports with a single system and in a decentralized way, the guarantee of the origin of documents, and the sharing and secure transmission of information and notifications of events between the port actors, while preserving privacy and data ownership, can significantly benefit from the Blockchain.

All the players may benefit from the use of the Blockchain in support of intermodal traffic, from the for-profit enterprises, such as freight forwarders, terminal operators, shipping companies, haulers, to the port authorities. The former can think of reducing direct costs and increasing efficiency, thanks to a greater automation, the reduction of intermediaries and delays, but also of reducing indirect costs, such as legal ones, thanks to the decrease in disputes guaranteed by Blockchain certification and smart contracts. The latter can also have advantages in terms of reducing costs and having greater guarantees that laws and regulations are respected.

The pros of introducing Blockchain to a cross-border port community have been clearly listed in the previous sections. As for the cons, there are probably no negative consequences related to the introduction of a technology that increases the transparency of operations, the trust of the parties, the automation of many activities, the security and confidentiality of data transfers. The only reasons, not all justifiable, why one party could negatively view the introduction of the Blockchain in a cross-border port community may include the need to bear the costs of technological adoption and transition, low propensity to increase transparency, little trust against a new technology, possible privacy or scalability issues or system governance issues.

Part IV

Cryptocurrencies price forecasting

7 Forecasting Bitcoin closing price series using linear regression and neural networks models

In Part IV I present a work to forecast daily closing price series of Bitcoin, Litecoin and Ethereum cryptocurrencies, using data on prices and volumes of prior days.

Cryptocurrencies price behavior is still largely unexplored, presenting new opportunities for researchers and economists to highlight similarities and differences with standard financial prices. I compared my results with various benchmarks: one recent work on Bitcoin prices forecasting that follow different approaches, a well-known paper that uses Intel, National Bank shares and Microsoft daily NASDAQ closing prices spanning a 3-year interval and another, more recent paper which gives quantitative results on stock market index predictions.

I followed different approaches in parallel, implementing both statistical techniques and machine learning algorithms: the Simple Linear Regression (*SLR*) model for uni-variate series forecast using only closing prices, and the Multiple Linear Regression (*MLR*) model for multivariate series using both price and volume data.

I used two artificial neural networks as well: Multilayer Perceptron (*MLP*) and Long short-term memory (*LSTM*).

While the entire time series resulted to be indistinguishable from a random walk, the partitioning of datasets into shorter sequences, representing different price "regimes", allows to obtain precise forecast as evaluated in terms of Mean Absolute Percentage Error (*MAPE*) and relative Root Mean Square Error (*relativeRMSE*). In this case the best results are obtained using more than one previous price, thus confirming the existence of time regimes different from random walks.

The models perform well also in terms of time complexity, and provide overall results better than those obtained in the benchmark studies, improving the state-of-the-art.

The content of this part was partially published in:

- "Forecasting Bitcoin closing price series using linear regression and neural networks models", N. Uras, L. Marchesi, M. Marchesi, R. Tonelli. In PeerJ Computer Science, Volume 6, pp. e279, PeerJ Inc., 2020. [211]

Even if it is not completely centered on the topic of this thesis, this work has to do with the study of the phenomenon of blockchain technology and is part of the work done during my doctoral studies, so I decided to include it, albeit marginally.

7.1 Introduction

Bitcoin is a new entry in currency markets, it is the world's most valuable cryptocurrency, though it is officially considered as a commodity rather than a currency, and its price behaviour is still largely unexplored, presenting new opportunities for researchers and economists to highlight similarities and differences with standard financial currencies, also in view of its very different nature with respect to more traditional currencies or commodities. The price volatility of Bitcoin is far greater than that of fiat currencies [212], providing significant potential in comparison to mature financial markets [213] [214] [215].

According to *coinmarketcap* website [216], one of the most popular sites that provides almost real-time data on the listing of the various cryptocurrencies in global exchanges, on May 2019 Bitcoin market capitalization value is valued at approximately 105 billion of USD. Hence, forecasting Bitcoin price has also great implications both for investors and traders. Even if the number of bitcoin price forecasting studies is increasing, it still remains limited [217].

In this work, I approach the forecast of daily closing price series of the Bitcoin cryptocurrency using data on prices and volumes of prior days. I compare my results with three well-known recent papers, one dealing with Bitcoin prices forecasting using other approaches, one forecasting Intel, National Bank shares and Microsoft daily NASDAQ prices and one on stock market index forecasting using fusion of machine learning techniques.

The first paper I compare to, tries to predict three of the most challenging stock market time series data from NASDAQ historical quotes, namely Intel, National Bank shares and Microsoft daily closed (last) stock price, using a model based on chaotic mapping, firefly algorithm, and Support Vector Regression (SVR) [218].

In the second one [217] Mallqui and Fernandez used different machine learning techniques such as Artificial Neural Networks (ANN) and Support Vector Machines (SVM) to predict, among other things, closing prices of Bitcoin.

The third paper proposes a two-stage fusion approach to forecast stock market index. The first stage involves SVR. The second stage uses ANN, Random Forest (RF) and SVR [219].

I decided to predict these three share prices to give a sense of how Bitcoin is different from traditional markets. Moreover, to enrich my work, I applied the models also to two other two well-known cryptocurrencies: Ethereum and Litecoin.

In this work I forecast daily closing price series of Bitcoin cryptocurrency using data of prior days following different approaches in parallel, implementing both statistical techniques and machine learning algorithms. I tested the chosen algorithms on two datasets: the first consisting only of the closing prices of the previous days; the second adding the volume data. Since Bitcoin exchanges are open 24/7, the closing price reported on *coinmarketcap* I used, refers to the price at 11:59 PM UTC of any given day. The implemented algorithms are Simple Linear Regression (SLR) model for univariate series forecast, using only closing prices; a Multiple Linear Regression (MLR) model for multivariate series, using both price and volume data; a Multilayer Perceptron and a Long Short-Term Memory neural network tested using both the datasets.

The first step consisted in a statistical analysis of the overall series. From this analysis I show that the entire series are not distinguishable from a random walk. If the series were truly random walks, it would not be possible to make any forecasts.

Since I am interested in prices and not in price variations, I avoided the time series differencing technique by introducing and using the novel presented approach.

Therefore, each time series was segmented in shorter overlapping sequences in order to find shorter time regimes that do not resemble a random walk so that they can be easily modeled. Afterwards, I run all the algorithms again on the partitioned dataset.

The reminder of this chapter is organized as follows. Section 7.2 gives an overview of the current literature. Section 7.3 presents the methodology, briefly describing the data, their pre-processing, and finally the models used. Section 7.4 presents and discuss the results. Section 7.5 concludes the chapter.

7.2 Literature Review

Over the years many algorithms have been developed for forecasting time series in stock markets. The most widely adopted are based on the analysis of past market movements [220]. Among the others, Armano proposed a prediction system using a combination of genetic and neural approaches, having as inputs technical analysis factors that are combined with daily prices [221].

Enke discussed a hybrid prediction model that combines differential evolution-based fuzzy clustering with a fuzzy inference neural network for performing an index level forecast [222].

Kazem presented a forecasting model based on chaotic mapping, firefly algorithm, and support vector regression (SVR) to predict stock market prices [218]. Unlike other widely studied time series, still very few researches have focused on bitcoin price prediction. In a recent exploration McNally tried to ascertain with what accuracy the direction of Bitcoin price in USD can be predicted using machine learning algorithms like LSTM (Long short-term memory) and RNN (Recurrent Neural Network) [223].

Naimy tried to forecast the volatility of the Bitcoin/USD exchange rate using GARCH (Generalized Auto-Regressive Conditional Heteroscedasticity) models [224].

Sutiksno studied and applied α-Sutte indicator and Arima (Autoregressive Integrated Moving Average) methods to forecast historical data of Bitcoin [225]. Stocchi proposed the use of Fast Wavelet Transform to forecast Bitcoin prices [226].

Yang examined a few complexity measures of the Bitcoin transaction flow networks, and modeled the joint dynamic relationship between these complexity measures and Bitcoin market variables such as return and volatility [227].

Bakar presented a forecasting Bitcoin exchange rate model in high volatility environment, using autoregressive integrated moving average (ARIMA) algorithms [228].

Catania studied the predictability of cryptocurrencies time series, comparing several alternative univariate and multivariate models in point and density forecasting of four of the most capitalized series: Bitcoin, Litecoin, Ripple and Ethereum, using univariate Dynamic Linear Models and several multivariate Vector Autoregressive models with different forms of time variation [229].

Vo used knowledge of statistics for financial time series and machine learning to fit the parametric distribution and model and forecast the volatility of Bitcoin returns, and analyze its correlation to other financial market indicators [230].

Other approaches try to predict stock market index using fusion of machine learning techniques [219].

Akcora introduced a novel concept of chainlets, or bitcoin subgraphs, to evaluate the local topological structure of the Bitcoin graph over time and the role of chainlets on bitcoin price formation and dynamics [231].

Greave predicted the future price of bitcoin investigating the predictive power of blockchain network-based, in particular using the bitcoin transaction graph [232].

Since the cryptocurrencies market is at an early stage, the cited papers that deals with forecasting bitcoin prices had the opportunity to train and test their models on a quite narrow dataset. In particular, bitcoin market has been at first characterized by an almost constantly ascending price trend, the so-called bull-market condition. However, since 2018, it has been characterized by a strong descending price trend, the so-called bear-market condition.

Therefore, the cited papers trained their models on data of the first market condition, and tested them on data of the second type. These market conditions are shown in Figure 36 (a: bull-market condition; b: bear-market condition).

This study spans over a period of more than 4 years, characterized by different price dynamics. Therefore, I was able to train and test the models, including in each stage both bull- and bear- market conditions. For these reasons, this study enriches the state-of-the-art, as it is the most updated and deals with the biggest and more complete dataset.

7.3 Methods

In this section I first introduce some notions on time series analysis, which helped me to take the operational decisions about the algorithms I used and to better understand the results presented in the following. Then, I present the used dataset, including its pre-processing analysis. Finally, I introduce the proposed algorithms with the metrics employed to evaluate their performance and the statistical tools I adopted.

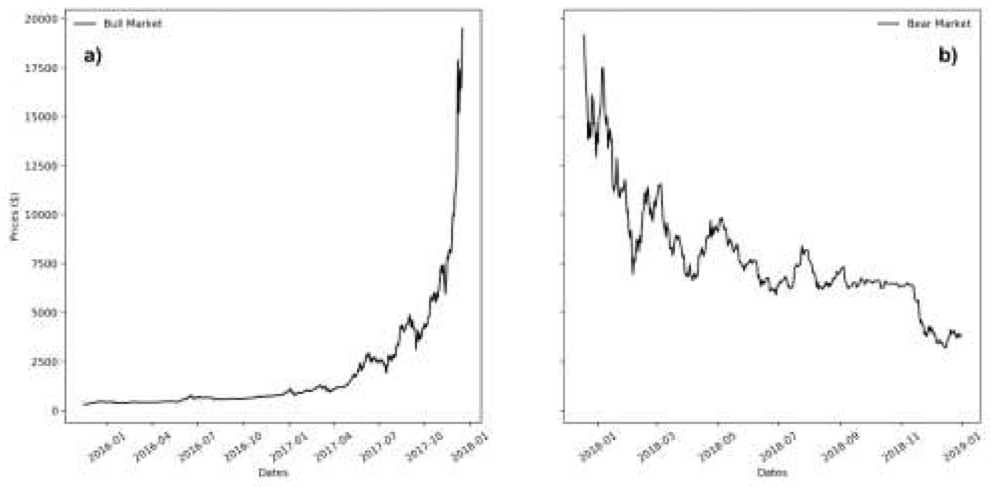


Figure 36 Bull (a) and Bear (b) price dynamics for Bitcoin market.

7.3.1 Time Series Analysis

7.3.1.1 Time Series Components

Any time series is supposed to consist of three systematic components that can be described and modelled. These are '*base level*', '*trend*' and '*seasonality*', plus one non-systematic component called *'noise'*. The *base level* is defined as the average value in the series. A *trend* is observed when there is an increasing or decreasing slope in the time series. *Seasonality* is observed when there is a repeated pattern between regular intervals, due to seasonal factors. *Noise* represents the random variations in the series. Every time series is a combination of these four components, where base level and noise always occur, whereas trend and seasonality are optional.

Depending on the nature of the trend and seasonality, a time series can be described as an additive or multiplicative model. This means that each observation in the series can be expressed as either a sum or a product of the components [233].

An additive model is described by following the linear equation:

$$y(t) = BaseLevel + Trend + Seasonality + Noise \tag{3}$$

A multiplicative model is instead represented by the following non-linear equation:

$$y(t) = BaseLevel * Trend * Seasonality * Noise \tag{4}$$

An additive model would be used when the variations around the trend does not vary with the level of the time series whereas a multiplicative model would be appropriate if the trend is proportional to the level of the time series. This method of time series decomposition is called "classical decomposition" [233].

7.3.1.2 Statistical Measures

The statistical measures calculated for each time series are the mean, labelled with μ, the standard deviation σ and the trimmed mean $\overline{\mu}$, obtained discarding a portion of data from both tails of the distribution. The trimmed mean is less sensitive to outliers than the mean, but it still gives a reasonable estimate of central tendency and can be very helpful for time series with high volatility.

7.3.2 Collected data

I tested the algorithms on six daily price series. Three of them are stock market series, all the data were extracted from the 'Historical Data' available on *yahoofinance* website [234]; the other ones are cryptocurrencies, namely Bitcoin, Ethereum and Litecoin price daily series, all the data were extracted from *coinmarketcap* website [216].

- Daily stock market prices for Microsoft Corporation (MSFT), from 9/12/2007 to 11/11/2011.
- Daily stock market prices for Intel Corporation (INTC), from 9/12/2007 to 11/11/2010.
- Daily stock market prices for National Bankshares Inc. (NKSH), from 6/27/2008 to 8/29/2011.
- Daily Bitcoin, Ethereum and Litecoin price series, from 15/11/2015 to 12/03/2020.

I state once more that I choose these price series and the related time intervals as benchmark to compare my results with well-known literature results obtained by using other methods.

Specifically, I have chosen for the stock market series the same time intervals chosen in [218].

The choice of Bitcoin as cryptocurrency is quite natural since it represents about 60% of the Total Market Capitalization. Ethereum and Litecoin were chosen since they are among the most important and well-known cryptocurrencies.

It is worth noting that, for the stock market series I used the same data of the work I compare to, whereas for the cryptocurrencies I used all the available data to have more significant results.

The dataset was divided into two sets, a training part and a testing part. After some empirical test the partition of the data which lead us to optimal solutions was 80% of the daily data for the training dataset and the remaining for the testing dataset.

7.3.3 Data pre-processing

For both models I prepared the dataset in order to have a set of inputs (X) and outputs (Y) with temporal dependence. I performed a one-step ahead forecast: the output Y is the value from the next (future) point of time while the inputs X are one or several values from the past, i.e. the so called *lagged* values. From now on I identify the number of used lagged values with the *lag* parameter. In the Linear Regression and *Univariate* LSTM models the dataset includes only the daily closing price series, hence there is only one single lag parameter for the *close* feature. On the contrary, in the Multiple Linear Regression and *Multivariate* LSTM models the dataset includes both *close* and *volume (USD)* series, hence I use two different lag parameters, one for the *close* and one for the *volume* feature. In both cases, I attempted to optimize the predictive performance of the models by varying the *lag* from 1 to 10.

7.3.4 Univariate versus Multivariate Forecasting

A univariate forecast consists of predicting time series made by observations belonging to a single feature recorded over time, in this case the closing price of the series considered. A multivariate forecast is a forecast in which the dataset consists of the observations of several features. Here I used:

- for BTC, ETH and LTC series all the features provided by *Coinmarketcap* website [216]: Open, High, Low, Close, Volume.
- for MSFT, INTC, NKSH series all the features provided by *Yahoofinance* website [234]: Date, Open, High, Low, Close, Volume.

I observed that adding features to the dataset did not lead to better predictions, but performance and sometimes also results worsened. For this reason, I decided to use in the multivariate analysis only the *close* and *volume* features, that provided the best results.

7.3.5 Statistical Analysis

As a first step I carried out a statistical analysis in order to check for non-stationarity in the time series, using the *augmented Dickey-Fuller test* and *autocorrelation plots* [235] [236].

A stochastic process with a *unit root* is non-stationary, namely shows statistical properties that change over time, including mean, variance and covariance, and can cause problems in predictability of time series models. A common process with *unit root* is the *random walk*. Often price time series show some characteristics which makes them indistinguishable from a random walk. The presence of such a process can be tested using a *unit root* test.

The *ADF* test is a statistical test that can be used to test for a *unit root* in a univariate process, such as time series samples. The null hypothesis H_0 of the *ADF* test is that there is a *unit root*, with the alternative H_a that there is no *unit root*. The most significant results provided by this test are the *observed test statistic*, the Mackinnon's approximate *p-value* and the *critical values* at the 1%, 5% and 10% levels.

The test statistic is simply the value provided by the *ADF* test for a given time series. Once this value is computed it can be compared to the relevant critical value for the Dickey-Fuller Test.

Critical values, usually referred to as α levels, are an error rate defined in the hypothesis test. They give the probability to reject the null hypothesis H_0. So if the observed test statistic is less than the critical value (keep in mind that ADF statistic values are always negative [235]), then the null hypothesis H_0 is rejected and no *unit root* is present.

The *p-value* is instead the probability to get a "more extreme" test statistic than the one observed, based on the assumed statistical hypothesis H_0, and its mathematical definition is shown in the equation:

$$p_{value} = P(\mathrm{t} \geq t_{observed} \mid \mathrm{H}_0) \tag{5}$$

The *p-value* is sometimes called *significance*, actually meaning the closeness of the *p-value* to zero: the lower the *p-value*, the higher the significance.

In my analysis I performed this test using the *adfuller()* function provided by the *statsmodels* Python library, and I chose a *significance level* of 5%.

Furthermore, the *autocorrelation plot*, also known as *correlogram*, allowed me to calculate the correlation between each observation and the observations at previous time steps, called *lag values*. In my case I employed the *autocorrelation_plot()* function provided by the python *Pandas* library [237].

7.3.6 Forecasting

I decided to follow two different approaches: the first uses two well-known statistical methods: Linear Regression (LR) and Multiple Linear Regression (MLR). The second uses two very common neural networks (NN): Multilayer Perceptron (MLP) NN and Long Short-Term Memory (LSTM) NN. The reasons of this choices are explained below.

7.3.6.1 Linear Regression and Multiple Linear Regression

Linear regression is a linear approach for modelling the relationship between a dependent variable and one independent variable, represented by the main equation:

$$y = b_0 + \overrightarrow{b_1} \cdot \overrightarrow{x_1} \tag{6}$$

where y and $\overrightarrow{x_1}$ are the dependent and the independent variable respectively, while b_0 is the intercept and $\overrightarrow{b_1}$ is the vector of slope coefficients. In my case the components of the vector $\overrightarrow{x_1}$, the independent variable, are the values of the closing prices of the previous days.

Therefore, $\overrightarrow{x_1}$ size is the value of the *lag* parameter. In this case y represents the closing price to be predicted.

This algorithm aims to find the curve that best fits the data, which best describes the relation between the dependent and independent variable. The algorithm finds the best fitting line plotting all the possible trend lines through our data and for each of them calculates and stores the amount $(y - \bar{y}^2)$, and then choose the one that minimizes the squared differences sum $\sum_i (y_i - \bar{y_i}^2)$, namely the line that minimizes the distance between the real points and those crossed by the line of best fit.

I then tried to forecast with multiple independent variables, adding to the *close* price feature the observations of several features, including *volume*, *highest value* and *lowest value* of the previous day. These information were gained from *coinmarketcap* website. In these cases, I used a Multiple Linear Regression model (MLR). The MLR equation is:

$$y = b_0 + \overrightarrow{b_1} \cdot \overrightarrow{x_1} + \cdots + \overrightarrow{b_n} \cdot \overrightarrow{x_n} = b_0 + \sum_{i=1}^{n} \overrightarrow{b_i} \cdot \overrightarrow{x_i} \tag{7}$$

where the index I refers to a particular independent variable and n is the dimension of the independent variables space.

I used the Linear and Multiple regression model of *scikit learn* [238]. I decided to use these two models for several reasons: they are simple to write, use and understand, they are fast to compute, they are commonly used models and fit well to datasets with few features, like mine. Their disadvantage is that they can model only linear relationships.

7.3.6.2 Multilayer Perceptron

A multilayer perceptron (MLP) is a feedforward artificial neural network that generates a set of outputs from a set of inputs. It consists of at least three layers of neurons: an input layer, a hidden layer and an output layer. Each neuron, apart from the input ones, has a nonlinear activation function. MLP uses backpropagation for training the network.

In my model I keep the structure as simple as possible, with a single hidden layer. The inputs are the closing prices of the previous days, where the number of values considered depends on the *lag* parameter. The output is the forecast price. The optimal number of neurons were found by optimizing the network architecture on the number of neurons itself, varying it in an interval between 5 and 100. I used the Python *Keras* library [239].

7.3.6.3 LSTM Networks

Long Short-Term Memory networks are nothing more than a prominent variation of Recurrent Neural Network (RNN). RNN's are a class of artificial neural network with a specific architecture oriented at recognizing patterns in sequences of data of various kinds: texts, genomes, handwriting, the spoken word, or numerical time series data emanating from sensors, markets or other sources [240]. Simple recurrent neural networks are proven to perform well only for short-term memory and are unable to capture long-term dependencies in a sequence.

On the contrary, LSTM networks are a special kind of RNN, able at learning long-term dependencies.

The model is organized in cells which include several operations. LSTM hold an internal state variable, which is passed from one cell to another and modified by Operation Gates (forget gate, input gate, output gate). These gates control how much of the internal state is passed to the output and work in a similar way to other gates. These three gates have independent weights and biases; hence the network will learn how much of the past output and of the current input to retain and how much of the internal state to send out to the output.

In my case the inputs are the closing prices of the previous days and the number of values considered depends on the *lag* parameter. The output is the forecast price. I used the *Keras* framework for deep learning. My model consists of one stacked LSTM layer with 64 units each and the densely connected output layer with one neuron. I used Adam optimizer and MSE (mean squared error) as a loss.

I optimized the *LSTM* model searching for the best set of *epochs* and *batch size* hyperparameters values. These hyperparameters strongly depend on the number of observations available for the experiment.

Due to the recently birth of the cryptocurrency markets, the dimensions of my datasets are quite limited (around 1000 observations), therefore I decided to vary the *epochs* hyperparameter from 300 to 800 with a step of 100.

The Keras LSTM algorithm I used sets as default value for *batch size* 32. So, for each fixed *epoch*, I trained the model varying the *batch size* within the interval [22, 82] with a step of 10.

I did not take into account values less than 300 epochs, nor greater than 800 in order to avoid *underfitting* and *overfitting* problems. Furthermore, I did not consider *batch size* values less than 22, since they would lead to extremely long training times. Similarly, *batch size* values greater than 82 would not allow to find a good local minimum point of the chosen loss function during the learning procedure. The results obtained during the hyperparameters tuning are shown in Figure 37.

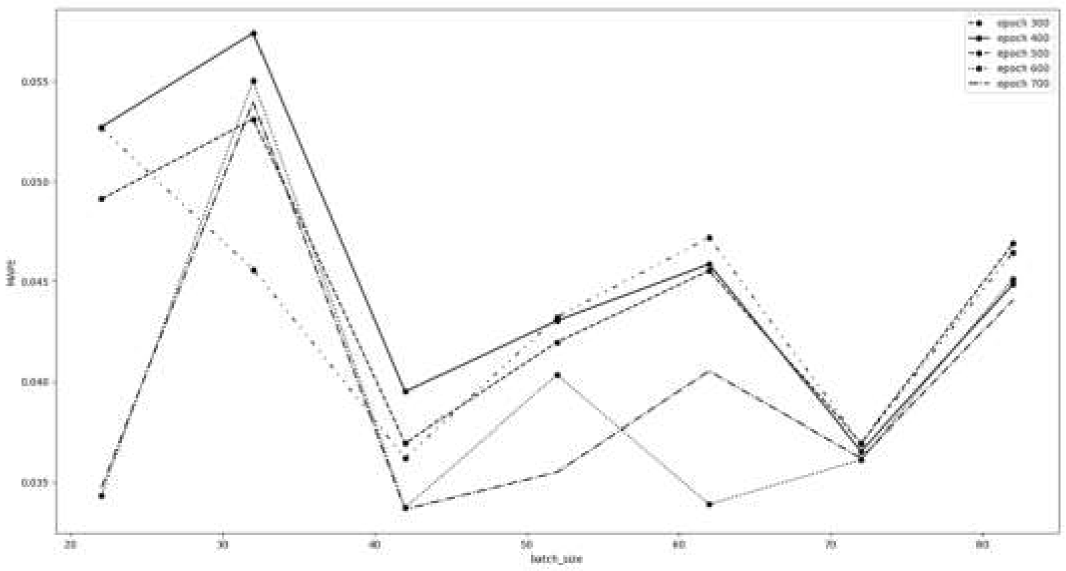


Figure 37 Bitcoin hyperparameters tuning results.

This figure shows the *MAPE* error as a function of the *batch size* hyperparameter, for each fixed epoch. As can be seen from the figure, I considered the *batch size* equal to 72 to be the optimal value. In fact, it is an excellent compromise, having a low MAPE value, which is also practically the same for all tested *epochs*. The optimal choice for the *epochs* hyperparameter is 600, which is the one that minimizes the MAPE error for *batch size* equal to 72, and is consistently among the best choices for almost all batch sizes considered. Therefore, the best set of *epochs* and *batch size* hyperparameters values chosen is 600 and 72, respectively.

7.3.7 Time Regimes

The time series considered are found to be indistinguishable from a random walk. This peculiarity is common for time series of financial markets, and in this case is confirmed by the predictions of the models, in which the best result is obtained considering only the price of the previous day.

The purpose is to find an approach that allows to avoid time series differencing technique, in view of the fact that I am interested in prices and not in price variations represented by integrated series of d-order. For this reason, each time series was segmented into short partially overlapping sequences, in order to find if shorter time regimes are present, where the series do not resemble a random walk. Finally, to continue with the forecasting procedure, a train and a test set were identified within each time regime.

For each regime I always sampled 200 observations - namely 200 daily prices. The beginning of the next regime is obtained with a shift of 120 points from the previous one. Thus, every regime is 200 points wide and has 80 points in common with the following one.

I chose a regime length of 200 days because, in this way, I obtain at least 5 regimes (from 5 to 12) for each time series to test the effectiveness of the algorithms, without excessively reducing the number of samples needed for training and testing. The choice was determined also according to the following: I performed the augmented Dickey-Fuller test on subsets of the data, starting from the whole set and progressively reducing the data window and sliding it through the data. The first subset of data that does not behave as random walks appears at time interval of 230 days, which I rounded to 200.

Since the time series considered have different lengths, the partition in regimes has generated:

- Bitcoin, Ethereum and Litecoin: 12 regimes
- Microsoft: 8 regimes
- Intel and National Bankshares: 5 regimes

From a mathematical point of view, the used approach can be described as follows. Let us target a vector $\overrightarrow{OA}$ along the t axis, with length 200. This vector is identified by the points $O(1,0), A(a,0) \equiv (200,0)$. The length of this vector represents the width of each time regime.

Let $\overrightarrow{OH}$ be a fixed translation vector along the t axis, identified by the points $O(1,0), H(h,0) \equiv (120,0)$. The length of $\overrightarrow{OH}$ represents the translation size.

For the sake of simplicity, let us label $\overrightarrow{OA}$ and $\overrightarrow{OH}$ vectors with $\vec{A}$ and $\vec{H}$.

Let $\overrightarrow{A'}$ be the vector $\vec{A}$ shifted by $\vec{H}$ and $\overrightarrow{A^n}$ the vector $\vec{A}$ shifted by *n* times $\vec{H}$. Therefore, the vector that identifies the *n*th sequence to be sampled along the series is given by:

$$\overrightarrow{A^n} = \vec{A} + n\vec{H} \tag{8}$$

where $n \in \left[0, \frac{D-A}{h}\right]$, being *D* the dimension of the sampling space, *A* the time regimes width and *h* the translation size. So, the *n*th time regime is given by:

$$R^n = f\left(\overrightarrow{A^n}\right) = f\left(\vec{A} + n\vec{H}\right) \tag{9}$$

where *f* is the function that maps the values along the *t* axis (dates) to the respective regimes *y* values (actual prices).

7.3.8 Performance Measures

To evaluate the effectiveness of different approaches, I used the *relative* Root Mean Square Error (rRMSE) and the Mean Absolute Percentage Error (MAPE), defined respectively as:

$$relativeRMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\frac{y_i - f_i}{y_i})^2} \quad (10)$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - f_i}{y_i}\right| \quad (11)$$

In both formulas y_i and f_i represent the actual and forecast values, and N is the number of forecasting periods. These are scale free performance measures, so that they are well appropriate to compare model performance results across series with different orders of magnitude, as in this study.

7.4 Results

7.4.1 Time Series Analysis

In Figure 38 I report the decomposition of Bitcoin (a-d) and Microsoft (e-h) time series, for comparison purposes, as obtained using the *seasonal_decompose()* method, provided by the Python *statsmodels* library [241].

The *seasonal_decompose()* method requires to specify whether the model is additive or multiplicative. In the Bitcoin time series, the trend of increase at the beginning is almost absent (from around 2016-04 to 2017-02); in later years, the frequency and the amplitude of the cycle appears to change over time.

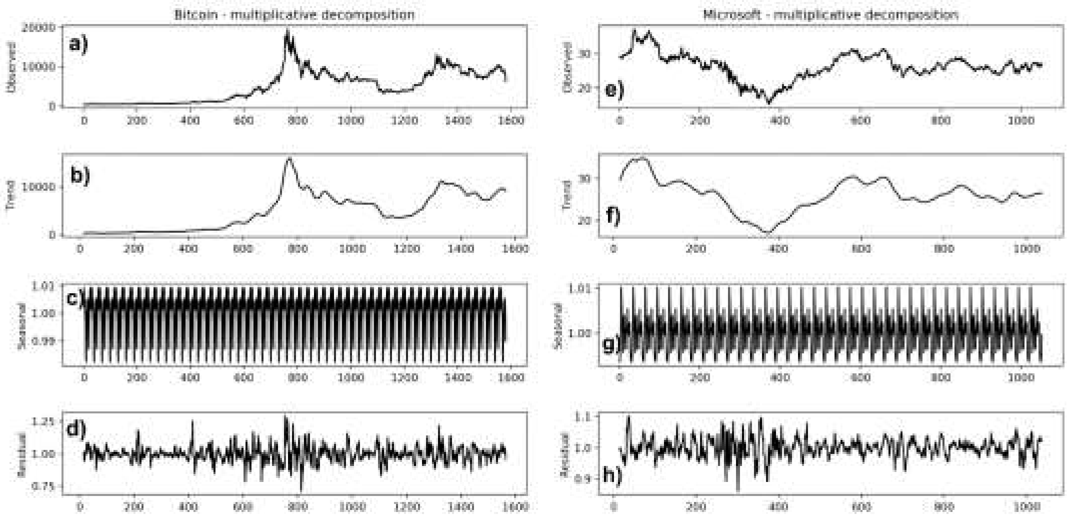


Figure 38 Decomposition of Bitcoin (a-d) and Microsoft (e-h) time series.

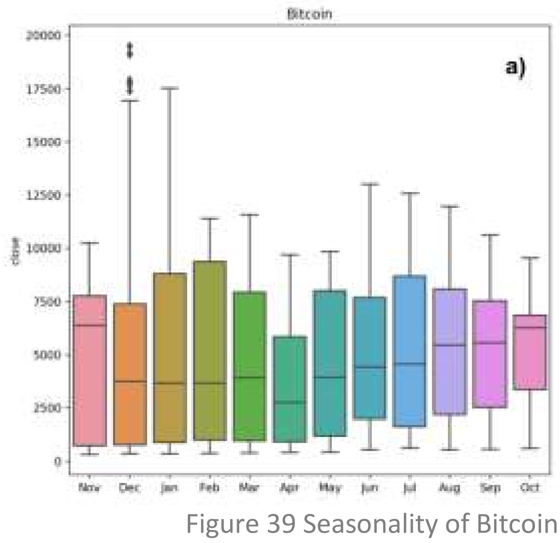
The Microsoft time series shows a non-linear seasonality over the whole period, with frequency and amplitude of the cycles changing over time. These considerations suggest that the model is multiplicative. Furthermore, if we look at the residuals, they look quite random, in agreement with their definitions. The Bitcoin residuals are likewise meaningful, showing periods of high variability in the later years of the series.

It is also possible to group the data at seasonal intervals, observing how the values are distributed and how they evolve over time. In this work I grouped the data of the same month over the considered years. This is achieved with the 'Box plot' of month-wide distribution, shown in Figure 39 (a: Bitcoin; b: Microsoft).

The Box plot is a standardized way of displaying the distribution of data based on five numbers summary: minimum, first quartile, median, third quartile and maximum. The box of the plot is a rectangle which encloses the middle half of the sample, with an end at each quartile. The length of the box is thus the inter-quartile range of the sample. The other dimension of the box has no meaning. A line is drawn across the box at the sample median. Whiskers sprout from the two ends of the box defining the outliers' range. The box length gives an indication of the sample variability, and for the Bitcoin samples shows a large variance, in almost all months, except for April, September and October. Not surprisingly, bitcoin volatility is much higher than Microsoft one. The line crossing the box shows where the sample is centered, i.e. the median.

The position of the box in its whiskers and the position of the line in the box also tell us whether the sample is symmetric or skewed, either to the right or to the left. The plot shows that the Bitcoin monthly samples are therefore skewed to the right. The top whisker is much longer than the bottom whiskers and the median is gravitating towards the bottom of the box. This is due to the very high prices that Bitcoin reached throughout the period between 2017 and 2018. These large values tend to skew the sample statistics.

In Microsoft, an alternation between samples skewed to the left and samples skewed to the right occurs, except for the sample of October that shows a symmetric distribution. Lack of symmetry entails one tail being longer than the other, distinguishing between heavy-tailed or light-tailed populations. In the Bitcoin case that the majority of the samples are left skewed populations with short tails. Microsoft shows an alternation between heavy-tailed and light-tailed distributions. Moreover, some Microsoft samples, particularly those with long tails, present outliers, representing anomalous values. This is due to the fact that heavy tailed distributions tend to have many outliers with very high values. The heavier the tail, the larger the probability that you will get one or more disproportionate values in a sample.



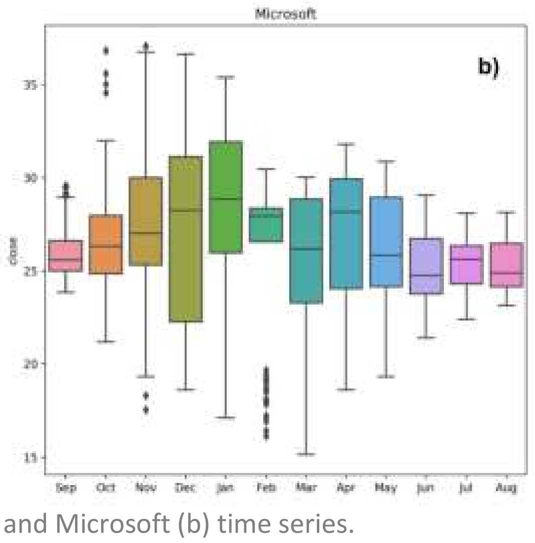


Figure 39 Seasonality of Bitcoin (a) and Microsoft (b) time series.

Table 24 and table in Figure 42 show the statistics calculated for each time series and for each short time regime. The unit of measurement of the values in the tables is the US dollar ($). In Table 24 we can observe that the only series for which the trimmed mean, obtained with *trim_mean()* method provided by the Python *scipy* library [242], with a cut-off percentage of 10%, is significantly different from the mean are Bitcoin, Ethereum and Litecoin. In particular the trimmed mean decreased. This is due to the fact that these cryptocurrencies, for a long period of time, registered a large price increment and this implies a shift of the mean to the right (i.e. to highest prices). This confirms that cryptocurrencies distribution is right-skewed. Table in Figure 42 shows that stock market series time regimes present a lower σ than BTC, ETH and LTC ones, namely that cryptocurrencies distribution has higher variance.

Table 24 Time Series Statistical Measures.

Series	**μ**	**σ**	$\bar{\mu}$
BTC	4931,3	3970,0	4593,1
ETH	216,8	239,8	171,2
LTC	55,9	58,0	45,6
MSFT	26,2	3,9	26,3
INTC	19,9	3,6	19,9
NKSH	24,3	3,9	24,5

Figure 40 and Figure 41 show the autocorrelation plots of BTC and MSFT series. The others stock market series are not presented because they show the same features of the MSFT series. Both autocorrelation plots (sub-figures c) show a strong autocorrelation between the current price and the closest previous observations and a linear fall-off from there to the first few hundred lag values.

I then tried to make the series stationary by taking the *first difference*. The autocorrelation plots of the 'differences series' (sub-figures d) show no significant relationship between the lagged observations. All correlations are small, close to zero and below the 95% and 99% confidence levels.

As regards the *augmented Dickey-Fuller* results, shown in

Table 25, looking at the observed *test statistics*, we can state that all the series follows a unit root process. I remind that the null hypothesis H_0 of the *ADF* test is that there is a *unit root*. In particular, all the observed *test statistics* are greater than those associated to all significance levels. This implies that I cannot reject the null hypothesis H_0, but does not imply that the null hypothesis is true.

Observing the *p-values*, we notice that for the stock market series there is a low probability to get a "more extreme" test statistic than the one observed under the null hypothesis H_0. Precisely, for both MSFT and INTC there is a probability of 29%, for NKSH a probability of 25%. The same considerations also apply to the Bitcoin, Ethereum and Litecoin cryptocurrency time series. So, H_0 cannot be rejected and so each time series present a *unit root* process.

We conclude that all the considered series show the statistical characteristics typical of a *random walk*.

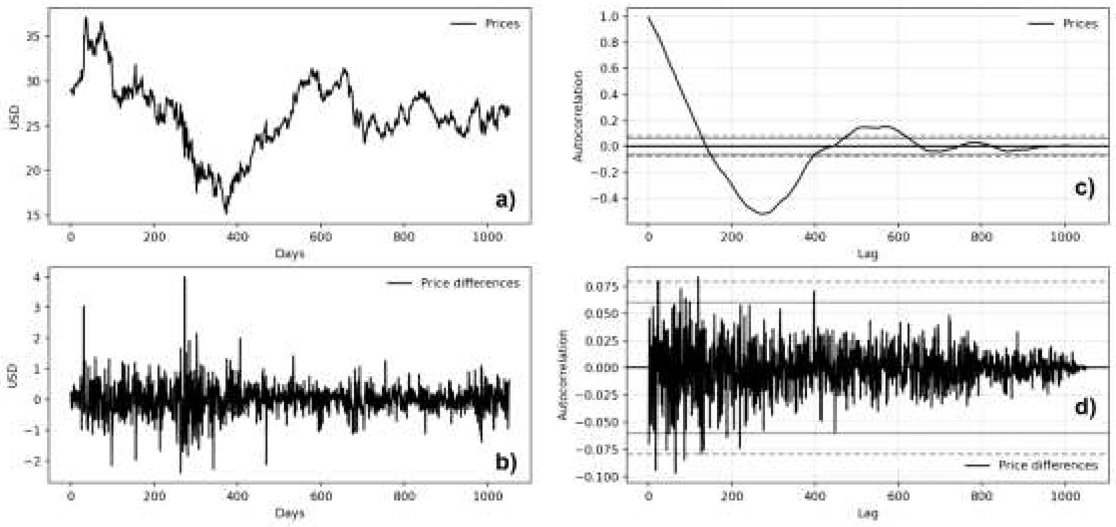


Figure 40 Microsoft time series autocorrelation plots.

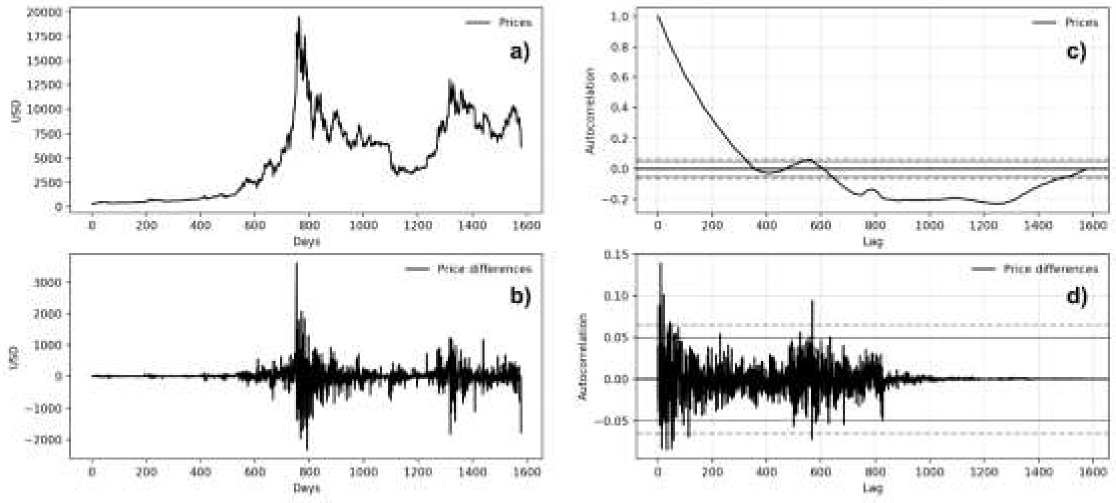
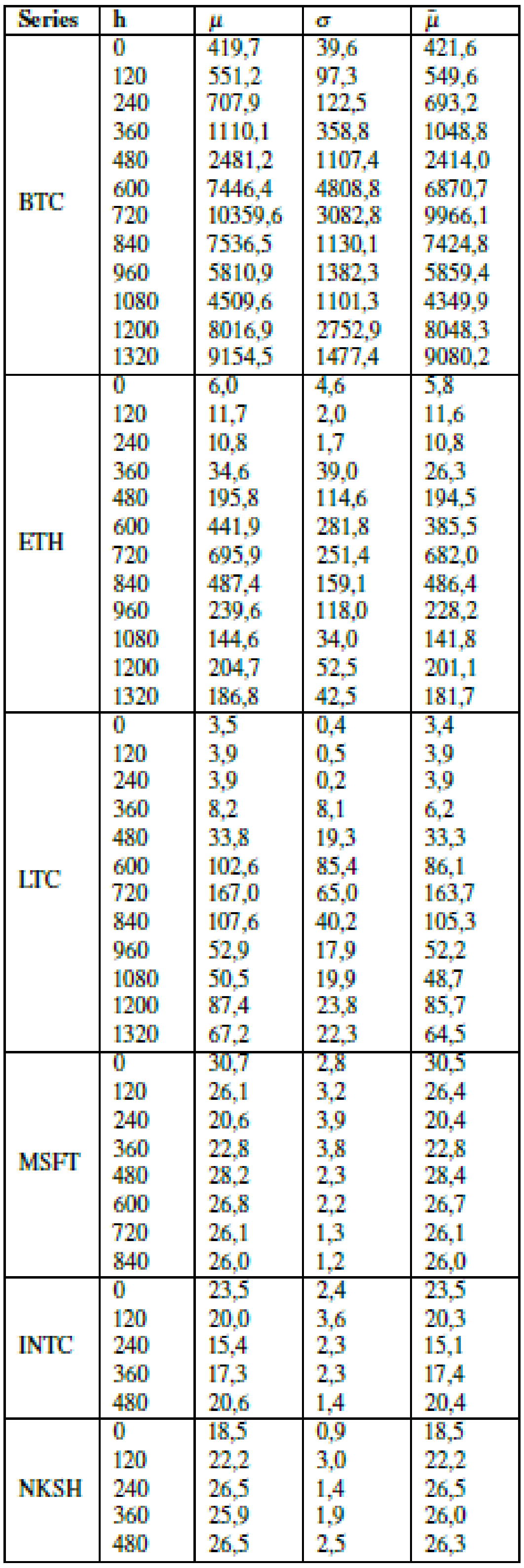


Figure 41 Bitcoin time series autocorrelation plots.

Table 25 Augmented Dickey-Fuller test results.

Series	**ADF statistic**	**p-value**
BTC	-2,12	0,24
ETH	-2,17	0,22
LTC	-2,34	0,16
MSFT	-1,98	0,29
INTC	-1,98	0,29
NKSH	-2,10	0,25



Series	h	μ	σ	$\tilde{\mu}$
BTC	0	419,7	39,6	421,6
	120	551,2	97,3	549,6
	240	707,9	122,5	693,2
	360	1110,1	358,8	1048,8
	480	2481,2	1107,4	2414,0
	600	7446,4	4808,8	6870,7
	720	10359,6	3082,8	9966,1
	840	7536,5	1130,1	7424,8
	960	5810,9	1382,3	5859,4
	1080	4509,6	1101,3	4349,9
	1200	8016,9	2752,9	8048,3
	1320	9154,5	1477,4	9080,2
ETH	0	6,0	4,6	5,8
	120	11,7	2,0	11,6
	240	10,8	1,7	10,8
	360	34,6	39,0	26,3
	480	195,8	114,6	194,5
	600	441,9	281,8	385,5
	720	695,9	251,4	682,0
	840	487,4	159,1	486,4
	960	239,6	118,0	228,2
	1080	144,6	34,0	141,8
	1200	204,7	52,5	201,1
	1320	186,8	42,5	181,7
LTC	0	3,5	0,4	3,4
	120	3,9	0,5	3,9
	240	3,9	0,2	3,9
	360	8,2	8,1	6,2
	480	33,8	19,3	33,3
	600	102,6	85,4	86,1
	720	167,0	65,0	163,7
	840	107,6	40,2	105,3
	960	52,9	17,9	52,2
	1080	50,5	19,9	48,7
	1200	87,4	23,8	85,7
	1320	67,2	22,3	64,5
MSFT	0	30,7	2,8	30,5
	120	26,1	3,2	26,4
	240	20,6	3,9	20,4
	360	22,8	3,8	22,8
	480	28,2	2,3	28,4
	600	26,8	2,2	26,7
	720	26,1	1,3	26,1
	840	26,0	1,2	26,0
INTC	0	23,5	2,4	23,5
	120	20,0	3,6	20,3
	240	15,4	2,3	15,1
	360	17,3	2,3	17,4
	480	20,6	1,4	20,4
NKSH	0	18,5	0,9	18,5
	120	22,2	3,0	22,2
	240	26,5	1,4	26,5
	360	25,9	1,9	26,0
	480	26,5	2,5	26,3

Figure 42 Regimes Statistical Measures.

7.4.2 Time Series Forecasting

Table 26 and Table 27 show the best results, in terms of MAPE and rRMSE, obtained with the different algorithms applied to the entire series. From now on, let us label the closing and the volume features *lag* parameters with k_p and k_v respectively.

In particular, Table 26 reports the results obtained using the *Linear Regression* algorithm for univariate series forecast, using only closing prices, and the *Multiple Linear Regression* model for multivariate series, using both price and volume data.

Table 27 shows the results obtained with the *LSTM* neural network, distinguishing between *univariate LSTM*, using only closing prices, and *multivariate LSTM*, using both price and volume data.

Small values of the MAPE and rRMSE evaluation metrics suggest accurate predictions and good performance of the considered model.

From the analysis of the series in their totality, it appears that linear models outperform neural networks. However, for both models, most best results are obtained for a *lag* of 1, thus confirming the hypothesis that the series are indistinguishable from a random walk.

To perform the time series forecasting, I also implemented a *Multi-Layer Perceptron* model. Since the LSTM network outperforms the MLP one, I decided to show only the LST} results. This is probably due to the architecture of the LSTM network, that can capture long-term dependencies in a sequence.

It should be noted that better predictions are obtained for stock market series rather than for the cryptocurrencies one. In particular, the best result is obtained for Microsoft series, with a MAPE of 0,011 and k_p equal to 1. This is probably due to the high price fluctuations that Bitcoin and the other cryptocurrencies have suffered during the investigated time interval. This is confirmed by the statistics shown in Table 24.

It must be noted that the addition of the *volume* feature to the dataset does not improve the predictions.

Table 26 Linear and Multiple Linear Regression results.

	Linear Regression			**Multiple Linear Regression**			
Series	**MAPE**	**rRMSE**	**k_p**	**MAPE**	**rRMSE**	**k_p**	**k_v**
BTC	0,026	0,040	1	0,026	0,037	1	1
ETH	0,031	0,049	1	0,039	0,053	6	3
LTC	0,034	0,050	1	0,045	0,058	2	2
MSFT	0,011	0,015	1	0,011	0,015	1	1
INTC	0,013	0,017	1	0,013	0,017	1	1
NKSH	0,014	0,019	12	0,013	0,018	7	5

Table 27 Univariate and Multivariate LSTM results.

	Univariate LSTM			**Multivariate LSTM**			
Series	**MAPE**	**rRMSE**	**k_p**	**MAPE**	**rRMSE**	**k_p**	**k_v**
BTC	0,027	0,041	1	0,038	0,048	2	1
ETH	0,034	0,052	6	0,057	0,076	2	1
LTC	0,035	0,051	1	0,039	0,054	1	1
MSFT	0,012	0,015	1	0,012	0,015	1	2
INTC	0,013	0,017	2	0,013	0,017	1	1
NKSH	0,014	0,020	7	0,013	0,018	1	2

To perform price forecast I changed the approach and decided to split the time series analysis using shorter time windows of 200 points, shifting the windows by 120 points, with the aim of finding local time regimes where the series do not follow the global random walk pattern.

Figure 43 and Figure 44 show the results obtained with the approach of partitioning the series into shorter sequences. Let us label the moving step forward with h. Particularly, in Figure 43 are presented the results obtained using the *Linear Regression* algorithm for univariate series forecast, using only closing prices, and the *Multiple Linear Regression* model for multivariate series, using both price and volume data. This approach, has the advantage of being simple to implement and requires low computational complexity. Nevertheless, has led to good results, similar to those present in the literature, if not better as in the Microsoft, Bitcoin and National Bankshares cases, where the MAPE error is lower that 1%.

Figure 44 shows the results obtained with the *LSTM* neural network, distinguishing between *univariate LSTM*, using only closing prices, and *multivariate LSTM*, using both price and volume data. For each time regimes I show the best results obtained on a specific time window defined by the k_p and k_v values reported in Figure 43 and Figure 44. Note that I highlighted the best results in bold. In particular, it is worth noting that introducing the time regimes, the best result is obtained for the Bitcoin time series, outperforming also the financial ones.

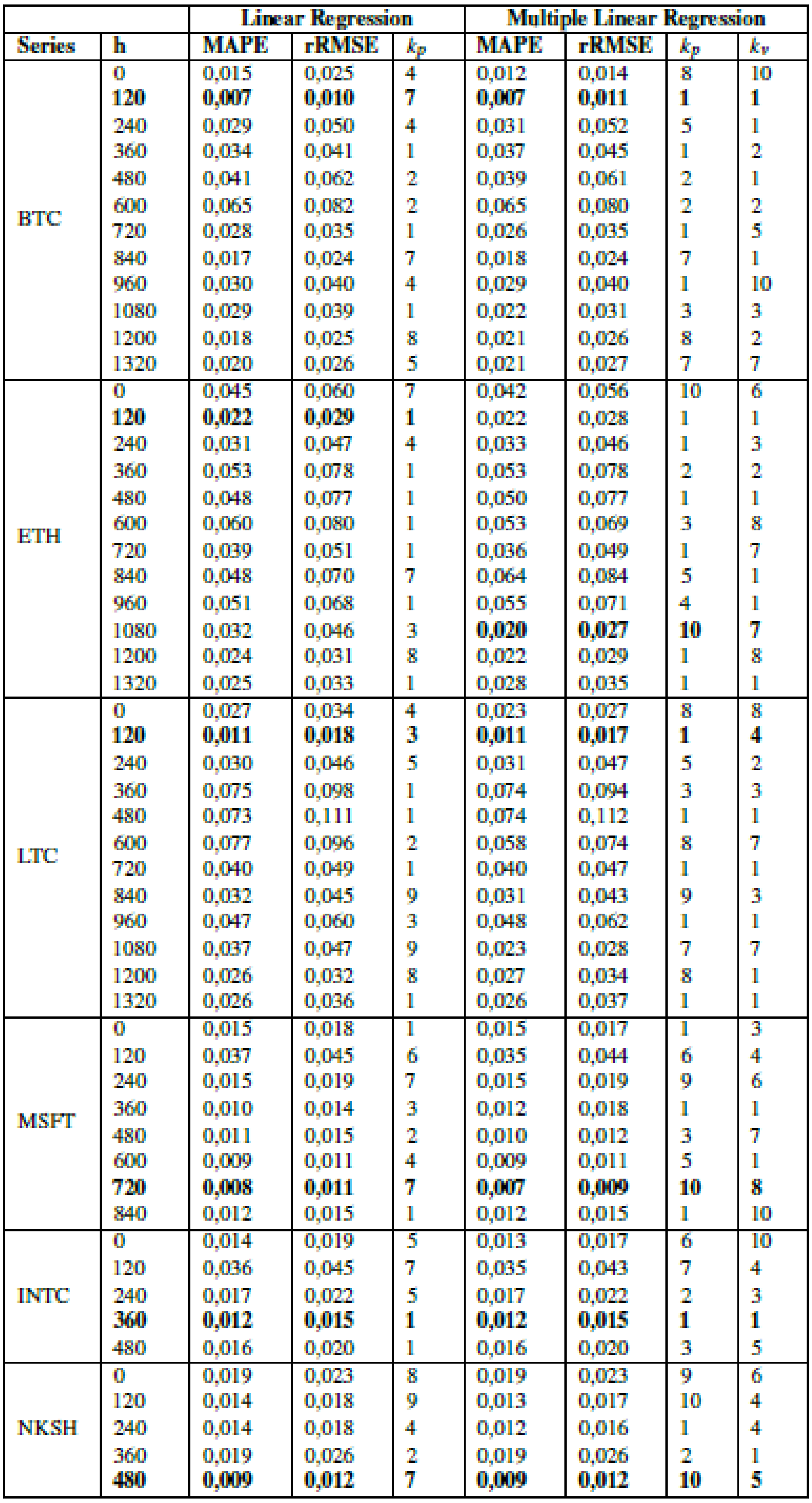
These results show how such innovative partitioning approach allowed us to avoid the ''random walk problem'', finding that best results are obtained using more than one previous price. Furthermore, this method leads to a significant improvement in predictions. It is worth noting that, from this analysis the best result arises from the Bitcoin series, with a MAPE error of 0,007, a temporal window k_p of 7 and a translation step h of 120, obtained using both regression models and LSTM network.

Another interesting consideration that arises from the results is that, as stated previously in the analysis of the series in their entirety, the linear regression models generally outperform the neural networks ones, while in the short-time regimes approach the different models yielded to similar results.

For direct feedback, I report in Table 28 the best results obtained in the papers I compared to and my best ones. In the event that the best MAPE error results from different models, I consider the model whose computational complexity is the least as best. It is noticeable that my results outperform those obtained in the benchmark papers, providing notable contribution to the literature.

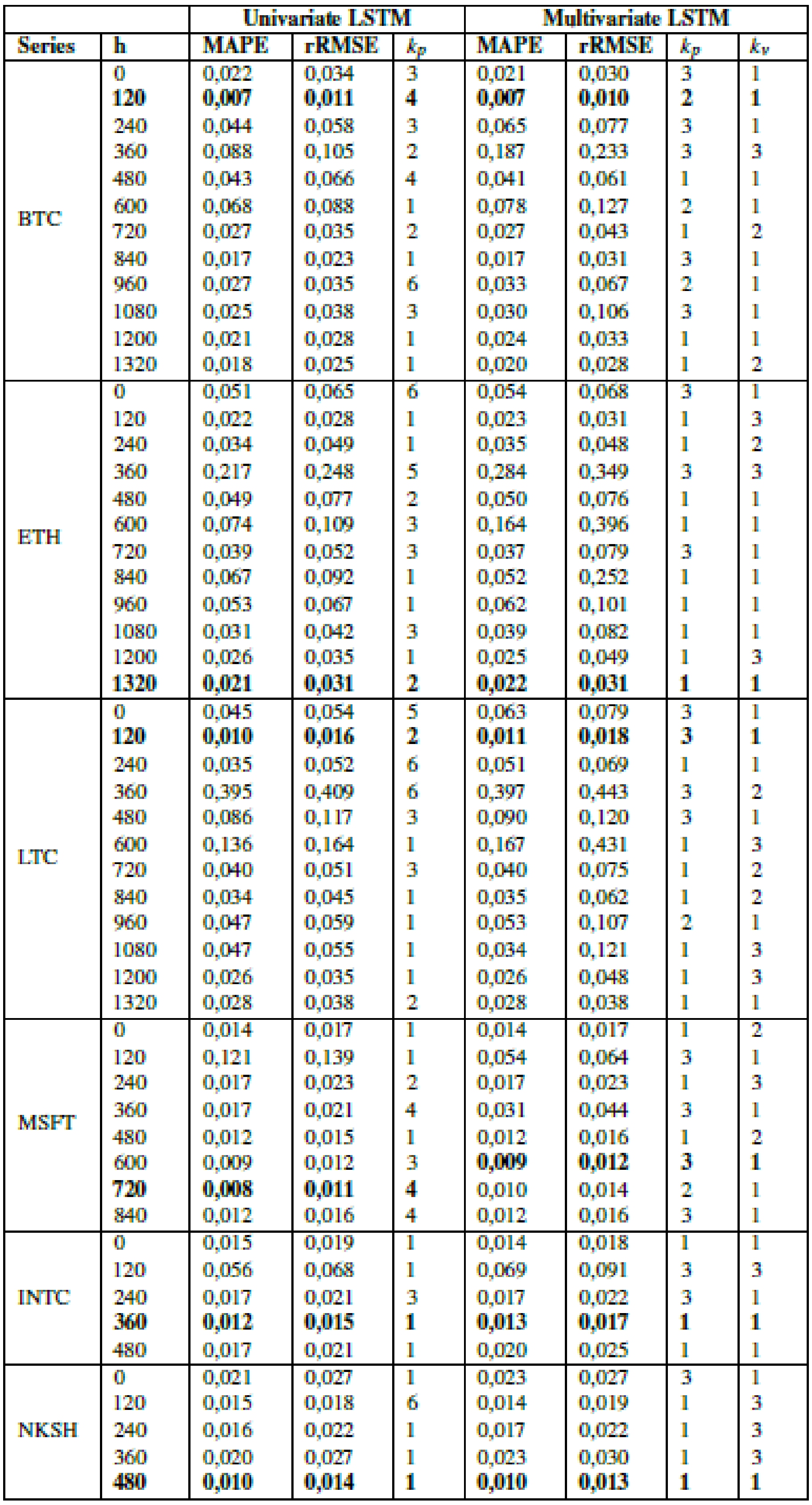
Table 28 Best Benchmarks Results compared to ours.

Reference	Series	Model	MAPE
[217]	BTC	SVM:0.9-1(Relief)	0,011
[219]	S&P BSE SENSEX	SVR	0,009
[218]	MSFT	SVR-CFA	0,052
	INTC	SVR-CFA	0,045
	NKSH	SVR-CFA	0,046
Our Work	BTC	LR	0,007
	ETH	MLR	0,020
	LTC	Univariate LSTM	0,010
	MSFT	MLR	0,007
	INTC	LR	0,012
	NKSH	LR	0,009



		Linear Regression			Multiple Linear Regression			
Series	h	MAPE	rRMSE	k_p	MAPE	rRMSE	k_p	k_v
	0	0,015	0,025	4	0,012	0,014	8	10
	120	**0,007**	**0,010**	**7**	**0,007**	**0,011**	**1**	**1**
	240	0,029	0,050	4	0,031	0,052	5	1
	360	0,034	0,041	1	0,037	0,045	1	2
	480	0,041	0,062	2	0,039	0,061	2	1
BTC	600	0,065	0,082	2	0,065	0,080	2	2
	720	0,028	0,035	1	0,026	0,035	1	5
	840	0,017	0,024	7	0,018	0,024	7	1
	960	0,030	0,040	4	0,029	0,040	1	10
	1080	0,029	0,039	1	0,022	0,031	3	3
	1200	0,018	0,025	8	0,021	0,026	8	2
	1320	0,020	0,026	5	0,021	0,027	7	7
	0	0,045	0,060	7	0,042	0,056	10	6
	120	**0,022**	**0,029**	**1**	0,022	0,028	1	1
	240	0,031	0,047	4	0,033	0,046	1	3
	360	0,053	0,078	1	0,053	0,078	2	2
	480	0,048	0,077	1	0,050	0,077	1	1
ETH	600	0,060	0,080	1	0,053	0,069	3	8
	720	0,039	0,051	1	0,036	0,049	1	7
	840	0,048	0,070	7	0,064	0,084	5	1
	960	0,051	0,068	1	0,055	0,071	4	1
	1080	0,032	0,046	3	**0,020**	**0,027**	**10**	**7**
	1200	0,024	0,031	8	0,022	0,029	1	8
	1320	0,025	0,033	1	0,028	0,035	1	1
	0	0,027	0,034	4	0,023	0,027	8	8
	120	**0,011**	**0,018**	**3**	**0,011**	**0,017**	**1**	**4**
	240	0,030	0,046	5	0,031	0,047	5	2
	360	0,075	0,098	1	0,074	0,094	3	3
	480	0,073	0,111	1	0,074	0,112	1	1
LTC	600	0,077	0,096	2	0,058	0,074	8	7
	720	0,040	0,049	1	0,040	0,047	1	1
	840	0,032	0,045	9	0,031	0,043	9	3
	960	0,047	0,060	3	0,048	0,062	1	1
	1080	0,037	0,047	9	0,023	0,028	7	7
	1200	0,026	0,032	8	0,027	0,034	8	1
	1320	0,026	0,036	1	0,026	0,037	1	1
	0	0,015	0,018	1	0,015	0,017	1	3
	120	0,037	0,045	6	0,035	0,044	6	4
	240	0,015	0,019	7	0,015	0,019	9	6
MSFT	360	0,010	0,014	3	0,012	0,018	1	1
	480	0,011	0,015	2	0,010	0,012	3	7
	600	0,009	0,011	4	0,009	0,011	5	1
	720	**0,008**	**0,011**	**7**	**0,007**	**0,009**	**10**	**8**
	840	0,012	0,015	1	0,012	0,015	1	10
	0	0,014	0,019	5	0,013	0,017	6	10
	120	0,036	0,045	7	0,035	0,043	7	4
INTC	240	0,017	0,022	5	0,017	0,022	2	3
	360	**0,012**	**0,015**	**1**	**0,012**	**0,015**	**1**	**1**
	480	0,016	0,020	1	0,016	0,020	3	5
	0	0,019	0,023	8	0,019	0,023	9	6
	120	0,014	0,018	9	0,013	0,017	10	4
NKSH	240	0,014	0,018	4	0,012	0,016	1	4
	360	0,019	0,026	2	0,019	0,026	2	1
	480	**0,009**	**0,012**	**7**	**0,009**	**0,012**	**10**	**5**

Figure 43 LR and MLR results with time regimes.



		Univariate LSTM			Multivariate LSTM			
Series	h	MAPE	rRMSE	k_p	MAPE	rRMSE	k_p	k_v
BTC	0	0,022	0,034	3	0,021	0,030	3	1
	120	**0,007**	**0,011**	**4**	**0,007**	**0,010**	**2**	**1**
	240	0,044	0,058	3	0,065	0,077	3	1
	360	0,088	0,105	2	0,187	0,233	3	3
	480	0,043	0,066	4	0,041	0,061	1	1
	600	0,068	0,088	1	0,078	0,127	2	1
	720	0,027	0,035	2	0,027	0,043	1	2
	840	0,017	0,023	1	0,017	0,031	3	1
	960	0,027	0,035	6	0,033	0,067	2	1
	1080	0,025	0,038	3	0,030	0,106	3	1
	1200	0,021	0,028	1	0,024	0,033	1	1
	1320	0,018	0,025	1	0,020	0,028	1	2
ETH	0	0,051	0,065	6	0,054	0,068	3	1
	120	0,022	0,028	1	0,023	0,031	1	3
	240	0,034	0,049	1	0,035	0,048	1	2
	360	0,217	0,248	5	0,284	0,349	3	3
	480	0,049	0,077	2	0,050	0,076	1	1
	600	0,074	0,109	3	0,164	0,396	1	1
	720	0,039	0,052	3	0,037	0,079	3	1
	840	0,067	0,092	1	0,052	0,252	1	1
	960	0,053	0,067	1	0,062	0,101	1	1
	1080	0,031	0,042	3	0,039	0,082	1	1
	1200	0,026	0,035	1	0,025	0,049	1	3
	1320	**0,021**	**0,031**	**2**	**0,022**	**0,031**	**1**	**1**
LTC	0	0,045	0,054	5	0,063	0,079	3	1
	120	**0,010**	**0,016**	**2**	**0,011**	**0,018**	**3**	**1**
	240	0,035	0,052	6	0,051	0,069	1	1
	360	0,395	0,409	6	0,397	0,443	3	2
	480	0,086	0,117	3	0,090	0,120	3	1
	600	0,136	0,164	1	0,167	0,431	1	3
	720	0,040	0,051	3	0,040	0,075	1	2
	840	0,034	0,045	1	0,035	0,062	1	2
	960	0,047	0,059	1	0,053	0,107	2	1
	1080	0,047	0,055	1	0,034	0,121	1	3
	1200	0,026	0,035	1	0,026	0,048	1	3
	1320	0,028	0,038	2	0,028	0,038	1	1
MSFT	0	0,014	0,017	1	0,014	0,017	1	2
	120	0,121	0,139	1	0,054	0,064	3	1
	240	0,017	0,023	2	0,017	0,023	1	3
	360	0,017	0,021	4	0,031	0,044	3	1
	480	0,012	0,015	1	0,012	0,016	1	2
	600	0,009	0,012	3	**0,009**	**0,012**	**3**	**1**
	720	**0,008**	**0,011**	**4**	0,010	0,014	2	1
	840	0,012	0,016	4	0,012	0,016	3	1
INTC	0	0,015	0,019	1	0,014	0,018	1	1
	120	0,056	0,068	1	0,069	0,091	3	3
	240	0,017	0,021	3	0,017	0,022	3	1
	360	**0,012**	**0,015**	**1**	**0,013**	**0,017**	**1**	**1**
	480	0,017	0,021	1	0,020	0,025	1	1
NKSH	0	0,021	0,027	1	0,023	0,027	3	1
	120	0,015	0,018	6	0,014	0,019	1	3
	240	0,016	0,022	1	0,017	0,022	1	3
	360	0,020	0,027	1	0,023	0,030	1	3
	480	**0,010**	**0,014**	**1**	**0,010**	**0,013**	**1**	**1**

Figure 44 Univariate and Multivariate LSTM results with time regimes.

7.5 Conclusions

The results, obtained considering the series in their totality, reflect the considerations made in the introduction. The predictions of the Bitcoin, Ethereum and Litecoin closing price series are worse, in terms of MAPE error, than those obtained for the benchmark series (Intel, Microsoft and National Bankshares). This is probably due to at least two reasons: high volatility of the prices and market immaturity for cryptocurrencies. This is confirmed by the statistics reported in Table 24 and Figure 42.

The results obtained partitioning the dataset into shorter sequences also confirmed the correctness of my hypothesis of identifying time regimes that do not resemble a random walk and that are easier to model, finding that best results are obtained using more than one previous price. It is worth noting that, with this novel approach, I obtained the best results for the Bitcoin price series rather than for the stock market series, as happened in the analysis of the series in their totality. As stated before, this is probably due to the high volatility of the Bitcoin price. In fact, it is no accident that the best result was found for the time regime identified by a translation step h of 120, where the Bitcoin prices are more distributed around the mean, showing a lower variance. This is confirmed by the standard deviation values shown in Figure 42.

It is important to emphasize that the innovative approach proposed here, namely the identification of short-time regimes within the entire series, allowed us to obtain leading-edge results in the field of financial series forecasting.

Comparing the best result with those obtained in the considered benchmark papers, my result represents one of the best found in the literature. I highlight that I obtained, both for the Bitcoin and the traditional market series, better results than the benchmark ones. Precisely, for Bitcoin I obtained a MAPE error of 0,007, while the benchmark best one [217] is 0,011.

For the stock market series my algorithms outperform those of benchmarks even more. In fact, my errors are as low as between 15% and 30% with respect to the reference errors reported in the literature.

Also for the Ethereum and Litecoin time series, the best results are those obtained with the time regimes approach, with a MAPE of 2% and 1% respectively.

As regards the implemented algorithms, the best results were found with both *regression models* and *LSTM* network. However, from the point of view of execution speed, the linear regression models outperform neural networks.

It is worth noting that, since Bitcoin and the other cryptocurrencies still are at an early stage, the length of the time series is limited, and future investigation might yield different results.

8 References

[1] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf. [Accessed January 2022].

[2] R. G. Brown, J. Carlyle, I. Grigg e M. Hearn, «Corda: An introduction,» 2016.

[3] A. Churyumov, «Byteball: a decentralized system for transfer of value,» 2016.

[4] G. Wood, «Ethereum: A secure decentralised generalised transaction ledger,» 2014.

[5] T. Chen, X. Li, X. Luo e X. Zhang, «Under-Optimized Smart Contracts Devour Your Money,» in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017.

[6] "Ethereum Network Status," [Online]. Available: https://ethstats.net. [Accessed January 2022].

[7] Z. Zheng, S. Xie, H.-N. Dai, X. Chen e H. Wang, «Blockchain challenges and opportunities: a survey,» *International Journal of Web and Grid Services ,* vol. 14, p. 352–375, 2018.

[8] S. Tikhomirov, «Ethereum: state of knowledge and research perspectives,» in *International Symposium on Foundations and Practice of Security*, 2017.

[9] "State of the dapps," 2019. [Online]. Available: https://www.stateofthedapps.com/stats. [Accessed 2022 January].

[10] C. Dannen, Introducing Ethereum and Solidity, Springer, 2017.

[11] L. Marchesi, M. Marchesi e R. Tonelli, «Abcde–agile block chain dapp engineering,» *Blockchain: Research and Applications,* vol. 1, p. 1000002, 2020.

[12] L. Marchesi, M. Marchesi, G. Destefanis, G. Barabino e D. Tigano, «Design Patterns for gas optimization in ethereum,» in *IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE 2020)*, 2020.

[13] L. Marchesi, M. Marchesi, L. Pompianu e R. Tonelli, «Security Pattern for Ethereum and Solidity,» [Online]. Available: http://tiny.cc/security_checklist.

[14] M. I. Lunesu, R. Tonelli, L. Marchesi e M. Marchesi, «Assessing the Risk of Software Development in Agile Methodologies Using Simulation,» *IEEE Access,* vol. 9, pp. 134240-134258, 2021.

[15] G. Fenu, L. Marchesi, M. Marchesi e R. Tonelli, «The ico phenomenon and its relationships with ethereum smart contract environment,» in *International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018.

[16] K. Beck, M. Beedle, A. V. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries e e. al., «Manifesto for agile software development,» 2001.

[17] K. Schwaber e M. Beedle, «Agile Software Development with Scrum,» Pearson, 2001.

[18] M. Cohn, User Stories Applied: For Agile Software Development, Addison-Wesley Professional, 2004.

[19] D. Janzen e H. Saiedian, «Test-driven development concepts, taxonomy, and future direction,» *Computer,* vol. 38, n. 9, p. 43–50, 2005.

[20] "Truffle website," 2019. [Online]. Available: https://www.trufflesuite.com/. [Accessed January 2022].

[21] M. Fowler, Refactoring: Improving the Design of Existing Code (2nd Edition), Addison-Wesley Professional, 2018.

[22] S. Porru, A. Pinna, M. Marchesi e R. Tonelli, «Blockchain-oriented software engineering: challenges and new directions,» in *39th International Conference on Software Engineering Companion*, 2017.

[23] J. Rumbaugh, G. Booch e I. Jacobson, The unified modeling language reference manual, Addison Wesley, 2017.

[24] X. Xu, I. Weber e M. Staples, Architecture for Blockchain Applications, Springer, 2019.

[25] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso e P. Rimba, «A taxonomy of blockchain-based systems for architecture design,» in *IEEE International Conference on Software Architecture (ICSA)*, 2017.

[26] F. Wessling, C. Ehmke, M. Hesenius e V. Gruhn, «How much blockchain do you need? towards a concept for building hybrid dapp architectures,» in *1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB 2018)*, 2018.

[27] G. Fridgen, J. Lockl, S. Radszuwill, A. Rieger, A. Schweizer e N. Urbach, «A solution in search of a problem: A method for the development of blockchain use cases,» in *24th Americas Conference on Information Systems (AMCIS)*, New Orleans, USA, 2018.

[28] M. Jurgelaitis, V. Drungilas, L. Ceponiene, R. Butkiene e E. Vaiciukynas, «Modelling principles for blockchain-based implementation of business or scientific processes,» in *International Conference on Information Technologies (IVUS 2019)*, Kaunas, Lithuania, 2019.

[29] M. Beller e J. Hejderup, «Blockchain-based software engineering,» in *41th International Conference on Software Engineering Companion*, 2019.

[30] V. Lenarduzzi, I. Lunesu, M. Marchesi e R. Tonelli, «Blockchain applications for agile methodologies,» in *19th International Conference on Agile Software Development: XP 2018 Companion*, New York, USA, 2018.

[31] P. Chakraborty, R. Shahriyar, A. Iqbal e A. Bosu, «Understanding the software development practices of blockchain projects: A survey,» in *12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2018)*, Oulu, Finland, 2018.

[32] A. Bosu, A. Iqbal, R. Shahriyar e P. Chakraborty, «Understanding the motivations, challenges and needs of blockchain software developers: a survey,» *Empirical Software Engineering,* n. 24, p. 2636–2673, 2019.

[33] P. Praitheeshan, L. Pan, J. Yu, J. Liu and R. Doss, "Security analysis methods on ethereum smart contract vulnerabilities: A survey," *arXiv preprint: 1908.08605 ,* 2019.

[34] Y. Huang, Y. Bian, R. Li, L. Zhao e P. Shi, «Smart contract security: A software lifecycle perspective,» *IEEE Access,* vol. 7, 2019.

[35] B. D. Win, R. Scandariato, K. Buyens, J. Grégoire e W. Joosen, «On the secure software development process: Clasp, sdl and touchpoints compared,» *Information and Software Technology,* vol. 51, p. 1152–1171, 2009.

[36] K. Rindell, S. Hyrynsalmi e V. Leppänen, «Busting a myth: Review of agile security engineering methods,» in *12th International Conference on Availability, Reliability and Security*, 2017.

[37] H. Baumeister, N. Koch e L. Mandel, «Towards a uml extension for hypermedia design,» in *International Conference on the Unified Modeling Language*, 1999.

[38] L. Baresi, F. Garzotto e P. Paolini, «Extending uml for modeling web applications,» in *34th Annual Hawaii International Conference on System Sciences*, 2001.

[39] H. Rocha e S. Ducasse, «Preliminary Steps Towards Modeling Blockchain-oriented Software,» in *1th Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB 2018)*, Gotheborg, Sweden, 2018.

[40] KPMG, «Agile transformation, Survey on agility,» 2019.

[41] D. J. Anderson, Kanban: successful evolutionary change for your technology business, Blue Hole Press, 2010.

[42] P. Coad e E. Yourdon, Object-oriented analysis, vol. 2, Englewood Cliffs, NJ: Yourdon press , 1991.

[43] B. Scriber, «A framework for determining blockchain applicability,» *IEEE Software,* vol. 35, p. 70–77, 2018.

[44] L. Marchesi, M. Marchesi, L. Pompianu e R. Tonelli, «Security checklists for ethereum smart contract development: patterns and best practices,» *arXiv preprint,* 2020.

[45] L. L. Constantine e L. A. Lockwood, Software for use: a practical guide to the models and methods of usage-centered design, Pearson Education, 1999.

[46] H. Sharp, Y. Rogers e J. Preece, Interaction design: beyond human computer interaction, 5th edition, John Wiley & Sons, 2019.

[47] "Solidity website," 2019. [Online]. Available: https://solidity.readthedocs.io. [Accessed January 2022].

[48] M. Bartoletti e L. Pompianu, «An empirical analysis of smart contracts: platforms, applications, and design patterns,» in *Financial Cryptography and Data Security (FC 2017)*, Malta, 2017.

[49] M. Wohrer e U. Zdun, «Smart contracts: Security patterns in the Ethereum ecosystem and solidity,» in *International Workshop on Blockchain Oriented Software Engineering (IWBOSE 2018)*, 2018.

[50] X. XU, C. PAUTASSO, L. ZHU, Q. LU e I. WEBER, «A pattern collection for blockchain-based applications,» in *23rd European Conference on Pattern Languages of Programs*, 2018.

[51] P. Zhang, J. White, D. C. Schmidt e G. Lenz, «Applying software patterns to address interoperability in blockchain-based healthcare apps.,» *arXiv preprint: 1706.03700,* 2017.

[52] Y. Liu, Q. Lu, X. Xu, L. Zhu e H. Yao, «Applying design patterns in smart contracts,» in *International Conference on Blockchain*, 2018.

[53] J. Liu e Z. Liu, «A survey on security verification of blockchain smart contracts,» *IEEE Access,* vol. 7, 2019.

[54] K. Anton, J. Manico and J. Bird, "Owasp proactive controls for developers," Open Web Application Security Project (OWASP) , 2018.

[55] "Consensys solidity best practices website," 2019. [Online]. Available: https://consensys.github.io/smart-contract-best-practices/. [Accessed January 2022].

[56] "Proxy patterns," 2019. [Online]. Available: https://blog.openzeppelin.com/proxy-patterns/. [Accessed January 2022].

[57] "Ethereum smart contract security best practices," 2019. [Online]. Available: https://ethereum-contract-security-techniques-and-tips.readthedocs.io/en/latest/. [Accessed January 2022].

[58] Gamma, Helm, Johnson e Vlissides, «Design Patterns: Elements of Reusable Object-Oriented Software,» Addison-Wesley, 1995.

[59] A. S. Podda e L. Pompianu, «An overview of blockchain-based systems and smart contracts for digital coupons,» in *IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020.

[60] N. Atzei, M. Bartoletti e T. Cimoli, «A survey of attacks on Ethereum smart contracts,» *Cryptology ePrint Archive,* p. Report 2016/1007, 2016.

[61] A. Mense e M. Flatscher, «Security vulnerabilities in Ethereum smart contracts,» in *20th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2018)*, New York, NY, USA, 2018.

[62] "Embark website," 2020. [Online]. Available: https://framework.embarklabs.io/. [Accessed January 2022].

[63] "Etherlime website," 2020. [Online]. Available: https://etherlime.gitbook.io/etherlime/. [Accessed January 2022].

[64] "Ropsten website," 2020. [Online]. Available: https://ropsten.etherscan.io/. [Accessed January 2022].

[65] "Rinkeby website," 2020. [Online]. Available: https://www.rinkeby.io. [Accessed January 2022].

[66] "Goerli website," 2020. [Online]. Available: https://github.com/goerli/testnet. [Accessed January 2022].

[67] "Ganache website," 2020. [Online]. Available: https://www.trufflesuite.com/ganache. [Accessed January 2022].

[68] Eattheblocks, "How to optimize gas cost in a Solidity smart contract? 6 tips," 2019. [Online]. Available: https://eattheblocks.com/how-to-optimize-gas-cost-in-a-solidity-smart-contract-6-tips/. [Accessed January 2022].

[69] M. Gupta, "Mudit Gupta’s Blog," 2018. [Online]. Available: https://mudit.blog/solidity-gas-optimization-tips/.

[70] M. Gupta, «Solidity tips and tricks to save gas and reduce bytecode size,» 2019. [Online]. Available: https://blog.polymath.network/solidity-tips-and-tricks-to-save-gas-and-reduce-bytecode-size-c44580b218e6. [Consultato il giorno January 2022].

[71] W. Shahda, "Gas Optimization in Solidity Part I: Variables," 2019. [Online]. Available: https://medium.com/coinmonks/gas-optimization-in-solidity-part-i-variables-9d5775e43dde. [Accessed January 2022].

[72] L. Marchesi, M. Marchesi e R. Tonelli, «An agile software engineering method to design blockchain applications,» in *Software Engineering Conference Russia (SECR 2018)*, Moscow, Russia, 2018.

[73] G. Baralla, A. Pinna e G. Corrias, «Ensure traceability in european food supply chain by using a blockchain system,» in *2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB 2019)*, 2019.

[74] W. Warren e A. Bandeali, «0x: An open protocol for decentralized exchange on the ethereum blockchain,» 2017.

[75] "Solidity patterns," 2019. [Online]. Available: https://fravoll.github.io/solidity-patterns/. [Accessed January 2022].

[76] R. Conradi e A. I. Wang, Empirical Methods and Studies in Software Engineering, vol. 2765 of LNCS, Springer Berlin Heidelberg, 2003.

[77] W. W. Lowrance, Of Acceptable Risk: Science and the Determination of Safety, Kaufmann Inc., 1976.

[78] C. Chittister e Y. Y. Haimes, «Assessment and management of software technical risk,» *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 24, n. 2, pp. 187-202, 1994.

[79] C. Alberts e A. J. Dorofee, «Risk management framework,» Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2010.

[80] P. M. Institute, A Guide to the Project Management Body of Knowledge (PMBOK Guide), vol. 2, Project Management Institute, 2017.

[81] L. Wallace, M. Keil e A. Rai, «How software project risk affects project performance: An investigation of the dimensions of risk and an exploratory model,» *Decision Sciences,* vol. 35, n. 2, pp. 289-321, 2004.

[82] A. Albadarneh, I. Albadarneh e A. Qusef, «Risk management in agile software development: A comparative study,» in *IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT 2015)*, 2015.

[83] J. Nyfjord e M. Kajko-Mattsson, «Commonalities in risk management and agile process models,» in *International Conference on Software Engineering Advances (ICSEA 2007)*, 2007.

[84] L. Siddique e B. A. Hussein, «Practical insight about risk management process in agile software projects in norway,» in *IEEE International Technology Management Conference*, 2014.

[85] D. Hilˇcíková, M. Dohnanská, D. Lajˇcin e G. Gabrhelová, «Agile project management as one of the critical success factors in project risk management in machinery industry in slovakiaAgile project management as one of the critical success factors in project risk management in machinery industry in slovakia,» *International Journal of Economics and Financial Issues,* vol. 1, pp. 37-54, 2020.

[86] S. Y. Chadli, A. Idri, J. N. Ros, J. L. Fernández-Alemán, J. M. C. d. Gea e a. A. Toval, «Software project management tools in global software development: a systematic mapping study,» *SpringerPlus,* vol. 5, n. 1, pp. 1-38, 2016.

[87] B. G. Tavares, C. E. S. d. Silva e A. D. d. Souza, «Risk management analysis in scrum software projects,» *International Transactions in Operational Research,* vol. 26, n. 5, pp. 1884-1905, 2019.

[88] S. d. S. Lopes, R. C. G. d. Souza, A. d. G. Contessoto, A. L. d. Oliveira e R. T. V. Braga, «A risk management framework for scrum projects,» in *23rd International Conference on Enterprise Information Systems (ICEIS 2021)*, 2021.

[89] T. Alencar, M. I. Cortés, N. L. Veras e L. Magno, «A proactive approach to support risk management in software projects using multi-agent systems,» in *20th International Conference on Enterprise Information Systems (ICEIS 2018)*, 2018.

[90] A. Mishra e D. Mishra, «Software project management tools: a brief comparative view,» *Software Engineering Notes,* vol. 38, n. 3, pp. 1-4, 2013.

[91] H. M. Sarkan, T. P. S. Ahmad e A. A. Bakar, «Using jira and redmine in requirement development for agile methodology,» in *2011 Malaysian Conference in Software Engineering*, 2011.

[92] D. Pfahl, «Prosim/ra—software process simulation in support of risk assessment,» *Value-based software engineering,* p. 263–286, 2006.

[93] J. García-García, J. Enríquez, M. Ruiz, C. Arévalo e A. J.-. Ramírez, «Software process simulation modeling: Systematic literature review,» *Computer Standards Interfaces,* vol. 70, 2020.

[94] B. Verma, M. Dhanda, B. Verma e M. Dhanda, «A review on risk management in software projects,» *International Journal for Innovative Research in Science and Technology,* vol. 2, pp. 499-503, 2016.

[95] Atlassian, "Jira," [Online]. Available: https://www.atlassian.com/software/jira. [Accessed January 2022].

[96] C. Lopez e J. L. Salmeron, «Dynamic risks modelling in erp maintenance projects with fcm,» *Information Sciences,* vol. 256, pp. 25-45, 2014.

[97] R. N. Charette, «Why software fails [software failure].,» *Ieee Spectrum,* vol. 42, n. 9, pp. 42-49, 2005.

[98] B. W. Boehm, «Software risk management: principles and practices,» *IEEE software,* vol. 8, n. 1, pp. 32-41, 1991.

[99] F. M. Dedolph, «The neglected management activity: Software risk management,» *Bell Labs Technical Journal,* vol. 8, n. 3, pp. 91-95, 2003.

[100] C. R. Pandian, Applied software risk management: A guide for software project managers, Auerbach Publications, 2006.

[101] M. d. O. Barros, C. M. L. Werner e G. H. Travassos, «Supporting risks in software project management,» *Journal of Systems and Software,* vol. 70, n. 1-2, pp. 21-35, 2004.

[102] B. Shahzad e A. S. Al-Mudimigh, «Risk identification, mitigation and avoidance model for handling software risk,» in *2nd International Conference on Computational Intelligence, Communication Systems and Networks*, 2010.

[103] L. Xiaosong, L. Shushi, C. Wenjun e F. Songjiang, «The application of risk matrix to software project risk management,» in *International Forum on Information Technology and Applications*, 2009.

[104] B. Roy, R. Dasgupta e N. Chaki, «A study on software risk management strategies and mapping with sdlc,» *Advanced Computing and Systems for Security,* pp. 121-138, 2016.

[105] C. A. Thieme, A. Mosleh, I. B. Utne e J. Hegde, «Incorporating software failure in risk analysis - part 2: Risk modeling process and case study,» *Reliability Engineering & System Safety,* 2020.

[106] B. G. Tavares, M. Keil, C. E. S. d. Silva e A. D. d. Souza, «A risk management tool for agile software development,» *Journal of Computer Information Systems,* pp. 1-10, 2020.

[107] M. I. Kellner, R. J. Madachy e D. M. Raffo, «Software process simulation modeling: why? what? how?,» *Journal of Systems and Software,* vol. 46, n. 2, pp. 91-105, 1999.

[108] H. Zhang, B. Kitchenham e D. Pfahl, «Reflections on 10 years of software process simulation modeling: a systematic review,» in *International Conference on Software Process*, 2008.

[109] D. Liu, Q. Wang e J. Xiao, «The role of software process simulation modeling in software risk management: A systematic review,» in *3rd International Symposium on Empirical Software Engineering and Measurement (ESEM 2009)*, 2009.

[110] T. Baum, F. Kortum, K. Schneider, A. Brack e J. Schauder, «Comparing pre-commit reviews and post-commit reviews using process simulation,» *Journal of Software: Evolution and Process,* vol. 29, n. 11, 2017.

[111] B. Zhao, J. Cao, S. Jiang e Q. Qi, «An agent based simulation system for open source software development,» in *IEEE World Congress on Services (SERVICES 2020)*, 2020.

[112] M. Melis, I. Turnu, A. Cau e G. Concas, «Evaluating the impact of test-first programming and pair programming through software process simulation,» *Software Process: Improvement and Practice,* vol. 11, n. 4, pp. 345-360, 2006.

[113] D. Anderson, G. Concas, M. I. Lunesu e M. Marchesi, «Studying lean-kanban approach using software process simulation,» in *International Conference on agile software development*, 2011.

[114] D. J. Anderson, G. Concas, M. I. Lunesu, M. Marchesi e H. Zhang, «A comparative study of scrum and kanban approaches on a real case study using simulation,» *Agile Processes in Software Engineering and Extreme Programming,* pp. 23-137, 2012.

[115] R. Turner, R. Madachy, D. Ingold e J. A. Lane, «Modeling kanban processes in systems engineering,» in *International Conference on Software and System Process*, 2012.

[116] R. Turner, D. Ingold, J. A. Lane, R. Madachy e D. Anderson, «Effectiveness of kanban approaches in systems engineering within rapid response environments,» *Procedia Computer Science,* vol. 8, pp. 309-314, 2012.

[117] Z. Wang, «Modelling and simulation of scrum team strategies: A multiagent approach,» in *Computational Methods in Systems and Software*, 2020.

[118] N. Fenton, M. Neil, W. Marsh, P. Hearty, L. Radlinski e P. Krause, «On the effectiveness of early life cycle defect prediction with bayesian nets,» *Empirical Software Engineering,* vol. 13, pp. 499-537, 2008.

[119] D. L. T. Rodríguez, V. H. M. García e G. A. M. Giraldo, «Dynamic model to manage threats in software development projects through artificial intelligence techniques,» in *Workshop on Engineering Applications*, 2012.

[120] H. R. Joseph, «Software development risk management: Using machine learning for generating risk prompts,» in *37th IEEE International Conference on Software Engineering*, 2015.

[121] W.-M. Han, «Discriminating risky software project using neural networks,» *Computer Standards Interfaces,* vol. 40, pp. 15-22, 2015.

[122] W. Min, Z. Jun e Z. Wei, «The application of fuzzy comprehensive evaluation method in the software project risk assessment,» in *International Conference on Management Engineering, Software Engineering and Service Sciences (ICMSS 17)*, New York, NY, USA, 2017.

[123] T. E. Abioye, O. T. Arogundade, S. Misra, A. T. Akinwale e O. J. Adeniran, «Toward ontology based risk management framework for software projects: An empirical study,» *Journal of Software: Evolution and Process,* vol. 32, n. 12, 2020.

[124] M. Asif e J. Ahmed, «A novel case base reasoning and frequent pattern based decision support system for mitigating software risk factors,» *IEEE Access,* vol. 8, p. 102278–102291, 2020.

[125] K. Ghane, «Quantitative planning and risk management of agile software development,» in *IEEE Technology & Engineering Management Conference (TEMSCON 2017)*, 2017.

[126] V. Singh, V. Malik e R. Mittal, «Risk analysis in software cost estimation: A simulation-based approach,» *Turkish Journal of Computer and Mathematics Education ,* vol. 12, n. 6, p. 2176–2183, 2021.

[127] Z. S. H. Abad, M. Noaeen, D. Zowghi, B. H. Far e K. Barker, «Two sides of the same coin: Software developers' perceptions of task switching and task interruption,» in *22nd International Conference on Evaluation and Assessment in Software Engineering (EASE18)*, New York, NY, USA, 2018.

[128] A. Tregubov, B. Boehm, N. Rodchenko e J. A. Lane, «Impact of task switching and work interruptions on software development processes,» in *International*

Conference on Software and System Process (ICSSP 2017), New York, NY, USA, 2017.

[129] G. Concas, M. I. Lunesu, M. Marchesi e H. Zhang, «Simulation of software maintenance process, with and without a work-in-process limit,» *Journal of Software: Evolution and Process,* vol. 25, n. 12, pp. 1225-1248, 2013.

[130] W. R. Shadish, T. D. Cook e D. T. Campbell, «Experimental and Quasi-Experimental Designs for Generalized Causal Inference,» *Cengage Learning,* 2001.

[131] L. Marchesi, K. Mannaro and R. Porcu, "Automatic Generation of Blockchain Agri-food Traceability Systems," in *IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB 2021)*, 2021.

[132] P. Serra, G. Fancello, R. Tonelli e L. Marchesi, «Can the Blockchain Facilitate the Development of an Interport Community?,» in *International Conference on Computational Science and Its Applications*, 2021.

[133] M. E. Peck, «Blockchain world - do you need a blockchain? this chart will tell you if the technology can solve your problem,» *IEEE Spectrum ,* vol. 54, n. 10, pp. 38-60, 2017.

[134] K. Wust e A. Gervais, «Do you need a blockchain?,» in *Crypto Valley Conference on Blockchain Technology (CVCBT 2018)*, 2018.

[135] V. Hassija, S. Zeadally, I. Jain, A. Tahiliani, V. Chamola e S. Gupta, «Framework for determining the suitability of blockchain: Criteria and issues to consider,» *Transactions on Emerging Telecommunications Technologies ,* 2021.

[136] B. Koteska, E. Karafiloski e A. Mishev, «Blockchain implementation quality challenges: A literature review,» in *6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2017)*, 2017.

[137] S. Maranhao, J. M. Seigneur e R. Hu, «Towards a standard to assess blockchain & other dlt platforms,» ITU, 2019.

[138] S. N. G. Gourisetti, M. Mylrea e H. Patangia, «Evaluation and demonstration of blockchain applicability framework,» *IEEE Transactions on Engineering Management ,* vol. 67, n. 4, p. 1142–1156, 2020.

[139] M. Garriga, S. D. Palma, M. Arias, A. D. Renzis, R. Pareschi e D. A. Tamburri, «Blockchain and cryptocurrencies: A classification and comparison of architecture drivers,» *Concurrency and Computation: Practice and Experience ,* vol. 33, n. 8, 2021.

[140] M. Woehrer e U. Zdun, «Architectural design decisions for blockchain-based applications,» in *3rd IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2021)*, 2021.

[141] S. S. Panda, B. K. Mohanta, U. Satapathy, D. Jena, D. Gountia e T. K. Patra, «Study of blockchain based decentralized consensus algorithms,» in *IEEE Region 10 Conference (TENCON 2019)*, 2019.

[142] M. Castro e B. Liskov, «Practical byzantine fault tolerance,» in *3rd Symposium on Operating systems design and implementation*, 1999.

[143] L. Gerrits, C. N. Samuel, R. Kromes, F. Verdier, S. Glock e P. Guitton-Ouhamou, «Experimental scalability study of consortium blockchains with bft consensus for iot automotive use case,» in *19th ACM Conference on Embedded Networked Sensor Systems, Association for Computing Machinery*, New York, NY, USA, 2021.

[144] G. Shapiro, C. Natoli e V. Gramoli, «The performance of byzantine fault tolerant blockchains,» in *IEEE 19th International Symposium on Network Computing and Applications (NCA 2020)*, 2020.

[145] A. Ahmad, M. Saad, J. Kim, D. Nyang e D. Mohaisen, «Performance evaluation of consensus protocols in blockchain-based audit systems,» in *International Conference on Information Networking (ICOIN 2021)*, 2021.

[146] C. N. Samuel, S. Glock, F. Verdier e P. Guitton-Ouhamou, «Choice of ethereum clients for private blockchain: Assessment from proof of authority perspective,» in *IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2021)*, 2021.

[147] Y. Wang, J. H. Han e P. Beynon-Davies, «Understanding blockchain technology for future supply chains: a systematic literature review and research agenda,» *Supply Chain Management: An International Journal,* 2019.

[148] Q. Lu e X. Xu, «Adaptable blockchain-based systems: A case study for product traceability,» *IEEE Software 34,* pp. 21-27, 2017.

[149] D. D. F. Maesa e P. Mori, «Blockchain 3.0 applications survey,» *Journal of Parallel and Distributed Computing ,* vol. 138, pp. 99-114, 2020.

[150] S. V. Akram, P. K. Malik, R. Singh, G. Anita e S. Tanwar, «Adoption of blockchain technology in various realms: Opportunities and challenges,» *Security and Privacy,* vol. 3, n. 5, 2020.

[151] E. J. D. Aguiar, B. S. Faical, B. Krishnamachari e J. Ueyama, «A survey of blockchain-based strategies for healthcare,» *ACM Computing Surveys,* vol. 53, n. 2, pp. 1-27, 2021.

[152] R. Colomo-Palacios, M. Sanchez-Gordon e D. Arias-Aranda, «A critical review on blockchain assessment initiatives: A technology evolution view point,» *Journal of Software: Evolution and Process ,* vol. 32, n. 11, 2020.

[153] J. Polge, J. Robert e Y. L. Traon, «Permissioned blockchain frameworks in the industry: A comparison,» *ICT Express,* vol. 7, n. 2, pp. 229-233, 2021.

[154] "Ethereum 2.0 website," 2021. [Online]. Available: https://ethereum.org/en/eth2/. [Accessed January 2022].

[155] "Hyperledger fabric," 2021. [Online]. Available: https://www.hyperledger.org/use/fabric. [Accessed January 2022].

[156] D. Li, W. E. Wong e J. Guo, «A survey on blockchain for enterprise using hyperledger fabric and composer,» in *6th International Conference on Dependable Systems and Their Applications (DSA)*, 2019.

[157] J. Sun, X. Yao, S. Wang e Y. Wu, «Blockchain-based secure storage and access scheme for electronic medical records in ipfs,» *IEEE Access ,* vol. 8, p. 59389–59401, 2020.

[158] "Regulation (eu) no 910/2014 of the european parliament and of the council," 2014.

[159] H. Huang, J. Lin, B. Zheng, Z. Zheng e J. Bian, «When blockchain meets distributed file systems: An overview, challenges, and open issues,» *IEEE Access,* vol. 8, p. 50574–50586, 2020.

[160] D. C. Nguyen, P. N. Pathirana, M. Ding e A. Seneviratne, «Integration of blockchain and cloud of things: Architecture, applications and challenges,» *IEEE Communications Surveys Tutorials,* vol. 22, n. 4, pp. 2521-2549, 2020.

[161] "Etherna blochain as a service," 2021. [Online]. Available: https://www.netservice.eu/en/productsand-solutions/etherna. [Accessed January 2022].

[162] T. Adisorn, L. Tholen e T. Gotz, «Towards a digital product passport fit for contributing to a circular economy,» *Energies,* vol. 14, n. 8, 2021.

[163] K. Demestichas, N. Peppes, T. Alexakis e E. Adamopoulou, «Blockchain in agriculture traceability systems: A review,» *Applied Sciences (Switzerland),* vol. 10, n. 12, pp. 1-22, 2020.

[164] C. Costa, F. Antonucci, F. Pallottino, J. Aguzzi, D. S.´a e P. Menesatti, «A review on agri-food supply chain traceability by means of rfid technology,» *Food and bioprocess technology,* vol. 6, n. 2, pp. 353-366, 2013.

[165] F. Tian, «An agri-food supply chain traceability system for china based on rfid & blockchain technology,» in *13th international conference on service systems and service management*, 2016.

[166] F. Antonucci, S. Figorilli, C. Costa, F. Pallottino, L. Raso e P. Menesatti, «A review on blockchain applications in the agri-food sector,» *Journal of the Science of Food and Agriculture,* vol. 99, n. 14, p. 6129–6138, 2019.

[167] P. Bottoni, N. Gessa, G. Massa, R. Pareschi, H. Selim e E. Arcuri, «Intelligent smart contracts for innovative supply chain management,» *Frontiers in Blockchain,* vol. 3, p. 52, 2020.

[168] L. Cocco e K. Mannaro, «Blockchain in agri-food traceability systems: a model proposal for a typical italian food product,» in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2021.

[169] L. Cocco, K. Mannaro, R. Tonelli, L. Mariani, M. B. Lodi, A. Melis, M. Simone e A. Fanti, «A blockchain-based traceability system inagri-food sme: Case study of a traditional bakery,» *IEEE Access,* vol. 9, p. 62899–62915, 2021.

[170] M. Alharby e A. V. Moorsel, «Blockchain-based smart contracts: A systematic mapping study,» *arXiv preprint:1710.06372,* 2017.

[171] W. Zou e e. al., «Smart Contract Development: Challenges and Opportunities,» *IEEE Transactions on Software Engineering ,* vol. 99, pp. 1-1, 2019.

[172] H. Rocha e S. Ducasse, «Preliminary steps towards modeling blockchain oriented software,» in *IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB 2018)*, 2018.

[173] A. B. Tran, Q. Lu e I. Weber, «Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management,» in *BPM (Dissertation/Demos/Industry)*, 2018.

[174] Q. Lu, A. B. Tran, I. Weber, H. O'Connor, P. Rimba, X. Xu, M. Staples, L. Zhu e R. Jeffery, «Integrated model-driven engineering of blockchain applications for business processes and asset management,» *Software: Practice and Experience,* 2020.

[175] C. K. Frantz e M. Nowostawski, «From institutions to code: Towards automated generation of smart contracts,» in *IEEE 1st International Workshops on Foundations and Applications of Self* Systems* , 2016.

[176] V. A. d. Sousa, C. Burnay e M. Snoeck, «B-merode: A model-driven engineering and artifact-centric approach to generate smart contracts,» in *Conference on Advanced Information Systems Engineering*, 2020.

[177] A. Mavridou e A. Laszka, «Designing secure ethereum smart contracts: A finite state machine based approach,» in *International Conference on Financial Cryptography and Data Security*, 2018.

[178] M. Tripoli e J. Schmidhuber, «Emerging opportunities for the application of blockchain in the agri-food industry,» *FAO and ICTSD: Rome and Geneva,* vol. 3, 2018.

[179] Y. Tribis, A. E. Bouchti e H. Bouayad, «Supply chain management based on blockchain: A systematic mapping study,» in *MATEC Web of Conferences*, 2018.

[180] M. S. Farooq, S. Riaz, A. Abid, T. Umer e Y. B. Zikria, «Role of iot technology in agriculture: A systematic literature review,» *Electronics,* vol. 9, n. 2, 2020.

[181] F. Tian, «A supply chain traceability system for food safety based on haccp, blockchain & internet of things,» in *International conference on service systems and service management*, 2017.

[182] J. Li e X. Wang, «Research on the application of blockchain in the traceability system of agricultural products,» in *2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2018.

[183] A. Iftekhar, X. Cui, M. Hassan e W. Afzal, «Application of blockchain and internet of things to ensure tamper-proof data availability for food safety,» *Journal of Food Quality,* vol. 2020, pp. 1-14, 2020.

[184] S. S. Kamble, A. Gunasekaran e R. Sharma, «Modeling the blockchain enabled traceability in agriculture supply chain,» *International Journal of Information Management,* vol. 52, p. 101967, 2020.

[185] Y. Chang, E. Iakovou e W. Shi, «Blockchain in global supply chains and cross border trade: a critical synthesis of the state-of-the-art, challenges and opportunities,» *International Journal of Production Research,* vol. 58, n. 7, pp. 2082-2099, 2020.

[186] "Agridigital website," 2015. [Online]. Available: https://www.agridigital.io/. [Accessed January 2022].

[187] M. P. Caro, M. S. Ali, M. Vecchio e R. Giaffreda, «Blockchain-based traceability in agri-food supply chain management: A practical implementation,» in *IoT Vertical and Topical Summit on Agriculture-Tuscany (IOT Tuscany)*, 2018.

[188] S. Wang, D. Li, Y. Zhang e J. Chen, «Smart contract-based product traceability system in the supply chain scenario,» *IEEE Access,* vol. 7, p. 115 122–115 133, 2019.

[189] B. Yu, P. Zhan, M. Lei, F. Zhou e P. Wang, «Food Quality Monitoring System Based on Smart Contracts and Evaluation Models,» *IEEE Access,* vol. 8, p. 12 479–12 490, 2020.

[190] T. H. Pranto, A. A. Noman, A. Mahmud e A. B. Haque, «Blockchain and smart contract for iot enabled smart agriculture,» *PeerJ Computer Science,* vol. 7, 2021.

[191] B. Haque, R. Hasan e O. M. Zihad, «Smartoil: Blockchain and smart contract-based oil supply chain management,» *IET Blockchain,* 2021.

[192] V. Andiappan e Y. K. Wan, «Distinguishing approach, methodology, method, procedure and technique in process systems engineering,» *Clean Technologies and Environmental Policy,* pp. 1-9, 2020.

[193] OpenZeppelin, "Openzeppelin: Contracts," 2020. [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts. [Accessed January 2022].

[194] P. Serra e G. Fancello, «Use of ICT for More Efficient Port Operations: The Experience of the EasyLog Project,» in *International Conference on Computational Science and Its Applications*, 2020.

[195] M. Jović, N. Kavran, S. Aksentijević and E. Tijan, "The transition of Croatian seaports into smart ports," in *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics*, 2019.

[196] M. Stopford, Maritime Economics, 3rd ed., London, UK: Routledge, 2017.

[197] L. Heilig, S. S. e V. S., «An analysis of digital transformation in the history and future of modern ports,» in *50th Hawaii International Conference on System Sciences*, 2017.

[198] M. O. Weernink, W. v. d. Engh, M. Francisconi and F. Thorborg, "The Blockchain potential for port logistics," 2017.

[199] M. Jović, M. Filipović, E. Tijan e M. Jardas, «A Review of Blockchain Technology Implementation in Shipping Industry,» *Pomorstvo,* vol. 33, n. 2, pp. 140-148, 2019.

[200] E. Tijan, S. Aksentijević, K. Ivanić e M. Jardas, «Blockchain technology implementation in logistics,» *Sustainability,* vol. 11, n. 4, p. 1185, 2019.

[201] K. Jabbar and P. Bjørn, "Infrastructural grind: Introducing Blockchain technology in the shipping domain," in *ACM Conference on Supporting Groupwork*, 2018.

[202] H. A. Gausdal, V. K. Czachorowski and Z. M. Solesvik, "Applying Blockchain technology: Evidence from Norwegian companies," *Sustainability,* vol. 10, no. 6, p. 1985, 2018.

[203] J. Lindman, K. T. V. e M. Rossi, «Opportunities and risks of Blockchain Technologies–a research agenda,» in *50th Hawaii International Conference on System Sciences*, 2017.

[204] P. Dutta, M. C. T., S. Somani and R. Butala, "Blockchain technology in supply chain operations: Applications, challenges and research opportunities," *Transportation Research Part E: Logistics and Transportation Review,* vol. 142, 2020.

[205] G. Bavassano, C. Ferrari and A. Tei, "Blockchain: How shipping industry is dealing with the ultimate technological leap," *Research in Transportation Business & Management,* 2020.

[206] IBM, "TradeLens: How IBM and Maersk Are Sharing Blockchain to Build a Global Trade Platform," 2018. [Online]. Available: https://www.ibm.com/blogs/think/2018/11/tradelens-how-ibm-and-maersk-are-sharing-Blockchain-to-build-a-global-trade-platform/. [Accessed January 2022].

[207] R. Ahmad, H. Hasan, R. Jayaraman, K. Salah e M. Omar, «Blockchain applications and architectures for port operations and logistics management,» *Research in Trasportation Business & Management,* 2021.

[208] E. Irannezhad, «The Architectural Design Requirements of a Blockchain-Based Port Community System,» *Logistics ,* vol. 4, n. 30, 2020.

[209] M. Laurent, N. Kaaniche, C. Le e M. V. Plaetse, «A blockchain-based access control scheme,» in *15th International Conference on Security and Cryptography (SECRYPT 2018)*, Porto, Portugal, 2018.

[210] V. Gatteschi, F. Lamberti, C. Demartini, C. Pranteda e V. Santamaria, «Blockchain and Smart Contracts for Insurance: Is the Technology Mature Enough?,» *MDPI: Future Internet,* vol. 10, n. 6, pp. 8-13, 2019.

[211] N. Uras, L. Marchesi, M. Marchesi e R. Tonelli, «Forecasting Bitcoin closing price series using linear regression and neural networks models,» *PeerJ Computer Science,* vol. 6, 2020.

[212] M. Briere, K. Oosterlinck e A. Szafarz, «Virtual currency, tangible return: Portfolio diversification with bitcoins,» *SSRN Electronic Journal,* vol. 16, n. 6, 2013.

[213] K. H. McIntyre e K. Harjes, «Order flow and the bitcoin spot rate,» *Applied Economics and Finance,* p. 42908–42920, 2014.

[214] L. Cocco, R. Tonelli e M. Marchesi, «An agent-based artificial market model for studying the bitcoin trading.,» *IEEE Access,* vol. 7, 2019.

[215] L. Cocco, R. Tonelli e M. Marchesi, «An agent based model to analyze the bitcoin mining activity and a comparison with the gold mining industry,» *Future Internet,* vol. 11, n. 1, 2019.

[216] "Coinmarketcap website," [Online]. Available: http://www.coinmarketcap.com. [Accessed January 2022].

[217] D. Mallqui e R. Fernandes, «Predicting the direction, maximum, minimum and closing prices of daily bitcoin exchange rate using machine learning techniques,» *Applied Soft Computing,* 2018.

[218] A. Kazem, E. Sharifi, F. K. Hussain, S. Morteza e O. K. Hussain, «Support vector regression with chaos-based firefly algorithm for stock market price forecasting,» *Applied soft computing,* vol. 13, n. 2, pp. 947-958, 2013.

[219] J. Patel, S. Shah, P. Thakkar e K. Kotecha, «Predicting stock market index using fusion of machine learning techniques,» *Expert Systems with Applications,* vol. 42, p. 2162–2172, 2015.

[220] J. Agrawal, V. Chourasia e A. Mittra, «State-of-the-art in stock prediction techniques,» *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering,* vol. 2, n. 4, pp. 1360-1366, 2013.

[221] G. Armano, M. Marchesi e A. Murru, «A hybrid genetic-neural architecture for stock indexes forecasting,» *Information Sciences,* vol. 170, n. 1, pp. 3-33, 2015.

[222] D. Enke e N. Mehdiyev, «Stock market prediction using a combination of stepwise regression analysis, differential evolution-based fuzzy clustering, and a fuzzy inference neural network,» *Intelligent Automation and Soft Computing,* vol. 19, n. 4, pp. 636-648, 2013.

[223] S. McNally, J. Roche e S. Caton, «Predicting the price of bitcoin using machine learning,» in *26th Euromicro International Conference on Parallel and Network-Based Processing*, 2018.

[224] V. Y. Naimy e M. R. Hayek, «Modelling and predicting the bitcoin volatility using garch models,» *International Journal of Mathematical Modelling and Numerical Optimisation,* vol. 8, pp. 197-215, 2018.

[225] D. U. Sutiksno, A. S. Ahmar, N. Kurniasih, E. Susanto e A. Leiwakabessy, «Forecasting historical data of bitcoin using arima and α-sutte indicator,» *Journal of Physics: Conference Series,* vol. 1028, n. 1, 2018.

[226] M. Stocchi e M. Marchesi, «Fast wavelet transform assisted predictors of streaming time series,» *Digital Signal Processing,* vol. 77, pp. 5-12, 2018.

[227] S. Y. Yang e J. Kim, «Bitcoin market return and volatility forecasting using transaction network flow properties,» in *IEEE Symposium Series on Computational Intelligence*, 2016.

[228] N. Bakar e S. Rosbi, «Autoregressive integrated moving average (arima) model for forecasting cryptocurrency exchange rate in high volatility environment: A new insight of bitcoin transaction,» *International Journal of Advanced Engineering Research and Science,* vol. 4, n. 11, 2017.

[229] L. Catania, S. Grassi e F. Ravazzolo, «Forecasting cryptocurrencies financial time series,» *BI Norwegian Business School, Centre for Applied Macro- and Petroleum Economics,* 2018.

[230] N. Vo e G. Xu, «The volatility of bitcoin returns and its correlation to financial markets,» in *International Conference on Behavioral, Economic, Socio-cultural Computing (BESC)*, 2017.

[231] C. Akcora, A. K. Dey, Y. R. Gel e M. Kantarcioglu, «Forecasting bitcoin price with graph chainlets,» in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2018.

[232] A. Greave e B. Au, «Using the bitcoin transaction graph to predict the price of bitcoin,» *Computer Science,* 2015.

[233] R. Hyndman e G. Athanasopoulos, «Chapter 6,» in *Forecasting: principles and practice*, 2014, pp. 157-182.

[234] "Yahoo Finance," [Online]. Available: http://www.finance.yahoo.com. [Accessed January 2022].

[235] A. Banerjee, J. Dolado, J. Galbraith e D. Hendry, «Chapter 4,» in *Cointegration, error correction, and the econometric analysis of non-stationary data*, Oxford: Oxford University Press, 1993.

[236] G. E. P. Box e G. Jenkins, Time series analysis: Forecasting and control, Holden-Day, 1976.

[237] W. Mckinney, «pandas: a foundational python library for data analysis and statistics,» *Python High Performance Science Computer,* 2011.

[238] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Muller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay,

"Scikit-learn: Machine learning in python," *Journal of Machine Learning Research,* vol. 12, 2012.

[239] F. Chollet, "Keras," 2015. [Online]. Available: https://keras.io. [Accessed January 2022].

[240] S. Hochreiter e J. Schmidhuber, «Long short-term memory,» *Neural Computation,* vol. 9, n. 8, pp. 1735-1780, 1997.

[241] S. Skipper e Perktold, «Statsmodels: Econometric and statistical modeling with python,» in *9th Python in Science Conference*, 2010.

[242] E. Jones, T. Oliphant e P. Peterson, «Scipy: Open source scientific tools for python,» 2001. [Online]. Available: http://www.scipy.org/.

[243] P. Praitheeshan, L. Pan, J. Yu, J. Liu e R. Doss, «Security analysis methods on ethereum smart contract vulnerabilities: A survey,» *arXiv preprint:1908.08605.*