future internet

MDPI

Article

Investigation on Self-Admitted Technical Debt in Open-Source Blockchain Projects

Andrea Pinna *, Maria Ilaria Lunesu , Stefano Orrù and Roberto Tonelli

Department of Mathematics and Computer Science, University of Cagliari, 09124 Cagliari, Italy; ilaria.lunesu@unica.it (M.I.L.); ste_1795@hotmail.it (S.O.); roberto.tonelli@unica.it (R.T.)
* Correspondence: pinna.andrea@unica.it

Abstract: Technical debt refers to decisions made during the design and development of software that postpone the resolution of technical problems or the enhancement of the software's features to a later date. If not properly managed, technical debt can put long-term software quality and maintainability at risk. Self-admitted technical debt is defined as the addition of specific comments to source code as a result of conscious and deliberate decisions to accumulate technical debt. In this paper, we will look at the presence of self-admitted technical debt in open-source blockchain projects, which are characterized by the use of a relatively novel technology and the need to generate trust. The self-admitted technical debt was analyzed using NLP techniques for the classification of comments extracted from the source code of ten projects chosen based on capitalization and popularity. The analysis of self-admitted technical debt in blockchain projects was compared with the results of previous non-blockchain open-source project analyses. The findings show that self-admitted design technical debt outnumbers requirement technical debt in blockchain projects. The analysis discovered that some projects had a low percentage of self-admitted technical debt in the comments but a high percentage of source code files with debt. In addition, self-admitted technical debt is on average more prevalent in blockchain projects and more equally distributed than in reference Java projects. If not managed, the relatively high presence of detected technical debt in blockchain projects could represent a threat to the needed trust between the blockchain system and the users. Blockchain projects development teams could benefit from self-admitted technical debt detection for targeted technical debt management.

Keywords: self-admitted technical debt; blockchain; NLP

Citation: Pinna , A.; Lunesu, M.I.; Orrù, S.; Tonelli, R. Investigation on Self-Admitted Technical Debt in Open-Source Blockchain Projects. *Future Internet* **2023**, *15*, 232. https://doi.org/10.3390/fi15070232

Academic Editors: Massimo Cafaro, Italo Epicoco and Marco Pulimeno

Received: 2 June 2023
Revised: 24 June 2023
Accepted: 27 June 2023
Published: 30 June 2023

1. Introduction

The idea of technical debt was conceived by Ward Cunningham in the 1990s. The idea evolved into a theory that concerns the quality of software projects with reference to the productivity of the development team [1]. In particular, code is of high quality if it can maintain high productivity even when the development team or project goals change. When developers deviate from this goal, taking shortcuts and delivering code that isn't quite right, technical debt builds up. As Thierry Coq et al. reported [2]:

> Code that is not quite right may include many types of issues. These issues may be related to architecture, structure, duplication, test coverage, comments and documentation, potential bugs, complexity, code smells, coding practices, and style. All these types of issues incur technical debt because they have a negative impact on productivity.

Cunningham conceives of debt as an analogy to what happens in financial terms. Getting a loan is positive because it allows us to do something immediately. However, we will pay the interest until the debt is paid off. Similarly, accumulating technical debt allows you to get to market quickly, but the accumulation of technical debt leads to a reduction in

productivity, due to coming across code that needs to be fixed, and an increase in costs since code refactoring is required to pay off the debt [3]. The rate of accumulation of technical debt may depend on the business strategy and the management of the process. Initiatives with many competitors that aim for a short time-to-market can accumulate technical debt to be able to get first to market and gain a competitive advantage. This is the scenario that characterizes initiatives related to blockchain technology [4].

It is evident that developers are personally involved in the technical debt pileup [5,6]. However, developers pile up debt more or less consciously, depending on their experience. The developer who realizes that the code he wrote is not quite right may have time to write a comment for posterity to be taken into account in the next refactoring. This phenomenon was initially referred to by Potdar and Shihab in 2014 as Self-Admitted Technical Debt (SATD) [7], and it was proposed to take into account debt that is purposely introduced (for example, by a temporary fix) and confessed by developers themselves. In ref. [8], the relationship between SATD and software quality has been empirically investigated by using data collected from five open-source projects. In particular, it has been looked at whether files with SATD have more flaws than files without SATD, whether SATD changes cause new flaws, and whether SATD-related changes tend to be more challenging. Currently, the number of studies that propose automatic methods for detecting SATD is growing, not only from comments but also from additional sources, such as commit messages and pull requests, or by fusing different sources, is growing.

Blockchain technology, since the mid-2010s, has given rise to numerous software projects. After the success of the Bitcoin system, thousands of systems based on blockchain or other types of distributed ledger technology (DLT) were put on the market, many of which were generated as forks or clones of previous projects. Considering only public blockchains, today there are about a thousand blockchains hosting a native cryptocurrency. One of the most striking phenomena was that of ICOs, where development teams competed to offer investors the most convincing ideas [4]. Today, the monitored tokens in circulation are over twenty thousand, and the total trading volume that revolves around blockchain technology is in the order of tens of billions of dollars a day. Furthermore, in order to convince investors and instill trust, blockchain projects are often open-source, which allows researchers to study the code to highlight development patterns, recognize vulnerabilities, and calculate statistics [9,10]. For the reason that this kind of system is intended to create trust, understanding TD in blockchain systems is crucial. Blockchain systems would be hard to maintain and thus unsustainable without proper TD management, undermining the trust that users have in them. For this reason, it is of particular interest to evaluate the presence of technical debt within software projects and to make a comparison with other types of projects. In this paper, we focus on the SATD detection of open-source blockchain projects and, in particular, on a selection of ten blockchain projects. The selected projects are the most relevant among the open-source ones in terms of the capitalization of the related cryptocurrency. The work reported in this paper was intended to answer the following two research questions.

- RQ1. To what extent is SATD present in open-source blockchain software?
- RQ2. Is the presence of SATD in open-source blockchain projects statistically different from other types of open-source projects?

To reach the goal of answering the research questions, we organized the work into several subsequent phases. First of all, ten blockchain projects whose source code was cloned were selected. For the extrapolation of the comments, a tool was created capable of parsing all the languages used in the projects and recording the results in serialized form. Furthermore, a technique for detecting the self-admitted technical debt was chosen among those available, and in particular, a technique based on NLP described by Maldonado et al. was used [11]. Subsequently, the comments were analyzed and classified. The results on the presence of self-admitted technical debt were then analyzed and compared with those of previous studies on non-blockchain projects.

The remainder of this paper is structured as follows: Section 2 introduces SATD and the state of the art on its detection in software projects and blockchain projects; Section 3 describes the process of detecting SATD in blockchain projects that characterizes this work, including project selection, comment extraction, and classifier training phase. Section 4 reports the results of the classification of the comments and the results of the comparison with the results of a reference study. Section 5 discusses the results obtained, and Section 6 draws conclusions.

2. Background

Cost cuts are a common practice in all industries, and software development is no exception. Consequently, development teams are often forced to prioritize new functionalities over internal product quality. It is precisely in these situations that technical debt is introduced, intended as faults produced as a result of rapid or temporary remedies [7,8]. However, if effectively managed, TD can help the project achieve its goals sooner or more cheaply [11]. While technical debt may have negative consequences for software quality, its influence is not tied to flaws, but rather to making the system more difficult to alter in the future [6,12]. For this reason, technical debt management is important both to guarantee the quality of the software and, especially, its maintainability. Both factors are important for the success of software projects and are especially important for blockchain-based projects, given the characteristic need to build trust [13]. Specific tools, instruments, and technologies applied to the software project help the development team to keep track of technical debt accumulation, and to gain additional insight into the internal quality of their software via an automatic indicator [11], and provide for effective TD removal by the developers [8,14].

There are two basic methods for detecting TD: TD measurement on source code and SATD detection [15]. Several works have been published on TD measurement [16–19], where specific tools and technologies are claimed to be able to evaluate and detect technical debt in source code lines and identify faulty design decisions or bad source code, by using static code analysis, code smell detection, and other rule-based techniques. SATD detection is an effective technique for TD detection that can be used alternatively or in a complementary manner to TD detection techniques based on source code analysis. In ref. [15], the comparison between TD measurement and SATD detection is shown, highlighting that the detection of SATD and TD measurement is aligned and coherent even if differences in some values exist. Data reveal that the average discrepancy between the SATD detected and the measured TD is 3% when measured with SonarQube and 4% when measured with Vector Squore. For comparison, the average disparity between SonarQube and Vector Squore measures is as high as 2%. In ref. [11], JDeodorant is used to compare SATD detection results with TD measurement via code analysis. Three typologies of code smells are examined: Long Method, God Class, and Feature Envy. Results show that 69.7% of files bearing SATD are also implicated in at least one of the three examined code smells. According to the findings, utilizing code comments to identify technical debt is a supplementary technique to utilizing code smells to detect technical debt. There is clearly overlap, but each technique reveals unique examples of technical debt, so both approaches (measuring and SATD detection) to identifying technical debt should be employed. While TD measurement is a great and useful tool, it cannot be the only insight into accumulated debt [11]. The studies related to the detection of the SATD, the techniques used, and the importance of detecting the TD in blockchain projects will be detailed below

2.1. Self-Admitted Technical Debt

The quality of the software is an issue that needs to be addressed because it affects the cost, reliability, and longevity of software products. Consequently, it is essential to have tools and metrics to keep quality under control during the entire product life cycle. When developers take shortcuts during software development, mostly for the sake of expediency, it is called technical debt. Technical Debt (TD) is unfinished work or tempo-

rary workarounds that can negatively impact software maintenance and overall quality. Technical debt impacts the quality of applications by generating structural weaknesses that translate into slowness and functional deficiencies at the development level. In the context of Agile programming, the more the technical debt grows, the more time it takes to compensate for errors, intervening at the syntax and refactoring levels.

By doing an exploratory investigation on source code comments that hint to instances of TD, Potdar and Shihab (2014) [7] changed the course of their research. They discovered that a significant portion of TD is explicitly admitted by developers via code comments. This phenomenon was initially referred to by the authors as Self-Admitted Technical Debt (SATD) which was proposed to take into account debt that is purposely introduced (for example, by a temporary fix) and confessed by developers themselves. SATD refers to technical debt incurred by developers themselves (for example, comments to indicate an imperfect implementation that should be replaced or improved later). In brief, it refers to when these shortcuts are explicitly admitted by developers, such as when they write a TODO or Fixme comment. Potdar and Shihab's exploratory study on TD focused primarily on three aspects: the amount of SATD quantification, the motivations for SATD introduction, and the actual removal percentage of SATD, as well as a deeper analysis of the comments on the source code, focusing on SATD in software development in open-source software projects. The study used data from four large open-source projects: Eclipse, Chromium OS, Apache HTTP server, and ArgoUML. Given the importance of code comments for programmers to communicate information about their code, and to be used to help improve the quality of a codebase, it is important to track the status of TD in order to make informed decisions about how to manage it.

Previous research has demonstrated that source code comments may effectively identify TD. The focus of earlier research has been on detecting SATD in issue trackers and source code comments. In ref. [8], the relationship between SATD and software quality has been empirically investigated by using data collected from five open-source projects. In particular, it has been looked at whether files with SATD have more flaws than files without SATD, whether SATD changes cause new flaws, and whether SATD-related changes tend to be more challenging. Currently, the number of studies that propose automatic methods for detecting SATD from additional sources, such as commit messages and pull requests, or by fusing different sources.

Several studies have focused on SATD detection and examined its impact on software quality. However, preliminary findings indicate that not all SATD is bad, and some may need to be removed while others may be acceptable, but SATD detection is a way of detecting technical debt through code comments and to remove it [14]. In particular, the median time for SATD to be removed is between 18 and 172 days. Developers mostly use SATD to track future bugs and areas of the code that need improvement. They also mostly remove SATD when they are fixing bugs or adding new features. These findings contribute to the body of empirical evidence on SATD, particularly its removal [14]. After a couple of years, Zampetti et al. in ref. [20], conducted an in-depth quantitative and qualitative study of how SATD is addressed in five Java open-source projects and how to reduce it. The results can be used to plan TD management, study patterns, and provide recommendations to developers.

Focusing on the prioritization aspect, an interesting work is the study conducted by Mensah et al. in ref. [12] where a prioritization scheme to minimize SATD in software development to aid in decision-making before software release to minimize high maintenance overheads is introduced. The scheme focuses on identifying, examining, and estimating rework effort for prioritized tasks. The results showed that design debts are highly prone to software bugs, requiring a rework effort of 10 to 25 commented LOC per SATD source file to address the few vital tasks. Developers often try to pay back design and test debt first, but system or infrastructure debt is often ignored. There are practices used to assist in SATD management, such as maintaining a list of prioritized SATD, grouping related technical debt items, and resolving development tasks jointly. However, determining the

priorities of SATD and other works and convincing developers not to introduce SATD can be challenging [12].

During the same time frame, in ref. [21], Sierra et al. published a survey about SATD where they analyzed current approaches and techniques for detection, comprehension, and repayment. It identifies open challenges, areas missing investigation, and potential future research avenues. However, the majority of modern state-of-the-art methods use pattern matching to recognize SATD remarks, achieving great accuracy but relatively low recall. As a result, they could overlook several SATD remarks and lack sufficient practicality [22].

Due to this, in ref. [6], a method for automatically identifying SATD that combines source code comments, commit messages, pull requests, and issue tracking systems is suggested and assessed. Li et al. [6] discuss technical debt, which is defined as the compromise of maintainability and evolvability of software systems in the long term for the sake of short-term goals. Technical debt can accumulate if not proactively managed, resulting in a maintenance crisis.

While there has been a fair amount of work studying SATD management in Open Source projects, SATD in industry is relatively unexplored. Li et al. in [23] discuss an exploratory case study with an industrial partner aimed at understanding core characteristics, developers' attitudes, triggers for SATD introduction, relations between sources, practices used, and challenges and tooling ideas for SATD management focusing on industrial context. The study, with the aim of understanding and improving the nature and management process of self-admitted technical debt in practice from the point of view of software engineers, analyzed self-admitted technical debt in source code comments, issue tracking systems, and commit messages in the context of the embedded systems industry.

2.2. SATD Detection Strategies and Techniques

Detecting the SATD essentially requires parsing the comments and classifying them based on the text content left by the developers. This operation can be performed manually and consists of associating each comment with a label that determines its type (for example, a specific category of SATD or the absence of SATD). However, manual classification is a time-consuming and skilled operation that is actually the basis for the creation of datasets used in classifier training. According to recent literature, two main techniques of Machine Learning are currently used for automatic SATD detection: Natural Language Processing (NLP)-based and Neural Network (NN)-based.

NLP techniques can be used to automatically identify the SATD within code comments to speed up the SATD identification process, making it more efficient and accurate. NLP-based classifiers use NLP features and a training dataset of labeled samples to build up classification rules [24]. In ref. [11], the authors use a maximum entropy classifier (Stanford Classifier) to classify the comments of the code, on the basis of a dataset they created manually through the attribution of labels that represent the types of SATD they considered. Through the use of NLP, authors obtain significantly more accurate results than searches based on keywords or phrases. To improve the F1 score of training results, the authors in ref. [25] use N-gram inverse document frequency (IDF) as a feature for the classification of requirements and design SATD. In refs. [5,22], authors adopt text-mining, and, in particular, use feature selection to improve performance in SATD detection.

Neural network-based techniques for SATD detection consist of using NN processes for efficient extraction of information from text useful for SATD recognition. The use of Convolutional Neural Networks (CNNs) for feature extraction from texts was initially proposed by Kim [26]. CNNs for SATD detection find application in ref. [27] for creating a trained CNN model that authors use for SATD pattern identification. In refs. [6,28], the authors adopt an approach based on CNN for text via multitasking learning and focus on the detection of the SATD from different sources, including source code comments, and comments in commits, issues, and pull requests. Neural Networks based on Generative Adversarial Networks are used in ref. [29] to detect different typologies of SATD and,

in particular, to solve the imbalance of sampled data (enhancing the features containing few number data). In ref. [30], authors adopt a deep learning approach to recommend how SATD should be removed, which involves the use of CNN for processing vectorized comments and Recurrent Neural Networks for processing the whole source code. Automatically Learning Patterns (Automatically Learning Patterns for Self-Admitted Technical Debt Removal)

2.3. Technical Debt in Blockchain Projects

Blockchain technology is known for its characteristics that allow for the creation of decentralized and distributed systems, based on the archiving of validated transactions according to a specific consensus mechanism. Blockchain systems include on-chain components (the blockchain itself and smart contracts) and also off-chain components, i.e., based on traditional computing or archiving systems.

Studying TD in blockchain systems is of considerable importance because this type of system is designed to generate trust. Without proper management of the TD, blockchain systems would be difficult to maintain and therefore unsustainable, undermining the trust that users place in them. The applications of the blockchain are not limited to cryptocurrencies given that the blockchain is considered one of the most innovative technologies available to companies, and there are an increasing number of companies that have adopted the blockchain in the most diverse sectors. Among these are the following [31]: food traceability, finance and defi, right management, self-sovereign identity, and, more generally, any web3 application.

The study of TD in blockchain projects has recently been undertaken from different points of view. In particular, in [32] 2021 the authors focus on the on-chain component, i.e., on smart contracts, and in particular on the need to assess the TD security. Previously, Ref. [33] proposed a taxonomy to better understand the Security TD in blockchain projects and avoid risks. Both studies are conceived to increase the visibility of security design issues. Recently, Yu et al. focused on SATD detection in blockchain projects. In particular, in [34], the authors focus on the use of NLP techniques and a maximum entropy classifier for the identification of different types of SATD in a selection of blockchain projects, while in ref. [35], a large-scale pre-trained model for NLP, and in particular BERT, was used to identify the presence or absence of SATD in each comment of a selection of blockchain projects.

3. SATD Detection in Blockchain Software Projects

The study of the SATD in the blockchain projects presented here is a process composed of several phases. In order, the first phase involved creating a sample of ten blockchain projects that would provide the code to examine. The second phase concerns the extraction of comments from the source code through the creation of a multilingual extractor capable of extracting the data in serialized form. The third phase concerns the choice of the detection technique of the SATD from the comments based on its applicability to the extracted data. Then the last two stages are about tracking and analyzing the results. A representative schematic of the phases of this study is shown in Figure 1.

3.1. Selection of Open-Source Blockchain Projects

To conduct our study on self-admitted technical debt in blockchain projects, we focused on the top ten open-source blockchain projects in terms of market capitalization and relevance in terms of repository content. Projects that do not provide source code have been rejected.

At the end of the selection, the following projects were chosen: Bitcoin, Ethereum, USD Coin, Binance, Terra, Xrp, Polkadot, Cardano, Avalanche, and Solana. These projects correspond to the cryptocurrencies among those with the highest capitalization and include the most relevant projects in terms of repository content. For each project, we describe the main features below.

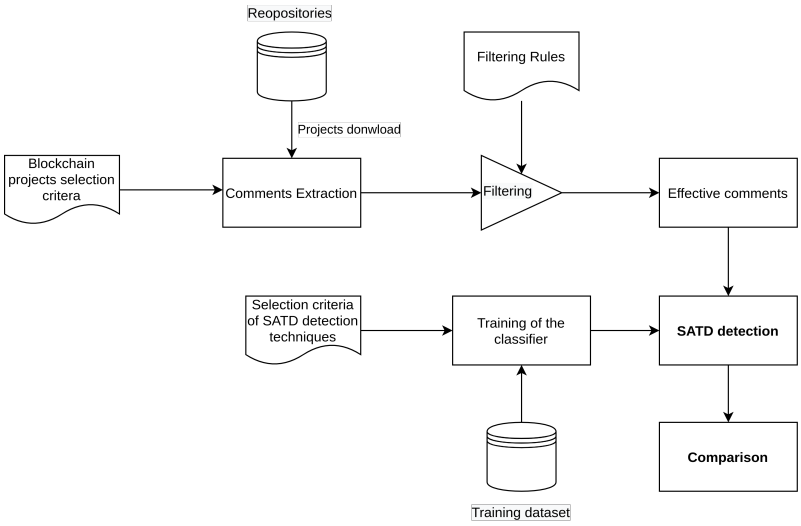

Figure 1. Representation of the execution phases of the study of the detection of the SATD in blockchain projects.

3.1.1. Bitcoin

The introduction of Bitcoin in 2009 began a revolution in the world of digital assets by allowing users to send and receive funds without the use of central intermediaries. Each block in the bitcoin blockchain contains a set of transactions and an encryption that binds it to the previous block. This structure gives rise to the term blockchain and is characterized by being secure and immutable. Posted transactions cannot be edited or deleted later. The verification of the transactions on the blockchain is carried out by all the nodes of the network, and the registration of new transactions is entrusted to the miners, encouraged through proof-of-work consensus by the possibility of earning Bitcoins. The block time is about 10 min on average. Bitcoin is also the first cryptocurrency in terms of capitalization.

3.1.2. Ethereum

Ethereum is a blockchain designed for the development of decentralized applications and is the most widely used blockchain 2.0, i.e., a blockchain that can be programmed through the use of smart contracts and can be used in different scenarios. The blockchain is public and permissionless, and it employs a consensus algorithm that was initially based on Proof of Work and is now based on Proof of Stake, with an average block time of around 15 s. Ethereum has its own currency, Ether (ETH), which is the second cryptocurrency in terms of capitalization and the first in terms of daily volume.

3.1.3. Xrp (Ripple)

Ripple is a peer-to-peer protocol created by Ripple Labs 53 in 2012 for the secure, instantaneous, and low-cost global transfer of funds. Conceived to be used by banks and financial institutions, it is defined by Ripple itself as "an infrastructure technology for interbank transactions". The native currency used in the protocol is the XRP token. Major Ripple partnerships include Accenture, American Express, Deloitte, Santander, UBS, and Unicredit. Ripple aims to solve some of the biggest problems facing banks and financial institutions in globally transferring money, especially for cross-border payments.

3.1.4. USD Coin

USD Coin is the most popular of the cryptocurrencies that fall under the definition of stablecoin, i.e., those cryptocurrencies designed to maintain a stable value over time based on the value of other currencies or commodities such as USD, gold, etc. The value of USDC is backed by collateral that equals the number of outstanding USDC, in this case, US dollars, and the value of 1 USDC always equals the value of 1 US dollar. The USD coin project

aims for full financial interoperability, and for this reason, the USD coin implementation is available on several blockchain platforms, including Ethereum, Solana, and Avalanche. The issuance of USDC is reserved for the Circle consortium.

3.1.5. Cardano

Cardano is a blockchain platform for executing smart contracts released in 2017 and built through peer-reviewed research. It uses a proof-of-stake consensus protocol called Ouroboros, and its blockchain is structured on different levels, each focused on particular characteristics such as scalability or security. Cardano is part of the projects trying to solve some intrinsic problems of the first generation of blockchains, such as the scalability and speed of transactions. The focus is therefore on building a more sustainable and balanced blockchain ecosystem.

3.1.6. Terra

Terra is a blockchain designed to provide a stable and scalable infrastructure for payments and financial apps. It was designed to be used as a platform for e-commerce financial applications and offers an efficient solution for low-cost peer-to-peer transactions. Terra is also a decentralized and secure system that uses blockchain technology to ensure the transparency and security of transactions. The native cryptocurrency is the stablecoin of the same name, which is associated with a reserve asset cryptocurrency called Luna.

3.1.7. Polkadot

Polkadot is a multi-chain blockchain designed to provide an infrastructure for building different blockchains and interconnecting them. Polkadot was designed to overcome the limitations of single blockchains, such as scalability and interoperability, by providing a flexible infrastructure for building many decentralized applications. The Polkadot blockchain uses a relay chain system to connect to different blockchains, making them able to exchange information and currencies with each other. This makes it possible to create an interconnected ecosystem of different blockchains and applications, paving the way for many new opportunities for blockchain technology. Additionally, Polkadot offers a secure and scalable platform for developing new blockchain and decentralized technologies.

3.1.8. Avalanche

Avalanche is a blockchain platform launched in 2020 with the aim of ensuring scalability and high performance with up to 6500 transactions per second. Conceived to be a platform for decentralized applications, it stands out for guaranteeing predictable and specific fees. The Avalanche network consists of three different blockchains: the X-Chain, C-Chain, and P-Chain. In addition, the Avalanche platform allows developers to build application-specific blockchains (called subnets). The consensus protocol is Proof of Stake.

3.1.9. Binance

Binance is a platform for financial services and cryptocurrency exchanges. In addition to offering an exchange platform, Binance has also launched its own blockchain, known as Binance Chain. Binance Chain was designed to provide a highly efficient and decentralized solution for exchanging digital tokens. Originally known as Binanche chain, the system is now called Build N Build chain (BNB chain) and is the blockchain used by Binance services. This is composed of two blockchains, the Beacon chain for block validation and governance and the Smart Chain, which is EVM-compatible and therefore capable of run smart contracts.

3.1.10. Solana

Solana is an open-source blockchain platform that uses a consensus protocol called Proof-of-History (PoH) to increase the speed and efficiency of transactions. Solana is

designed to be a high-performance platform for building Apps (Decentralized Applications) and DeFi (Decentralized Finance). The smart contract programming language is rust.

Table 1 summarizes the blockchain projects chosen for SATD analysis, with the related programming languages used in the project and the links to the repositories

Table 1. The selected blockchain open-source projects, programming languages and the repositories URL.

Project	Number of Contributors	Languages	Link [3]
Bitcoin	113	c++, python	https://github.com/bitcoin
Ethereum	2095	go, c, javascript, c++, solidity, python, typescript	https://github.com/ethereum
Binance	NA [1]	java, javascript	https://github.com/binance-exchange
USD coin	34	javascript, solidity, typescript	https://github.com/centrehq
Solana	343 [2]	rust, typescript, go, cue, rust, python	https://github.com/solana-labs
XRP (Ripple)	152	c++, c , css, javascript	https://github.com/ripple
Avalanche	113	javascript, go, typescript, python	https://github.com/ava-labs/
Cardano	238	haskell, markdown, ruby, c, typescript	https://github.com/input-output-hk
Terra	67	javascript, go, python, typescript	https://github.com/terra-money
Polkadot	697	rust	https://github.com/paritytech

[1] The number of contributors is not available for this project. [2] This number is net of the 5000 contributors registered for getting the solana token. [3] Projects were downloaded on 1 April 2022. Last accessed on 10 June 2023.

3.2. Comment Extraction

This phase includes extracting the comments from the source files, serializing the comments into csv files, and filtering the comments to remove sources of noise.

To extract comments, a copy of each project's code was downloaded first. The following step was to extract the comments from the source code. In terms of the programming languages used, each blockchain project turned out to be a diverse set. For the reason that there are so many languages, a tool (available at github.com/StefanoOr/RecuperoCommentiDeBitoTecnio, accessed on 10 June 2023)was developed by us for this work that extrapolates the comments for any language in Table 1. The tool allows the extraction of comments and their characterization in terms of length, number of lines, position in the file.

The operation of the extractor is as follows:

1. Once the path of the folder containing the repository has been assigned, the program recursively enters all the subdirectories, and whenever it finds a code file (recognized through a list of extensions), the file is read, and the comments are extrapolated based on the types of syntax of language comments.
2. Whenever a file with a specific extension is found, the program reads the file line by line, and when it finds a comment based on the syntax of the language, it inserts the comment into a structured list.
3. After reading the source file, the list contains all extracted comments, and for each comment contains information about the line number, column number, and length of the comment.
4. Once the previous operation is finished, a CSV file is created with the same name as the source file, to which the ".csv" extension is added. It contains all the comments

extrapolated from the file and some relevant information, such as the position of the comment within the file. The newly created file will be placed in a specific folder.

The result of this operation is a number of CSV files equal to the number of source files present for each repository, and each CSV file will contain all the comments of the corresponding source file. The set of CSV files represents our initial dataset to analyze.

For the SATD analysis, once all the comments from all the files have been extracted, the files are merged into a single file using a script created for the purpose. The script recursively takes the ".csv" files and, for each file containing the comments extracted above, removes the character structures used in the syntax of the language to indicate the comments (for example, '//' or '/ *' and '*/'), punctuation characters (for example, ',', '. . . ', ';', ':'), and any extra white space characters (for example, ' ', '\t', '\n'), and finally we convert all the comments to lowercase, the comment once it has been cleaned of punctuation marks is placed in another CSV file. However, we have decided not to remove the exclamation and question marks. These specific punctuations are helpful when identifying comments with SATD. Once this process is finished, we will have, as a final result, a single CSV file per project that contains all the comments of the repository.

Table 2 shows an example of the contents of each CSV file. The "Line" and "Column" data indicate where the comment is located within the source file, and the "Number of lines" indicates how many lines the comment consists of (in the case of multiline comments).

Table 2. Example of csv file content created by the comment extractor.

Line	Column	Number of Lines	Text of the Comment
1	1	1	Copyright (C) 2019–2021, Ava Labs, Inc. All rights reserved.
2	1	1	See the file LICENSE for licensing terms.
12	1	1	This file contains structs used in arguments and responses in services.
14	1	1	EmptyReply indicates that an api doesn't have a response to return.

Multi-line comments were also processed. Most programming languages support multi-line comments. However, developers can use a single-line comment set to leave a single comment. These multi-line comments often need to be combined to represent the final intention of the developer. For our study, these comments were collected and merged into a single comment.

3.3. Comments Filtering Rules

Before proceeding to the comment classification, we have chosen to remove from the comment dataset all the lines that contain the following types of comments:

- License specification comments (such as "mit software licens", "Spdx-license-identifier", or "License New BSD License").
- Comments generated by the development environment.
- Documentation comments (such as long comments that describe source code elements).
- Copyright comments (such as "Copyright c 2018–2020 The Bitcoin Core developer", "Copyright c 2007 2015 University of Illinois at Urbana Champaign")
- Comments containing blockchain addresses (sucha s "5DhDcHqwxoes5s89AyudGMjt ZXx1nEgrk5P45X88oSTR3iyx").
- Comments with only numbers.

At the end of the filtering process, a reduced number of actual comments to be analyzed is obtained compared with the initial one. Table 3 reports, for each project, the total number of comments extracted from the source codes, the number of comments filtered and eliminated, and the number of effective comments to be subjected to classification.

Table 3. Number of extracted comments before and after the filtering process.

Project	Total Comments	Removed Comments	Effective Comments	Filtering Percentage
Bitcoin	18,858	971	17,887	5.14%
Ethereum	43,655	1092	42,563	2.50%
Binance	1294	11	1283	0.85%
USD coin	1575	42	1533	2.67%
Solana	27,534	14	27,520	0.05%
XRP	14,224	625	13,599	4.39%
Avalanche	13,145	1265	11,880	9.62%
Cardano	24,325	213	24,112	0.87%
Terra	1575	7	1568	0.44%
Polkadot	56,007	6419	49,588	11.46%

3.4. Choice of Detection Technique and Configuration

The choice of detection technique is essentially based on considering two main aspects. The first regards the possibility of replicating the process in existing studies, i.e., accessing the method and the training dataset and the second regards having comparable results about the presence of the SATD in open source projects, which we can take as a reference for the results that will be obtained in this study. Among the techniques presented in Section 2.2, we decided to use the technique used in [11], as the authors provide the tool specifications and the training dataset, and discuss the results of the detection of the SATD in a selection of open source projects. We will then use NLP techniques and the Stanford Classifier, which is a Java implementation of a maximum entropy classifier [36].

The Stanford Classifier is a machine learning tool developed by Stanford University designed to perform text classification. The classification algorithm used by this tool is based on the Maximum Entropy (MaxEnt) method [37]. The algorithm uses several binary functions (features) that correspond to specific properties or attributes of the input text that are distinctive enough with respect to the class it belongs to. In other words, features are elementary pieces of evidence that link the fact that we observe a given d (for example, a word) with a category c that we want to predict for a new text. Examples of features are: previous word, current word; presence of suffixes; type of word (adjective, noun, verb, etc.); presence of capital letters in a specific order; tags that precede or follow the word (in, to) [38]. The learning phase of the algorithm consists of optimizing the weights of the characteristic functions in order to maximize the joint probability of the observed data and the corresponding class labels. This process is usually completed using an optimization algorithm, such as gradient descent or coordinate optimization. The underlying principle of the MaxEnt algorithm is to find the probability distribution model that maximizes entropy (a concept related to the degree of uncertainty in a distribution) subject to specific given constraints. In other words, we try to find the most uniform model possible that is consistent with the observed information. Once trained, the Stanford Classifier can be used to assign class labels to new data based on the learned probability distribution model. The algorithm calculates the probability of each possible class label given the observation and selects the one with the highest probability as the predicted label.

To that end, a tool capable of configuring the classifier via property files, training the classifier via the training dataset, and processing the data collected in the CSV files to detect the presence of SATD in the comments has been developed. The tool takes as input the configuration property file for the classifier, the training set, and the CSV file containing the comments extracted from each repository. The classifier, once trained, will be used to assign a class label to each comment in our comments dataset. The tool makes use of the "ColumnDataClassifier" belonging to the library "edu.stanford.nlp.classify", which allows the processing of labeled structured data organized into columns, as in the case of our csv file-based dataset.

The training dataset described in ref. [11] is composed of 57,676 source code comments, manually categorized into SATD classification labels. Although the set pf dataset labels includes multiple categories of SATD, in order to avoid an excessive imbalance, we decide to consider only the three most represented categories: "design", "requirements" and "withoutclassification" (i.e., absence of SATD), where "design" indicates that the comment refers to suboptimal design, including workarounds and features that need to be extended to support more features, and "requirement" indicates that the comment refers to some software requirements that have not been fully satisfied by the implementation. The training phase was repeated several times, and at each training of the classifier, the training dataset was randomly divided into two sets, training set (90% of labeled comments) and the test set (the remaining 10%).

The configuration of the classifier consists of the selection of the NLP properties the classifier has to use in its training. A configuration file (named prop) consists of a series of properties described in the "ColumnDataClassifier" class [36].

In our study, we defined two different props. A first prop was extracted from the reference study [11], while we created the second prop aiming to optimize training performance, i.e., to reach the highest possible score of the "Micro-averaged accuracy/F1" and "Macro-average F1", These two configuration files describe the configuration of two different SATD detection models. The two configurations differ in that in the configuration used in ref. [11], the use of N-Grams is set to false and the use of lowercase words is set to true, while in the prop we created, the use of N-Grams is set to true, the use of split words is set to true, and the length of the N-Grams is limited between 1 and 5. In summary, the main differences between the two configuration files are the use of N-Grams, the form of the split words, and the length of the N-Grams.

For each class, the results show the number of true positives (TP), false negatives (FN), false positives (FP), and true negatives (TN). True positives are cases where the maximum entropy classifier correctly identifies, while true negatives are comments without technical debt that are classified as such. The table also shows the accuracy, precision, recall and F1 measure for each class. The classes in question are the three classes present in the dataset, namely WITHOUTCLASSIFICATION, REQUIREMENT and DESIGN.

Using the TP, TN, FP, and FN values, we are able to evaluate the performance of different classifier configurations. Performances are evaluated in terms of

$$Precision\ P = TP/(TP + FP),\ Recall\ R = TP/(TP + FN),$$

$$Accuracy = (number\ of\ correct\ predictions)/(total\ predictions),$$

$$F1\ score = (2 \times ((P \times R)/(P + R)).$$

The Micro-averaged F1 score is used to evaluate the overall performance of a classifier and has a value between 0 and 1. Table 4 lists the performance of the training by using the same configuration as in [11], while Table 5 lists the performance of the training obtained using the optimized configuration.

We can see from tables that the performances obtained through the optimized configuration outperform those of the reference configuration. This is essentially given by a reduction in false positives and this is reflected in a substantial reduction in the number of comments classified as SATD compared with what was obtained using the reference configuration.

Although our configuration has better results in terms of score, we felt it necessary to make some considerations. As mentioned, the optimized classifier uses a different model than the reference study. By running a test classification on one of the blockchain projects under review (Avalanche) with both classifiers, the results obtained highlight the differences between the two models. In particular, as shown in Table 6, the optimized classifier detects a number of comments with a SATD 85% lower than the classifier of the reference configuration.

Additionally, if we manually examine comments that were classified as SATD using the reference setup but not the optimized configuration, we discover that these are not necessarily false positives. For example, comments that have keywords like TODO, FIXME, and XXX are labeled "WITHOUT CLASSIFICATION" by the classifier with the optimized configuration. The same comments are detected as SATD by the classifier with the reference configuration. Some of these comments and their classification labels are shown in Table 7.

As a further consideration, we must mention that this study aims to compare the presence of SATD in blockchain projects with that in other open-source projects. Having chosen the study in [11] as a reference, the same configuration must be maintained.

Both considerations lead us to decide to use the reference configuration in [11] for SATD detection in blockchain projects.

Table 4. Classification Result with the configuration in [11] of 5768 samples in test set.

Class	TP	FN	FP	TN	Acc	P	R	F1
DESIGN	189	96	303	5183	0.913	0.380	0.660	0.482
REQUIREMENT	42	30	39	5657	0.988	0.519	0.583	0.549
WITHOUT CLASSIFICATION	5100	314	98	256	0.929	0.981	0.942	0.961
Micro-averaged accuracy/F1:	0.92372							
Macro-average F1:	0.66423							

Table 5. Classification result with the optimized configuration prop of 5768 samples in test set.

Class	TP	FN	FP	TN	Acc	P	R	F1
DESIGN	209	73	48	5438	0.979	0.813	0.741	0.776
REQUIREMENT	42	30	16	5680	0.992	0.724	0.583	0.646
WITHOUT CLASSIFICATION	5380	34	73	281	0.981	0.987	0.994	0.990
Micro-averaged accuracy/F1:	0.97625							
Macro-average F1:	0.80394							

Table 6. Comparison of classification results with the two configuration props, namely the reference configuration and the optimized configuration.

Project	Number of Comments	SATD Design. Reference | Optimized	SATD Requirement. Reference | Optimized
Avalanche	12,071	909 | 131	179 | 29

Table 7. Samples of comments and the comparison of the classification obtained with the models created via the reference prop file and via the optimized prop file.

Comment	Classification—Reference Prop	Classification—Optimized Prop
TODO actually disable crypto verification	REQUIREMENT	WITHOUT CLASSIFICATION
TODO Shutdown VM if an error occurs	REQUIREMENT	WITHOUT CLASSIFICATION
Probably because signature wrong	DESIGN	WITHOUT CLASSIFICATION
If p node Dispatch panics then we should log the panic and then re raise the panic This is why the above defer is broken into two parts	DESIGN	WITHOUT CLASSIFICATION

4. Results

This section presents the results of the study of the presence of the SATD in blockchain projects. The first part is intended to show the results of the classification of project comments, while the second part aims to compare what was obtained with the SATD detection measures in the baseline study.

4.1. Classification Results

Table 8 provides details on each of the projects analyzed in our study. The columns of the table show the results of the detection both in terms of the presence of SATD in the single comments and in terms of the presence of SATD in the files (where it is established that a file has SATD if at least one comment in it has been classified as SATD).

In general, the percentage of self-admitted technical debt for all blockchain projects is between 7.12% and 11.80% of comments, averaging 8.35%, with Terra and XRP projects having the highest percentage of SATD in comments, respectively 11.79% and 11.80% of comments. While the Ethereum and Bitcoin projects have the lowest percentages, at 7.13% and 7.72%, respectively. In all projects, the SATD classified as Design is more present than the SATD Requirement, and is on average about 4 times more present. However, the gap is more pronounced in projects like Binance and Terra where SATD Design is about 6.4 times more present, while it is smaller in Cardano where SATD Design is 1.9 times more present than the SATD Requirement. In terms of the percentage of files with SATD, the variation is more significant, ranging from 14.50% for Ethereum to 41.09% for Bitcoin, with an average of 23.02% of files where there is at least one comment classified as SATD. This means that among all the blockchain repositories, the Ethereum project has the lowest percentage of files containing SATD, while Bitcoin has the highest percentage of files containing SATD.

Table 8. Results of SATD detecting in Blockchain projects.

Project	Number of Comments	Number of SATD Design Comments	Number SATD Requirement Comments	Percentage of SATD Comments	Number of Examined Files	Files with SATD	Total % of Files with SATD
Bitcoin	17,887	1130	251	7.72%	988	406	41.09%
Ethereum	42,563	2486	546	7.13%	8334	1214	14.5%
Binance	1283	89	14	8.02%	174	39	22.4%
Usd coin	1533	109	28	8.93%	290	60	20.6%
Solana	27,520	1770	682	8.90%	2048	570	27.9%
Xrp	13,599	1316	289	11.80%	757	310	40.9%
Avalanche	11,880	870	176	8.80%	1472	381	25.9%
Cardano	24,112	1309	691	8.29%	2283	406	38.3%
Terra	1568	160	25	11.79%	542	88	16.2%
Polkadot	49,588	3023	1024	8.16%	2376	960	40.04%
Total	191,533	12,262	3726		19,264	4434	
Average	19,153.3	1226.2	372.6	8.35%	1926.4	443.4	23.02%

Table 9 shows the distribution of comments with SATD in the respective project files, showing that in the SATD files, the average number of comments detected varies from 2.10 to 5.19. In correspondence with projects with higher averages, we can also notice higher standard deviation values, indicating a possible high concentration of SATD comments in a limited number of files.

Table 9. Distribution of SATD comments in files.

Project	Files with SATD	Average Number of SATD Comments per File	Standard Deviation
Bitcoin	406	3.59	5.26
Ethereum	1214	2.52	4.75
Binance	39	2.53	3.37
Usd-coin	60	2.26	2.63
Solana	570	4.38	11.37
Xrp	310	5.19	10.53
Avalanche	381	2.84	3.35
Cardano	406	3.57	4.01
Terra	88	2.10	3.27
Polkadot	960	4.59	10.69

Based on the number of files with SATD, we can divide the projects into three sets. In particular, as reported in Table 10, four projects have a percentage of files with SATD lower than 25%, four projects have a percentage of files with SATD greater than 30%, while two projects have intermediate values.

Table 10. Projects by the percentage of files with SATD.

Concentration of Files with SATD Less than 25%	Concentration of Files with SATD between 25% and 30%	Concentration of Files with SATD over 30%
Ethereum	Avalanche	Cardano
Terra	Solana	Polkadot
Usd coin		Xrp
Binance		Bitcoin

4.2. Comparison of SATD Detection Results in Blockchain Projects and in Open-Source Java Projects

We compared the results previously reported with those contained in the reference research [11] for Java projects. The baseline study examines ten open source projects from different application domains, namely Ant, ArgoUML, Columba, EMF, Hibernate, JEdit, JFreeChart, JMeter, JRuby, and SQuirrel SQL. These selected projects are all written in JAVA. Table 11 reports the result of the SATD identification in this set of open-source project.

The comparison between blockchain projects and Java projects allows deducing whether the blockchain projects are more or less exposed to the presence of SATD. This is useful for understanding whether blockchain projects are more or less threatened by TD. In particular, the main difference between the two sets of projects concerns, in addition to the technology used, the fact that blockchain projects are characterized by the need to generate trust—trust that resides in the code—in users as a fundamental requirement for their success. In both sets of projects, the presence of SATD is a sign of unpaid TD and can be a symptom of future difficulties in maintaining the project, which manifests itself in delays in fixing malfunctions and vulnerabilities. In blockchain projects, this is intolerable and would lead to a loss of trust and, therefore, project failure.

First of all, some differences between the two sets of projects should be noted. First, unlike the baseline study, our selected blockchain projects contain source code written in several programming languages. Furthermore, the average number of contributors in blockchain projects is, on average, about five times higher than that of the Java projects examined in the benchmark study. Finally, the average percentage of comments filtered by us in the blockchain repositories is approximately 5%, while the average percentage of comments filtered in the reference study is 24%.

The results of the SATD detection show that the blockchain repositories selected in this study have an average percentage of SATD in comments of 8.25%, which is therefore

higher than the percentage found in the repositories selected in the reference study, where this percentage is 5.52%. So blockchain projects have on average nearly 50% more SATD comments than projects selected by the reference study, and all blockchain projects have a higher SATD comment rate than the average percentage calculated in the reference projects. From these results, it can be deduced that blockchain projects could contain more TDs than the average of the reference projects, which could be detrimental to the long-term maintainability of the projects and therefore to the generation of trust.

It should be noted that the reference projects are characterized by greater variability in the percentage of SATD, with a minimum SATD detection value of 2.05% of JEdit and a maximum of 14% of Hibernate, compared with the blockchain projects. Three of the reference projects have a SATD comment rate that exceeds the average value for blockchain projects, which could indicate a higher accumulation of technical debt compared with both blockchain projects and the other reference projects.

For both studies, it is noted that the percentages of comments classified as SATD Design are higher than those of SATD Requirement, but, while in blockchain projects the presence of SATD Design is about 4 times higher than that of Requirement, in the reference projects there is a presence of SATD Design about 7 times higher than that of SATD Requirement.

This indicates that in blockchain projects, compared with the reference study, the presence of debt requirements is more pronounced, and this is a symptom of a greater incidence of requirements that developers have not been able to fully implement.

Table 11. Results of SATD Detection in Open-Source Java Projects form [11].

Project	Contributors	Effective Comments	Number of SATD Design Comments	Number of SATD Requirement Comments	Percentage of SATD Comments
Ant	74	4137	95	12	2.58%
ArgoUML	87	9548	801	411	12.69%
Columba	9	6478	125	42	2.57%
EMF	30	4401	78	16	2.13%
Hibernate	226	2968	355	64	14.11%
JEdit	57	10,322	196	14	2.05%
JFreeChart	19	4423	184	15	4.49%
JMeter	33	8162	316	21	4.12%
JRuby	328	4897	343	110	9.25%
SQuirrel	46	7230	209	50	3.58%
Total	909	62,566	2702	755	-
Average	91	6256.6	270.2	75.5	5.52%

5. Discussion

The detection of SATD in blockchain projects has made it possible to evaluate the presence of two types of SATD, i.e., requirement and design, in ten selected projects. The results examined in the previous section allow us to answer RQ1: the findings reveal the common existence of SATD in the selected blockchain software systems, revealing and making visible a part of TD that should be paid off. Results showed the amount of SATD detected and evaluated its presence both at the level of comments and at the level of files. What emerges is that a percentage of comments between 7.13% and 11.40% was detected as SATD. As far as files are concerned, this percentage varies between 14.5% and 41.09%.

In projects where the SATD is distributed over many files (Bitcoin, XRP, Polkadot, and Cardano), the TD is distributed over a larger portion of the project. This can be problematic for long-term project maintainability if the development team does not adopt a TD monitoring strategy. In these projects, the identification of the TD through SATD is particularly useful as it allows to precisely locate the points of accumulation of TD and intervene for its removal. The Bitcoin and Polkadot projects, while not having the highest

percentage of SATD in comments, have a higher diffusion of SATD comments in files. This may require more future work to maintain and improve those projects.

Thanks to the results obtained in the comparison phase, it is also possible to answer RQ2. Blockchain projects have a higher average SATD percentage detected and are more evenly distributed than the projects in the reference study. It was also found that in blockchain projects, the detection rate of the SATD Design is less dominant than in the projects of the reference study. This indicates that blockchain developers are more likely to report inconsistencies between their implementation and project requirements. The results of this study allow us to observe that blockchain projects have a greater presence of SATD than the reference projects. This means that the developers of the blockchain projects are accumulating technical debt, and some of this is being detected through the identification of the SATD [15].

The high presence of SATD in blockchain projects can be seen as a threat to the bond of trust that must be generated between the blockchain system and the users for the project to be successful. For this reason, blockchain project development teams can benefit from SATD detection. The detection results allow the development team to locate the source of TD in the code and understand the problem that generated it thanks to the comment left by the developer.

Therefore, the detection of the SATD can be seen as a practice to allow the development team to make the payment of the TD (or its removal) faster and more punctual [14]. The SATD detection result can be used to monitor TD and gain insight into which aspects of blockchain projects require the most attention in order to be maintained and improved. Thanks to this, in the refactoring phase, it would be possible to effectively remove TD by punctually acting on the identified lines of code without requiring a review of all the project source codes. Adopting practices for timely monitoring and repaying TD can enable blockchain project development teams to gain trust from users, which is a key factor in the success of blockchain projects.

However, detected SATD reveals TD introduced proactively by programmers during the development process, and these programmers may be able to address these issues consciously during a future software refactoring. However, because blockchain systems are currently in a state of rapid development, application developers who use these open-source frameworks could have difficulty keeping up with these potential issues.

5.1. Threats to Validity

In this section, threats to the validity of our study will be examined. In particular, threats to internal, external, and construct validity are examined.

5.1.1. Internal Validity

Internal validity evaluates the causal relationship between an activity and observable changes. In our case, the data comes from a selection of ten projects that recall the characteristics of the chosen blockchains. So, according to this, examined projects may contain unknown and hidden elements that could influence the outcome; for example, different attitudes toward annotating the SATD or the manners used to annotate the SATD may have influenced the quality of the results. Likewise, the method adopted to detect SATD in the projects could also affect the outcomes. Although the SATD detection process replicates the reference work, it could not fully encompass the aspects related to the presence of different programming languages in blockchain projects.

5.1.2. External Validity

External validity represents the utility of the outcomes of research and their applicability in real-world settings. If a study has external validity, its findings will generalize to a larger population that was not included in the experiment. In our case, we limited the number of examined blockchain projects to ten. The limited number of projects could jeopardize the external validity of our findings. However, the set of projects was popu-

lated, including the most relevant blockchain projects and various typologies of blockchain projects. In addition, the SATD detection process we used has already been tested on a large number of projects, even if they are non-blockchain, and has been studied and analyzed in depth. The findings obtained from the blockchain projects confirmed, in some way, those obtained from other studies conducted in the reference study regarding the prevalence of SATD design over SATD requirements. So we can assume that the results are generalizable. However, more research is needed to corroborate the project-related findings and determine whether they can be generalized. In order to maximize performance, the real-world application of SATD detection techniques for TD management in the blockchain projects may necessitate revised training dataset creation and configuration phases.

5.1.3. Construct Validity

Construct validity refers to the degree to which inferences can be drawn from observed phenomena about the constructs that these instances may represent. The first threat to construct validity is that, in our work, our conclusions could be influenced by the quality of the dataset used for training the classifier, which was constructed by other people. In fact, we adopted the dataset used in the reference study. Another threat to construct validity is the fact that we focused our discussion on SATD detection and management as a relevant factor for blockchain projects success. However, there are numerous additional aspects that could affect the success of blockchain projects, such as, for example, the high number of contributors or the various tools and languages.

6. Conclusions

In this work, the study of the presence of SATD in blockchain projects has been addressed. The need for this study arises from the particular characteristics of blockchain projects, which could be sensitive to an accumulation of TD. First, blockchain technology is believed to be a new and fast-evolving technology, and furthermore, blockchain technology is designed to be trusted. The creation of SATDs by developers, if properly detected, can help the development team make the TD visible and manage it. In this study, SATD was detected in ten blockchain projects using NLP-based machine learning techniques. The results of the study reveal that all ten blockchain projects examined have comparable percentages of SATD in the comments, ranging between 7 and 11%, with the presence of SATD Design 4 times higher than that of SATD requirement. The major differences are seen at the file level with SATD, where it is noted that the distribution of SATD is more concentrated in some projects than others. By comparing what emerged by examining the SATD in ten projects in a reference study, it can be seen that the percentage of SATD in blockchain projects is on average higher, and more evenly distributed between the two types of SATD surveyed. This may indicate that developers of blockchain projects are more likely to leave explanatory comments about their choices when introducing TDs than those of the reference study projects. The SATD detection result can be used to track TD and learn which elements of blockchain projects need the greatest attention in order to be maintained and developed. As a result, during the refactoring phase, it would be easy to successfully remove TD by acting on the recognized lines of code on a regular basis without having to review all of the project source codes. Adopting policies for prompt monitoring and repayment of TD could help blockchain project development teams maintain user trust, which is critical to project success.

Author Contributions: Conceptualization, A.P.; methodology, A.P., R.T. and M.I.L.; software, S.O. and A.P.; validation, R.T. and M.I.L.; investigation, S.O., A.P. and M.I.L.; formal analysis, S.O. and A.P.; data curation, S.O. and A.P.; writing—original draft preparation, A.P. and M.I.L.; writing—review and editing, A.P. and M.I.L.; supervision, R.T.; project administration, R.T.; funding acquisition, R.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union-NextGenerationEU. We acknowledge financial support under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.5-Call for tender No. 3277 published on 30 December 2021 by the Italian Ministry of University and Research (MUR) funded by the European Union-NextGenerationEU. Project Code ECS0000038—Project Title eINS Ecosystem of Innovation for Next Generation Sardinia—CUP F53C22000430001-Grant Assignment Decree No. 1056 adopted on 23 June 2022 by the Italian Ministry of University and Research (MUR). This work was partially funded by the "W.E. B.E.S.T. Wine EVOO Blockchain Et Smart ContracT" PRIN 2020 financed by the Italian Ministry of University and Research (MUR), CUP: F73C22000430001.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lilienthal, C. *Sustainable Software Architecture: Analyze and Reduce Technical Debt*; Rocky Nook: Heidelberg, Germany, 2019.
2. Alliance, A.; Letouzey, J.-L.; Whelan, D. Introduction to the Technical Debt Concept. 2016. Available online: https://www.agilealliance.org/introduction-to-the-technical-debt-concept (accessed on 10 May 2023).
3. Cunningham, W. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* **1992**, *4*, 29–30. [CrossRef]
4. Ibba, S.; Pinna, A.; Lunesu, M.I.; Marchesi, M.; Tonelli, R. Initial coin offerings and agile practices. *Future Internet* **2018**, *10*, 103. [CrossRef]
5. Huang, Q.; Shihab, E.; Xia, X.; Lo, D.; Li, S. Identifying Self-Admitted Technical Debt in Open Source Projects Using Text Mining. *Empir. Softw. Eng.* **2018**, *23*, 418–451. [CrossRef]
6. Li, Y.; Soliman, M.; Avgeriou, P. Automatic identification of self-admitted technical debt from four different sources. *Empir. Softw. Eng.* **2023**, *28*, 65. [CrossRef]
7. Potdar, A.; Shihab, E. An Exploratory Study on Self-Admitted Technical Debt. In Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, 29 September–3 October 2014; pp. 91–100. [CrossRef]
8. Wehaibi, S.; Shihab, E.; Guerrouj, L. Examining the impact of self-admitted technical debt on software quality. In Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Osaka, Japan, 14–18 March 2016; Volume 1, pp. 179–188.
9. Ibba, G.; Ortu, M.; Tonelli, R. Analysis of Topics Related To Smart Contracts on Social Media. In Proceedings of the PoEM'2022 Workshops and Models at Work Papers, London, UK, 23–25 November 2022.
10. Ortu, M.; Ibba, G.; Conversano, C.; Tonelli, R. Smart Topics: Designing an Ethreurm Smart Contracts Environment Knowledge-Base Using Natural Language Processing, Social Media and Complex Network Theory. *SSRN* **2023**. [CrossRef]
11. Maldonado, E.d.S.; Shihab, E.; Tsantalis, N. Using Natural Language Processing to Automatically Detect Self-Admitted Technical Debt. *IEEE Trans. Softw. Eng.* **2017**, *43*, 1044–1062. [CrossRef]
12. Mensah, S.; Keung, J.; Svajlenko, J.; Bennin, K.E.; Mi, Q. On the value of a prioritization scheme for resolving Self-admitted technical debt. *J. Syst. Softw.* **2018**, *135*, 37–54. [CrossRef]
13. Casey, M.J.; Vigna, P. In blockchain we trust. *MIT Technol. Rev.* **2018**, *121*, 10–16.
14. Maldonado, E.D.S.; Abdalkareem, R.; Shihab, E.; Serebrenik, A. An empirical study on the removal of self-admitted technical debt. In Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 17–22 September 2017; pp. 238–248.
15. Pavlič, L.; Hliš, T.; Heričko, M.; Beranič, T. The Gap between the Admitted and the Measured Technical Debt: An Empirical Study. *Appl. Sci.* **2022**, *12*, 7482. [CrossRef]
16. Khomyakov, I.; Makhmutov, Z.; Mirgalimova, R.; Sillitti, A. An analysis of automated technical debt measurement. In Proceedings of the Enterprise Information Systems: 21st International Conference, ICEIS 2019, Heraklion, Greece, 3–5 May 2019; Revised Selected Papers 21; Springer: Berlin/Heidelberg, Germany, 2020; pp. 250–273.
17. Lenarduzzi, V.; Martini, A.; Taibi, D.; Tamburri, D.A. Towards surgically-precise technical debt estimation: Early results and research roadmap. In Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, Tallinn, Estonia, 27 August 2019; pp. 37–42.
18. Tsoukalas, D.; Siavvas, M.; Jankovic, M.; Kehagias, D.; Chatzigeorgiou, A.; Tzovaras, D. Methods and Tools for TD Estimation and Forecasting: A State-of-the-art Survey. In Proceedings of the 2018 International Conference on Intelligent Systems (IS), Funchal, Portugal, 25–27 September 2018; pp. 698–705.
19. Alfayez, R.; Winn, R.; Alwehaibi, W.; Venson, E.; Boehm, B. How SonarQube-identified technical debt is prioritized: An exploratory case study. *Inf. Softw. Technol.* **2023**, *156*, 107147. [CrossRef]
20. Zampetti, F.; Serebrenik, A.; Di Penta, M. Was self-admitted technical debt removal a real removal? An in-depth perspective. In Proceedings of the 15th International Conference on Mining Software Repositories, Gothenburg, Sweden, 28–29 May 2018; pp. 526–536.
21. Sierra, G.; Shihab, E.; Kamei, Y. A survey of self-admitted technical debt. *J. Syst. Softw.* **2019**, *152*, 70–82. [CrossRef]

22. Liu, Z.; Huang, Q.; Xia, X.; Shihab, E.; Lo, D.; Li, S. SATD detector: A text-mining-based self-admitted technical debt detection tool. In Proceedings of the 40th ACM/IEEE International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 27 May–3 June 2018; Volume 3, pp. 9–12.
23. Li, Y.; Soliman, M.; Avgeriou, P.; Somers, L. Self-Admitted Technical Debt in the Embedded Systems Industry: An Exploratory Case Study. *IEEE Trans. Softw. Eng.* **2023**, *49*, 2545–2565. [CrossRef]
24. Brownlee, J. *Deep Learning for Natural Language Processing: Develop Deep Learning Models for Your Natural Language Problems*; Machine Learning Mastery: Victoria, Australia, 2017.
25. Wattanakriengkrai, S.; Maipradit, R.; Hata, H.; Choetkiertikul, M.; Sunetnanta, T.; Matsumoto, K. Identifying Design and Requirement Self-Admitted Technical Debt Using N-gram IDF. In Proceedings of the 2018 9th International Workshop on Empirical Software Engineering in Practice (IWESEP), Nara, Japan, 4 December 2018; pp. 7–12. [CrossRef]
26. Kim, Y. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; Association for Computational Linguistics: Doha, Qatar, 2014; pp. 1746–1751. [CrossRef]
27. Ren, X.; Xing, Z.; Xia, X.; Lo, D.; Wang, X.; Grundy, J. Neural Network-Based Detection of Self-Admitted Technical Debt: From Performance to Explainability. *ACM Trans. Softw. Eng. Methodol.* **2019**, *28*, 1–45. [CrossRef]
28. Li, Y.; Soliman, M.; Avgeriou, P. Identifying Self-Admitted Technical Debt in Issue Tracking Systems Using Machine Learning. *Empir. Softw. Eng.* **2022**, *27*, 131. [CrossRef]
29. Yu, J.; Zhou, X.; Liu, X.; Liu, J.; Xie, Z.; Zhao, K. Detecting Multi-Type Self-Admitted Technical Debt with Generative Adversarial Network-Based Neural Networks. *Inf. Softw. Technol.* **2023**, *158*, 107190. [CrossRef]
30. Zampetti, F.; Serebrenik, A.; Di Penta, M. Automatically Learning Patterns for Self-Admitted Technical Debt Removal. In Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 18–21 February 2020; pp. 355–366. [CrossRef]
31. Wüst, K.; Gervais, A. Do you need a blockchain? In Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), Zug, Switzerland, 20–22 June 2018; pp. 45–54.
32. Ahmadjee, S.; Mera-Gómez, C.; Bahsoon, R. Assessing Smart Contracts Security Technical Debts. In Proceedings of the 2021 IEEE/ACM International Conference on Technical Debt (TechDebt), Madrid, Spain, 19–21 May 2021; pp. 6–15. [CrossRef]
33. Ahmadjee, S.; Bahsoon, R. A Taxonomy for Understanding the Security Technical Debts in Blockchain Based Systems. *arXiv* **2019**, arXiv:1903.03323.
34. Qu, Y.; Bao, T.; Chen, X.; Li, L.; Dou, X.; Yuan, M.; Wang, H. Do we need to pay technical debt in blockchain software systems? *Connect. Sci.* **2022**, *34*, 2026–2047.
35. Qu, Y.; Wong, W.E.; Li, D. Empirical Research for Self-Admitted Technical Debt Detection in Blockchain Software Projects. *Int. J. Perform. Eng.* **2022**, *18*, 149. [CrossRef]
36. Group, S.N. ColumnDataClassifier (Stanford JavaNLP API). Available online: https://nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/classify/ColumnDataClassifier.html (accessed on 10 June 2023).
37. Manning, C.; Klein, D. Optimization, Maxent Models, and Conditional Estimation without Magic. In Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: Tutorials, Edmonton, AB, Canada, 27 May–1 June 2003.
38. Curran, J. *Maximum Entropy Models for Natural Language Processing*; Australasian Language Technology Summer School: Sydney, Australia, 2004.