Empirical Software Engineering
https://doi.org/10.1007/s10664-019-09796-5

An exploratory study of smart contracts in the Ethereum blockchain platform

Gustavo A. Oliva[1] · **Ahmed E. Hassan**[1] · **Zhen Ming (Jack) Jiang**[2]

Published online: 12 March 2020

Abstract

Ethereum is a blockchain platform that supports smart contracts. Smart contracts are pieces of code that perform general-purpose computations. For instance, smart contracts have been used to implement crowdfunding initiatives that raised a total of US$6.2 billion from January to June of 2018. In this paper, we conduct an exploratory study of smart contracts. Differently from prior studies that focused on particular aspects of a subset of smart contracts, our goal is to have a broader understanding of all contracts that are currently deployed in Ethereum. In particular, we elucidate how frequently used the contracts are (activity level), what they do (category), and how complex they are (source code complexity). To conduct this study, we mined and cross-linked data from four sources: Ethereum dataset on the Google BigQuery platform, Etherscan, State of the DApps, and CoinMarketCap. Our study period runs from July 2015 (inception of Ethereum) until September 2018. With regards to activity level, we notice that it is concentrated on a very small subset of the contracts. More specifically, only 0.05% of the smart contracts are the target of 80% of the transactions that are sent to contracts. New solutions to cope with Ethereum's limited scalability should take such an activity imbalance into consideration. With regards to categories, we highlight that the new and widely advertised rich programming model of smart contracts is currently being used to develop very simple applications that tend to be token-centric (e.g., ICOs, Crowdsales, etc). Finally, with regards to code complexity, we observe that the source code of high-activity verified contracts is small, with at most 211 instructions in 80% of the cases. These contracts also commonly include at least two subcontracts and libraries in their source code. The comment ratio of these contracts is also significantly higher than that of GitHub top-starred projects written in Java, C++, and C#. Hence, the source code of high-activity verified smart contracts exhibit particular complexity characteristics compared to other popular programming languages. Further studies are necessary to uncover the actual reasons behind such differences. Finally, based on our findings, we propose an open research agenda to drive and foster future studies in the area.

Keywords Smart contracts · Ethereum · Blockchain

Communicated by: Arie van Deursen

✉ Gustavo A. Oliva
gustavo@cs.queensu.ca

Extended author information available on the last page of the article.

1 Introduction

Ethereum is a blockchain platform (Wood 2017). A blockchain platform is a distributed, chronological database of transactions that is shared and maintained across nodes that participate in a peer-to-peer network (Swan 2015). The decentralized architecture of a blockchain enables transactions to be confirmed without the need of a central entity (e.g., a bank or a credit card company).

The Ethereum platform was crowdfunded in the second semester of 2014 and raised approximately US$15.6 million.[1] As of August 2018, the Ethereum platform has achieved an impressive market capitalization of US$42.3 billion.[2] The Ethereum platform is slowly gaining more and more attention from the media, with articles about it in traditional media outlets such as The Economist (2018) and The New York Times (2017). Enterprise versions of the Ethereum platform are also already being conceived with the support of industry-leading companies such as Intel, Microsoft, J.P. Morgan, and Accenture.[3]

At the heart of the Ethereum platform are *smart contracts* (Szabo 1994). A smart contract is simply a non-modifiable general purpose computer program. Different from app stores, Ethereum not only hosts *smart contracts*, but also executes them. Smart contracts are written in Solidity, whose syntax is similar to that of JavaScript. Smart contracts are commonly used to create a tradeable digital *token*, which can represent a currency, an asset, a virtual share, a proof of membership, etc. Smart contracts also commonly define how these tokens are meant to be distributed. For instance, a smart contract might define a token with a fixed supply or even act as a central bank that can issue tokens.

Smart contracts have been successfully used to implement *crowdfunding initiatives* for startup companies. Typically, a company creates a smart contract that defines a new token (in this case, a cryptocurrency), sets a certain supply for this token, defines the presale period, defines safeguards measures for investors (e.g., the contract might hold the money of an investor until a specific date or goal is reached), and operationalizes the sale itself. And all of this happens without the need of a centralized arbitrator or clearinghouse. Such crowdfunding contracts are often referred to as *initial coin offerings* (ICOs). According to the ICO WatchList[4] website, a record of US$6.2 billion were raised via ICOs from January to June in 2018. According to the same website, 82.2% of all their tracked ICOs were implemented as smart contracts on the Ethereum platform.

The former example shows how smart contracts can be used as stand-alone applications. However, smart contracts can also be used to implement the *backend* of a certain application. Such blockchain-powered applications are advertised by the Ethereum platform creators as *decentralized applications* (ĐApps). One prominent example is the CryptoKitties game,[5] which allows users to purchase, collect, breed, and sell virtual cats. The craze

[1] The dollar amount reported considers the exchange rate during which the crowdfunding took place (Jul 22th 2014 until Sep 2nd 2014). We proceed analogously for all other crowdfunding amounts reported in this paper. More information is available at: https://cryptoslate.com/coins/ethereum/

[2] *Market capitalization* is the multiplication of a company's shares by its current stock price. In the virtual coin world, a company's share corresponds to the total coin supply. As of August 3rd 2018, Ethereum has a total ether supply of 101,104,524 with a market price of US$418.26 per ether, yielding an impressive market capitalization of US$42.3 billion. More information is available at https://coinmarketcap.com/currencies/ethereum/historical-data

[3] https://entethalliance.org

[4] https://icowatchlist.com/

[5] https://cryptokitties.co

around the game was so intense that it has once managed to significantly slow down all transaction processing on Ethereum (BBC 2017). Some virtual cats have been sold for more than US$100,000 (exchange rate taken at time of sale[6]).

With the recent popularity of smart contracts and their inherent association with financial transactions, the software engineering research community has primarily focused on examining the security aspects of smart contracts (Luu et al. 2016; Kalra et al. 2018; Chen et al. 2018; Tikhomirov et al. 2018; Grishchenko et al. 2018; Wan et al. 2017). As such, a more global view of smart contracts remained uncharted. In this paper, we conduct an exploratory study in which we aim to characterize three fundamental properties of smart contracts, namely: *activity level*, *category*, and *source code complexity*. Our investigation of activity level aims to explore how intensively smart contracts have been used and how concentrated this usage is. Our investigation of categories aims to uncover what smart contracts have been used for. Finally, our investigation of code complexity aims to quantify complexity according to several dimensions.

We study all smart contracts in the Ethereum platform that were created via *contract creation transactions* (1.9M contracts). We characterize smart contracts by cross-linking data from several sources: Ethereum dataset in Google BigQuery,[7] Etherscan[8] (primary Ethereum explorer website), State of the ĐApps[9] (curated list of ĐApps), and CoinMarketCap[10] (coin market capitalization tracker). In the following, we list our research questions and key results:

RQ1) What is the activity level of smart contracts? *Activity is concentrated on a very small subset of the contracts. More specifically, only 0.05% of the smart contracts are the target of 80% of the transactions that are sent to contracts. The source code is available for 73.1% of these high-activity contracts.*

RQ2) What are the categories of high-activity smart contracts? *Despite the hype around blockchain-powered applications and their presumable suitability for several use-cases, our results suggest that at least 41.3% of the high-activity contracts revolve around transferring, selling, and distributing tokens. In particular, 5 out of the top-10 contracts with highest activity belong to Currency Exchange ĐApps.*

RQ3) How complex is the source code of verified smart contracts? *The source code of high-activity verified contracts is small, with at most 211 instructions in 80% of the cases. These contracts also commonly include at least two subcontracts and libraries in their source code. The comment ratio of these contracts is significantly higher than that of GitHub top-starred projects written in Java, C++, and C#.*

In summary, our empirical results led us to conclude that: (i) new solutions to cope with Ethereum's limited scalability should take the activity imbalance into consideration, (ii) researchers and practitioners should be aware that the main application of smart contracts at this time is constrained to simple token management, and (iii) source code of high-activity verified smart contracts exhibit particular complexity characteristics compared to other popular programming languages.

[6] https://kittysales.herokuapp.com
[7] https://bigquery.cloud.google.com/dataset/bigquery-public-data:ethereum_blockchain
[8] https://etherscan.io
[9] https://www.stateofthedapps.com
[10] https://coinmarketcap.com

Key contributions:

1. Provision of empirically sound evidence from cross-linked data regarding the usage of smart contracts (in contrast to the large amount of grey literature found in the Internet).
2. Elucidation of key properties of smart contracts currently deployed in Etherscan, as a way to characterize the state-of-the-practice.
3. Proposal of an open research agenda to guide future software engineering research in the topic.
4. Provision of a replication package with our preprocessed data[a] as a means to enable more in-depth and varied studies on this important area throughout our community.

[a] https://github.com/SAILResearch/replication-smart_contracts_overview

Paper Organization The remainder of this paper is organized as follows. Section 2 defines key concepts surrounding smart contracts. Section 3 explains the data collection procedures that we employed to obtain the smart contracts and their associated metadata. Section 4 presents and discusses the results of our research questions. Section 5 introduces an open research agenda to guide future studies in smart contracts. Section 6 presents the different perspectives from which prior research has studied smart contracts. Section 7 discusses the threats to the validity of our study. Finally, Section 8 concludes the paper.

2 Background: Blockchain, Smart Contracts, and Related Concepts

In this section, we describe concepts that are key to our study. Sections 2.1 and 2.2 define *blockchain* and *smart contracts* respectively. Sections 2.3 and 2.4 explain how one *interacts with* and *verifies* smart contracts respectively. Section 2.5 defines *token*, *token contracts*, and *mintable token contracts*. Finally, Section 2.6 introduces Ð*Apps*.

2.1 Blockchain

A blockchain is a distributed, chronological database of transactions. This database is shared and maintained across nodes that participate in a peer-to-peer network. The name *blockchain* comes from the manner in which transactions are stored. More specifically, transactions are packaged into blocks and these blocks are linked to one another as a chain. There are currently several offerings of blockchain platforms. Popular ones include: Bitcoin,[11], Ethereum,[12] EOS,[13] POA,[14] Nxt,[15] and Hyperledger Fabric.[16]

Adding a new transaction to a blockchain requires confirmation from several nodes of the network, which all abide to a certain consensus protocol. The Ethereum platform uses

[11] https://bitcoin.org
[12] https://ethereum.org
[13] https://eos.io
[14] https://www.poa.network
[15] https://www.jelurida.com/nxt
[16] https://www.hyperledger.org/projects/fabric

the computationally costly *Proof-of-Work (PoW)* consensus protocol (Jakobsson and Juels 1999), which requires nodes to solve a hard mathematical puzzle. The computing power required to solve the hard mathematical puzzle ensures that tampering with the data is infeasible. In particular, the PoW consensus protocol ensures that there is no better strategy to find the solution to the mathematical puzzle than enumerating the possibilities (i.e., brute force). On the other hand, verification of a solution is trivial and cheap. Ultimately, the PoW consensus protocol ensures that a trustworthy third-party (e.g., a bank) is *not* needed in order to validate transactions, enabling entities who do not know or trust each other to build a dependable transaction ledger.

Other consensus protocols include Proof-of-Stake (e.g., used by Nxt) and Delegated Proof-Of-Stake (e.g., used by EOS). In Proof-of-Stake protocols,[17] "a set of validators take turns proposing and voting on the next block, and the weight of each validator's vote depends on the size of its deposit (i.e. stake)". Proof-of-Stake thus do not consume large quantities of electricity and discourage centralized cartels. Ethereum developers plan to transition from Proof-of-Work to Proof-of-Stake in 2020.

Independently of the consensus protocol, once a block is appended to a blockchain, its contents cannot be altered without changing every other block that came after it. In practice, a transaction in Ethereum is deemed final and irreversible after six block confirmations (i.e., after six new blocks have been added to blockchain). More generally, due to the PoW consensus protocol, it is impossible to change the contents of old blocks without owning more than 50% of the computing power that runs Ethereum (a.k.a., a 51% attack).

An introduction to blockchain platforms and surrounding concepts can be found in the books by Swan (2015) and Richmond (2018).

2.2 Smart Contracts

Ethereum supports *smart contracts*. Blockchain platforms that support smart contracts are often referred to as *programmable blockchains* (Bitcoin is *not* a programmable blockchain).

The term *smart contract* was coined by Szabo (1994). According to him, "a smart contract is a computerized transaction protocol that executes the terms of a contract." More recently, with the advent of Ethereum and other sophisticated blockchain platforms, the concept of smart contracts has become much broader, representing any general-purpose computation.

Ethereum both hosts and executes smart contracts. This is a key distinction compared to mobile app stores, which only host applications. The execution aspect, coupled with the ledger characteristics of a blockchain, enables keeping track of the activity of smart contracts.

Source Code and Bytecode The source code of a smart contract is typically written in Solidity, whose syntax is similar to that of Javascript. The structure of a Solidity smart contract resembles that of a class (as in object-oriented programming). An illustrative example is shown in Fig. 1. Similarly to the Java compiler, the Solidity compiler also produces a bytecode version of the source code, which is executed by the Ethereum Virtual Machine (EVM). The Ethereum bytecode is an assembly language made up of several opcodes (low-level instructions). Most importantly, only the bytecode of a smart contract is stored in

[17] https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ


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

Fig. 1 An example of a smart contract written in Solidity

Ethereum. More details about Solidity and its bytecode format can be found in Solidity's official documentation.[18]

Subcontracts and Libraries In order to enable separation of concerns, the Solidity language provides *subcontracts* and *libraries* constructs. Similarly to nested classes (as in object-oriented programming), subcontracts and libraries have each a specific purpose. Subcontracts enable developers to establish object-oriented relationships between contracts, such as inheritance and interface implementation. Libraries often provide a set of utility methods that are mindful of corner-cases or optimize processing time (e.g., a library to perform mathematical operations without incurring into underflows and underflows). In the example depicted in Fig. 1, there are two subcontracts (`ERC20` and `MyCoin`) and two libraries (`SafeMath` and `VectorSum`).

2.3 Interacting with Smart Contracts

The Ethereum platform supports two types of accounts: *user accounts* and *smart contract accounts*. Both accounts are uniquely identified with a 40-digit hexadecimal ID, which is often referred to as the *address* of the account. Similarly to accounts, all transactions are also uniquely identified by a 40-digit hexadecimal ID.

A user account can *deploy* contracts to the blockchain and *interact* with them. The deployment is done by means of a transaction[19] sent to the blockchain. This transaction is commonly referred to as the *contract creation transaction*. A smart contract receives its address as a result of the execution of the contract creation transaction. A smart contract can also be deployed by an existing contract. This process occurs through a mechanism commonly known as an *internal transaction*. Internal transactions are not real transactions, as they are not kept on the blockchain. Our study does not cover contracts created via internal transactions.

A user account interacts with a contract by sending transactions to its functions. These transactions are ultimately *function calls*. Each smart contract transaction burns a certain amount of *gas* units depending on which and how many instructions are executed during runtime. This concept is known as the *gas usage* of a transaction. Hence, in order to send a transaction, the user has to set two parameters: *gas price* and *gas limit*. Gas price is the amount of money that the user is willing to pay for one unit of gas. Gas price is given in Ether, which is Ethereum's cryptocurrency (Section 2.5). Gas limit is the maximum amount of gas units that the user is willing to pay for. The transaction fee that the user has to pay is thus $gasprice \times gasusage$, which is at most $gasprice \times gaslimit$. Gas payment covers the costs of the computing power.

2.4 Verifying Smart Contracts

When a developer uploads a smart contract to the Ethereum platform, only its bytecode version is stored. Therefore, it is up to the developer to publish the corresponding source code somewhere. The Etherscan website, which is the primary Ethereum explorer website,

[18] https://solidity.readthedocs.io

[19] Example of a transaction that created a smart contract: https://etherscan.io/tx/0xebcbe706f9959c8b98a72bcd42fed545d3cf60fe3fa801186d5fef2249dac91a

provides a code transparency mechanism known as *contract verification*. This mechanism offers developers the possibility of publishing the source code of their contracts in Etherscan, so that it is available to the whole community that is interested in the Ethereum platform. The code verification mechanism works as follows: (i) the developer uploads the source code and indicates a particular version of the Solidity compiler, (ii) Etherscan compiles the code using the developer-indicated compiler version, (iii) Etherscan checks if the generated bytecode matches the bytecode that is stored on the blockchain. If there is a perfect match, then the contract is deemed as *verified* and the source code becomes publicly available on Etherscan in less than 5 minutes.[20]

2.5 Cryptocurrency, Tokens, and Coins

A *cryptocurrency* is digital and represents money. A cryptocurrency is native to its own blockchain. In the case of Ethereum, the cryptocurrency is called Ether and is abbreviated as ETH. Ether can be transferred between user accounts. In addition, as mentioned in Section 2.3, Ether must also be paid by anyone who wants to run a transaction on the Ethereum platform, as it covers the costs of computing power. In terms of usage, Ether (ETH) is not much different than traditional currencies like USD Dollars (USD) and Euros (EUR). The only practical difference is that we have metal coins and pieces of paper to represent dollars and euros in the physical world. Instead, cryptocurrencies are purely virtual.

Tokens are created on top of existing blockchains. Tokens are used to represent digital assets that are tradeable (and usually fungible), including everything from commodities to voting rights. Every token has a name and an acronym (popularly known as a *symbol*) and any smart contract can define a new token. It is common for tokens to represent money. Therefore, in practice, (crypto)coins and tokens are frequently used interchangeably. For instance, a crowdfunding initiative that is implemented as a distribution of tokens is more commonly referred to as an ICO (Initial *Coin* Offering) instead of an ITO (Initial *Token* Offering). There are several physical and virtual currency exchanges[21] around the world that buy and sell (crypto)coins, as well as exchange one (crypto)coin for another.

Token Contract A *token contract* is a special kind of smart contract that defines a token and keeps track of its balance across user accounts. Ethereum has two main technical standards for the implementation of tokens, known as the *ERC20 interface* and the *ERC721 interface*. The standardization allows contracts to operate on different tokens seamlessly, while also fostering interoperability between contracts. From an implementation perspective, ERC20 and ERC721 interfaces are object-oriented interfaces defining several functions, such as `totalSupply()`, `balanceOf(address who)`, and `transfer(address to, uint256 value)` (check IERC20 in Fig. 1). In this paper, we study both token contracts and non-token contracts.

Mintable Token and Mintable Token Contact A *mintable token* is a special kind of token that has a *non-fixed total supply*. Most *mintable token contracts* are ERC20 token contracts with an added `mint()` function, which increases the total token supply upon invocation.

[20] https://etherscan.io/apis#contracts

[21] Examples include IDEX (https://idex.market), ForkDelta (https://forkdelta.app), and Bancor (https://www.bancor.network). IDEX is described in Appendix.

Optionally, a `burn()` function is also included to decrease the total supply. Bitcoin (BTC), the official cryptocurrency of the homonymous blockchain plaform, is a mintable token. In particular, 12.5 newly created BTCs are given as a reward to those who put the next block on the Bitcoin blockchain platform.

2.6 Decentralized Applications (ĐApps)

A smart contract can be an application in itself. For instance, a smart contract might define a token (coin) and implement an ICO for it. Alternatively, a smart contract might be used as the backend of a multi-tiered application. In this sense, a smart contract can also be a building block of something larger.

Any application that relies on one or more smart contracts as its backend is known as a *decentralized application* (ĐApp). Therefore, ĐApps contrast to more traditional applications in which the backend code runs on centralized servers. The frontend of ĐApps can be implemented in any programming language. Commonly, it is implemented with a combination of HTML5, CSS, and `web3.js` (a Javascript API for Ethereum).

As we mentioned earlier, a list of curated ĐApps can be seen on the State of the ĐApps website.

3 Data Collection

Conducting our study involved cross-linking data from several sources, namely: Google BigQuery, Etherscan, CoinMarketCap, and State of the ĐApps. Google BigQuery is a scalable data analytics platform that contains several public datasets, including one for the Ethereum platform. Google set up nodes on their cloud infrastructure that synchronize with the Ethereum blockchain. The data is updated daily. Etherscan is the most popular Ethereum explorer. Similarly to Google, Etherscan collects the data by having nodes set up on the Ethereum blockchain. The data is updated on their website in real time, allowing users to explore Ethereum through a web browser. CoinMarketCap is a website that tracks the market capitalization of cryptocurrencies. Finally, The State of the ĐApps is a website with a curated list of ĐApps. Each ĐApp has an associated webpage in State of the ĐApps, where relevant metadata is shown, including the category of the ĐApp, its description, and the address(es) of the smart contract(s) it uses (if available).

In the following, we describe the specific pieces of data on which we relied in order to answer our research questions. An overview of the data collection process is shown below in Fig. 2.

1: **Retrieve addresses, number of transactions, and token symbol of all smart contracts.** We relied on the live public Ethereum dataset available in Google BigQuery to determine the list of all contracts and their associated metadata, including number of received transactions and token symbol (for token contracts). We do note that this dataset does not track contracts created via internal transactions (Section 2.3).
2: **Identify verified contracts.** We used the REST APIs provided by Etherscan to determine the list of verified contracts.[22] We obtained 42,256 verified contracts.

[22] https://etherscan.io/apis#contracts

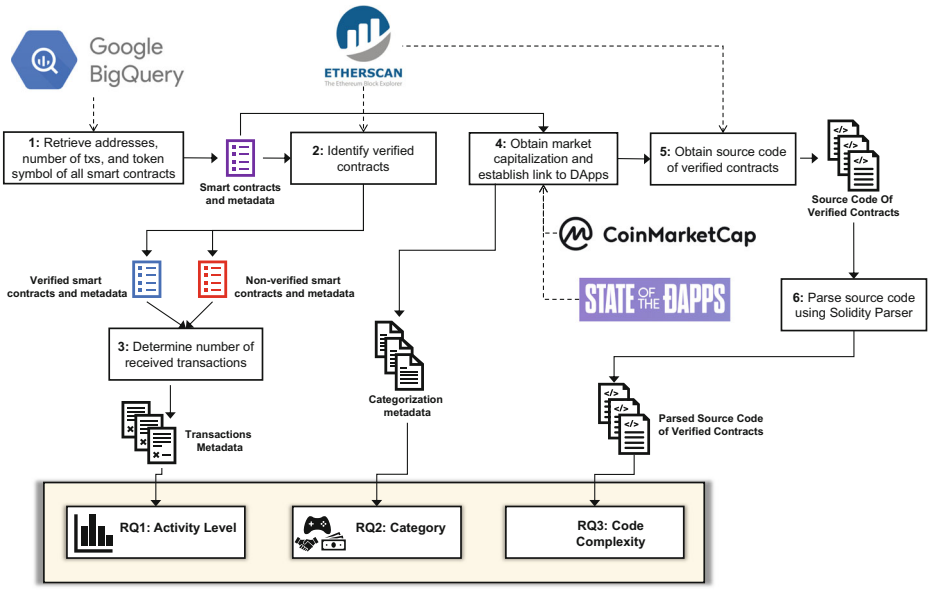

Fig. 2 An overview of our data collection process. Dashed lines indicate a connection to a data source

3: Determine number of received transactions. We analyzed the number of received transactions by verified and non-verified contracts. The output of this step was the main source of information for tackling RQ1.

4. Obtain market capitalization and establish link to ĐApps. Since cryptocurrencies play a key role in Ethereum, we determined their market capitalization using data from the CoinMarketCap website. In addition, we wanted to discover the category of each smart contract. To this end, we also linked smart contracts to the list of ĐApps from State of the ĐApps, using the contract address field as the key. Performing this linkage involved crawling the webpage of each ĐApp available on State of the ĐApps. We emphasize, however, that not all contracts belong to a ĐApp. The output of this step was the main source of information for tackling RQ2.

5: Obtain source code of smart contracts. We used Etherscan's REST API to download the source code of verified contracts. We highlight that verified contracts are provided with the *explicit* expectation that they are not private (i.e., they are provided so that others can examine and scrutinize them). In addition, we do not reuse nor redistribute source code, and thus we do not infringe on copyright or code licenses. Hence, our usage of the source code is in accordance to Etherscan's usage terms.[23]

6: Parse source code using Solidity Parser. In order to investigate the complexity of the source code, as well as to identify subcontracts, libraries, and code comments, we wrote a JavaScript program that works on top of a Solidity parser made available by Federico Bond.[24] The parsed source code pieces were the main source of information for tackling RQ3.

[23] https://etherscan.io/source-code-usage-terms

[24] https://github.com/federicobond/solidity-parser-antlr

Data collection summary:

- Analysis period: July 30th 2015 until September 15th 2018
- Data sources: Google Big Query, Etherscan, State of the ĐApps, CoinMarketCap
- Pieces of data collected: smart contract metadata, smart contract source code, ĐApps metadata, market capitalization of tokens
- Number of studied contracts: 1,953,337
- Number of verified contracts: 42,256 (2.2% of all contracts)

4 Results

4.1 RQ1: What is the Activity Level of Smart Contracts?

Motivation The hype around blockchain technology and the success of games like CryptoKitties might persuade researchers and industries into thinking that platforms such as Ethereum host a vast number of killer applications. Hence, in this research question, we analyze the activity level of smart contracts and determine how concentrated it is.

Approach We use the number of transactions that a contract received as a proxy for their activity level. We analyze the distribution of the number of received transactions in order to gain insights into contract usage.

As described in Section 3, our data contains the number of transactions that a contract received since its creation until the day before we started the data collection process. We refer to this timeframe as the *age* of a contract. As part of this research question, we also investigate whether there is correlation between the number of transactions that a contract received and its age.

Finally, we investigate the *current idle time* of contracts, which we define as the time elapsed between a contract's last transaction and the data collection date. We analyze current idle times in order to determine the fraction of contracts that are no longer active. More specifically, we cluster the current idle time observations using the *Jenks natural breaks optimization* method (Jenks and Caspall 1971) with the support of the `classInt` R package.[25] The Jenks method is designed to determine the best arrangement of values into different classes by reducing the variance within classes and maximizing the variance between classes. The number of clusters n_c is an input parameter. We analyze the empirical cumulative distribution of the current idle time variable in order to pick n_c and then evaluate our choice using the goodness-of-fit measure, which indicates how well the Jenks clustering fits the data.

We use the Spearman's rank correlation coefficient (ρ) to evaluate the relationship between variables (e.g., number of transaction and the age of a contract). We choose this correlation coefficient because it does not require normally distributed data. We assess Sperman's ρ using the following thresholds (Evans 1995), which operate on 2 decimal places only: *very weak* for $|\rho| \leq 0.19$, *weak* for $0.20 \leq |\rho| \leq 0.39$, *moderate* for $0.40 \leq |\rho| \leq 0.59$, *strong* for $0.60 \leq |\rho| \leq 0.79$, and *large* for $|\rho| \geq 0.80$.

[25] https://cran.r-project.org/web/packages/classInt

In the following, we summarize the three metrics used in this investigation:

- *Activity level of a contract:* Number of transactions that a contract received from its creation until the day before our data collection date.
- *Age of a contract:* Number of days elapsed between contract creation and the day before our data collection date.
- *Current idle time of a contract:* Number of days elapsed between the last transaction received by a contract and the day before our data collection date.

Results **Result 1)** ***67.5% of all contracts never received any transaction. 94.7% of all contracts received less than 10 transactions.*** Figure 3a depicts the empirical cumulative distribution function (ECDF) for the number of received transactions per contract. 67.5% of the contracts have never received transactions. Moreover, 94.7% of the contracts received less than 10 transactions. This result makes it clear that a remarkable proportion of the contracts in the Ethereum platform have been seldom used.

Result 2) ***0.05% of all contracts received 80% of all transactions sent to contracts.*** We sort the contracts in descending order of number of transactions and plotted the curve shown in Fig. 3b. As the curve indicates, only an exceptionally small portion of the contracts concentrate the vast majority of all transactions sent to contracts. Throughout the remainder of this paper, we call them *high-activity contracts*.

Result 3) ***Only 8% of the high-activity contracts are currently inactive.*** In Fig. 4, we show the *current idle time* of the high-activity contracts. As the curve indicates, the vast majority of these contracts have a very short current idle time, which indicates that they are all still active. After a pronounced knee in the curve, we notice that there are contracts that have been idle from 100 days to as long as 964 days. From this analysis, we conclude that it would make sense to cluster the data in three groups: active contracts, recent-inactive contracts, and long-inactive contracts. We thus ran the Jenks clustering procedure with $n_c = 3$, which produced the following intervals: $[0, 96.0]$, $(96.0, 448.8]$, $(448.8, +\infty]$ (check red dashed lines in Fig. 4). The obtained goodness-of-fit is 0.95, thus indicating that the clustering appropriately fits the data. Our analysis thus suggests that contracts that have been idle for at least 96 days (approximately 3 months) can be considered inactive. Most importantly, only 8% of the high-activity contracts are inactive.

Result 4) ***Older contracts do not necessarily receive more transactions.*** One may hypothesize that older contracts would be more likely to receive more transactions.

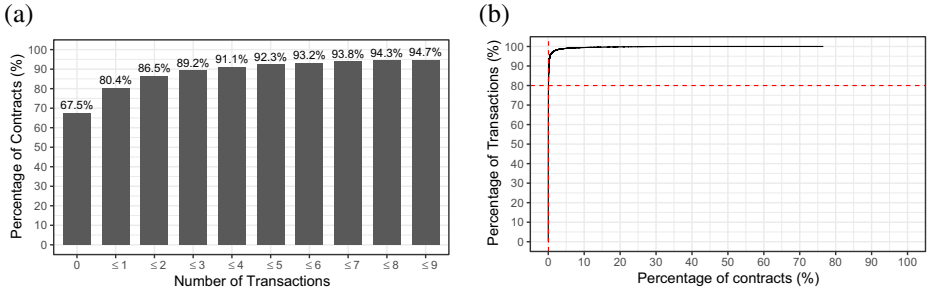

Fig. 3 **a** 67.5% of the contracts have never been used. 94.7% of the contracts received less than 10 transactions. **b** 0.05% of the contracts received 80% of the transactions sent to contracts

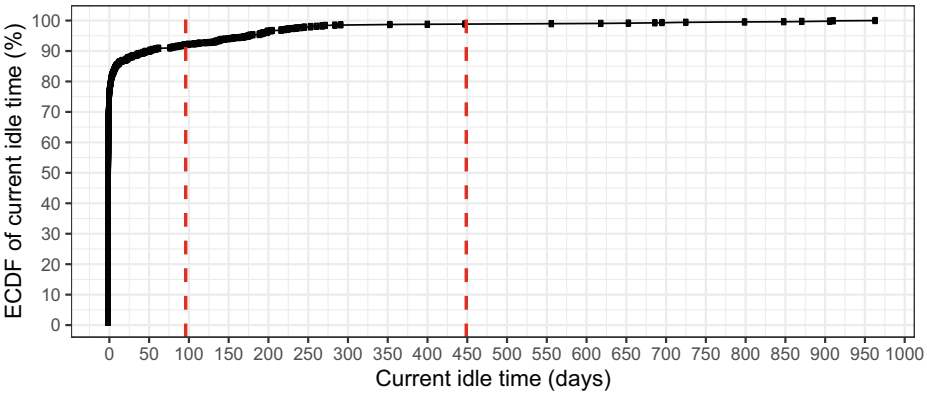

Fig. 4 The dashed lines delimit the three intervals produced by the Jenks clustering procedure. These intervals denote active contracts, inactive contract, and long inactive contracts (from left to right). Only 8% of the high-activity contracts are currently inactive

However, this hypothesis does not hold for our data, as we observed only a very weak correlation between a contract's age and its number of transactions ($\rho = 0.13$). To gain a deeper insight into the relationship between a contract's age and its number of transactions, we plot a heatmap (Fig. 5). Analysis of this map reveals that there are few observations in the top-right corner position. In fact, the contract with the highest number of transactions is 582 days old (i.e., it is a middle-aged contract).

To confirm our results, we perform an additional analysis: we compare the age of high-activity contracts against the average age over all contracts. The violin plot depicted in Fig. 6

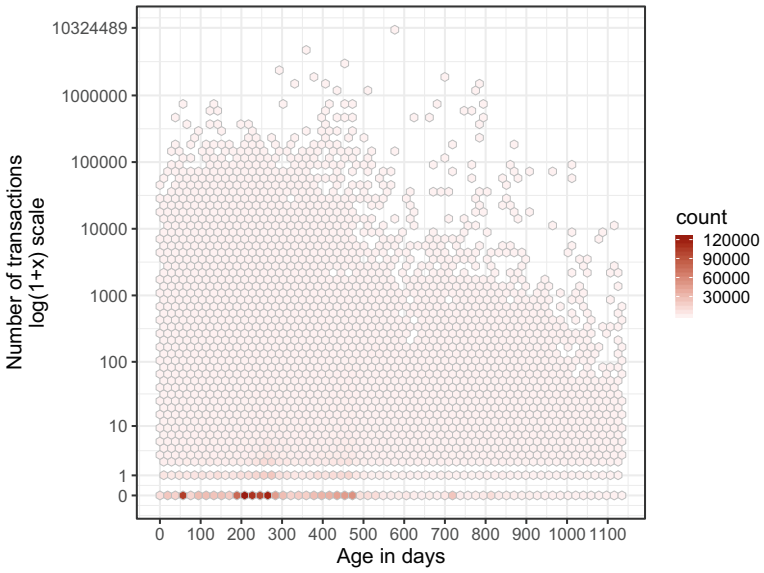

Fig. 5 Heatmap plot showing the relationship between a contract's age and its number of received transactions

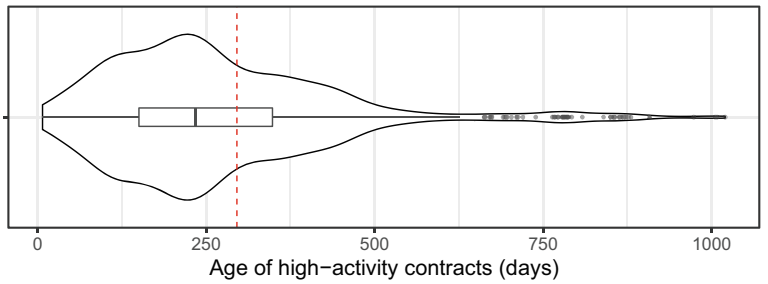

Fig. 6 Distribution of the age of high-activity contracts. The red dashed line indicates the average age of a contract

shows the distributions of age for high-activity contracts. A violin plot is a compact visualization that highlights the density of a continuous distribution (with an optional boxplot included inside it). We use a red dashed line to indicate the average age over all contracts (295.6 days). What we observe is that the median age of high-activity contracts is actually lower than the average age of contracts. Hence, older contracts do not necessarily receive more transactions.

Result 5) ***Verified contracts represent only 2.2% of all contracts. Yet, 71.9% of all transactions sent to contracts target verified contracts.*** As shown in the summary box of the data collection section (Section 3), verified contracts account for only 2.16% of all contracts. Yet, as shown in Fig. 7, verified contracts received more transactions (median = 3, mean = 2,247, sd = 62,888) than non-verified contracts (median = 0, mean = 19.4, sd = 2,155). A two-sided Mann-Whitney test indicates that the difference is statistically significant at $\alpha = 0.05$ (p-value $<$ 2.2e-16). In order to understand the practical significance of this difference, we compute the Cliff's Delta (δ) effect-size score. We assess it using the following thresholds (Romano et al. 2006): *negligible* for $|\delta| \leq 0.147$, *small* for $0.147 < |\delta| \leq 0.33$, *medium* for $0.33 < |\delta| \leq 0.474$, and *large* otherwise. We observe that the effect size is large ($\delta = 0.55$). More generally, we observe that 71.9% of all transactions sent to contracts target verified contracts. In addition, 73.1% of the high-activity contracts are verified contracts. Hence, verified contracts play a key role in the Ethereum platform with regards to activity level.

Since correlation does not imply causation, it is possible that there is a confounding factor behind the high-activity of verified contracts (i.e., the identified high activity is not related to the availability of the source code per se). Further studies are necessary to better understand the characteristics of verified contracts.

Implications Ethereum's scalability is severely limited. According to Vitalik Buterin, co-founder and inventor of Ethereum:

> *As far as the big problems, my top three at this point are probably scalability, privacy, and usability. So scalability, Ethereum blockchain right now can process 15 transactions a second, really we need like 100,000.*
>
> (transcribed from an interview given to ABRA[26] in March 2019)

[26] https://www.youtube.com/watch?v=u-i_mTwL-FI

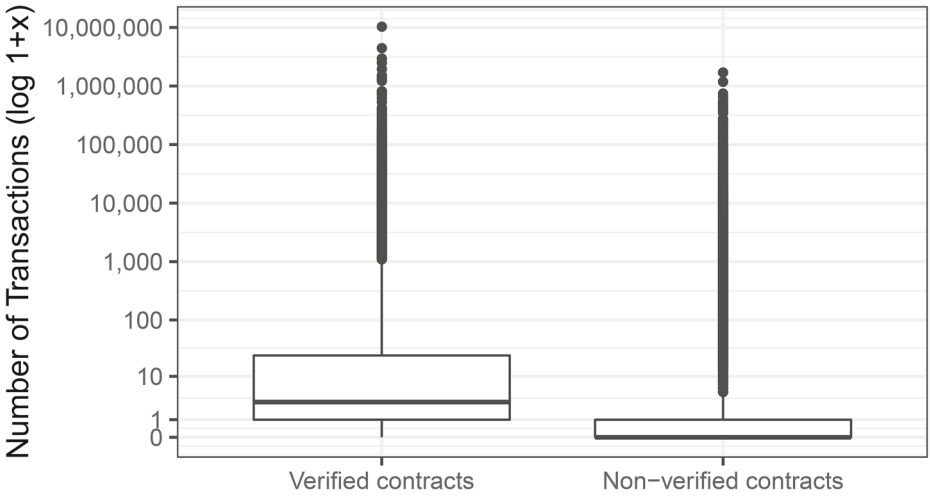

Fig. 7 Verified contracts received more transactions than non-verified contracts

There are currently two types of solutions being proposed to scale Ethereum (Buterin 2018; Gudgeon et al. 2019): Layer-1 solutions (i.e., those that modify Ethereum itself, such as Ethereum 2.0) and Layer-2 solutions (i.e., those that are built on top of Ethereum). While both types of solutions are likely to substantially improve Ethereum's throughput, they do not consider our main finding: activity level is severely skewed. We see the following immediate implications:

1. Understanding and monitoring the (contract) transaction validation workload is trivial, since this workload is driven by a very small number of contracts.
2. There is currently no way to ensure a fair utilization of the Ethereum processing power. In particular, there have been a series of events where the Ethereum blockchain got congested due to the activity of a single smart contract. For instance, a Chinese company developed a gambling game in which a player "wins the jackpot" if such a player is the last one to "buy a ticket" (i.e., send a transaction). A blog post[27] commented:

   > *The gambling game FOMO congested the Ethereum network once again earlier this month. [...]. Though just recently the small contract once again consumed some 20% [of all the gas] before peaking up to 36.4% at some point. To get a taste what this means for the Ethereum network, the regular fees for an Ethereum transaction surged from $0.01 to $0.50. Notably, the Asian daytime hours correspond with high activity rates.*

   In other words, if a single smart contract strongly attracts a niche of users who abruptly start sending a remarkable number of transactions towards such a contract, then Ethereum gets massively impacted. We believe that the ability of users to have their transactions processed in a timely manner (provided that they pay reasonable transaction fees) should not be hindered due to a niche of users who decided to play FOMO or breed CryptoKitties.

[27] http://ftreporter.com/attention-please-fomo-games-are-clogging-ethereum/

We envision that an activity-imbalance aware architecture would be able to provide separate classes of service depending on the current activity level of a contract (or a contract's function). For instance, a given smart contract could receive progressively lower priority (e.g., by means of an extra processing fee) as more and more transactions are sent to it. In this scenario, we also foresee wallets monitoring and communicating these priorities to users. More generally, we believe that the designers of future versions of Ethereum should be mindful of *blockchain utilization fairness* (e.g., preventing a single contract from utilizing most of the processing power of the blockchain).

3. The high skew in the activity level of contracts poses questions to the ability of Ethereum to handle denial of service (DoS) attacks. While such attacks would be costly (since the costs of such attack transactions must be paid), the impact on the whole blockchain (and the world economy) would be substantial – imagining a world where blockchains are the backbone of all commerce. We also note that such attacks are considerably trivial since one does not even need to target a large number of contracts. Today, we are already seeing signs of the potential impact of DoS attacks, where ĐApp developers are communicating manually through side-channels and advising their users to manipulate their fees in order to cope with blockchain cloggings. The figure below shows an example from CryptoKitties: the development team increased the cost of a smart contract function in order to incentivize miners (Fig. 8).

Future work opportunities that derive from our results and the aforementioned implications are listed in Section 5.

Summary: Activity level is concentrated on a very small subset of the contracts and almost three quarters of them are verified. In particular:

- 67.5% all contracts never received any transaction. 94.7% of all contracts received less than 10 transactions.
- 0.05% of all contracts received 80% of all transactions sent to contracts. We call them *high-activity* contracts.
- Only 8% of these high-activity contracts are inactive as of the data collection date.
- Older contracts do not necessarily receive more transactions.
- Verified contracts represent 2.2% of all contracts. Yet, 71.9% of all transactions sent to contracts target verified contracts.

Implications: New solutions to cope with Ethereum's limited scalability should take the activity imbalance into consideration. In particular:

- Understanding and monitoring the (contract) transaction validation workload is trivial, since this workload is driven by a very small number of contracts.
- There is currently no way to ensure a fair utilization of the Ethereum processing power.
- The high skew in the activity level of contracts poses questions to the ability of Ethereum to handle denial of service (DoS) attacks.

4.2 RQ2: What are the Categories of High-Activity Smart Contracts?

Motivation A blockchain platform such as Ethereum has distinct properties (e.g., fully-distributed, immutability of recorded information, ledger capabilities) compared traditional

Fig. 8 Cryptokitties tweet during Ethereum congestion

software architectures. Although media outlets have systematically published about promising blockchain use-cases (Vilner 2018; Marr 2018), there is limited evidence-supported knowledge about the state of this well-hyped technology. In particular, the categories of currently deployed smart contracts remain fairly unknown. In this research question, we investigate the categories of high-activity contracts (i.e., the 0.05% of the contracts that received 80% of the transactions sent to contracts). Our goal is to uncover the types of smart contracts that are of interest to the user base of the Ethereum platform.

Approach Given the key role of cryptocurrencies in blockchain platforms, we first determine the percentage of token contracts in the set of high-activity contracts. We detect token contracts by checking whether they implement either the ERC20 interface or the ERC721 interface (Section 2.5). Subsequently, we match the name of these token contracts against the regular expression shown below in order to spot *token managers*:

```
(ICO)|(Sale)|(.+Coin$)|(.?Token$)|(ERC20)|(ERC721)
```

Token managers are contracts that focus primarily on the manipulation of tokens. For instance, they offer functions to transfer tokens, mint tokens, or manage an ICO. The Ethereum tutorial on how to create cryptocurrencies[28] shows several examples of generic token management functions. In order to uncover the financial relevance of the tokens defined in the token contracts, we checked their market capitalization in the CoinMarketCap website.

As part of this research question, we also investigate how frequently high-activity contracts are linked to a ĐApp (Section 2.6). For the cases in which the link is established, we report the category of the ĐApp (as recorded by State of the ĐApps). As explained in Section 3, we link contracts to ĐApps from State of the ĐApps using the contract address as key. The rationale behind establishing this link is to obtain a broader understanding of the practical context in which smart contracts are used.

[28] https://www.ethereum.org/token

Lastly, we identify and describe the top-10 contracts with highest activity level. If a top-10 contract belongs to a ĐApp, then we report the category as recorded by State of the ĐApps. Adversely, if a top-10 contract is *not* linked to a ĐApp, then we determine the category ourselves based on an inspection of the smart contract code and its associated metadata.

Results **Result 6)** ***72.9% of the high-activity contracts are token contracts. These token contracts account for an impressive market capitalization of US$12.7 billion.*** 72.9% of all high-activity contracts are token contracts. In particular, 60.4% of all high-activity contracts are verified token contracts. Token contracts typically constitute a technological solution for selling and managing the shares of a company (e.g., via an ICO). We highlight, however, that the product offered by the company does not necessarily live on the Ethereum platform. For instance, the EOS token defined in a token contract represents the shares of a company proposing a brand new blockchain platform called EOS. To get an idea of the financial relevance of token contracts, we searched for their associated cryptocurrency symbols in the CoinMarketCap website. We were able to find the market capitalization of 50% of the cryptocurrencies. The violin plot shown in Fig. 9 depicts the market capitalization values as of September 16th 2018 (1 ETH = US$220.59). The sum of the market capitalization values is US$12.7 billion.

Result 7) ***At least 41.3% of all high-activity contracts are likely simple token managers.*** As mentioned in the approach, we matched the name of high-activity contracts with a regular expression in order to detect token managers. We observed that 41.3% of all high-activity contracts are (verified) token contracts that match the regular expression. This result suggests that almost half of the high-activity contracts are likely simple token managers.

Result 8) ***Games, Currency Exchanges, and Gambling are the prominent categories of high-activity contracts that belong to a ĐApp. Although only 8.1% of the high-activity contracts belong to a ĐApp, these ĐApp contracts received more transactions than non-ĐApp contracts.*** Only 8.1% of the high-activity contracts are linked to a ĐApp listed in State of the ĐApps. The remaining high-activity contracts either do not really belong to any ĐApp or the ĐApp to which they are linked is not listed in State of the ĐApps. In Fig. 10, we show the counts for each category. The names of the categories come straight from State of the ĐApps. The prominent categories are Games, (Currency) Exchanges, and Gambling.

Although only 8.1% of the high-activity contracts belong to a ĐApp, we highlight that these contracts received a remarkable number of transactions. We compared the number of transactions received by ĐApp contracts and non-ĐApp contracts (in the universe of high-activity contracts) using a two-tailed Mann-Whitney test ($\alpha = 0.05$) and the results

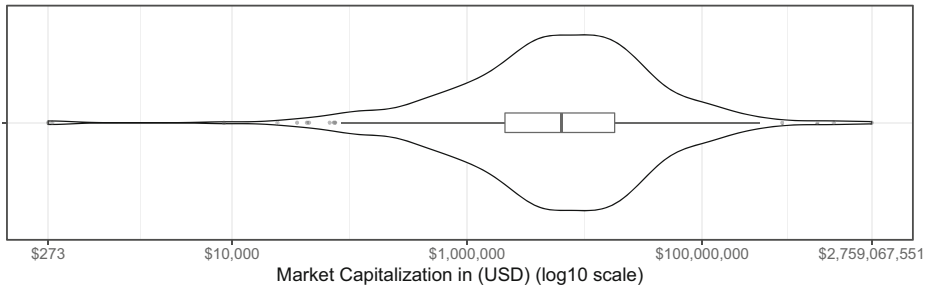

Fig. 9 The market capitalization of verified token contracts with high activity shown on a log10 scale (median = US$6.3M, mean = US$32.6M, sd = US$166.1M)

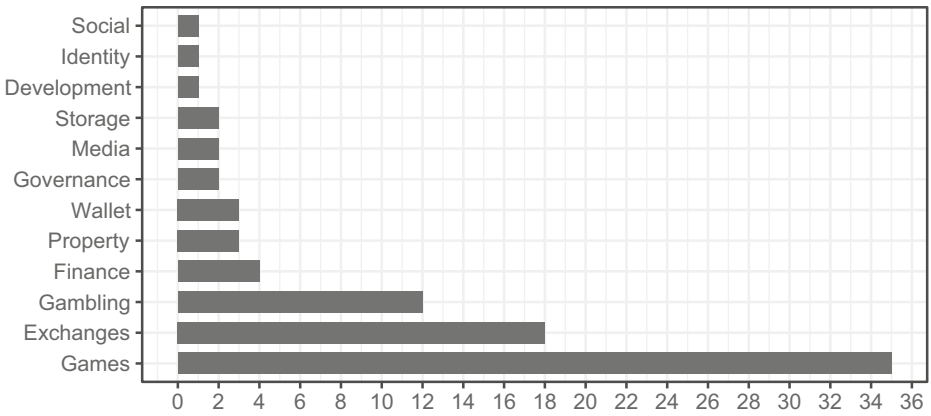

Fig. 10 Number of verified contracts with high activity and that belong to a ĐApp grouped by category. Games, (Currency) Exchanges, and Gambling are the prominent categories

indicated that the difference is statistically significant (p-value = 0.005808). Cliff's delta ($\delta = 0.17$) indicates that the difference is small (nevertheless non-negligible).

Result 9) ***5 out of the top-10 contracts with highest activity level are part of Exchange ĐApps.*** The top-10 contracts in terms of activity level (number of received transactions) are shown in Table 1. Together, these contracts received 22.5% of all transactions sent to contracts in Ethereum. The column Tx. denotes the number of received transactions. The column LoC denotes the counts for lines of code (excluding code comments). The last column indicates whether the contract is verified. We notice a lack of heterogeneity, as 5 out of the top-10 contracts are Exchange ĐApps. In Appendix, we describe these top-10 contracts in more details.

Implications Our findings indicate that, despite the hype around blockchain-powered applications and their presumable suitability for an infinite range of use-cases, the heterogeneity of deployed applications seems constrained. From a software development

Table 1 The top-10 contracts with highest activity level in Ethereum

Rank	Contract name	DApp name	Category	Token symbol	Creation date	Tx.	LoC	Verif. (Y/N)
1	EtherDelta	ForkDelta	Exchange	–	09-02-2017	10.3M	233	Y
2	Exchange	IDEX	Exchange	–	27-09-2017	4.4M	151	Y
3	DSToken	–	ICO	EOS	20-06-2017	2.9M	263	Y
4	KittyCore	CryptoKitties	Game	CK	23-11-2017	2.5M	971	Y
5	TronToken	-	Token mgt.	TRX	28-08-2017	1.9M	71	Y
6	[Unknown]	Poloniex	Exchange	–	18-10-2016	1.7M	-	N
7	ReplaySafeSplit	Bittrex	Exchange	–	25-07-2016	1.5M	20	Y
8	Controller	Bittrex	Exchange	–	12-08-2017	1.5M	132	Y
9	Bitcoinereum	Bitcoinereum	Mintable Token	BTCM	10-10-2017	1.4M	132	Y
10	OMGToken	OmiseGO	Wallet	OMG	05-07-2017	1.3M	185	Y

perspective, 41.3% are simple token managers. In addition, 5 of the top-10 contracts are currency exchanges. Hence, most of the usage in Ethereum is still tied to the simple manipulation of tokens. It is also noteworthy that a simple application such as CryptoKitties remains as one of the key drivers of the traffic on the network.

Our results leave us questioning what the rationale behind such simple applications is. The CryptoKitties team reported technical challenges. Bryce Bladon (CryptoKitties Founding Team member) questions whether the whole application should be on the blockchain:

> *One of the big things we learned was that although a lot of the very interesting aspects of blockchain technology have to do with decentralization. At the same time, we believe that certain centralized features can be a way to alleviate decentralized demand, and having much of the transactional weight tied to a sidechained scaling solution is a very interesting option. While interacting with the blockchain is what makes this interesting, it isn't necessarily everything. We found that there were a lot users who buy and breed cats – stuff that requires directly transacting on the network – but there are also users who just wanna browse or upvote or engage with the community.*

The following quote[29] from the CryptoKitties team discusses a bug that could not be fixed in practice:

> *We could fix Unexpected Kitty Fleas by making a change to a single line of solidity code. However, if we push the fix through, all Siring Auctions would need to be cancelled and reposted (that's over 40,000 auctions), and our users wouldn't be able to sire their cats for 12+ hours. Simply put: it makes more sense for us to issue refunds at a loss AND donate the entirety of the overpayment to charity than it does to fix such a minor issue (and disrupt a key part of the game for nearly a full day).*

Hence, there might be technical challenges behind the lack of more sophisticated applications. Further studies are necessary in order to extensively uncover the reasons behind the simplified usage of Ethereum. Future work opportunities that derive from our results and the aforementioned implications are listed in Section 5.

Summary: Most of the high-activity contracts are token contracts and 5 out of the top-10 contracts with highest activity belong to a Currency Exchange ĐApp. In particular:

- 72.9% of the high-activity contracts are token contracts. Together, they account for a market capitalization of US$12.7 billion.
- At least 41.3% of all high-activity contracts are likely simple token managers.
- Only 8.1% of the high-activity contracts belong to a ĐApp. Nevertheless, ĐApp contracts received more transactions than non-ĐApp contracts.
- Games, Currency Exchanges, and Gambling are the prominent categories of high-activity contracts that belong to a ĐApp.
- 5 out of the top-10 contracts with highest activity are part of Exchange ĐApps.

Implications: The new, widely advertised, rich programming model of smart contracts is currently being mainly used to develop very simple applications.

[29] https://medium.com/cryptokitties/unexpected-kitty-fleas-91c565547b11

4.3 RQ3: How Complex is the Source Code of Verified Smart Contracts?

Motivation The Ethereum platform creators advertise smart contracts as a mechanism that enables the transparent specification and enforcement of business rules, financial clauses, or any general governing rule (Wood 2017). However, smart contracts are ultimately computer code, meaning that it can get complex and not easily digestible by non-specialists who wish to simply use these contracts. For instance, as we showed in RQ2, *EtherDelta* is a popular Exchange ĐApp. Although its developers advocate transparency of the business rules (as they are all exposed in the smart contract), understanding the contract itself is far from trivial. In fact, the developers of EtherDelta posted a thread on reddit[30] to explain what their contract does and how. In this research question, we set out to investigate the code complexity of smart contracts and how it evolved. As such, we analyze verified contracts only. As observed in RQ1, verified contracts play a major role in the Ethereum platform.

We interpret *code complexity* as a broad concept. We thus investigate it from four distinct angles: (a) code size (number of instructions), (b) complexity of control flow (cyclomatic complexity), (c) code organization (number of subcontracts and libraries), and (d) documentation. Investigating these four complexity angles will help us better understand the coding practices adopted by smart contract developers.

Approach Our approach involves investigating code complexity according to the four aforementioned angles. In the following, we explain how we measured complexity:

- *Size.* We determine the size of a contract by counting the *number of instructions*. We use the number of instructions instead of the more popular metric *number of lines of code* (LoC) because the source code of some contracts published in Etherscan do not have line breaks.[31] In addition, LoC is influenced by code indentation styles. Therefore, we compute the number of instructions as follows: *numbers of semicolons* (`;`) + *number of open curly brackets* ({).
- *Cyclomatic complexity.* We compute it as the *number of decision points in the source code* + 1, while taking into account compound conditions (McCabe 1976). We leverage our Solidity parser to reliably detect decision points and conditions.
- *Code organization.* We used our parser to detect subcontracts and libraries. In the remainder of this subsection, we use the general term *code sections* to refer to both subcontracts and libraries indistinctly. We report the number of code sections that we found for each studied contract.
- *Documentation.* We used our parser to detect code comments, including line comments and block comments. We report the comment ratio of each contract, which we define as $\frac{ncom}{ninst+ncom}$ (`ncom` and `ninst` stand for *number of comments* and *number of instructions* respectively). We count contiguous line comments (i.e., subsequent lines starting with `//`) as a single comment.

To answer this research question, we quantify complexity according to the metrics above and analyze their evolution over time. We focus on high-activity verified contracts, since they are inherently relevant to the community using Ethereum. Analogously to RQ1, high-activity verified contracts are defined as the set of verified contracts that receive 80% of all transactions sent to verified contracts. As shown in Fig. 11, high-activity verified contracts

[30] https://www.reddit.com/r/EtherDelta/comments/6kdiyl/smart_contract_overview

[31] https://etherscan.io/address/0xe68b7aa92f5a71184bfcb42f2a5ec5711e16afe7#code

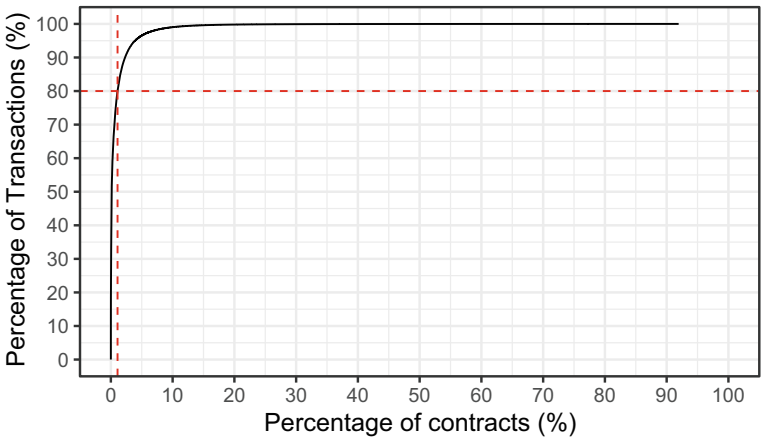

Fig. 11 1.1% of the verified contracts received 80% of the transactions sent to verified contracts. We call them high-activity verified contracts

account for only 1.1% of all verified contracts. In other words, a remarkably small portion of the verified contracts receive a high number of transactions.

Results Result 10) ***The complexity of high-activity verified contracts is slightly higher than that of other verified contracts.*** As illustrated in Fig. 12, the complexity of high-activity verified contracts is higher than that of other verified contracts. For each complexity metric, we performed a two-sided non-paired Mann-Whitney test with $\alpha = 0.05$ and calculated Cliff's Delta (δ). Table 2 summarizes the results that we obtained. In summary, the difference is statistically significant in all cases. The effect size is non-negligible for all metrics but comment ratio. Therefore, we conclude that high-activity verified contracts are more complex than the other verified contracts.

Result 11) ***The source code of high-activity verified contracts is small, commonly uses at least 2 code blocks, and is extensively documented.*** In Fig. 13a, we show the ECDF for the number of instructions in high-activity contracts. As indicated by the dashed lines, 80%

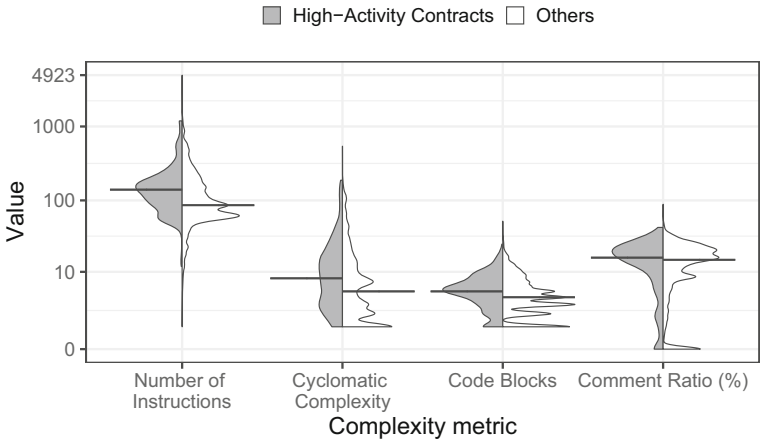

Fig. 12 Complexity of high-activity verified contracts is higher than that of other verified contracts

Table 2 Statistical results for the comparison of complexity between high-activity verified ontracts and other verified contracts

Metric	Statistically significant difference	Cliff's delta
Number of instructions	Yes (p-value < 2.2e-16)	Small (0.25)
Cyclomatic complexity	Yes (p-value < 2.2e-16)	Small (0.22)
Code blocks	Yes (p-value = 1.664e-11)	Small (0.18)
Comment ratio	Yes (p-value = 0.00778)	Negligible (0.07)

We compared distributions using a two-sided non-paired Mann-Whitney test with $\alpha = 0.05$

of the contracts have at most 211 instructions. We consider 211 instructions to be a really small number: even if a contract called 10 other contracts, the number of instructions would still be smaller than most real world applications written in Java or C#.

In Fig. 13b, we show the ECDF for the number of code blocks in high-activity contracts. Analysis of this figure reveals that only 10% of these contracts define a single code block. In particular, the median of code blocks is 5. These results thus indicate that high-activity contracts typically have high (internal) modularity.

Finally, we investigated the comment ratio of high-activity contracts. We argue that these contracts are extensively documented. Relying on the dataset provided by Munaiah et al. (2017), we compared the comment ratio of high-activity contracts to that of GitHub projects written in Java, C++, and C#. For each of these 3 languages, we ordered them according to the number of stars and selected the top 1,000 projects. In order to ensure a reliable comparison, we recalculated the comment ratio of our solidity contracts using the same metric and tool used by Munaiah et al. (2017). The results are shown in Fig. 14. As the violin plots indicate, the comment ratio in high-activity contracts is higher than that of projects written in other languages. Table 3 summarizes the results of the statistical analysis we performed. The difference was statistically significant in all cases with small to medium effect sizes.

We conclude this analysis by highlighting that extensive documentation does not necessarily imply that the comments are of high-quality or that the code being commented is of high-quality. Indeed, the most extensively documented contract, with a comment ratio of 42.6% (using our original metric) is *The DAO*. The DAO launched with US$150 million in crowdfunding in June 2016. The contract was shortly-after hacked by exploiting a recursion

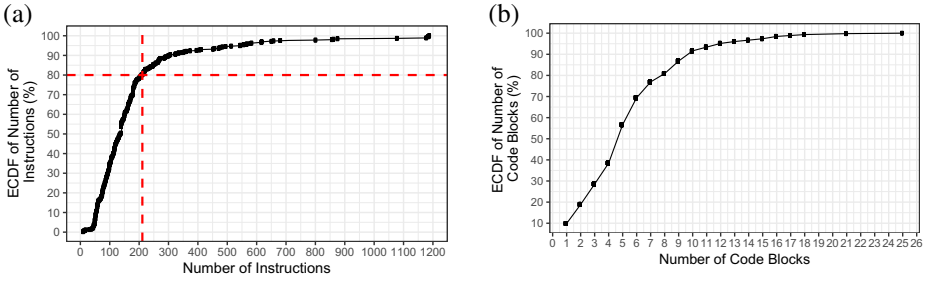

Fig. 13 **a** 80% of the high-activity contracts have at most 211 source code instructions. **b** Only 10% of the high-activity contracts define a single code block

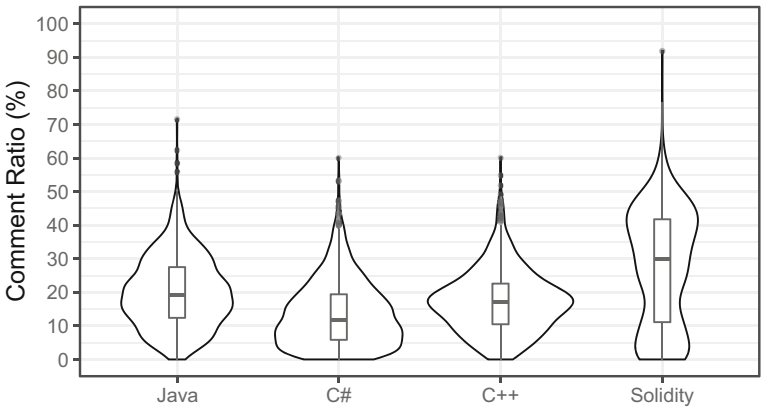

Fig. 14 The comment ratio in high-activity contracts (Solidity) is higher than that of projects written in other languages

call vulnerability. The hacker managed to drain US$50 million worth of cryptocurrency. The unprecedented event resulted in a large debate in the Ethereum community regarding whether the prior transactions on Ethereum should be rewritten to revert the hacker's operation, as it would somehow contradict Ethereum's own immutability principle. Following the "too big to fail" financial lemma (Sorkin 2010), the Ethereum platform ended up being split into two at block 1,920,000: the Ethereum Classic Platform (with transactions kept as is) and the Ethereum Platform (which reverted the hacker's operation).

Result 12) ***The number of instructions and code blocks of high-activity contracts has remained roughly stable over time. On the other hand, the comment ratio has a downward trend starting from 2017-07-01.*** Figure 15 shows the number of instructions, number of code blocks, and comment ratio for all contracts created every trimester (and that ended up receiving a high number of transactions). The dashed line indicates the overall median. Analysis of this figure reveals that the number of transactions and code blocks increased in the trimester starting on 2017-01-01 (yyyy-mm-dd). However, these two metrics have remained roughly stable ever since. On the other hand, the comment ratio has a clear downtrend starting from 2017-07-01. For instance, a certain contract created on 2018-07-16[32] has 486 instructions (i.e., above the overall median of number of instructions) and a comment ratio of 5.6% (i.e., below the overall median of comment ratio). The reason behind this downward trend in comment ratio is unknown and requires further investigation.

Lastly, we note that we do not analyze cyclomatic complexity since it is strongly correlated with number of instructions ($\rho = 0.71$).

Implications Smart contract development entails a new programming paradigm, and thus understanding the key characteristics of the source code is key to shaping and informing future research and improvement efforts. We observed that the source code of high-activity verified smart contracts exhibit particular complexity characteristics compared to other popular programming languages. With relation to the small code size of contracts, we conjecture that it is due to many contracts being simple token managers. The rationale for the simplified usage of such a new and powerful programming paradigm is an important follow

[32] https://etherscan.io/address/0x0777f76d19579526838789343068e4fcd286919#code

Table 3 Comparison of comment ratio for projects written in Java, C#, C++, and Solidity

Comparison	Statistically significant difference	Cliff's Delta
Solidity vs Java	Yes (p-value = 8.727e-16)	Small (0.26)
Solidity vs C#	Yes (p-value < 2.2e-16)	Medium (0.44)
Solidity vs C++	Yes (p-value < 2.2e-16)	Medium (0.34)

We use high-activity contracts to represent Solidity. We compared distributions using a two-sided non-paired Mann-Whitney test with $\alpha = 0.05$

up step to our study. With regards to high internal modularity, we wonder whether Solidity developers are generally experienced programmers (e.g., good object-oriented skills). With regards to the extensive documentation of contracts, we conjecture that this might be an effort to promote transparency of the code behavior. However, there might be technical

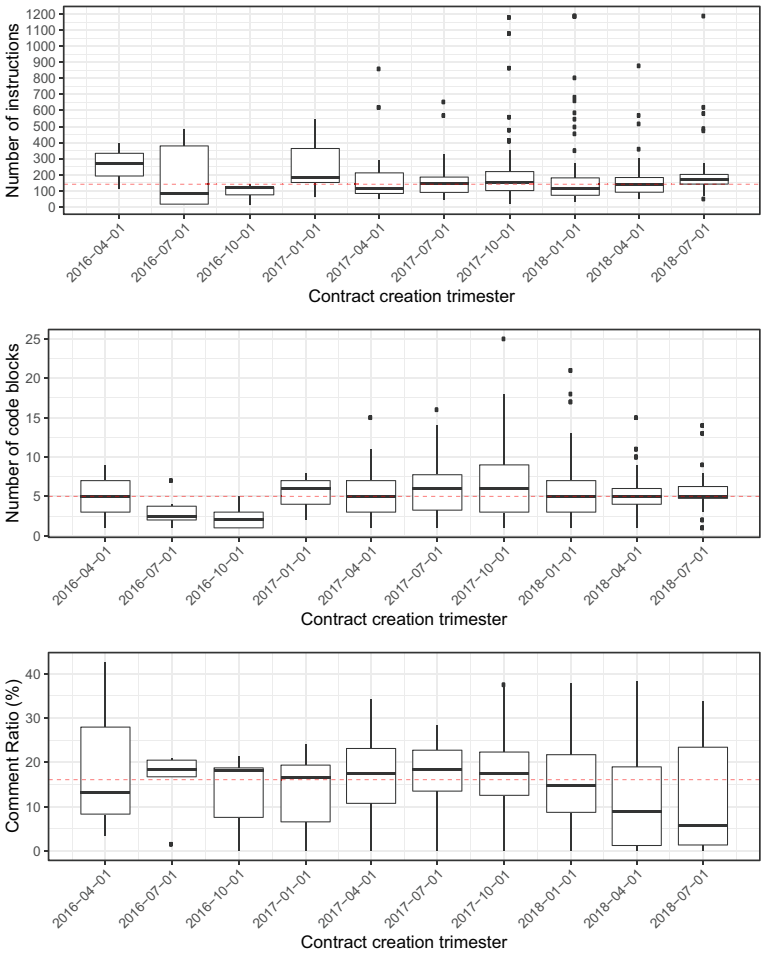

Fig. 15 Evolution of number of instructions, number of code blocks, and comment ratio for high-activity verified contracts. The dashed lines indicate the overall median

reasons behind it as well (e.g., extensive commenting should help code review activities and make it easier to estimate gas consumption). Again, future studies need to uncover the rationale behind these findings. A more extensive list of open research questions is presented in Section 5.

Summary: The source code of high-activity verified contracts is small, commonly uses at least 2 code blocks, and is extensively documented. In particular:

- The complexity of high-activity verified contracts is slightly higher than that of other verified contracts (except for comment ratio).
- 80% of the high-activity verified contracts have at most 211 instructions.
- The median of code blocks (subcontracts and libraries) in high-activity verified contracts is 5.
- The comment ratio in high-activity verified contracts is higher than that of GitHub top-starred projects written in Java, C++, and C#. On the other hand, the median comment ratio in high-activity verified contracts has been decreasing since 2017-07-01.
- The size and number of code blocks have remained roughly stable over time for high-activity verified contracts.

Implications: The source code of high-activity verified smart contracts exhibit particular complexity characteristics compared to other popular programming languages. Further studies are necessary to uncover the actual *reasons* behind such differences.

5 An Open Research Agenda

Our study explores three key dimensions of smart contracts: activity level (RQ1), category (RQ2), and source code complexity (RQ3). For each RQ, we discussed the implications of our findings. Nevertheless, we were also left with several open questions and conjectures. Hence, in this section, we introduce an *open research agenda*. We hope that such an agenda can bootstrap future work in the area and thus foster a deeper understanding of the Ethereum platform.

(OQ01) Why are there so many zero-transaction contracts?

As an additional motivation behind this open question, we investigated the total amount of Ether that was spent to deploy zero-transaction contracts (i.e., contract creation fee). In Fig. 16, we show the distribution of contract creation fee. We observe that developers managed to create 0.48% of the contracts with virtually zero Ether[33] (note the small blob at the bottom of the violin plot).

The median is 0.005246 Ether and the total is 3.564150 Ether. If we consider the exchange rate from the day that we performed our data collection (September 15th 2019, 1 Ether = US$208.87), the median becomes US$1.10 and the total becomes US$2.5 million. Since the Ether to USD exchange rate has fluctuated considerably over time,[34] using more

[33] Example tx: `0x1fc8cd67cbbf6e96d64c0dca84b5cb420b0837ff74bfe2f1c9547d45a58baa0a`

[34] https://etherscan.io/chart/etherprice

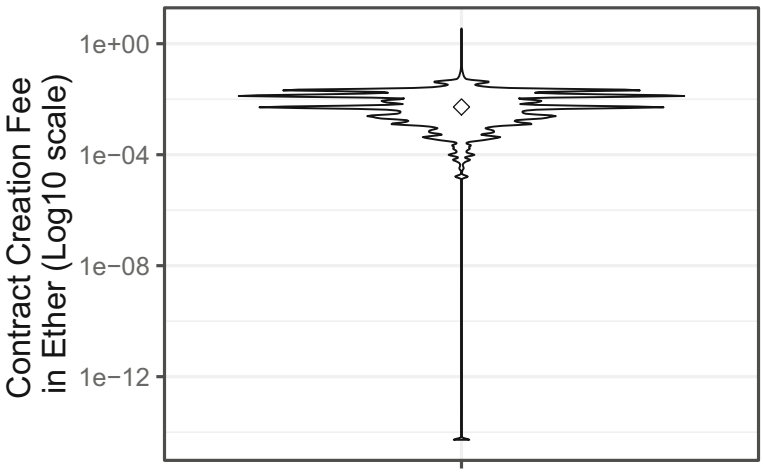

Fig. 16 Distribution of contract creation fee in Ether. The diamond indicates the median

precise exchange rates would likely show an even higher total. Hence, we conclude that a noteworthy amount of money was spent to deploy zero-transaction contracts. Our follow-up questions are listed below.

(a) What is the practical relevance of zero-activity contracts?
(b) How many are toy contracts, simple tutorial contracts, and incorrect contracts? Are there rare-cases/corner-case contracts?
(c) What kind of information do they store and/or process?
(d) Are they storing information that is valuable to a particular customer or groups of customers who might start interacting with such contracts in the future?
(e) How many contracts represent simple data stores? By searching for contracts that had 0 transactions, 1 subcontract, and 0 libraries (i.e., a heuristic), we manually spot a few examples of data-store contracts. In one contract,[35] the constructor hardcodes the balance of several user addresses. In another contract,[36] the constructor hardcodes data related to life settlement.
(f) Should future designs for Blockchains provide special support for deprecating contracts?

We conjecture that many zero-transaction contracts exist simply due to developers trying out Ethereum and deploying toy or test contracts (i.e., because of the hype). In Fig. 5, we show the relationship between the number of received transactions and the contract's age. A remarkable number of contracts of ages between 200-300 days ended up never receiving transactions (note the dark red hexbins). These ages encompass the period of November 19th 2017 to February 27th 2018. As depicted in Fig. 17, such a period maps precisely to when the hype around Ethereum exploded (which, in turn, encompasses the period in which CryptoKitties exploded).

[35]Contract address: `0xc244d24a3293150709913ce8377dc2854a3ec4a1`
[36]Contract address: `0xac9efefb9de2d2aa0e1bcaada95480fe29f23c42`

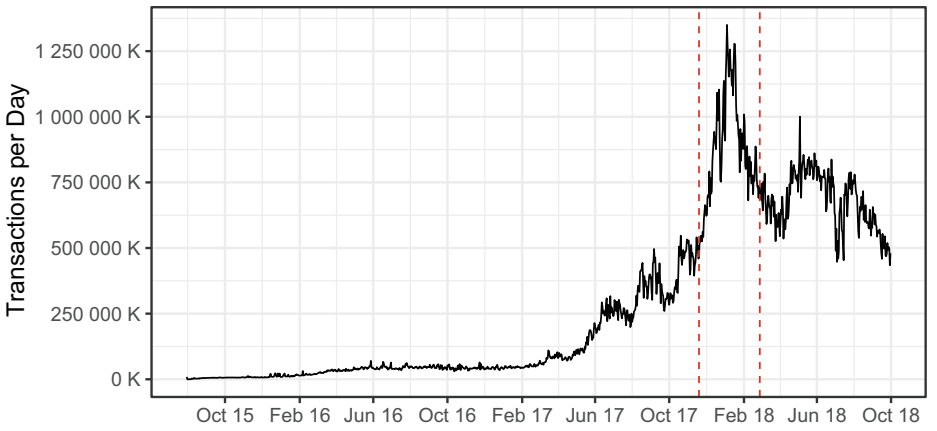

Fig. 17 The number of transactions processed by Ethereum per day. The red dashed lines delimit the period of November 19th 2017 to February 27th 2018.

(OQ02) Why are there so many low-activity contracts?

As an additional motivation behind this open question, we investigated trends in contract creation. In Fig. 18, we show that the number of contracts created has grown considerably since 2017-02-01 (despite remarkable fluctuation). Yet, as Fig. 19 also indicates, the rate at which developers deploy contracts that end up receiving less than 10 transactions has remained roughly stable over time. The dashed line indicates the overall median, which is 92.3%. In particular, at least 77.6% of the contracts deployed every month received less than 10 transactions.

Our follow-up questions are listed below.

(a) What is the practical relevance of low-activity contracts (1 to 10 received transactions)?
(b) Are they toy contracts or simple tutorial contracts? Are they pre-release testing contracts? Are they examples of rare-case/corner-case contracts?
(c) How much Ether do they hold? How much Ether did they move? What is the market capitalization of the tokens/coins that they hold/move?

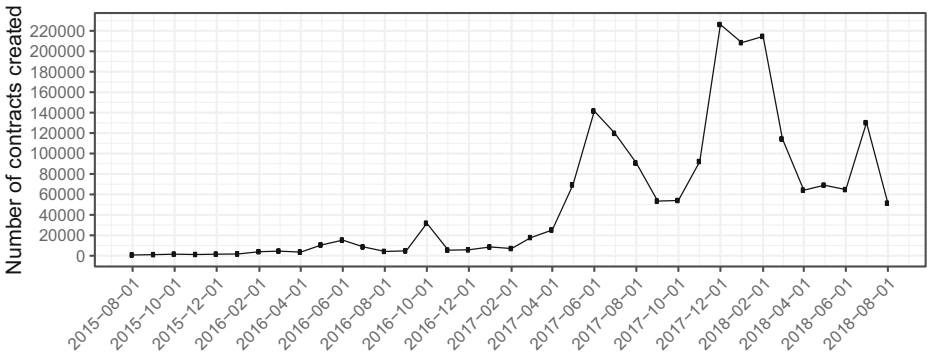

Fig. 18 Number of contracts created over time

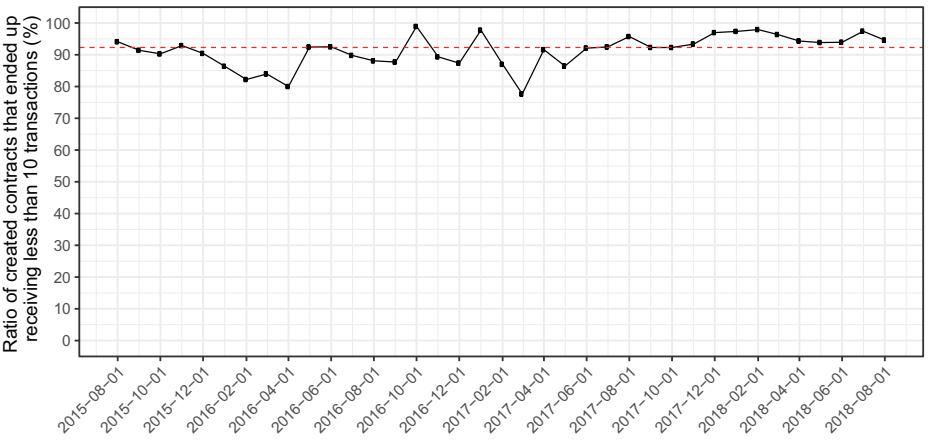

Fig. 19 The creation of contracts that ended up receiving less than 10 transactions has been roughly stable over time. The red dashed line indicates the median (61.3)

(d) Should designs for future blockchains provide special treatment for such low-transaction contracts?
(e) Can knowledge about contract usage intensity help optimize blockchains?
(f) Should contracts be tagged by their developers about their expected usage intensity?

We conducted a preliminary study to understand the practical relevance of low-activity contracts. More specifically, we focused on the financial perspective and analyzed how much Ether was sent to low-activity contracts. Using data from Google BigQuery, we find that 95% of the low-activity contracts received at most 17.84766 Ether (95th percentile). As we noted before, the Ether to USD exchange rate has fluctuated considerably over time. If we take the maximum exchange rate that was ever recorded (1 Ether = US$1,359.48), we obtain US$24,263.54. This result thus indicates that the vast majority of low-activity contracts have not received much Ether. However, if we (i) sort the low-activity contracts per amount of received Ether, (ii) take the top five,[37] and (iii) perform the same currency conversion, we obtain the following amounts (hundreds of millions): US$441.8M, US$373.9M, US$255.6M, US$169.2M, and US$141.6M. If we take the exchange rate of the day that we performed our data collection (September 15th 2019, 1 Ether = US$208.87), the numbers remain high (tens of millions): US$67.9M, US$57.4M, US$39.3M, US$26.0M, and US$21.8M. Hence, there seems to be very few low-activity contracts that received a noteworthy amount of money. More studies are necessary in order to better understand the financial relevance of low-activity contracts. For instance, contracts might

[37] The addresses of these top-5 contracts are:

`0xcea2b9186ece677f9b8ff38dc8ff792e9a9e7f8a` (325,000 ETH),
`0x69c6dcc8f83b196605fa1076897af0e7e2b6b044` (275,010 ETH),
`0xeca56d04546affcec0b3ce61971136f497866a3b` (188,000 ETH),
`0x4b25b370aa62d408bc2c87598289b59d1140545f` (124,424 ETH),
`0xeb2227d932aa85a0855613f870bb1b7fdc4b8af6` (104,145.1 ETH)

have not received Ether, but they might hold tokens (coins) that have a significant market capitalization.

(OQ03) What architectural changes could Ethereum undergo in order to account for such an imbalanced workload and thus ensure a fair usage of the platform? How can such changes be implemented and validated?

(a) What are the implications of such an imbalanced workload on the ability of the blockchain to cope with denial of service attacks?

(OQ04) While verified contracts are key players in terms of activity level, why are so few contracts being verified?

(a) Is the low proportion of verified contracts due to technical (the verification process is cumbersome and time consuming) or non-technical (e.g., developers do not wish to share their code widely or developers do not see value in performing such verifications) challenges?
(b) Has the rate of contract verifications increased over time?

(OQ05) Why do high-activity contracts capture a very limited number of use cases?

(a) Is it due to technical limitations about the programming model of smart contracts, their inflexible nature (i.e., code cannot be changed), or simply due to the slow adoption of such a programming model by other domains?
(b) What are the complexities with developing, testing and maintaining smart contracts? CryptoKitties team mentioned scalability problems and difficulties to fix bugs. Do other development teams share the same view?
(c) We employed a heuristic to detect contracts that are token managers. We also linked contracts to DApps (using State of the DApps). What other approaches can be used to determine a finer-grain categorization of a smart contract? More generally, what data sources could be mined and what approaches could be employed in order to discover contextual information about a smart contract (e.g., GitHub repository, development team information, project information)?

(OQ06) Why is the complexity of high-activity verified contracts (slightly) higher than that of other verified contracts?

(a) Do high-activity verified contracts provide more functionality than other verified contracts (e.g. larger API)?
(b) Do high-activity verified contracts implement gas-optimized functions (hence the larger complexity)?

(OQ07) Why is the source code of high-activity verified contracts generally small?

(a) What is the rationale for the simplified usage of such a new and powerful programming paradigm? Is there a simple explanation or is it a combination of factors? Is it due to many contracts being simple token managers (i.e., lack of heterogeneity)?
(b) Are smart contracts small because they work together to implement more complex functionality? What is the typical size of ĐApps?
(c) How similar to each other is the source code of high-activity contracts (e.g., clone detection)? Is code cloning the reason behind the stability of size metrics (i.e., number of instructions, code blocks) over time?

(OQ08) Why is the source code of high-activity verified contracts highly modular (internally)?

(a) Is there a simple explanation or is it a combination of factors? Is it to ease development and maintenance of new versions? Is it to foster behavior transparency? Is it to ease gas usage estimation (Section 2.3)?
(b) Are Solidity developers more experienced (e.g., good object-oriented design skills)?

(OQ09) Why is the source code of high-activity verified contract extensively documented?

(a) Is there a simple explanation or is it a combination of factors? Is it an effort towards promoting transparency of the code behavior?
(b) What is the extent to which such documentation helps code review activities or promote code reuse?
(c) Why has the comment ratio went down recently? Is it because code is becoming more adherent to standards (e.g., ERC20) and thus requires less explanation?

(OQ10) What are the software engineering practices employed by the developers of smart contracts (especially high-activity contracts)?

(a) Do developers employ known best practices to develop smart contracts (e.g., those defined by ConsenSys' experts?[38])
(b) Do developers reuse reliable building blocks to develop smart contracts (e.g., those provided by the DappSys[39] and OpenZeppelin[40] projects)?

6 Related Work

Exploratory Analyses of Smart Contracts Bartoletti and Pompianu (2017) downloaded and manually analyzed the source code of 811 verified smart contracts available on the Etherscan website as of January 1st, 2017. They focused on two aspects: (i) categories and (ii) design patterns. As a result of their manual analysis, the authors classified contracts into the following categories: financial, notary, game, wallet, library, and unclassified. They observed that 82% of the transactions in Ethereum are processed by contracts in the financial category, which corroborates our findings from RQ2. In terms of design patterns, the authors identified the following instances: token, authorization, oracle, randomness, poll, time constraint, termination, math, fork check, and none. Similarly to object-oriented design patterns, the aforementioned patterns are not mutually exclusive. The paper includes a discussion on which categories use which patterns the most (and vice-versa). In our paper, in addition to performing a manual analysis of the source code, we also relied on the categories and tags informed by the developers themselves when registering their ĐApps on State of the ĐApps. Nevertheless, we acknowledge that there is still a vast field of research in terms of understanding how contracts are built (e.g., discovering which subcontracts and libraries tend to be reused across contracts).

Tonelli et al. (2018) downloaded the source code of the 12,000 verified smart contracts that were available on Etherscan at the time the paper was written. The authors focused

[38] https://consensys.github.io/smart-contract-best-practices
[39] https://github.com/dapphub/dappsys
[40] https://openzeppelin.com/contracts

on calculating and analyzing several source code metrics, including: lines of code (LoC), blank lines, comment lines, number of static calls to events, number of modifiers, number of functions, number of payable functions, cyclomatic complexity, number of mappings to addresses, size of the contract's ABI vector, and size of the contract's bytecode. For each of these metrics, the authors calculated descriptive statistics (e.g., mean, median, sd) and tried to fit the metrics to standard statistical distributions (Power Law and Log-Normal). Their main conclusion is that only LoC is well-fitted by a Power Law. In particular, they mention that for most variables, all values are generally into a range of few standard deviations from the mean (although this is expected for most statistical distributions). According to the authors, the source code of smart contracts have different characteristics compared to that of other programming languages. In our study, we focused on the *complexity* aspect of the source code and we observed that the number of instructions, number of code sections (subcontracts and libraries), and the comment per instruction ratio are different from what one would expect for languages such as Java, C#, and C++. Therefore, our results corroborate the claims of Tonelli et al. (2018), in the sense that the source code of smart contracts have specific complexity characteristics.

Security of Smart Contracts Since blockchains applications typically operated on currencies of some form, there is a large concern in the communities of both researchers and practitioners around the security aspect of smart contracts. Such a concern became even more relevant after the incident with "The DAO". Luu et al. (2016) wrote a symbolic execution tool called OYENTE[41] to find potential security bugs. According to the authors, the tool flagged 45.6% of the contracts in Ethereum as potentially vulnerable. Kalra et al. (2018) also leverage symbolic execution to verify the correctness and fairness of smart contracts. Correctness is defined as adherence to safe programming practices, while fairness is adherence to agreed upon higher-level business logical (i.e., does the contract do what the author says it does?) The fairness evaluation is the main novelty compared to previous work. According to the authors, 94.6% of the smart contracts are vulnerable to one or more correctness issues. The authors claim that ZEUS has zero false negatives and a low false positive rate. Chen et al. (2018) focus on discovering Ponzi schemes on Ethereum using a machine learning classifier built with features from user accounts and *op codes* from the smart contract bytecode. A Ponzi scheme is a classic type of fraud, similar to the pyramid scheme. Authors claim to have found more than 400 Ponzi schemes running on Ethereum. Bartoletti et al. (2017) have also written a paper on the same topic. The infestation of Ponzi schemes in Ethereum was also discussed in a Financial Times article (Kaminska 2017). Finally, most recently, researchers have started to build static analysis to detect bugs in smart contracts (Tikhomirov et al. 2018; Grishchenko et al. 2018). In industry, auditing companies for smart contracts have emerged, promising to ensure that newly written smart contracts are as free of bugs as possible. Some of these companies include the Solified Team[42] and Securify.[43]

Others Zheng et al. (2018) propose high-level and low-level performance metrics for different blockchain systems (including Ethereum), including a real-time monitoring framework. The authors claim that the monitoring framework has much lower overhead and offers richer performance information compared with previous approaches. Fröwis and Böhme (2017)

[41] https://github.com/melonproject/oyente
[42] https://www.youtube.com/channel/UCpEUyenjL908MFMCO-J_yhw
[43] https://securify.ch

investigate the immutability of the control flow of smart contracts by means of static analysis. According to the authors, not only code immutability is necessary for trustlessness, but also control flow immutability (i.e., call relationships between contracts should not change on runtime). To find immutability violations, the authors extract call relationships between smart contracts and search for contract addresses that are provided as input parameters or that are read from state variables. The authors conclude that two out of five smart contracts require trust in at least one third party.

7 Threats to Validity

Construct Validity In this paper, we generally focused our analysis on contracts with high activity. The way we defined the threshold choice potentially excludes some relevant contracts (e.g., those that almost reached the minimum number of received transactions to be included in our dataset). Still, we believe that our inclusion criterion reflects a large portion of contracts that are inherently relevant to the Ethereum platform user base and software engineering researchers.

Our identification of ĐApps rely solely on State of the ĐApps. Not all existing ĐApps might be registered in such a website. Also, the developer of a ĐApp might choose not to inform the addresses of the used contracts. Given these constraints, we were able to automatically retrieve domain-related information for only a subset of the studied contracts. Similarly, our classification of ĐApp categories come straight from the State of the ĐApps website. In particular, this website has a predefined list of categories from which the developer has to select when submitting their ĐApp to the website. Therefore, our classification of ĐApps is directly influenced by the categories made available by State of the ĐApps.

We use a regular expression to spot smart contracts that primarily manage tokens. While we believe that our expression is conservative and thus precise, it might still miss contracts that solely manage tokens (i.e., it may have a low recall). In this sense, our quantification of token contracts likely serves as a lower bound. Additional work is required in order to better classify token contracts.

Our analysis of code complexity relies on the source code that is published as a result of the verification process employed by Etherscan (as described in Section 2.4). We acknowledge that investigating code complexity directly on the bytecode or on decompiled code are complementary approaches that could be investigated as part of future work. We refrained from operating directly on bytecode because it does not contain code comments (which we analyze as part of RQ3). In addition to suffering from the same limitation, decompilation takes time and tool support is still a research topic (Grech et al. 2019). Therefore, we choose to operate directly on the source code of verified contracts. Nevertheless, we acknowledge that the source code that we collected is susceptible to any bugs that the verification process from Etherscan might have.

The notion of code complexity is frequently associated with how pieces of code are connected to each other (code dependencies). Our Solidity parser currently does not detect cross-contract calling relationships (independently of whether the contracts reside in the same address or not). Enhancing our parser and investigating how coupled smart contracts are is part of our future work.

External Validity The population of smart contracts studied in this paper includes all Solidity contracts deployed in Ethereum. The only exception are contracts created via internal transactions, which are not present in our dataset. In addition, the characteristics of smart

contracts deployed in other blockchain platforms (e.g., EOS) are potentially different from those that we studied. Therefore, we acknowledge that additional replication studies are required in order to further generalize our results. We emphasize, nevertheless, that the goal of this paper is not to build a theory that applies to all smart contracts of all blockchain platforms, but rather to make developers and researchers aware of key characteristics of smart contracts that are frequently used in Ethereum.

8 Conclusion

The growing number of smart contracts being deployed in the Ethereum blockchain platform has attracted the attention of media outlets, industries, and researchers. Nonetheless, prior research has focused on particular aspects of smart contracts, such as their security, while rarely putting findings in a bigger context (e.g., how frequently a smart contract is being used).

In this paper, we take a more holistic view towards smart contracts in an attempt to characterize them. We focus on three key characteristics of these contracts, namely: activity level (Section 4.1), category (Section 4.2), and code complexity (Section 4.3). Relying on cross-linked data collected from Google BigData, Etherscan, State of the ĐApps, and CoinMarketCap, we conclude that:

- Activity level is concentrated on a very small subset of the contracts and almost three quarters of them are verified.
- Despite the hype around blockchain-powered applications, the main application of smart contracts is still constrained to token management (e.g., ICOs, Crowdsales, etc).
- The source code of high-activity verified contracts is small, commonly includes at least 2 libraries/subcontracts, and is extensively documented.

We believe that by (i) providing empirically sound evidence from cross-linked data regarding the usage of smart contracts, (ii) elucidating key properties of smart contracts currently deployed in Ethereum, and (iii) defining an open research agenda, we foster a deeper understanding of the Ethereum platform and support future research in the area of smart contracts.

Acknowledgements This research has been supported by the Natural Sciences and Engineering Research Council (NSERC). This study leveraged the computational resources provided by the Microsoft Azure for Research program.

Appendix: The Top-10 Most Active Contracts on Ethereum

The top two contracts are part of decentralized currency exchange ĐApps. The webpage of IDEX is shown in Fig. 20. On the left-hand side, we can see the list of cryptocurrencies that are supported by IDEX. On the right-hand side, it shows the price chart and volume of exchanges for the cryptocurrency that we selected (AURA). Right below the price chart, we can see options for buying and selling aura (using Ether). We highlight that both etherdelta_2 and IDEX_1 contracts do not define a token of their own (check column "Own Token"). Instead, they operate on cryptocurrency tokens created by other contracts.

The third contract was an ICO for the EOS token. It was a crowdfunding initiative for the EOSIO project, whose goal is to build a new blockchain platform that can process millions

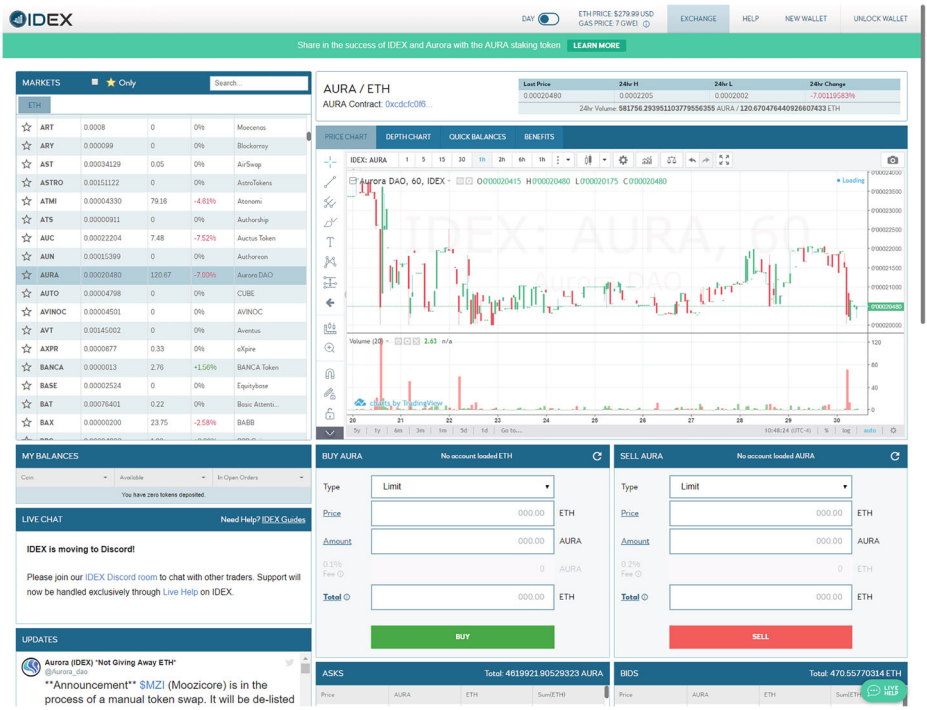

Fig. 20 A screenshot of the IDEX currency exchange ĐApp

of transactions per second. As of August 2018, the EOS token is one of the most valuable cryptocurrencies with a market capitalization of US$5 billion.[44] The EOSIO blockchain has been released as open source software and stable versions are already available. A convenient quickstart guide providing a Docker image is available at the EOSIO Developer portal.[45]

The fourth contract is part of a game ĐApp called CryptoKitties. This contract deals with core aspects of the game and has a higher number of lines of code compared to the others in our list. CryptoKitties is considered the first game to achieve widespread success on the Ethereum platform. As described in their website, "CryptoKitties is a game centered around breedable, collectible, and oh-so-adorable creatures we call CryptoKitties! Each cat is one-of-a-kind and 100% owned by you; it cannot be replicated, taken away, or destroyed". The game is clearly in the realm of digital collectibles, allowing people to buy, sell, and trade CryptoKitties (similarly to traditional collectibles like trading cards). While the vast majority of cryptokitties sell for less than US$100, a few *rare* kitties sell for far more money (Galea 2017). For instance, one of the rarest kitties is the very first one created by the developers. The token representing this kitty was sold on December 2nd 2017 for US$113,082.15.[46] Figure 21 shows kitties for sale and Fig. 22 shows the profile of the second kitty (id #1044853).

[44] https://coinmarketcap.com/currencies/eos

[45] https://developers.eos.io/eosio-nodeos/docs/docker-quickstart

[46] The genesis cat is displayed at https://www.cryptokitties.co/kitty/1. The purchase transaction can be seen at https://etherscan.io/tx/0xf365be10a326b894cc13ddd3edf55a2db6ec517e1af83741df61fb9b09b37118.

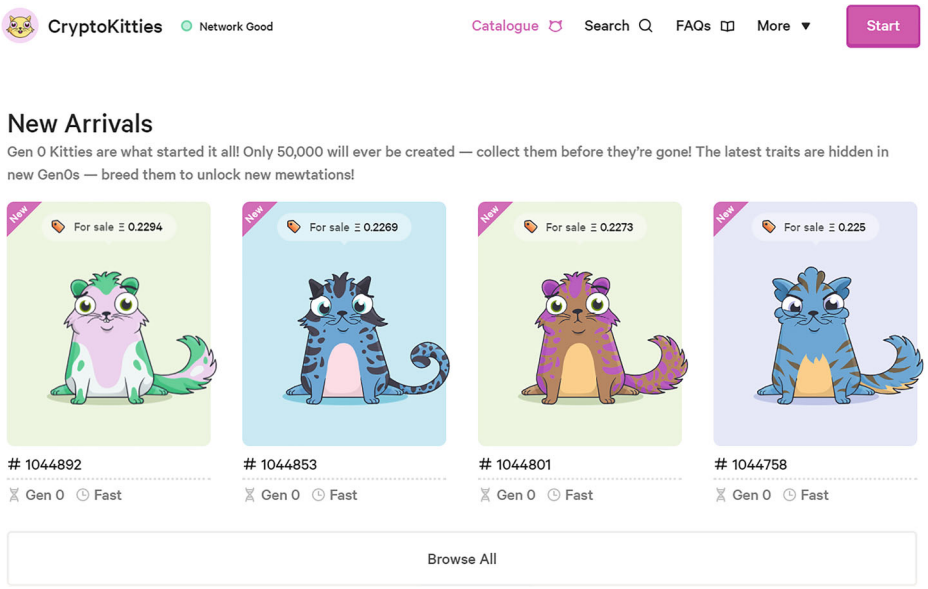

Fig. 21 The CryptoKitties game. The image shows rare Generation 0 kitties (i.e., those created by the developers of the game) for sale. Below each kitty is its unique identifier. After a kitty breeds with another kitty, it will be temporarily unable to breed again. The "Fast" tag below these kitties indicates that this recovery time is short for them (1 minute)

The fifth contract is called TronToken. It defines and manages the Tronix (TRX) token. This token is the cryptocurrency that was sold in the ICO to bootstrap the Tron project, which advertises itself as "one of the largest blockchain-based operating systems in the world". The project raised US$70 million in the ICO (all tokens were sold). Ultimately, TRON is a domain-specific blockchain platform. Tron aims to be a content distribution platform for the digital entertainment industry, in which creators have the power to freely publish, store, and own their content, interacting directly with consumers. The selling point of Tron is making entertainment content easier to sell and cheaper to consume by removing the man-in-the-middle. The Tron project is already operational and people use the token contract to operate on the tokens (e.g., transfer tokens between accounts). The Tron blockchain can be explored by means of the TronScan website,[47] which operates analogously to the Etherscan.io website (Ethereum blockchain explorer).

The sixth contract is not verified, so its name is not available. However, searching for its address on Etherscan revealed that such a contract is part of the Poloniex Exchange ĐApp.

The seventh and eighth contracts are part of the Bittrex ĐApp, which is yet another cryptocurrency exchange. Interestingly, the Controller contract is used internally by the Bittrex company, as it manages the creation of wallets and other managerial tasks.

The ninth contract is an attempt to have a Bitcoin-like token in Ethereum. The key difference compared to regular Ethereum tokens is that it is mintable. Instead of issuing a supply of coins via an ICO or similar mechanism, the contract offers a `mine()` function that delivers 1 BTCM per call. And that's the only way to generate coins. The contract allows only 50 calls to `mine()` per 10 minutes (across the whole Ethereum platform, not per client). The

[47] https://tronscan.org

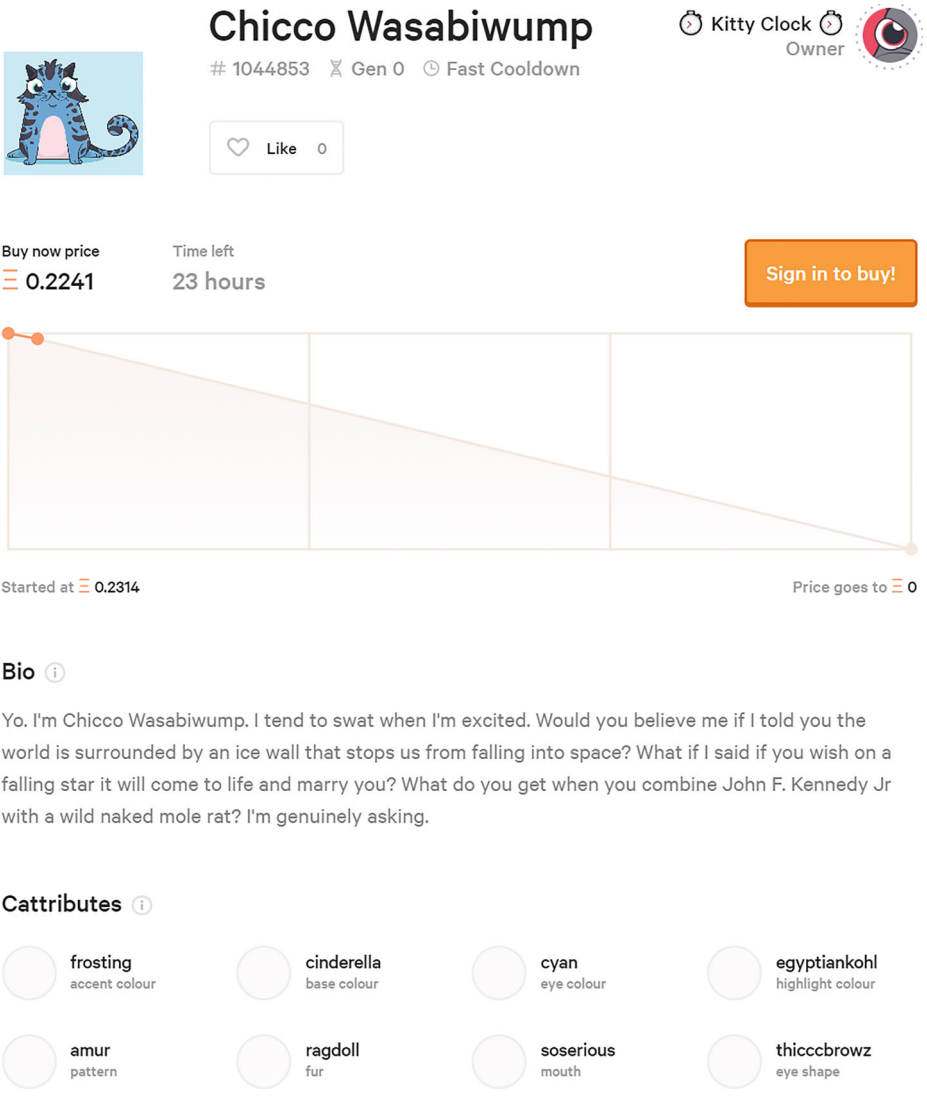

Fig. 22 The profile of the kitty with ID 1044853 (check Fig. 21). At the top of the page, we can see the kitty's name and its owner. The owner defines the start and end prices (similarly to an auction). Bio is a simple biography of the kitty. Catttributes are the attributes of the kitty, which indicate its rarity and also influence the profile of its children once it breads with another kitty

maximum supply allowed by the contract is capped at 21,000,000 BTCM (same as Bitcoin). This supply is projected to be achieved (minted) in 132 years.

Finally, the last contract OMGToken is the token contract for the OMG token, which was sold in an ICO to crowdfund the OmiseGO ĐApp. OmigoGo is yet another cryptocurrency exchange. The advisors of OmiseGo include Vitalin Buterin and Gavin Wood, who are the co-founders of Ethereum. The OmiseGo (OMG) token was the first Ethereum cryptocurrency to surpass a market capitalization of US$1 billion (later on, other coins achieved similar status) (Russell 2017).

References

Bartoletti M, Pompianu L (2017) An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Brenner M, Rohloff K, Bonneau J, Miller A, Ryan PY, Teague V, Bracciali A, Sala M, Pintore F, Jakobsson M (eds) Financial cryptography and data security. Springer International Publishing, Cham, pp 494–509

Bartoletti M, Carta S, Cimoli T, Saia R (2017) Dissecting ponzi schemes on Ethereum: identification, analysis, and impact. arXiv:1703.03779

BBC (2017) CryptoKitties craze slow down transactions on Ethereum. https://www.bbc.com/news/technology-42237162, [Online; accessed 26-August-2018]

Buterin V (2018) Layer 1 should be innovative in the short term but less in the long term. https://vitalik.ca/general/2018/08/26/layer_1.html, [Online (constantly evolving); accessed 10-August-2018]

Chen W, Zheng Z, Cui J, Ngai E, Zheng P, Zhou Y (2018) Detecting ponzi schemes on Ethereum: towards healthier blockchain technology. In: Proceedings of the 2018 world wide web conference, international world wide web conferences steering committee. Republic and Canton of Geneva, Switzerland, WWW '18, pp 1409–1418. https://doi.org/10.1145/3178876.3186046

Economist T (2018) Blockchain technology may offer a way to re-decentralise the internet. https://www.economist.com/special-report/2018/06/30/blockchain-technology-may-offer-a-way-to-re-decentralise-the-internet, [Online; accessed 10-August-2018]

Evans JD (1995) Straightforward statistics for the behavioral sciences. Brooks/Cole Pub Co

Fröwis M, Böhme R (2017) In code we trust? In: Garcia-Alfaro J, Navarro-Arribas G, Hartenstein H, Herrera-Joancomartí J (eds) Data privacy management, cryptocurrencies and blockchain technology. Springer International Publishing, Cham, pp 357–372

Galea A (2017) Ethereum CryptoKitties: by the numbers. https://medium.com/@galea/crypto-kitties-by-the-numbers-6d3bbd791aac, [Online; accessed 26-August-2018]

Grech N, Brent L, Scholz B, Smaragdakis Y (2019) Gigahorse: thorough, declarative decompilation of smart contracts. In: Proceedings of the 41st international conference on software engineering, ICSE '19. IEEE Press, Piscataway, pp 1176–1186, https://doi.org/10.1109/ICSE.2019.00120

Grishchenko I, Maffei M, Schneidewind C (2018) Foundations and tools for the static analysis of Ethereum smart contracts. In: Chockler H, Weissenbacher G (eds) Computer aided verification. Springer International Publishing, Cham, pp 51–78

Gudgeon L, Moreno-Sanchez P, Roos S, McCorry P, Gervais A (2019) Sok: off the chain transactions. IACR Cryptology ePrint Archive 2019:360. https://eprint.iacr.org/2019/360

Jakobsson M, Juels A (1999) Proofs of work and bread pudding protocols. In: Proceedings of the IFIP TC6/TC11 joint working conference on secure information networks: communications and multimedia security, CMS '99. Kluwer, B.V., Deventer, pp 258–272. http://dl.acm.org/citation.cfm?id=647800.757199

Jenks GF, Caspall FC (1971) Error on choroplethic maps: definition, measurement, reduction. Ann Assoc Am Geogr 61(2):217–244. https://doi.org/10.1111/j.1467-8306.1971.tb00779.x

Kalra S, Goel S, Dhawan M, Sharma S (2018) ZEUS: analyzing safety of smart contracts. In: 25th Annual network and distributed system security symposium, NDSS 2018. The Internet Society, NDSS '18, San Diego

Kaminska I (2017) It's not just a Ponzi, it's a 'smart' Ponzi. https://ftalphaville.ft.com/2017/06/01/2189634/its-not-just-a-ponzi-its-a-smart-ponzi/, [Online; accessed 26-August-2018]

Luu L, Chu DH, Olickel H, Saxena P, Hobor A (2016) Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, CCS '16. ACM, New York, pp 254–269, https://doi.org/10.1145/2976749.2978309

Marr B (2018) 35 Amazing real world examples of how blockchain is changing our world. https://www.forbes.com/sites/bernardmarr/2018/01/22/35-amazing-real-world-examples-of-how-blockchain-is-changing-our-world, [Online; accessed 26-August-2018]

McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 2(4):308–320. https://doi.org/10.1109/TSE.1976.233837

Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating github for engineered software projects. Empir Softw Eng 22(6):3219–3253. https://doi.org/10.1007/s10664-017-9512-6

Popper N (2017) Understanding Ethereum, Bitcoin's virtual cousin. https://www.nytimes.com/2017/10/01/technology/what-is-Ethereum.html, [Online; accessed 10-August-2018]

Richmond F (2018) Exposing blockchain: an inside look at the technology behind smart contracts, cryptocurrency wallets, cryptocurrency mining, Bitcoin and other digital coins (Ethereum Litecoin, Ripple and More). Independently published

Romano J, Kromrey J, Coraggio J, Skowronek J (2006) Appropriate statistics for ordinal level data: should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys? In: Annual meeting of the Florida Association of Institutional Research, pp 1–3

Russell J (2017) The first ICO unicorns are here. https://techcrunch.com/2017/08/31/the-first-ico-unicorns-are-here/, [Online; accessed 26-August-2018]

Sorkin AR (2010) Too big to fail: the inside story of how wall street and washington fought to save the financial system–and themselves. Penguin Books

Swan M (2015) Blockchain: blueprint for a new economy, 1st edn. O'Reilly Media, Inc

Szabo N (1994) Smart contracts. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html, [Online; accessed 26-August-2018]

Tikhomirov S, Voskresenskaya E, Ivanitskiy I, Takhaviev R, Marchenko E, Alexandrov Y (2018) Smartcheck: static analysis of Ethereum smart contracts. In: Proceedings of the 1st international workshop on emerging trends in software engineering for blockchain, WETSEB '18. ACM, New York, pp 9–16, https://doi.org/10.1145/3194113.3194115

Tonelli R, Destefanis G, Marchesi M, Ortu M (2018) Smart contracts software metrics: a first study. arXiv:1802.01517

Vilner Y (2018) 5 blockchain product use cases to follow this year. https://www.forbes.com/sites/yoavvilner/2018/06/27/5-blockchain-product-use-cases-to-follow-this-year, [Online; accessed 26-August-2018]

Wan Z, Lo D, Xia X, Cai L (2017) Bug characteristics in blockchain systems: a large-scale empirical study. In: Proceedings of the 14th international conference on mining software repositories, MSR '17. IEEE Press, Piscataway, pp 413–424, https://doi.org/10.1109/MSR.2017.59

Wood G (2017) Ethereum: a secure decentralised generalised transaction ledger - EIP-150 revision. http://yellowpaper.io/, [Online; accessed 10-August-2018]

Zheng P, Zheng Z, Luo X, Chen X, Liu X (2018) A detailed and real-time performance monitoring framework for blockchain systems. In: Proceedings of the 40th international conference on software engineering: software engineering in practice, ICSE-SEIP '18. ACM, New York, pp 134–143, https://doi.org/10.1145/3183519.3183546

Dr. Gustavo A. Oliva is a Post-Doctoral Fellow at the School of Computing of Queen's University in Canada under the supervision of professor Dr. Ahmed E. Hassan. Gustavo leads the blockchain research team at the Software Analysis and Intelligence Lab (SAIL). In his research, Gustavo leverages his expertise on mining software repositories, social network analysis, and machine learning to investigate blockchain technology through the lens of Software Engineering. Gustavo received his MSc and PhD degrees from the University of São Paulo (USP) in Brazil under the supervision of professor Dr. Marco Aurélio Gerosa. More information at: https://www.gaoliva.com/.

Dr. Ahmed E. Hassan is an IEEE Fellow, an ACM SIGSOFT Influential Educator, an NSERC Steacie Fellow, the Canada Research Chair (CRC) in Software Analytics, and the NSERC/BlackBerry Software Engineering Chair at the School of Computing at Queen's University, Canada. His research interests include mining software repositories, empirical software engineering, load testing, and log mining. He received a PhD in Computer Science from the University of Waterloo. He spearheaded the creation of the Mining Software Repositories (MSR) conference and its research community. He also serves/d on the editorial boards of IEEE Transactions on Software Engineering, Springer Journal of Empirical Software Engineering, and PeerJ Computer Science. More information at: https://sail.cs.queensu.ca.

Dr. Zhen Ming (Jack) Jiang is an associate professor at the Department of Electrical Engineering and Computer Science, York University in Toronto, Canada. His research interests lie within software engineering and computer systems, with special interests in software performance engineering, software analytics, source code analysis, software architectural recovery, software visualizations, and debugging and monitoring of distributed systems. Some of his research results are already adopted and used in practice on a daily basis. He is the cofounder of the annually held International Workshop on Load Testing and Benchmarking of Software Systems (LTB). He received several Best Paper Awards including ICST 2016, ICSE 2015 (SEIP track), ICSE 2013, and WCRE 2011. He received the BMath and MMath degrees in computer science from the University of Waterloo, and the PhD degree from the School of Computing at the Queen's University. More information at: http://www.cse.yorku.ca/~zmjiang.

Affiliations

Gustavo A. Oliva[1] · **Ahmed E. Hassan**[1] · **Zhen Ming (Jack) Jiang**[2]

Ahmed E. Hassan
ahmed@cs.queensu.ca

Zhen Ming (Jack) Jiang
zmjiang@cse.yorku.ca

[1] Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Kingston, Ontario, Canada

[2] Department of Electrical Engineering, Computer Science, York University, Toronto, Ontario, Canada