An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns

Article *in* Lecture Notes in Computer Science · March 2017

DOI: 10.1007/978-3-319-70278-0_31

An empirical analysis of smart contracts: platforms, applications, and design patterns

Massimo Bartoletti and Livio Pompianu

Università degli Studi di Cagliari, Cagliari, Italy
{bart,livio.pompianu}@unica.it

Abstract. Smart contracts are computer programs that can be consistently executed by a network of mutually distrusting nodes, without the arbitration of a trusted authority. Because of their resilience to tampering, smart contracts are appealing in many scenarios, especially in those which require transfers of money to respect certain agreed rules (like in financial services and in games). Over the last few years many platforms for smart contracts have been proposed, and some of them have been actually implemented and used. We study how the notion of smart contract is interpreted in some of these platforms. Focussing on the two most widespread ones, Bitcoin and Ethereum, we quantify the usage of smart contracts in relation to their application domain. We also analyse the most common programming patterns in Ethereum, where the source code of smart contracts is available.

1 Introduction

Since the release of Bitcoin in 2009 [40], the idea of exploiting its enabling technology to develop applications beyond currency has been receiving increasing attention [26]. In particular, the public and append-only ledger of transaction (the *blockchain*) and the decentralized consensus protocol that Bitcoin nodes use to extend it, have revived Nick Szabo's idea of *smart contracts* — i.e. programs whose correct execution is automatically enforced without relying on a trusted authority [47]. The archetypal implementation of smart contracts is Ethereum [28], a platform where they are rendered in a Turing-complete language. The consensus protocol of Ethereum ensures that all and only the valid updates to the contract states are recorded on the blockchain, so ensuring their correct execution.

Besides Bitcoin and Ethereum, a remarkable number of alternative platforms have flourished over the last few years, either implementing crypto-currencies or some forms of smart contracts [1, 7, 9, 30, 37]. For instance, the number of crypto-currencies hosted on coinmarketcap.com has increased from 0 to more than 600 since 2012; the number of github projects related to blockchains and smart contracts has reached, respectively, 2,715 and 445 units (see Figure 1). In the meanwhile, ICT companies and some national governments have started dealing with these topics [41, 48], also with significant investments.

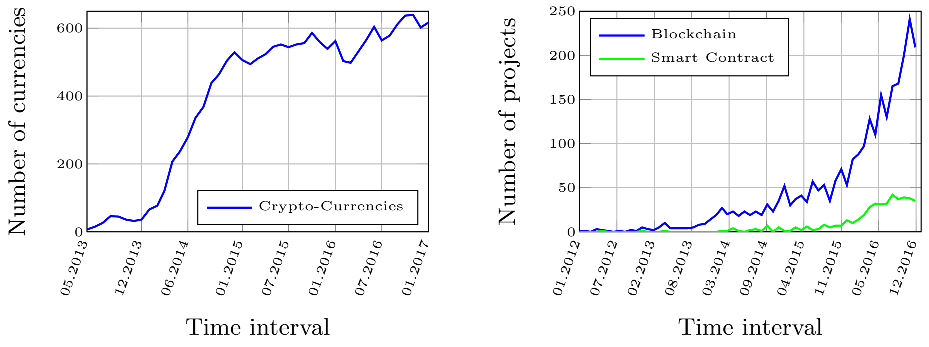

Fig. 1: On the left, monthly trend of the number of crypto-Currencies hosted on coinmarketcap.com. On the right, number of new projects related to blockchains and smart contracts which are created every month on github.com.

Despite the growing hype on blockchains and smart contracts, the understanding of the actual benefits of these technologies, and of their trustworthiness and security, has still to be assessed. In particular, the consequences of unsafe design choices for the programming languages for smart contracts can be fatal, as witnessed by the unfortunate epilogue of the DAO contract [13], a crowdfunding service plundered of $\sim 50M$ USD because of a programming error. Since then, many other vulnerabilities in smart contract have been reported [12,14,18,37].

Understanding how smart contracts are used and how they are implemented could help designers of smart contract platforms to create new domain-specific languages (not necessarily Turing complete [27,29,33,42]), which *by-design* avoid vulnerabilities as the ones discussed above. Further, this knowledge could help to improve analysis techniques for smart contracts (like e.g. the ones in [25,37]), by targeting contracts with specific programming patterns.

Contributions. This paper is a methodic survey on smart contracts, with a focus on Bitcoin and Ethereum — the two most widespread platforms currently supporting them. Our main contributions can be summarised as follows:

– in Section 2 we examine the Web for news about smart contracts in the period from June 2013 to September 2016, collecting data about 12 platforms. We choose from them a sample of 6 platforms which are amenable to analytical investigation. We analyse and compare several aspects of the platforms in this sample, mainly concerning their usage, and their support for programming smart contracts.
– in Section 3 we propose a taxonomy of smart contracts, sorting them into categories which reflect their application domain. We collect from the blockchains of Bitcoin and Ethereum a sample of 834 smart contracts, which we classify according to our taxonomy. We then study the usage of smart contracts, measuring the distribution of their transactions by category. This allows us

to compare the different usage of Bitcoin and Ethereum as platforms for smart contracts.

- in Section 4 we consider the source code of the Ethereum contracts in our sample. We identify 9 common design patterns, and we quantify their usage in contracts, also in relation to the associated category. Together with the previous point, ours constitutes the first quantitative investigation on the usage and programming of smart contract in Ethereum.

All the data collected by our survey are availble online at: goo.gl/pOswL8.

2 Platforms for smart contracts

In this section we analyse various platforms for smart contracts. We start by presenting the methodology we have followed to choose the candidate platforms (Section 2.1). Then we describe the key features of each platform, pinpointing differences and similarities, and drawing some general statistics (Section 2.2).

2.1 Methodology

To choose the platforms subject of our study, we have drawn up a candidate list by examining all the articles of coindesk.com in the "smart contracts" category[1]. Starting from June 2013, when the first article appeared, up to the 15th of September 2016, 175 articles were published, describing projects, events, companies and technologies related to smart contracts and blockchains. By manually inspecting all these articles, we have found references to 12 platforms: Bitcoin, Codius, Counterparty, DAML, Dogeparty, Ethereum, Lisk, Monax, Rootstock, Symbiont, Stellar, and Tezos.

We have then excluded from our sample the platforms which, at the time of writing, do not satisfy one of the following criteria: (i) have already been launched, (ii) are running and supported from a community of developers, and (iii) are publicly accessible. For the last point we mean that, e.g., it must be possible to write a contract and test it, or to explore the blockchain through some tools, or to run a node. We have inspected each of the candidate platforms, examining the related resources available online (e.g., official websites, white-papers, forum discussions, *etc.*) After this phase, we have removed 6 platforms from our list: Tezos and Rootstock, as they do not satisfy condition (i); Codius and Dogeparty, which violate condition (ii), DAML and Symbiont, which violate (iii). Summing up, we have a sample of 6 platforms: Bitcoin, Ethereum, Counterparty, Stellar, Monax and Lisk, which we discuss in the following.

2.2 Analysis of platforms

We now describe the general features of the collected platforms, focussing on: (i) whether the platform has its own blockchain, or if it just piggy-backs on an

[1] http://www.coindesk.com/category/technology/smart-contracts-news

already existing one; (ii) for platforms with a public blockchain, their consensus protocol, and whether the blockchain is public or private to a specific set of nodes; (iii) the languages used to write smart contracts.

Bitcoin [40] is a platform for transferring digital currency, the bitcoins (BTC). It has been the first decentralized cryptocurrency to be created, and now is the one with the largest market capitalization. The platform relies on a public blockchain to record the complete history of currency transactions. The nodes of the Bitcoin network use a consensus algorithm based moderately hard *"proof-of-work"* puzzles to establish how to append a new block of transactions to the blockchain. Nodes work in competition to generate the next block of the chain. The first node that solves the puzzle earns a reward in BTC.

Although the main goal of Bitcoin is to transfer currency, the immutability and openness of its blockchain have inspired the development of protocols that implement (limited forms of) smart contracts. Bitcoin features a non-Turing complete scripting language, which allows to specify under which conditions a transaction can be redeemed. The scripting language is quite limited, as it only features some basic arithmetic, logical, and crypto operations (e.g., hashing and verification of digital signatures). A further limitation to its expressiveness is the fact that only a small fraction of the nodes of the Bitcoin network processes transactions whose script is more complex than verifying a signature[2].

Ethereum [28] is the second platform for market capitalization, after Bitcoin. Similarly to Bitcoin, it relies on a public blockchain, with a consensus algorithm similar to that of Bitcoin[3]. Ethereum has its own currency, caller *ether* (ETH). Smart contracts are written in a stack-based bytecode language [49], which is Turing-complete, unlike Bitcoin's. There also exist a few high level languages (the most prominent being *Solidity*[4]), which compile into the bytecode language. Users create contracts and invoke their functions by sending transactions to the blockchain, whose effects are validated by the network. Both users and contracts can store money and send/receive ETH to other contracts or users.

Counterparty [32] is a platform without its own blockchain; rather, it embeds its data into Bitcoin transactions. While the nodes of the Bitcoin network ignore the data embedded in these transactions, the nodes of Counterparty recognise and interpret them. Smart contracts can be written in the same language used by Ethereum. However, unlike Ethereum, no consensus protocol is used to validate the results of computations[5]. Counterparty has its own currency, which can be transferred between users, and be spent for executing contracts. Unlike

[2] As far as we know, currently only the *Eligius* mining pool accepts more general transactions (called *non-standard* in the Bitcoin community). However, this pool only mines $\sim 1\%$ of the total mined blocks [20].

[3] The consensus mechanism of Ethereum is a variant of the GHOST protocol in [46].

[4] Solidity: `http://solidity.readthedocs.io/en/develop/index.html`

[5] See FAQ: How do Smart Contracts "form a consensus" on Counterparty? `http://counterparty.io/docs/faq-smartcontracts/#how-do-smart-contracts-form-a-consensus-on-coun`

Ethereum, nodes do not obtain fees for executing contracts; rather, the fees paid by clients are destroyed, and nodes are indirectly rewarded from the inflation of the currency. This mechanism is called *proof-of-burn*.

Stellar [10] features a public blockchain with its own cryptocurrency, governed by a consensus algorithm inspired to federated Byzantine agreement [11]. Basically, a node agrees on a transaction if the nodes in its neighbourhood (that are considered more trusted than the others) agree as well. When the transaction has been accepted by enough nodes of the network, it becomes unfeasible for an attacker to roll it back, and it is considered as confirmed. Compared to *proof-of-work*, this protocol consumes far less computing power, since it does not involve solve cryptographic puzzles. Unlike Ethereum, there is no specific language for smart contracts; still, it is possible to gather together some transactions (possibly ordered in a chain) and write them atomically in the blockchain. Since transactions in a chain can involve different addresses, this feature can be used to implement basic smart contracts. For instance, assume that a participant A wants to pay B only if B promises to pay C after receiving the payment from A. This behaviour can be enforced by putting these transactions in the same chain. While this specific example can be implemented on Bitcoin as well, Stellar also allows to batch operations different from payments[6], e.g. creating new accounts. Stellar features special accounts, called *multisignature*, which can be handled by several owners. To perform operations on these accounts, a threshold of consensus must be reached among the owners. Transaction chaining and multisignature accounts can be combined to create more complex contracts.

Monax [8] supports the execution of Ethereum contracts, without having its own currency. Monax allows users to create private blockchains, and to define authorisation policies for accessing them. Its consensus protol[7] is organised in rounds, where a participant proposes a new block of transactions, and the others vote for it. When a block fails to be approved, the protocol moves to the next round, where another participant will be in charge of proposing blocks. A block is confirmed when it is approved by at least 2/3 of the total voting power.

Lisk [6] has its own currency, and a public blockchain with a *delegated proof-of-stake* consensus mechanism[8]. More specifically, 101 active delegates, each one elected by the stakeholders, have the authority to generate blocks. Stakeholders can take part to the electoral process, by placing votes for delegates in their favour, or by becoming candidates themselves. Lisk supports the execution of Turing-complete smart contracts, written either in JavaScript or in Node.js. Unlike Ethereum, determinism of executions is not ensured by the language: rather, programmers must take care of it, e.g. by not using functions like *Math.random*. Although Lisk has a main blockchain, each smart contract is executed on a separated one. Users can deposit or withdraw currency from a contract to the

[6] `https://www.stellar.org/developers/guides/concepts/operations.html`
[7] `https://tendermint.com/`
[8] `https://lisk.io/documentation?i=lisk-handbooks/DelegateHandbook`

Platform	**Blockchain**			**Contract Language**	**Total Tx**	**Volume**	**Marketcap**
	Type	**Size**	**Block int.**			(K USD)	(M USD)
Bitcoin	Public	96 GB	10 min.	Bitcoin scripts + signatures	184,045,240	83,178	15,482
Counterparty				EVM bytecode	12,170,386	33	4
Ethereum	Public	17-60 GB	12 sec.	EVM bytecode	14,754,984	10,354	723
Stellar	Public	?	3 sec.	Transaction chains + signatures	?	35	17
Monax	Private	?	Custom	EVM bytecode + permissions	?	n/a	n/a
Lisk	Private	?	Custom	JavaScript	?	45	15

Table 1: General statistics of platforms for smart contracts.

main chain, while avoiding double spending. Contract owners can customise their blockchain before deploying their contracts, e.g. choosing which nodes can participate to the consensus mechanism.

Table 1 summarizes the main features of the analysed platforms. The question mark in some of the cells indicates that we were unable to retrieve the information (e.g., we have not been able to determine the size of Monax blockchains, since they are private). The first three columns next to the platform name describe features of the blockchain: whether it is public; its size; the average time between two consecutive blocks. Note that Bitcoin and Counterparty share the same cell, since the second platform uses the Bitcoin blockchain. Measuring the size of the Ethereum blockchain depends on which client and which pruning mode is used. For instance, using the Geth client, we obtain a measure of 17GB in "fast sync" mode, and of 60GB in "archive" mode[9]. In platforms with private blockchains, their block interval is custom. The fifth column describes the support for writing contracts. The sixth column shows the total number of transactions[10]. The last two columns show the daily volume of currency tranfers, and the market capitalisation of the currency (both in USD, rounded, respectively, to thousands and millions)[11]. All values reported on Table 1 are updated to January 1st, 2017.

3 Analysing the usage of smart contracts

In this section we analyse the usage of smart contracts, proposing a classification which reflects their application domain. Then, focussing on Bitcoin and Ethereum, we quantify the usage of smart contracts in relation to their application domain. We start by presenting the methodology we have followed to sample and classify Bitcoin and Ethereum smart contracts (Section 3.1). Then, we introduce our classification and our statistical analysis (Sections 3.2 and 3.3).

[9] `https://redd.it/5om2lw`

[10] Sources: `https://blockchain.info/charts/n-transactions-total` (for Bitcoin), `https://blockscan.com` (Counterparty), and `https://etherscan.io` (Ethereum).

[11] Market capitalization estimated by `http://coinmarketcap.com`.

3.1 Methodology

We sample contracts from Bitcoin and Ethereum as follows:

– for Ethereum, we collect on January 1st, 2017 all the contracts marked as "verified" on the blockchain explorer `etherscan.io`. This means that the contract bytecode stored on the blockchain matches the source code (generally written in a high level language, such as Solidity) submitted to the explorer. In this way, we obtain a sample of 811 contracts.
– for Bitcoin, we start by observing that many smart contracts save their metadata on the blockchain through the OP_RETURN instruction of the Bitcoin scripting language [1,2,7,23]. We then scan the Bitcoin blockchain on January 1st 2017, searching for transactions that embed in an OP_RETURN some metadata attributable to a Bitcoin smart contract. To this purpose we use an explorer[12] which recognises 23 smart contracts, and extracts all the transactions related to them.

3.2 A taxonomy of smart contracts

We propose a taxonomy of smart contracts into five categories, which describe their intended application domain. We then classify the contracts in our sample according to the taxonomy. To this purpose, for Ethereum contracts we manually inspect the Solidity source code, while for Bitcoin contracts we search their web pages and related discussion forums. After this manual investigation, we distribute all the contracts into the five categories, that we present below.

Financial. Contracts in this category manage, gather, or distribute money as preeminent feature. Some contracts certify the ownership of a real-world asset, endorse its value, and keep track of trades (e.g., Colu currently tracks over 50,000 assets on Bitcoin). Other contracts implement crowdfunding services, gathering money from investors in order to fund projects (the Ethereum DAO project was the most representative one, until its collapse due to an attack in June 2016). High-yield investment programs are a type of Ponzi schemes [22] that collect money from users under the promise that they will receive back their funds with interest if new investors join the scheme (e.g., Government, KingOfTheEtherThrone). Some contracts provide an insurance on setbacks which are digitally provable (e.g., Etherisc sells insurance policies for flights; if a flight is delayed or cancelled, one obtains a refund). Other contracts publish advertisement messages (e.g., PixelMap is inspired to the Million Dollar Homepage).

Notary. Contracts in this category exploit the immutability of the blockchain to store some data persistently, and in some cases to certify their ownership and provenance. Some contracts allow users to write the hash of a document on the blockchain, so that they can prove document existence and integrity (e.g., Proof of Existence). Others allow to declare copyrights on digital arts files,

[12] `https://github.com/BitcoinOpReturn/OpReturnTool`

Category	Platform	Contracts	Transactions
Financial	Bitcoin	6	470,391
	Ethereum	373	624,046
Notary	Bitcoin	17	443,269
	Ethereum	79	35,253
Game	Bitcoin	0	0
	Ethereum	158	58,257
Wallet	Bitcoin	0	0
	Ethereum	17	1,342
Library	Bitcoin	0	0
	Ethereum	29	37,034
Unclassified	Bitcoin	0	0
	Ethereum	155	3,679
Total	Bitcoin	23	913,660
	Ethereum	811	759,611
	Overall	834	1,673,271

Table 2: Transactions by category.

like photos or music (e.g., Monegraph). Some contracts (e.g., Eternity Wall) just allow users to write down on the blockchain messages that everyone can read. Other contracts associate users to addresses (often represented as public keys), in order to certify their identity (e.g., Physical Address).

Game. This category gathers contracts which implement *games of chance* (e.g., LooneyLottery, Dice, Roulette, RockPaperScissors) and *games of skill* (e.g., Etherization), as well as some games which mix chance and skill (e.g., PRNG challenge pays for the solution of a puzzle).

Wallet. The contracts in this category handle keys, send transactions, manage money, deploy and watch contracts, in order to simplify the interaction with the blockchain. Wallets can be managed by one or many owners, in the latter case requiring multiple authorizations (like, e.g. in Multi-owned).

Library. These contracts implement general-purpose operations (like e.g., math and string transformations), to be used by other contracts.

3.3 Quantifying the usage of smart contracts by category

We analyse all the transactions related to the 834 smart contracts in our sample. Table 2 displays how the transactions are distributed in the categories of Section 3.2. For both Bitcoin and Ethereum, we show the number of detected contracts (third column), and the total number of transactions (fourth column).

Overall, we have 1,673,271 transactions. Notably, although Bitcoin contracts are fewer than those running on Ethereum, they have a larger amount of transactions each. A clear example of this is witnessed by the financial category, where 6 Bitcoin contracts[13] totalize two thirds of the transactions published by the 373 Ethereum contracts in the same category.

[13] Bitcoin financial contracts: Colu, CoinSpark, OpenAssets, Omni, SmartBit, BitPos.

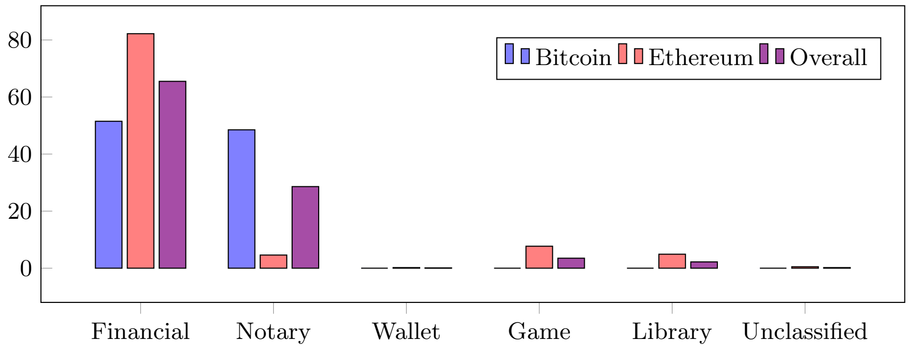

Fig. 2: Distribution of transactions by category.

While both Bitcoin and Ethereum are mainly focussed on financial contracts, we observe major differences about the other categories. For instance, the Bitcoin contracts in the Notary category[14] have an amount of transactions similar to that of the Financial category, unlike in Ethereum. The second most used category in Ethereum is Game. Although some games (e.g., lotteries [16, 17, 19, 24] and poker [36]) which run on Bitcoin have been proposed in the last few years, the interest on them is still mainly academic, and we have no experimental evidence that these contracts are used in practice. Instead, the greater flexibility of the Ethereum programming language simplifies the development of this kind of contracts (although with some quirks [31] and limitations[15]).

Note that in some cases there are not enough elements to categorise a contract. This happens e.g., when the contract does not link to the project webpage, and there are neither comments in online forums nor in the contract sources.

4 Design patterns for Ethereum smart contracts

In this section we study design patterns for Ethereum smart contracts. To this purpose, we consider the sample of 811 contracts collected through the methodology described in Section 3. By manually inspecting the Solidity source code of each of these contracts, we identify some common design patterns. We start in Section 4.1 by describing these patterns. Then, in Section 4.2 we measure the usage of the patterns in the various categories of contracts identified in Section 3.

[14] Bitcoin notary contracts: Factom, Stampery, Proof of Existence, Blocksign, Crypto-Copyright, Stampd, BitProof, ProveBit, Remembr, OriginalMy, LaPreuve, Nicosia, Chainpoint, Diploma, Monegraph, Blockai, Ascribe, Eternity Wall, Blockstore.

[15] Although the Ethereum virtual machine is designed to be Turing-complete, in practice the limitations on the amount of gas which can be used to invoke contracts also limit the set of computable functions (e.g., verifying checkmate exceeds the current gas limits of a transaction [35]).

4.1 Design patterns

Token. This pattern is used to distribute some fungible goods (represented by tokens) to users. Tokens can represent a wide variety of goods, like e.g. coins, shares, outcomes or tickets, or everything else which is transferable and countable. The implications of owning a token depend on the protocol and the use case for which the token has been issued. Tokens can be used to track the ownership of physical properties (e.g., gold [3]), or digital ones (e.g., cryptocurrency). Some crowdfunding systems issue tokens in exchange for donations (e.g., the Congress contract). Tokens are also used to regulate user authorizations and identities. For instance, the DVIP contract specifies rights and term of services for owners of its tokens. To vote on the poll ETCSurvey, users must possess a suitable token. Given the popularity of this pattern, its standardisation has been proposed [5]. Notably, the majority of the analysed Ethereum contracts which issue tokens already adhere to it.

Authorization. This pattern is used to restrict the execution of code according to the caller address. The majority of the analysed contracts check if the caller address is that of the contract owner, before performing critical operations (e.g., sending ether, invoking suicide or selfdestruct). For instance, the owner of Doubler is authorized to move all funds to a new address *at any time* (this may raise some concerns about the trustworthiness of the contract, as a dishonest owner can easily steal money). Corporation checks addresses to ensure that every user can vote only once per poll. CharlyLifeLog uses a white-list of addresses to decide who can withdraw funds.

Oracle. Some contracts may need to acquire data from outside the blockchain, e.g. from a website, to determine the winner of a bet. The Ethereum language does not allow contracts to query external sites: otherwise, the determinism of computations would be broken, as different nodes could receive different results for the same query. Oracles are the interface between contracts and the outside. Technically, they are just contracts, and as such their state can be updated by sending them transactions. In practice, instead of querying an external service, a contract queries an oracle; and when the external service needs to update its data, it sends a suitable transaction to the oracle. Since the oracle is a contract, it can be queried from other contracts without consistency issues. One of the most common oracles is Oraclize[16]: in our sample, it is used by almost all the contracts which resort to oracles.

Randomness. Dealing with randomness is not a trivial task in Ethereum. Since contract execution must be deterministic, all the nodes must obtain the same value when asking for a random number: this struggles with the randomness requirements wished. To address this issue, several contracts (e.g., Slot) query oracles that generate these values off-chain. Others (e.g., Lottery) try to generate the numbers locally, by using values not predictable *a priori*, as the hash of a block not yet created. However, these techniques are not generally considered secure [18].

[16] http://www.oraclize.it/

	Token	**Auth.**	**Oracle**	**Random.**	**Poll**	**Time**	**Termin.**	**Fork**	**Math**	**None**
Financial	24-51	51-39	2-15	1-2	5-29	23-31	14-30	8-69	4-47	29-66
Notary	13-6	52-9	1-2	0-0	8-9	20-6	29-13	0-0	1-3	30-15
Game	3-3	84-27	25-74	72-93	25-57	73-43	21-19	1-3	2-9	1-1
Wallet	18-2	100-3	0-0	0-0	0-0	94-6	100-10	0-0	12-6	0-0
Library	0-0	31-2	0-0	14-3	0-0	24-3	24-4	34-24	21-19	17-3
Unclassified	43-39	66-21	3-9	1-1	3-6	18-10	28-25	28-25	1-5	15-15
Total	*21-100*	*61-100*	*7-100*	*15-100*	*9-100*	*33-100*	*22-100*	*5-100*	*4-100*	*20-100*

Table 3: Relations between design patterns and contract categories. A pair (p, q) at row i and column j means that $p\%$ of the contracts in category i use the pattern of column j, and $q\%$ of contracts with pattern j belong to category i.

Poll. Polls allows users to vote on some question. Often this is a side feature in a more complex scenario. For instance, in the Dice game, when a certain state is reached, the owner issues a poll to decide whether an emergency withdrawal is needed. To determine who can vote and to keep track of the votes, polls can use tokens, or they can check the voters' addresses.

Time constraint. Many contracts implement time constraints, e.g. to specify when an action is permitted. For instance, BirthdayGift allows users to collect funds, which will be redeemable only after their birthday. In notary contracts, time constraints are used to prove that a document is owned from a certain date. In game contracts, e.g. Lottery, time constraints mark the stages of the game.

Termination. Since the blockchain is immutable, a contract cannot be deleted when its use has come to an end. Hence, developers must forethink a way to disable it, so that it is still present but unresponsive. This can be done manually, by inserting ad-hoc code in the contract, or automatically, calling `selfdestruct` or `suicide`. Usually, only the contract owner is authorized to terminate a contract (e.g., as in SimpleCoinFlipGame).

Math. Contracts using this pattern encode the logic which guards the execution of some critical operations. For instance, Badge implements a method named `subtractSafely` to avoid subtracting a value from a balance when there are not enough funds in an account.

Fork check. The Ethereum blockchain has been forked four times, starting from July 20th, 2016, when a fork was performed to contrast the effect of the DAO attack [4]. To know whether or not the fork took place, some contracts inspect the final balance of the DAO. Other contracts use this check to detect whether they are running on the main chain or on the fork, performing different actions in the two cases. AmIOnTheFork is a library contract that can be used to distinguish the main chain from the forked one.

4.2 Quantifying the usage of design patterns by category

We now study how the design patterns identified in Section 4.1 are used in smart contracts. Out of the 811 analysed contracts, 648 use at least one of the 9 patterns presented, for a grand total of 1427 occurrences of usage.

Table 3 shows the correlation between the usage of design patterns and contract categories, as defined in Section 3. A cell at row i and column j shows a pair of values: the first value is the percentage of contracts of category i that use the pattern of column j; the second one is the percentage of contracts with pattern j which belongs to category i. So, for instance, 24% of the contracts in the financial category use the token pattern, and 51% of all the contracts with the token pattern are financial ones.

We observe that *token*, *authorization*, *time constraint*, and *termination* are generally the most used patterns. Some patterns are spread across several categories (e.g., *termination* and *time constraint*), while others are mainly adopted only in one. For instance, *oracle* and *randomness* patterns are peculiar of game contracts, while the *token* pattern is mostly used in financial contracts. Although *math* is the less used, it appears in each category. Some contracts do not use any pattern (29% of financial and 30% of notary); almost all the contracts in game and wallet categories uses at least one. Further, only 15% of all the unclassified contracts do no use any pattern at all.

The most frequent patterns in financial contracts are *token* (24%), *authorization* (51%), and *time constraint* (23%). Due to the presence of contracts which implement assets and crowdfunding services, we have that half of contracts using *token* and *math* patterns belong to the financial category. For instance, these services use *token* for representing goods or developing polls. Moreover, a great 69% of contracts that use the *fork check* pattern is financial. This is caused by the necessity of knowing the branch of the fork before deciding to move funds. Finally, several financial applications (29%) perform simple operations (e.g. sending a payment) without using any of our described patterns.

The *authorization* pattern is used in many notary contracts to ensure that only the owner of a document can add or modify its data, in order to avoid tampering. Most gambling games involve players who pay fees to join the game, and rewards that can be collected by the winner of the game. The *authorization* pattern is used to let the owner to be the only one able to redeem participants' fees or to perform administrative operations, and to let the winner withdraw his reward. The *time constraint* pattern is used to distinguish the different phases of the game. For instance, within a specific time interval players can join the game and/or bet; then, bets are over, and the game determines a winner. To choose the winner, some gambling games resort to random numbers, which are often generated through an oracle. Indeed, 25% of games use the *oracle* pattern, and the pattern itself is used 74% of cases by a game contract. Since *all* game contracts invoking an *oracle* (25%) ask for random values, and since 72% of contracts use the *random* pattern, we can deduce that 47% of them generate random numbers without resorting to oracles.

Notably, 100% of wallet contracts adopt both *authorization* and *termination* design patterns. A high 94% also uses *time constraint*. On the contrary, *oracle*, *poll*, and *randomness* patterns are of little use when developing a wallet, while *math* is sometimes used for securing operations on the balance.

5 Conclusions

We have analysed the usage of smart contracts from various perspectives. In Section 2 we have examined a sample of 6 platforms for smart contracts, pinpointing some crucial technical differences between them. For the two most prominent platforms — Bitcoin and Ethereum — we have studied a sample of 834 contracts, categorizing each of them by its application domain, and measuring the relevance of each of these categories (Section 3). The availability of source code for Ethereum contracts has allowed us to analyse the most common design patterns adopted when writing smart contracts (Section 4).

We believe that this survey may provide valuable information to developers of new, domain-specific languages for smart contracts. In particular, measuring what are the most common use cases allows to understand which domains deserve more investments. Furthermore, our study of the correlation between design patterns and application domains can be exploited to drive the correct choice of programming primitives of domain-specific languages for smart contracts.

Due to the mixed flavour of our analysis, which compares differents platforms and studies how smart contracts are interpreted on each them, our work relates to various topics. The work [38] proposes design patterns for altering and undoing of smart contracts; so far, our analysis in Section 4.2 has not still found instances of these patterns in Ethereum. Among the works which study blockchain technologies, [15] compares four blockchains, with a special focus on the Ethereum one; [45] examines a larger set of blockchains, including also some which does not fit the criteria we have used in our methodology (e.g., RootStock and Tezos). Many works on Bitcoin perform empirical analyses of its blockchain. For instance, [43, 44] study users deanonymization, [39] measures transactions fees, and [21] analyses Denial-of-Service attacks on Bitcoin. The work [34] investigates whether Bitcoin users are interested more on digital currencies as asset or as currency, with the aim of detecting the most popular use cases of Bitcoin contracts, similarly to what we have done in Section 3.3. Our classification of Bitcoin protocols based on OP_RETURN transactions is inspired from [23], which also measures the space consumption and temporal trend of OP_RETURN transactions.

Recently, some authors have started to analyse the security of Ethereum smart contracts: among these, [18] surveys vulnerabilities and attacks, while [37] and [25] propose analysis techniques to detect them. Our study on design patterns for Ethereum smart contracts could help to improve these techniques, by targeting contracts with specific programming patterns.

Acknowledgments. This work is partially supported by Aut. Reg. of Sardinia project P.I.A. 2013 "NOMAD".

References

1. Bitcoin contract, https://en.bitcoin.it/wiki/Contract. Last accessed 2017/01/14
2. Bitcoin OP_RETURN wiki page, https://en.bitcoin.it/wiki/OP_RETURN. Last accessed 2017/01/14
3. Dgx website, https://www.dgx.io/. Last accessed 2017/01/14
4. Ethereum hard fork 20 july 2016, https://blog.ethereum.org/2016/07/20/hard-fork-completed/. Last accessed 2017/01/14
5. Ethereum request for comment 20, https://github.com/ethereum/wiki/wiki/Standardized_Contract_APIs. Last accessed 2017/01/14
6. Lisk, https://lisk.io/. Last accessed 2017/01/14
7. Making sense of blockchain smart contracts, http://www.coindesk.com/making-sense-smart-contracts/. Last accessed 2017/01/14
8. Monax, https://monax.io/. Last accessed 2017/01/14
9. Smart contracts: The good, the bad and the lazy, http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/. Last accessed 2017/01/14
10. Stellar, https://www.stellar.org/. Last accessed 2017/01/14
11. The Stellar consensus protocol, https://www.stellar.org/papers/stellar-consensus-protocol.pdf. Last accessed 2017/01/14
12. Thinking about smart contract security, https://blog.ethereum.org/2016/06/19/thinking-smart-contract- Last accessed 2017/01/14
13. Understanding the DAO attack, http://www.coindesk.com/understanding-dao-hack-journalists/. Last accessed 2017/01/14
14. Another bug in the ens, you can win with an unlimited high bid without paying for it (2017), https://www.reddit.com/r/ethereum/comments/5zctus/another_bug_in_the_ens_you_can_win_with_an/. Last accessed 2017/03/17
15. Anderson, L., Holz, R., Ponomarev, A., Rimba, P., Weber, I.: New kids on the block: an analysis of modern blockchains. CoRR abs/1606.06530 (2016)
16. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. In: IEEE S & P. pp. 443–458 (2014)
17. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. Commun. ACM 59(4), 76–84 (2016), http://doi.acm.org/10.1145/2896386
18. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts. Cryptology ePrint Archive, Report 2016/1007 (2016), http://eprint.iacr.org/2016/1007
19. Back, A., Bentov, I.: Note on fair coin toss via Bitcoin. http://www.cs.technion.ac.il/~idddo/cointossBitcoin.pdf (2013)
20. Banasik, W., Dziembowski, S., Malinowski, D.: Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In: ESORICS. pp. 261–280 (2016)
21. Baqer, K., Huang, D.Y., McCoy, D., Weaver, N.: Stressing out: Bitcoin "stress testing". In: Bitcoin Workshop. pp. 3–18 (2016)

22. Bartoletti, M., Carta, S., Cimoli, T., Saia, R.: Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact. CoRR abs/1703.03779 (2017), https://arxiv.org/abs/1703.03779
23. Bartoletti, M., Pompianu, L.: An analysis of Bitcoin OP_RETURN metadata. CoRR abs/1702.01024 (2016), https://arxiv.org/abs/1702.01024, to appear in Bitcoin Workshop 2017
24. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: CRYPTO. pp. 421–439 (2014)
25. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., Zanella-Beguelin, S.: Formal verification of smart contracts. In: PLAS (2016)
26. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In: IEEE S & P. pp. 104–121 (2015)
27. Brown, R.G., Carlyle, J., Grigg, I., Hearn, M.: Corda: An introduction. http://r3cev.com/s/corda-introductory-whitepaper-final.pdf (2016)
28. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper (2013)
29. Churyumov, A.: Byteball: a decentralized system for transfer of value. https://byteball.org/Byteball.pdf (2016)
30. Clack, C.D., Bakshi, V.A., Braine, L.: Smart contract templates: foundations, design landscape and research directions. CoRR abs/1608.00771 (2016)
31. Delmolino, K., Arnett, M., Miller, A., Kosba, A., Shi, E.: Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In: Bitcoin Workshop (2016)
32. Dermody, R., Krellenstein, A., Slama, O., Wagner, E.: Counterparty: Protocol specification (2014), http://counterparty.io/docs/protocol_specification/. Last accessed 2017/01/14
33. Frantz, C.K., Nowostawski, M.: From institutions to code: towards automated generation of smart contracts. In: Workshop on Engineering Collective Adaptive Systems (eCAS) (2016)
34. Glaser, F., Zimmermann, K., Haferkorn, M., Weber, M.C.: Bitcoin - asset or currency? revealing users' hidden intentions. In: European Conference on Information Systems (ECIS) (2014)
35. Grau, P.: Lessons learned from making a chess game for Ethereum (2016), https://medium.com/@graycoding/lessons-learned-from-making-a-chess-game-for-ethereum-6917c0117
Last accessed 2017/01/14
36. Kumaresan, R., Moran, T., Bentov, I.: How to use Bitcoin to play decentralized poker. In: ACM CCS. pp. 195–206 (2015)
37. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: ACM CCS (2016), http://eprint.iacr.org/2016/633
38. Marino, B., Juels, A.: Setting standards for altering and undoing smart contracts. In: RuleML. pp. 151–166 (2016)
39. Möser, M., Böhme, R.: Trends, tips, tolls: A longitudinal study of bitcoin transaction fees. In: Financial Cryptography and Data Security. pp. 19–33 (2015)
40. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008)
41. Nomura Research Institute: Survey on blockchain technologies and related services, http://www.meti.go.jp/english/press/2016/pdf/0531_01f.pdf
42. Popejoy, S.: The Pact smart contract language. http://kadena.io/pact (2016)

43. Reid, F., Harrigan, M.: An analysis of anonymity in the Bitcoin system. In: Security and privacy in social networks, pp. 197–223. Springer (2013)
44. Ron, D., Shamir, A.: Quantitative analysis of the full Bitcoin transaction graph. In: Financial Cryptography and Data Security. pp. 6–24. Springer (2013)
45. Seijas, P.L., Thompson, S., McAdams, D.: Scripting smart contracts for distributed ledger technology. Cryptology ePrint Archive, Report 2016/1156 (2016), http://eprint.iacr.org/2016/1156
46. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in bitcoin. In: Financial Cryptography and Data Security. pp. 507–527 (2015)
47. Szabo, N.: Formalizing and securing relationships on public networks. First Monday 2(9) (1997), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/548
48. UK Government Chief Scientific Adviser: Distributed ledger technology: beyond block chain, https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/492972/gs-16-1-distr
49. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. gavwood.com/paper.pdf (2014)