Technology and Regulation

Smart Contracts:
Tales of Trust and Certainty

Eliza Mik*

smart contracts, digital contracts, crypto-regulation, blockchain, Ethereum, smart contract regulation

eliza@elizamik.io

Given the continuing fascination with "magic computers" and "self-executing code," it is necessary to revisit the promises – and premises - of technology-driven improvements to transacting practices purportedly introduced by smart contracts. Contrary to the popular narrative, smart contracts do not eliminate the need for trust and are technically incapable of guaranteeing performance. The fascination with clear and unbreakable rules that are executed by code obfuscates the fact that such rules may be suboptimal and may incorrectly represent what was agreed. It also obscures the fact that it is impossible to write perfect code. Being in plain view and impossible to modify, changes nothing in this regard. Trust and certainty do not magically emerge from immutability or transparency. Legal analyses of smart contracts must be based on facts, not on fairy tales.

1. Introduction

Tales of smart contracts feature evil bankers, corrupt judges, over-priced lawyers, and opportunistic contracting parties – none of them can be trusted to do what they are supposed or have promised to do. If one is to believe the popular smart contract narrative, the only solution to the resulting problems lies in code. In the world of smart contracts, trust and certainty are based on computation, not on the traditional protections offered by the legal system or on the reputation of the counterparty. Secure, infallible, and unbiased code replaces unpredictable and biased humans. Smart contracts are set to "revolutionize private ordering"[1] by providing certainty[2] and "absolute confidence in possible outcomes."[3] In the world of smart contracts, performance is guaranteed by code[4] and legal enforcement is enhanced by - if not altogether replaced with - technological enforcement.[5] Unlike humans, code cannot change its mind, refuse to act, or deviate from the agreed terms.[6] The purported superiority of smart contracts rests on the assumption that, once set in motion, they will execute exactly as written – without the possibility of subsequent modification or interference – and ensure absolute adherence to the rules embodied in them. Smart contracts reflect a quest for certainty – and in an uncertain world plagued by financial crises, wars, and pandemics nothing seems more certain than code.[7]

Despite their purported importance, the technological characteristics of smart contracts remain largely misunderstood, at least in legal scholarship. It is, after all, easier to believe in tales of techno-solutionism than to analyze technical facts. While smart contracts are generally associated with the broader ideologically-driven narrative surrounding blockchains and cryptocurrencies, it must be remembered that they predate blockchains.[8] It must also be remembered that the popular claims made in relation to blockchains do not necessarily hold true for smart contracts. Arguably, *some* smart contract scholarship may have progressed too quickly, with their transformative potential and their ability to eliminate the need for trust being

1 Mark Verstraete, 'The Stakes of Smart Contracts' (2019) 50 *Loyola U Chic L J* 743.

2 Jan Kalbantner and others, 'A DLT-based Smart Contract Architecture for Atomic and Scalable Trading' (2021) https://arxiv.org/abs/2105.02937 accessed 1 October 2022; see generally: Michael J Casey and Peter Vigna, *The Truth Machine: The Blockchain and the Future of Everything* (St Martin's Press 2018).

3 Gavin Wood, 'Ethereum: A Secure Decentralised Generalised Transaction Ledger' (2015) 1, http://gavwood.com/paper.pdf accessed 1 October 2022.

4 Sara Green, 'Smart Contracts, Interpretation and Rectification' (2018) 24 LMCLQ 234, 236; Kevin Werbach and Nicholas Cornell, 'Contracts Ex Machina' (2017) 67 *Duke L J* 313, 352

5 Oliver R Goodenough, 'Integrating Smart Contracts with the Legacy Legal System: A US perspective' in Benedetta Cappiello and Gherardo Carullo (eds), *Blockchain, Law and Governance* (2021 Springer) 191, 193, 194; Helen Eenmaa-Dimitrieva and Maria Jose Schmidt-Kessen, 'Creating Markets in No-Trust Environments: The Law and Economics of Smart Contracts' (2019) 35 *CLSR* 69.

6 Law Commission, *Smart Legal Contracts Advice to Government* (Law Com 401, 2021) 12.

7 Simon Dingle, *In Math We Trust: Bitcoin, Crypto Currency and the Journey to Being Your Own Bank* (Tracey McDonald Publishers 2018); Kevin Werbach, 'Trust, But Verify: Why the Blockchain Needs the Law' (2018) 33 *Berkeley Tech L J* 489; Eenmaa-Dimitrieva and Schmidt-Kessen (n 5); Primavera De Filippi, Morshed Mannan and Wessel Reijers, 'Blockchain as a Confidence Machine: The Problem of Trust & Challenges of Governance' (2020) 62 *Tech in Soc* 101284.

8 Nick Szabo, 'Formalizing and Securing Relationships on Public Networks' (1996) 2 *First Monday*.

* Eliza Mik is assistant professor at the Faculty of Law at the Chinese University of Hong Kong.

Received 31 Mar 2022, Accepted 12 Oct 2022, Published: 27 Oct 2022.

taken for granted. It is thus necessary to pause, take a step back and re-examine the popular claims surrounding this phenomenon. This paper can thus be regarded as an overdue, critical evaluation of their technical attributes. It aims to put legal analyses of smart contracts, including any regulatory efforts in this area,[9] on a firmer factual footing. The legal response to smart contracts, if any, should depend on their capabilities and on the risks they may present[10] - not on abstract, unverified claims. Although the technologies are complex, the points made here are simple: smart contracts cannot eliminate the need for trust and cannot guarantee commercial outcomes. Irrespective of whether they are treated as contracts in the legal sense, as technologies that facilitate contracting or as the back-end of distributed applications, smart contracts may increase, rather than alleviate, certain transactional risks.

The idea of technologically enhancing the enforcement of legal rules is, of course, not new. It is reflected in Lessig's "code is law"[11] approach, as well as in the concept of normative technologies.[12] In the context of smart contracts, code ensures adherence to certain rules, such as those deriving from contractual obligations, by automatically executing them on blockchains – not by forcing or nudging humans to obey them. While "code is law" and normative technologies are predominantly discussed in the context of public law, smart contracts seem to dominate in the area of private agreement.

1.1 Roadmap

After introducing various descriptions of smart contracts and commenting on the scant definitions of trust in the blockchain narrative, the discussion commences with some important technical distinctions. To evaluate the common claims made with regards to smart contracts, it is necessary to understand their technical attributes, particularly their relationship with blockchains. It is also necessary to appreciate the different types and roles of blockchains. The latter can function as databases and/or as computational platforms. To this end, this paper focuses on the Ethereum blockchain, which was specifically designed to enable smart contracts,[13] and compares it with the original Bitcoin blockchain, which was designed as an alternative payment system.[14] Taking Bitcoin and Ethereum as examples, this paper focuses on permissionless rather than permissioned blockchains as the latter are generally governed by one or more organizations and cannot be considered as decentralized or trustless.[15] Given that much of the existing confusion seems to derive from the conflation of certain features of smart contracts with certain features of blockchains, it is necessary to draw a clear distinction between these two. The fact that smart contracts execute on blockchains does not mean that they are *like* blockchains. The paper proceeds to describe the difficulties of ensuring the technical correctness of code. Next, as the elimination of trust and the promise of certainty seem to derive from a combination of transparency and immutability, these two attributes are investigated in more detail. The paper proceeds to examine oracles, entities providing smart contracts with information about the fulfillment of payment conditions. Oracles also illustrate the difficulties of creating a system that tries to replace trust with technology alone.

Two caveats before proceeding. First, this paper does not join the debate whether smart contracts are contracts in the legal sense. Instead, it acknowledges the terminological uncertainty surrounding smart contracts and critically revisits the claims regarding their potential to revolutionize commercial interactions. Before debating the legal implications of smart contracts (including the questions whether they are contracts and whether they require regulation) it is necessary to understand their actual characteristics and capabilities.[16] Just because a technology is called a smart *contract* does not make it a contract.[17] Maybe smart contracts are contracts and maybe they are "only" computer programs that facilitate contracting.[18] In principle, there are no doctrinal obstacles for smart contracts to be contracts in the legal sense. [19] Their legal classification depends on the jurisdiction, whether a legal system permits the automation of the formation and/or performance of a contract and whether it allows for agreements to be expressed in code.

Second, some readers may object to the level of technical detail. Some detail is, however, indispensable to present the complexity of the issues involved - especially when it comes to difficulties of writing

9 For general descriptions of such efforts see: Thiebault Schrepel, 'Smart Contracts and the Digital Single Market Through the Lens of a "Law + Technology" Approach' (European Commission Report 2021); Julian Mouton, 'Regulating Smart Contracts in the Domain of Financial Trading' (2021) 57 *Cal West L Rev* 441; Agata Ferreira, 'Regulating smart contracts: Legal revolution or simply evolution?' (2021) 45 *Telecomm Policy* 102081, 9-14;; for specific regulatory attempts in the United States see, for example: Arizona Revised Statutes Title 44, para 44-7061(2017); Nevada State Bill 398 (2017); California Assembly Bill 2658 (2018)).

10 Andres Guadamuz, 'All Watched Over by Machines of Loving Grace: A Critical Look at Smart Contracts' (2019) 35 *CLSR* 1.

11 Lawrence Lessig, *Code version 2.0* (Basic Books 1999).

12 Mireille Hildebrandt, *Smart Technologies and the End(s) of Law, Novel Entanglements of Law and Technology* (Edward Elgar 2015); Roger Brownsword, *Rights, Regulation, and the Technological Revolution* (OUP 2008); Bibi van den Berg and Ronald E Leenes, 'Abort, Retry, Fail: Scoping Techno-Regulation and Other Techno-Effects' In M. Hildbrandt, & A. M. P. Gaakeer (Eds.), *Human law and computer law: Comparative perspectives* (Springer 2013).

13 At present, Ethereum remains the second most popular blockchain and the dominant smart contract platform, with the vast majority of legal and technical writings treating it as a point of reference when discussing smart contracts. Other public blockchains, such as Cardano, Hyperledger or Avalanche also offer smart contracts; see generally: Andreessen Horowitz, 'State of Crypto – An Overview Report' (2022) https://a16zcrypto.com/state-of-crypto-report-a16z-2022/ accessed 1 October 2022.

14 Satoshi Nakamoto, 'Bitcoin: A Peer-to-Peer Electronic Cash System' (2008) 1 https://bitcoin.org/bitcoin.pdf accessed 1 October 2022.

15 There are blockchains that do not fall into either categorization as they constitute a hybrid model. Sometimes, the term "permissioned" is used interchangeably with "private" and the term "permissionless" with "public" but these terms are not used consistently, see generally D J Yaga and others, 'Blockchain Technology Overview,' *National Institute of Standards and Technology Internal/Interagency Report 8202* (2018) 5; X Xu and others 'A Taxonomy of Blockchain-Based Systems for Architecture Design'in (2017) *IEEE International Conference on Software Architecture (ICSA)* 234.

16 Jake Goldenfein and Andrea Leiter, 'Legal Engineering on the Blockchain: Smart Contracts as Legal Conduct' (2018) 29 *Law & Critique* 141, 143; Guadamuz (n 10) 2: "Most of the scholarly and practical analysis so far has been taken the claims of this technology being akin to a contract at face value, with legal analysis of contract formation, performance, and enforcement at the forefront of the debate. ... [W]hile smart contracts may pose some interesting legal questions, most of these are irrelevant, and smart contracts should be understood almost strictly from a technical perspective, and that any legal response is entirely dependent on the technical capabilities of the smart contract."

17 Carla L Reyes, 'A Unified Theory of Code-Connected Contracts' (2021) 46 *J Corp L* 981, 998; Nataliia Filatova, 'Smart Contracts From the Contract Law Perspective: Outlining New Regulative Strategies' (2020) 28 IJILT 217, 225; Farshad Ghodoosi, 'Contracting in the Age of Smart Contracts' (2021) 96 *Wash L Rev* 51.

18 Reyes (n 17) 998.

19 For a detailed discussion see: Eliza Mik, 'Smart Contracts: Terminology, Technical Limitations and Real-World Complexity' (2017) 8 *LIT* 1; Andrea Stazi, *Smart Contracts and Comparative Law* (Springer 2021) particularly 84-87.

code as well as the actual implications of immutability and transparency. The "technicalities" are kept to a minimum, illustrating broader points, rather than claiming to be exhaustive. Further technical details can be found in the technical references.

1.2 Definitions & Descriptions

Smart contracts have multiple, inconsistent definitions. They have been described as "the execution of a digital contract,"[20] "systems which automatically move digital assets according to arbitrary pre-specified rules,"[21] "transactional scripts,"[22] "conditional instruments for transferring money,"[23] "low-level code stored and running on a blockchain"[24] or the embedding of legal terms in hardware and software to prevent breach.[25] The US National Institute of Standards and Technology defines a smart contract as a "collection of code and data (sometimes referred to as functions and state) that is deployed using cryptographically signed transactions on the blockchain network,"[26] or as "attestable application processes,"[27] whereas a popular cryptocurrency publication refers to them as "contracts expressed as a piece of code that are designed to carry out a set of instructions."[28] Despite such inconsistent descriptions, the claims (or *tales*?) concerning their transformative potential are surprisingly consistent: smart contracts provide certainty and obviate the need for trust. There are also consistent (or *persistent*?) references to two of their attributes: the immutability and visibility of their code. Paradoxically then, there is little agreement as to *what* smart contracts are but there is agreement as to their key characteristics and as to their potential to revolutionize various aspects of commerce, if not society in general.

When analyzing smart contracts, legal scholarship faces a challenge - not only due to the lack of a single, universally accepted definition but also due to the difficulty of selecting the best sources describing what smart contracts are and what they can do. On one hand, there is the broader blockchain narrative, with its ideological attachment to decentralization and "trustlessness," as well as sweeping claims concerning the irrefutable significance of smart contracts. On the other, there is technical scholarship, which contains not only complex descriptions of the relevant technologies but also ill-informed references to legal concepts – including the unfortunate term smart "contract." To date, legal scholarship has faced the challenge of "reconciling" ideology with technology – not to mention the challenge of establishing the factual veracity of the popular claims concerning smart contracts.[29] For present purposes, it is worth remembering that the exact meaning of the term "smart contract" is context-sensitive. Moreover, legal analyses will differ depending on which definition is adopted as a point of departure and whether one choses to rely on technical or on legal literature. Unfortunately, the latter often "recycles" a series of common statements without confirming their correctness. It is, of course, possible to analyze smart contracts from a legal perspective, just like it is possible to analyze computer programs from the perspective of intellectual property law, contract law or criminal law, amongst others. Such analyses may become unavoidable when, for example, smart contracts do not execute as intended and cause financial losses. The specific legal approach will depend on the context in which a given smart contract is used and on the financial losses or legal infringement involved. Comparing smart contracts to contracts or to contractual documents may, however, often be unnecessary. To execute a contract means to sign the final version of a formalized agreement. To execute a *smart* contract means to run its code on a blockchain.[30] It is also worth remembering that technical literature refers to smart contracts as being *created, deployed*, *called* or *used* rather than formed or performed.[31]

In practice, most smart contracts constitute mechanisms facilitating the transfer of cryptocurrencies[32] or the back-end of so-called distributed applications, that is, programs running on multiple computers within a network. [33] While the latter description ("back-end of distributed applications"!) is the least exciting, it is probably the most important. If smart contracts are synonymous with the back-end of distributed applications, then they are an indispensable component of Decentralized Finance (DeFi),[34] Non-fungible Tokens (NFTs),[35] Decentralized Autonomous Organization (DOAs)[36] or, in fact, any "traditional application" that is put on a blockchain in order to become decentralized. If such broad but purely technical definition of smart contracts is adopted, their actual and potential role becomes impossible to trivialize. At the same time, it renders it more difficult to analyze smart contracts "as if" they were contracts in the legal sense.

1.3 A Word on Trust

The blockchain narrative does not provide an unequivocal definition of trust.[37] This is unsurprising given that blockchains are supposed to eliminate the very need for trust or, to use the core term in the blockchain ideology, to be "trustless." If trust is eliminated, it need not be defined. The meaning of trust is context-dependent and not

20 Kalbantner (n 2).
21 Vitalik Buterin, 'Ethereum White Paper: A Next Generation Smart Contract and Decentralized Application Platform' (2015) github.com/ethereum/wiki/wiki/White-Paper accessed 1 October 2022.
22 Shaanan Cohney, David A Hoffman, 'Transactional Scripts in Contract Stacks' (2020) 105 *Minn L Rev* 319, 323.
23 Mouton (fn 9) 461.
24 Alessandro Brighente, Mauro Conti and Satish Kumar, 'Extortionware: Exploiting Smart Contracts Vulnerabilities for Fun and Profit' (2022) https://arxiv.org/abs/2203.09843 accessed 1 October 2022.
25 Szabo (n 8).
26 Yaga (n 15) 54.
27 Yaga (n 15) 3.
28 Matt Hussey and Daniel Phillips, 'What are Smart Contracts and how do they Work?' (*Decrypt*, April 28, 2022) https://decrypt.co/resources/smart-contracts accessed 1 October 2022.
29 For an overview of such claims, see Kelvin FK Low and Eliza Mik, 'Pause the Blockchain Legal Revolution' (2020) 69 *ICLQ* 135; Michèle Finck, *Blockchain: Regulation and Governance in Europe* (2018 CUP).
30 Technically, the code is not run "on the blockchain" but, in most instances, on the Ethereum Virtual Machine, for a detailed description of code execution in the context of Ethereum, see: Cohney and Hoffman (n 22) 340-344.
31 Examples of smart contracts given in technical literature are casinos, games, lotteries, distributed applications, DOAs, tokens and cryptocurrencies – none of which are contracts in the legal sense; see, for example: Simon Joseph Aquilina and others, 'EtherClue: Digital investigation of Attacks on Ethereum Smart Contracts' (2021) 2 *Blockchain: Research and Applications* 1; Zibin Zheng and others, 'An Overview on Smart Contracts: Challenges, Advances and Platforms' (2020) 105 *FGCS* 475.
32 A cryptocurrency is "a digital asset or unit within the system, which is cryptographically sent from one blockchain network user to another", Yaga (n 15) 49.
33 Andreas Antonopoulos and Gavin Wood, *Mastering Ethereum* (2nd edn, O'Reilly 2020) 269.
34 Fabian Schär, 'Decentralized Finance: On Blockchain- and Smart Contract-based Financial Markets' (2021) 103 *FRB St Louis Rev* 153.
35 Dan Chirtoaca, Joshua Ellul and George Azzopardi, 'A Framework for Creating Deployable Smart Contracts for Non-fungible Tokens on the Ethereum Blockchain' (2020) *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)* 100.
36 Vimal Dwivedi and others, 'Legally enforceable smart-contract languages: A systematic literature review' (2021) 54 *ACM CSUR* 1.
37 The original Bitcoin Whitepaper predominantly refers to trusted third parties but does not discuss trust per se, see: Nakamoto (n 14).

necessarily based on facts. [38] Trust often derives from beliefs - and blockchain enthusiasts *believe* that something is "trustless" if it is decentralized.[39] If no single entity controls a system, and if each node in the system is controlled by a different entity,[40] then such system obviates the need for trust in general. In the words of Antonopoulos: "No one actor is trusted, and no one needs to be trusted."[41] The underlying theories are inconsistent: blockchains (and, in some contexts, smart contracts) remove the need to trust a central entity, the other contracting party or a third party who acts as an intermediary. In essence, the blockchain narrative seems to regard the relationship between "decentralization" and "trust" as self-explanatory: when no single entity controls of the system, the system itself can be trusted. While this "explanation" may be disappointing, it must be remembered that blockchain narratives, unlike the technology itself, often lack precision and cohesiveness.[42] Many narratives can in fact be regarded as post-factum justifications for a particular type of system design, rather than fully conceptualized theoretical frameworks. To complicate matters, the term "decentralized" may often be difficult distinguish from "distributed." [43] The latter concerns the architecture of a system, the former concerns its governance or organization. At times, decentralization may also come dangerously close to disintermediation or to a general independence from humans and/or the legal system.[44] The original Bitcoin narrative is relatively restrained and aims at eliminating trust in *payment* intermediaries.[45] The smart contract narrative has, however, progressively expanded to encompass not just payment intermediaries but intermediaries in general – as well as the contracting parties themselves. This reflects the broader problem of pinpointing the exact entity that must normally be trusted and that should hence be replaced with code. After all, the other contracting party is neither an intermediary nor a third party and the vast majority of contracts do not require intermediaries or central authorities! Trusting a payment intermediary, who may assist in the transfer of money in the case of payment obligations, differs from trusting that a counterparty will perform. Similarly, the relationship between central authorities and intermediaries is difficult to comprehend, especially given that even lawyers are often portrayed as intermediaries. Instead of trying to reconcile these inconsistencies, it is best to accept their existence and adopt a broad, context-dependent understanding of "trust." It is also advisable to focus on the technology (purportedly) eliminating trust, not on the ideology proclaiming its elimination.

1.4 Technical Distinctions

To understand the capabilities of smart contracts, some technical distinctions are necessary. More specifically, the traditional role of blockchains as databases, or distributed ledgers, must be distinguished from their more recent use as computing platforms. At a basic level, databases store data and computing platforms execute code. Designed as an alternative payment system, the original Bitcoin blockchain provides mechanisms for the storage, generation, and transfer of its native cryptocurrency. The focus is on creating a secure, immutable record and simple, irreversible transfers between accounts represented as publicly visible addresses. Contrary to popular belief, the Bitcoin blockchain does not enable or support complex computations although each individual block is the product of extremely complex and expensive computations performed by the network of nodes running the consensus algorithm. In Bitcoin, the early equivalents of smart contracts take the form of cryptographic functions such as hash-locks, time-locks, and multi-signatures.[46] Such Bitcoin-based smart contracts are simple scripts that provide a set of pre-defined transfer conditions.[47] In response to the growing demand not just to hold but to transact in crypto, the Ethereum blockchain has been purposefully created to equip blockchains with customizable (that is: user-defined) business logic enabling a wider range of transactions.[48] What is important in the present context is that although Ethereum is a blockchain, it also provides a decentralized general-purpose computing infrastructure that executes smart contracts and is therefore often referred to as a "world computer"[49] or "a magic computer that anyone can upload programs to."[50] To complicate matters, smart contracts are not executed directly on Ethereum blockchain but on the Ethereum Virtual Machine ("EVM"), which provides a hardware- and operating system-agnostic abstraction of computation and storage. The EVM enables the execution of smart contracts written in different high-level programming languages,[51] but also requires their compilation into low-level bytecode, an important technical detail that has broader implications. In principle, Bitcoin can be regarded as a secure distributed database equipped with limited transacting functionality, while Ethereum can be regarded as a distributed database with unlimited transacting functionality – at least theoretically.[52] Ethereum

38 For a discussion of the different meanings of trust see: De Filippi, Mannan and Reijers (n 7), Caitlyn Lustig and Bonnie Nardi, 'Algorithmic Authority: the Case of Bitcoin' in S Misra and others (eds), *2015 48th Hawaii International Conference on System Sciences*, IEEE 243, 259; Kevin Werbach, *The Blockchain and the New Architecture of Trust* (MIT Press 2018) 17-25.

39 Antonopoulos and Wood (n 33) 10; Arvind Narayan and others, *Bitcoin and Cryptocurrency Technologies* (Princeton University Press 2016) 278-283.

40 Angela Walch, 'The Path of the Blockchain Lexicon (and the Law)' (2016) 36 *RBFL* 713, 720.

41 Andreas Anotonopoulos, 'Bitcoin Security Model: Trust in Code' (O'Reilly Radar, February 20, 2014) http://radar.oreilly.com/2014/02/bitcoin-security-model-trust-by-computation.html accessed 1 October 2022.

42 For descriptions see: Finck (n 29) 12, 183-185; Werbach (n 7) 509-512.

43 Angela Walch, 'Deconstructing Decentralization' in Chris Brummer (ed), *Cryptoassets: Legal, Regulatory, and Monetary Perspectives* (OUP 2019) 39.

44 Werbach (n 7) 509-512.

45 Nakamoto (n 14) 1, 2, 8 2.

46 Nicholas Kannengießer and others, 'Trade-offs between Distributed Ledger Technology Characteristics' (2020) 53 ACM Computing Surveys 2; while it could be argued that the scripts supported by the Bitcoin blockchain are not smart contracts, this remains an isolated view. Technical literature speaks of "simple" Bitcoin smart contracts, acknowledging their limited ability to express complex spending conditions; see generally: Malte Moser, Ittal Eyal, Emin Gün Sirer, 'Bitcoin Covenants' in: *International Conference on Financial Cryptography and Data Security*(Springer 2016) 216; Manuel Manuel T Chakravarty and others, 'The Extended UTXO Model' in: *Financial Cryptography and Data Security, Lecture Notes in Computer Science* (vol 12063 Springer 2020).

47 Buterin (n 21) 11; Werbach (n 7) 506, 507: "While Bitcoin operates based on smart contracts, it strictly limits their capabilities to basic fund transfers for security."

48 Ethereum has been originally described as "a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions" see: Buterin (n 21) 13

49 Antonopoulos and Wood, (n 33); Werbach (n 7) 55, 64, 66.

50 Vitalik Buterin, 'Visions, Part 1: The Value of Blockchain Technology' (2015) https://blog.ethereum.org/2015/04/13/visions-part-1-the-value-of-blockchain-technology/ accessed 1 October 2022.

51 Kannengießer (n 46) 8; the EVM also enables cross-interoperability with other blockchains, such as Avalanche.

52 The Bitcoin model relies on the so-called Unspent Transaction Output (UTXO) and is regarded as more secure and scalable but also less programmable. In contrast, the Ethereum model is based on so-called "accounts," addressable on-chain objects that contain a balance, a nonce, code and optional storage. Cardano introduced a smart contract approach that relies on the Extended Unspent Transaction Output (EUTXO) Model, which derives from Bitcoin's UTXO model. The EUTXO permits more

supports smart contracts that can encode more complex transfer conditions and render the relevant crypto-currencies into "programmable money" or protocols to exchange value.[53] The Bitcoin blockchain enables the encoding of some simple rules, the Ethereum blockchain – of any rule.[54] The following quote is informative: "smart contracts provide a way to define custom ways of moving Ether between accounts, enabling the development of sub-currencies (token), wallets, autonomous governance, and decentralized gambling/lottery applications, among others."[55] It demonstrates that smart contracts equip blockchains with additional functionality *and* that the debate whether they are contracts in the legal sense may be redundant. Neither wallets, nor lottery applications can, after all, be analyzed as contracts. What is important here is that blockchains can be used as databases *and/or* as computational platforms. Certain technical features, including transparency and immutability, may be indispensable when blockchains function as databases. To recall, the original purpose of making the blockchain's contents universally visible is to prevent the double spending of crypto-currencies. The inability to change such contents aims to ensure that transactions cannot be reversed. As demonstrated below, transparency and immutability are less important or even detrimental in the context of smart contracts, when blockchains function as computational platforms.

2. Trusting Code

It is useful to recall the basic fact that, notwithstanding the multitude of inconsistent definitions, smart contracts are computer programs and that computer programs are expressed in code. Consequently, any claims to the effect that smart contracts can replace the need to trust humans with the ability to trust their code,[56] can hold true if – and *only if* – the execution of such code guarantees the achievement of some agreed or expected result. This, however, requires that such code be perfect and predictable. Moreover, the common assumption is that by executing legal terms "on" the blockchain, smart contracts will "do" what was agreed. It is overlooked, however, that smart contracts will only do what they were coded to do.[57] As stated in the leading book on Ethereum: "[a] smart contract will execute exactly what is written, which is not always what the programmer intended"[58] or "what the agreeing parties thought it would do."[59] Executing "as coded" and "achieving what was agreed" or "intended" are different things. Blockchains do not discriminate between good code and bad code. If the smart contract contains errors or vulnerabilities, those will also be executed. To understand the difficulties involved in creating code that *could* guarantee the achievement of certain results and change the trust assumptions underlying existing transacting practices, it is important to distinguish the attributes of smart contracts from the attributes of blockchains – and to acknowledge the challenges of writing programs for distributed computing platforms. Throughout the discussion it must be remembered that if smart contracts are supposed to replace trust and ensure certainty, it seems crucial to be able to predict how they will execute *before* they are deployed.

2.1 Smart contracts vs Blockchains

Although smart contracts execute *on* blockchains, they are not *like* blockchains. The indiscriminate association of smart contracts with blockchains may underlie the misconceptions regarding their technical characteristics and hence their actual capabilities. It is, for example, commonly assumed that if blockchains are "trustless," then smart contracts are also "trustless." Technically, however, only blockchains can have this attribute.[60] "Trustlessness" denotes the ability to determine the truth of certain information without recourse to a trusted third party in an adversarial environment.[61] In the blockchain context, such information concerns the crypto-currencies stored therein or, more specifically, the state of the distributed database. Crypto-currencies are generated by the consensus algorithm as a reward provided to those who expand computational resources to append blocks and thus to maintain the blockchain.[62] Being a product of its underlying consensus algorithm, crypto-currencies can only exist "in" the blockchain. Consequently, the information *about* the cryptocurrencies is synonymous with the cryptocurrencies themselves – and it can be trusted to be correct. The blockchain is the only authoritative source of information about the crypto-currencies generated thereby and stored therein.[63] These "trust assumptions" cannot, however, be extrapolated to smart contracts. The latter are created by human developers and subsequently deployed on the blockchain. Smart contracts are *executed* by the nodes which run the consensus algorithm that generates blocks and the associated cryptocurrencies - but they are not *generated or validated* by such algorithm. The latter operates at a foundational layer, far below the abstraction of smart contracts.[64] Consensus algorithms are technically incapable of determining whether the code of a smart contract is correct, free of vulnerabilities, not to mention whether it correctly represents legal terms.[65] The "trustlessness" of certain information stored *in* the blockchain does not translate into the "trustlessness" of the code executing *on* the blockchain. Smart contracts are susceptible to programming errors like all other computer programs. Technically, they do not – and *cannot* - eliminate the need to trust someone. Multiple factors render them unsuitable for the encoding of legal terms or, more generally, for controlling any valuable resources.

2.2 Programming errors & vulnerabilities

Smart contracts are notorious for vulnerabilities and coding errors.[66] The latter are mistakes in the implementation of a system design that

expressive (speak: complex) smart contracts while providing security in a complex, distributed computing environment; see: Chakravarty (n 46).

53 Finck (n 29) 9,10.

54 Buterin (n 21) 19; it is worth observing that Rootstock (or "RSK") equips the Bitcoin blockchain with the ability to support the EVM and thus more complex smart contracts; see: Sergio Lerner, 'Rootstock Platform: Bitcoin Powered Smart Contracts', RSK Whitepaper, (revised Version, 29 January 2019) at: https://blog.rsk.co accessed 1 October 2022.

55 Aquilina (n 31).

56 Werbach (n 47) 514, 517.

57 With some exceptions pertaining to unexpected interactions with other smart contracts.

58 Antonopoulos and Wood (n 33) 171.

59 The Ethereum website states: "A smart contract is code that lives on the Ethereum blockchain and runs exactly as programmed. Once smart contracts are deployed on the network you can't change them. (...) This also means you need to design your contracts very carefully and test them thoroughly." https://ethereum.org/en/developers/docs/dapps/ accessed 1 October 2022.

60 This relates to permissionless blockchains!

61 Leslie Lampert and others, 'The Byzantine Generals Problem' (1982) 4 *ACM TOPLAS* 382.

62 For a detailed technical description see: Andreas Antonopoulos, *Mastering Bitcoin* (2nd edn, O'Reilly 2017) 26; for legal explanations see: Cohney and Hoffman (n 22) 339.

63 Low and Mik (n 29) 145.

64 Antonopoulos and Wood (n 33) 320.

65 Buterin (n 21) 18.

66 The Smart Contract Weakness Classification (SWC) Registry collects information about various vulnerabilities, https://swcregistry.io accessed 30 March 2022; Sam Werner et al., 'SoK: Decentralized Finance (DeFi)' (2021) https://arxiv.org/abs/2101.08778; Nicola Atzei, et al 'A Survey of At-

result in programs not providing the intended functionality or producing unexpected output. The former enable the malicious misuse of a system implementation to achieve effects contrary to its goals (so-called "exploits"). Not all vulnerabilities derive from programming errors and not all programming errors can be exploited. Some programming errors resemble those found in traditional programs (e.g. integer overflows), others are specific to smart contracts. [67] For example, "greedy contracts" can lock Ether indefinitely and "gasless send bugs" can prevent the execution of smart contracts. As any public function (that is: a task-specific code module) of a smart contract can be called (that is: activated) by another contract, malicious users can attack vulnerable functions with relative ease. This so-called "reentrancy bug" enables an attacker to call a contract's function multiple times before the initial call is terminated. If the contract's internal state is not securely updated, cryptocurrencies can be drained from the contract by recursively calling the function.[68] Reentrancy attacks, which are often regarded as a proxy for smart contract robustness, have gained notoriety since the 2016 DAO hack, which caused a loss of over 60 million US dollars. [69] Attacks on, or hacks of, smart contracts resulting in financial losses are a common occurrence.[70] If, however, smart contracts not only control high-value assets but are also supposed to lay the foundations for a new, decentralized economy, then it seems crucial that their code be reliable.[71] Admittedly, while their significance cannot be measured by the price of the cryptocurrencies they control, such price highlights the risks involved if a smart contract does not execute as expected or is otherwise exploited. Programming errors and vulnerabilities translate into monetary losses:

> "Smart contract code is unforgiving. Every bug can lead to monetary loss. You should not treat smart contract programming the same way as general-purpose programming. [...] You should apply rigorous engineering and software development methodologies, as you would in aerospace engineering or any similarly unforgiving discipline. Once you 'launch' your code, there's little you can do to fix any problems."[72]

While not every vulnerability will be exploited and result in financial loss,[73] it can be assumed that smart contracts controlling crypto assets with a high market value are more likely to be attacked. Before extolling their role in replacing traditional payment intermediaries, such as banks,[74] it must be remembered that software used in financial transactions is rigorously tested before deployment to ensure its robustness.[75] Moreover, financial transactions occur in a highly regulated environment, with clear rules of risk allocation and the ability to seek recourse against an identifiable entity. There is, however, no assurance of the quality of smart contracts and there may be no-one to be held liable if their execution results in losses – at least if one follows the dominant blockchain narrative, which emphasizes decentralization and disintermediation.

2.3 A new execution environment

It is rarely appreciated that many problems with smart contracts derive from the blockchain itself: the decentralized and distributed network of nodes where their code run.[76] As indicated, decentralization means that no single entity controls the system. It also means that no-one is responsible for its operations or liable for its malfunctions. The fascination with decentralisation obfuscates the fact that the individual nodes in permissionless blockchains are controlled by unknown parties, which are *by definition* considered potentially hostile.[77] It is true that blockchains provide a reliable execution environment in terms of availability and resistance to external interference. This means that, once deployed, smart contracts cannot be stopped or tampered with. It does not mean, however, that each smart contract will execute as planned or that its execution can be predicted. Given that permissionless blockchains, such as Ethereum, are open to all, anyone – including malicious and/or incompetent coders - can create a smart contract that will interact with other smart contracts.[78] Consequently, while smart contracts are immune from external interference, they are not immune from unexpected interactions with other smart contracts and from the inadvertent execution of untrusted code.[79] Perfect code (assuming, for the sake of argument, that such can exist) may produce unexpected results when it encounters imperfect or malicious code. Moreover, arbitrary smart contract invocations may progressively increase the number of interconnected contracts

tacks on Ethereum Smart Contracts (SoK)' in: M Maffei and M Ryan (eds), *Proceedings of the 6th International Conference on Principles of Security and Trust* (Springer 2017) 10; Ivica Nikolic et al., 'Finding the Greedy, Prodigal, and Suicidal Contracts at Scale' (2018) https://arxiv.org/pdf/1802.06038.pdf accessed 30 March 2022; for an excellent explanation from a legal perspective see: Cohney and Hoffman (n 22) 328.

67 Aquilina (n 31).

68 Oliver Lutz, et al., 'ESCORT: Ethereum Smart Contracts Vulnerability Detection using Deep Neural Network and Transfer Learning' (2021) https://arxiv.org/abs/2103.12607 accessed 1 October 2022.

69 Muhammad Izhar Mehar et al., 'Understanding a revolutionary and flawed grand experiment in blockchain: the DAO Attack' (2019) 21 *JCIT* 19; the DAO was a collection of smart contracts implementing an automated crowd-funding mechanism. An attacker exploited a bug in the withdraw functionality, effectively stealing the invested funds. For a more detailed description of the DOA hack and its immediate "successors;" see also Cohney and Hoffman (n 22) 320.

70 For example, in March 2021, Binance Smart Chain-based lending protocol Meerkat Finance lost $31 million in user funds when an attacker called a function in the contract that made their address become the vault owner, draining over $17.4 million worth of stable coins and tokens; in August 2021 a hack of the Poly Network exposed security flaws in smart contracts in three blockchains: BSC, Polygon, and Ethereum. The hacker aimed to showcase the risks of running unverified contracts. In April 2022, an exploit in the smart contract underpinning the algorithmic stable coin, Beanstalk, resulted a loss of over $120 million; for an overview of the most prominent smart contract hacks see: https://decrypt.co/93874/biggest-defi-hacks-heists accessed 1 October 2022; for more examples of financial losses attributable to vulnerable smart contracts see: Chainalysis, 'The 2022 Crypto Crime Report' (February 2022) 75, https://go.chainalysis.com/rs/503-FAP-074/images/Crypto-Crime-Report-2022.pdf accessed 1 October 2022.

71 The "reliability" of software generally denotes the probability of a failure-free operation for a specified period of time in a specified environment; see: Michael R Lyu, *Handbook of Software Reliability Engineering* (Vol. 222 IEEE Computer Society Press 1996) 5-7.

72 Antonopoulos and Wood (n 33) 172.

73 see: Daniel Perez, Benjamin Livshits, 'Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited' (2021) *USENIX Security Symposium*.

74 Mouton (n 9) 461-463.

75 See generally: the 'Principles for Financial Services Infrastructure'Bank for International Settlements and International Organization of Securities Commissions (2012), www.bis.org/cpmi/info_pfmi.htm accessed 1 October 2022.

76 Loi Luu and others, 'Making smart contracts smarter,' in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communication Security* (ACM 2016) 254.

77 Clara Schneidewind, Markus Scherer and Matteo Maffei, 'The Good, the Bad and the Ugly: Pitfalls and Best Practices in Automated Sound Static Analysis of Ethereum Smart Contracts,' in *Applications: 9th International Symposium on Leveraging Applications of Formal Methods* (ACM 2020) 9.

78 Antonopoulos and Wood (n 33) 166, 171.

79 Kannengießer (n 46) 6, 7.

making it impossible to anticipate how a specific smart contract will execute. It is hence unclear how smart contracts could provide "certainty of outcomes" if their actual execution cannot be predicted, not to mention guaranteed.[80]

2.4 New programming languages

Leaving aside blockchains, the difficulty of ensuring that smart contracts execute as intended derives from the shortcomings of the programming languages in which they are written.[81] The academic debates surrounding such languages often focus on their ability to reflect the semantic richness of legal prose and/or the complexity of the underlying transaction.[82] The technical reality is, however, more mundane. Issues pertaining to expressiveness, seem secondary to the security and predictability of smart contracts written in a particular programming language, not to mention its suitability for a distributed execution environment. To demonstrate the associated trade-offs, it is useful to start with Bitcoin's scripting language, which is secure but also limited to simple evaluations of spending conditions. It is also non-Turing complete, which means that it cannot simulate any general-purpose computer or computer language.[83] The Bitcoin blockchain is, after all, first and foremost a secure, distributed database – not a computing platform. In contrast, Ethereum's smart contract language (Solidity) is Turing-complete. Unfortunately, while Turing-complete languages are more expressive and enable more sophisticated smart contracts, they also increase the likelihood of vulnerabilities and coding errors.[84] They also render it more difficult to ensure that smart contracts execute as intended or, more generally, to anticipate their operation. Turing-completeness comes at the expense of security and predictability. To aggravate matters, while many academics ponder the optimal programming approach for smart contracts,[85] blockchain start-ups profess the "move-fast-and-break-things" ideology and often create smart contracts in novel, often hastily designed high-level programming languages.[86] The latter are created to be (relatively) easy to learn to encourage even less experienced developers to create applications (aka: smart contracts!) for specific blockchains. The popularity of many blockchains - and hence the price of their native cryptocurrencies - generally depends on the number of applications they support and on the size of their developer community.[87] Consequently, promoting a particular smart contract programming language is an indirect way of promoting a particular blockchain. At present, many smart contract programming languages "borrow" features from traditional programming languages, which are often unsuitable for distributed execution environments in terms of resource consumption and security. The former concerns the need to limit the computations required to execute an operation, the latter concerns the fact that many smart contracts control high-value crypto-assets and are therefore susceptible to attacks. The combination of "half-baked" programming languages with inexperienced programmers, has resulted in smart contracts being notoriously insecure.[88] The attentive reader might inquire why smart contracts cannot be written in any of the existing high-level programming languages. Some blockchains do in fact, adopt this approach.[89] In principle, however, the problem lies in adapting such language to the constraints of the execution environment. Such adaptation would require the compilation of the source code written in such "traditional" high-level programming language to low-level bytecode, which could be executed by the EVM. This would, in turn, require the creation of reliable compilers. Creating new compilers for various high-level programming languages is, however, a complex and time-consuming task. Moreover, being programs themselves, compilers introduce the risk "mistranslating" the code from a high-level to a low-level programming language.[90]

2.5 Trusting Coders

The blockchain narrative associates "trustlessness" with decentralization and disintermediation – the elimination of the need to trust a single entity. While the individual blocks in a blockchain are *generated* by a decentralized network of independent nodes, smart contracts are *created* by human coders who make them available for general use. In other words: those very same biased, opportunistic, and unpredictable humans are creating the code that is supposed to eliminate the need to trust biased, opportunistic, and unpredictable humans. The decentralized nature of the consensus algorithm underlying the blockchain changes nothing in this regard. Abstracting from questions of technical competence, there is no reason to assume that coders are inherently more trustworthy than bankers, judges, lawyers, or the contracting parties. This seems particularly important given that, in practice, the process of creating smart contracts may be centralized, as would be the case when a specific smart contract is written by a single coder. What is even more important is that smart contracts are used by parties who have not written them and lack the technical skills to evaluate their technical correctness. As smart contracts are written by coders but relied on, or used, by non-coders, those who use them must trust those who

80 Cardano's EUTXO model is deterministic, making it is easier to ensure that the smart contract will be execute as intended because its execution can be simulated before deployment - both in terms of the execution outcome and the amount of resources consumed thereby, see: Chakravarty (n 46) 9.

81 We are dealing with a more complex programming paradigm – the program will be executed in many dispersed locations, not on a single, centralized server. Some smart contracts are written in traditional programming languages, but this seems to be the exception given Ethereum's dominance of the field and ongoing efforts to create domain-specific programming languages, or DSL, that are tailored to the needs of blockchains; see: Reza Parizi and others, 'Smart Contract Programming Languages on Blockchains: An empirical Evaluation of Usability and Security' in S Chen, H Wang and LJ Zhang (eds), *Blockchain – ICBC 2018 Lecture Notes in Computer Science* (Springer 2018).

82 Guido Governatori and others, 'On Legal Contracts, Imperative and Declarative Smart Contracts, and Blockchain Systems' (2018) 26 *Artif Intell L* 377; Firas Al Khalil and others, 'Trust in Smart Contracts is a Process as Well' in Michael Brenner and others (eds), *Financial Cryptography and Data Security* (Springer 2017) 4; William Brown, 'Limitations of Code in Contracts: What we can learn from the Plain English Movement' (2019) 9 *VULJ* 57.

83 Buterin (n 21) 28; in simpler terms, Turing-completeness means that any application that runs on a conventional computer can be executed on the blockchain.

84 Massimo Bartoletti, et al, Verification of Recursive Bitcoin Contracts (2020) https://arxiv.org/abs/2011.14165 accessed 1 October 2022.

85 Governatori (n 82) 378.

86 see generally: Antonopoulos and Wood (n 33) 128, 129.

87 The popularity, or commercial significance of a smart contract platform is often evaluated based on the highest monthly average of commits, the number of monthly active developers or by the market value of the assets controlled thereby. Newer blockchains, such as Solana, are likely to boast more monthly commits and a growing influx of new developers; see: 'Blockchain Development Trends,'2022 Edition,' Outlier Ventures (January 2021 – December 2021).

88 Grigore Rosu, 'Formal Design, Implementation and Verification of Blockchain Languages' (2018) *3rd International Conference on Formal Structures for Computation and Deduction*.

89 See, for example, Cardano's programming language Plutus, which builds on Haskell (a functional programming language that ensures greater predictability of the code written therein) and enables the re-use of Haskell libraries.

90 Torben E Mogensen, *Introduction to Compiler Design* (2nd ed, Springer 2017) Cohney and Hoffman (n 22) 331.

created them.[91] In his acceptance speech for the Turing award, Ken Thompson famously stated: "You can't trust code that you did not totally create yourself."[92] The problem is that in public blockchains, such as Ethereum, anybody is allowed to create a smart contract and make it available for others to use. While this approach exemplifies the spirit of decentralization, it does not guarantee that smart contract creators have the necessary expertise and are not malicious. As indicated, blockchains do not provide any form of quality assurance of the smart contracts executed thereon. A competent but dishonest coder may deliberately create a smart contract containing a vulnerability in order to exploit it once the smart contract controls more crypto-currencies. An incompetent but well-meaning coder may create such vulnerability unintentionally and render the smart contract susceptible to exploits by more technologically competent coders. Irrespective of whether the error or vulnerability is intentional or not, hapless users will face the risk of financial losses.
To clarify, smart contract developers must not be conflated with so-called blockchain core developers, this is, programmers with 'commit access' to the source code of a given blockchain that allows them to make changes to the underlying consensus algorithms. The latter constitute a small group of elite programmers, with established reputations and a track record of reliable performance.[93] In contrast, many smart contracts developers are anonymous, inexperienced novices.[94]

In sum, code is always written by humans. If such humans are dishonest or incompetent, then the code they create cannot be trusted – not to mention eliminate the need for trust. Moreover, if such humans cannot be identified, then there is no one to be held liable for any losses caused by "malfunctioning" smart contracts.

2.6 Correctness of Expression

The previous paragraphs addressed the challenges of ensuring the technical correctness of code. A separate question is whether a particular smart contract accurately reflects the agreement. If, following the dominant narrative, smart contracts embody and/or execute legal terms, then these terms must be expressed in code exactly as agreed. Any discrepancy between the code and the terms would defeat the very purpose of smart contracts. Two problems arise.

The first concerns the expression of legal terms in any of the available programming languages. The difficulties of expressing contractual terms in code are usually underplayed, the simplistic assumption being that such terms can be *translated* into code. Legal terms are, however, written in natural languages - and natural languages are expressive but imprecise. The meaning of each word depends on its context, there being no standardized mapping from symbols to objects or a formalized grammar.[95] In contrast, programming languages have a formalized syntax, strict rules defining the semantics of a program and a closed set of permitted inputs. [96] They are designed to eliminate ambiguity.[97] As no programming language is as expressive as a natural language[98] and no natural language has the semantic and syntactic rigor to instruct computers, perfect *translations* are technically impossible.[99] Given that smart contracts aim to provide certainty by guaranteeing performance, the programming approach should focus on achieving the agreed results, not on ensuring the semantic and syntactic equivalence between the terms and the code. It seems unnecessary to "translate" the entire legal prose of a contract rather than encoding its operational provisions, such as payment or delivery obligations. The focus should be on actions and results, rather than on words or provisions. Arguably, in some contexts technical literature may be using the words "legal terms" interchangeably with "rules" in general. Still, the focus on expressing legal terms, or legal prose, in code may be misplaced given the inherent difficulty of mapping natural languages onto formal languages.

The second problem concerns the permitted amount of so-called "on-chain computation," that is: the number of computational steps that can be executed in a distributed computing environment represented by a blockchain. In principle, to prevent overloading Ethereum, computationally intensive (speak: complex) smart contracts are also more expensive: the computations are constrained by the price of so-called gas (a sub-unit of Ether, Ethereum's native crypto-currency) and by the amount of gas available in any given block. [100] Ether is used to meter and restrict the use of computing resources. In contrast to Bitcoin, its use as a payment mechanism is secondary. Given these technical limitations, smart contracts must be kept simple and, contrary to popular assumptions, cannot execute more sophisticated transactions or embody more elaborate legal terms. The creation of an expressive, Turing-complete programming language to enable complex smart contracts may hence be pointless if they cannot be executed due to the limits of on-chain computation. At the same time, moving intensive computations off-chain, to systems existing outside of the blockchain, contradicts the very purpose of smart contracts – ensuring that no-one can interfere with their execution. In sum, concerns regarding the ability to express legal terms in code ignore the inherent computational limitations of blockchains.

3. Transparency

The blockchain narrative associates transparency with the universal visibility of the contents of blockchains. In the context of smart contracts, trust and certainty are supposed to derive from the ability to view their code: "smart contracts are publicly available for anyone to audit or scrutinize."[101] Admittedly, transparency provides the ability to discover *obvious* errors or patently malicious designs. It also "enforces correct behavior through embarrassment."[102] Coders

91 De Filippi, Mannan and Reijers (n 7) ; Cohney and Hoffman (n 22) 328; Guadamuz (n 10) 14.
92 Ken Thompson, 'Reflections on Trusting Trust,' Turing Award Lecture' (1984) 27 *Communications of the ACM* 760, 763.
93 Angela Walch, 'In Code(rs) we Trust: Software Developers as Fiduciaries in Public Blockchains' in Philip Hacker and others (eds), *Regulating Blockchain: Techno-Social and Legal Challenges* (OUP 2019) 58; Gili Vidan and Vili Lehdonvirta, 'Mine the gap: Bitcoin and the maintenance of trustlessness' (2018) *New Media & Soc* 42.
94 Gabriel de Sousa Matsumura and others, 'Vulnerabilities and Open Issues of Smart Contracts: A Systematic Mapping' in: *Computational Science and Its Applications* (Springer 2021).
95 Stuart J Russell and Paul Norvig, *Artificial Intelligence: A Modern Approach* (4th edn, Pearson 2021) 270, 874, 875.
96 Russell and Norvig (n 95) 269-272.
97 Casey Casalnuovo, Kenji Sagae and Prem Devanbu, 'Studying the difference between natural and programming language corpora' (2019) 24 *Emp Softw Eng J* 1823.
98 To clarify, even a Turing Complete programming language will never be as expressive as a natural language 4.
99 For a more detailed explanation, see: Eliza Mik, 'Contracts in Code?' (2021) 13 *LIT* 1.
100 Buterin (n 21) 29; Cohney and Hoffman (n 22) 339.
101 Nic Carter and Linda Jeng, 'DeFi Protocol Risks: the Paradox of DeFi' (2021) https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3866699 accessed 1 October 2022; Adam J Kolber, 'Not-So-Smart Blochchain Contracts and Artificial Responsibility' (2018) *Stan Tech L Rev* 198, 208; Cohney and Hoffman (n 22) 322; Schär (n 34) 169; Nakamoto (n 14) 2.
102 This expression has been suggested by my colleague Stanley Yong.

who publicly deploy their smart contracts would want to ensure that their code is well written and appears professional. This assumption, however, works only for coders concerned about their reputation and is unlikely to discourage dishonest – and anonymous - coders. The glorification of transparency is questionable for multiple reasons. As described below, it does not imply that the code is not only readable but also understandable. An inspection of the code will not necessarily reveal the rules (or "legal terms") embodied in a smart contract ("this code does X"), not to mention whether such rules will actually be followed ("this code has been extensively tested and formally verified to ensure it does X").

3.1 Transparency is not understandability

The transparency of a smart contract does not translate into its understandability. Most smart contracts are written in high-level programming languages and are generally expressed as source code, which takes the form of text. Such text cannot, however, be read in the same manner people read traditional contracts. Source code facilitates programming as it abstracts from the technical details of the underlying processor and relies on standardized, textual expressions. Source code is, however, designed to convey instructions to compilers or interpreters – not to convey meaning to humans. Moreover, the meaning and *effect* of individual words and expressions in source code are specific to a programming language. Without an in-depth knowledge of the relevant programming language, the code is not "understandable," especially if some programmers use unusual names for variables or functions.[103] Moreover, although programs often contain comments in natural language explaining specific lines of code, this is generally done to facilitate future revisions by *other* coders, not to facilitate understanding in general. Moreover, if the transparency narrative is to be followed, then the code of the compiler should also be made available. After all, the visibility of the smart contract's source code seems of limited usefulness if such code must be converted and if the "converting mechanism" (that is: the compiler) may affect its execution. The compiled bytecode can, in fact, be regarded as more important than the source code – especially given that the compilation process may introduce errors into the final, executable code. Notably, many smart contracts are only published in bytecode,[104] which is not human readable. Although bytecode is easily decompiled into source code, the mere need to perform this operation may overwhelm the average user – not to mention the fact that such operation only leads back to square one: to human-readable but incomprehensible computer instructions. Transparency aside, the idea of an average user inspecting *and* testing the smart contract (and its compiler!) is unrealistic[105] as it would, amongst others, require a high level of technical competence and, ideally, access to easy-to-use *automatic* analysis tools. Alternatively, users could pay to have the code audited or tested. The latter option, however, introduces a human intermediary and hence a potential source of bias and uncertainty. In effect, those who decide to use smart contracts must be technology literate or hire expert assistance. Their only other option is to trust the coder who created the smart contract. The fact that the code is in viewable and accessible seems of limited relevance.

3.2 Transparency is not clarity of rules

The transparency of the smart contract gives little assurance as to the rules it will execute. Smart contracts are supposed to provide trust by executing as coded. This does not mean that they will execute as intended. It may, in fact, be difficult to establish *what* the smart contract was actually intended to do. This seemingly small detail is easily overlooked: when stating that the smart contract may not execute "as intended," it is necessary to inquire "as intended *by whom* – its creator or its user?" There may be a discrepancy between the description of the smart contract, which will form a user's primary motivation to deploy it, and what the code is actually written, or intended *by the creator*, to do. To illustrate: a smart contract is accompanied by a description in natural language, "this smart contract will transfer payments of $X from account A to account B on the 1st of every month for a period of 12 months." The actual code, however, contains a rule that a small percentage of each payment (eg., 0.1% of $X) will be sent to an anonymous account C. In such instance, the intent of the coder differs from the intent of the user and may remain hidden or discoverable only after extensive testing. The user will most likely be unable to determine what the smart contract is actually programmed to do and rely on its description in natural language. Although the dominant blockchain narrative assumes that if the code is visible, then the rules expressed thereby are visible and hence known, it must be acknowledged that such rules may be difficult to "reverse engineer" from the code. In "traditional" transactions, when the rules, such as the terms and conditions of a contract, are written in a natural language, it is usually unnecessary to distinguish between the rules and their expression. In such instance providing the rules in English is, well, synonymous with providing the rules. If the language is clear, then the rules described therein can be discerned. The problem will lie in ensuring that the rules are followed. In contrast, the smart contract expresses certain rules, but it does not clearly communicate them because its code is not understandable without technical expertise. When presented with a smart contract, it may be difficult to establish whether its code correctly represents the agreed legal terms or conforms with its description in natural language. In case of a discrepancy between the code and its description, it becomes necessary to establish which expression will prevail. Consequently, contrary to the popular transparency narrative, the visibility of the code is of limited practical relevance.

3.3 Transparency is not predictability

Transparency may facilitate the discovery of obvious coding errors or vulnerabilities,[106] but it cannot reveal *how* the smart contract will actually execute.[107] The visibility of the code provides little insight as to what it will do or what output it will produce. [108] Vulnerabilities are not easily detectable so that code that appears technically correct may still contain them. To predict their future behavior, it does not suffice

103 Emanuel Regnath and Sebastian Steinhorst, 'SmaCoNat: Smart Contracts in Natural Language' (2018) *Forum on Specification & Design Languages* (FSDL) 5.
104 Zheng (n 31) 475.
105 Walch (n 43) 14.
106 Asem Ghaleb and Karthik Pattabiraman, 'How Effective are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools Using Bug Injection,' in (2020) *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*; Josselin Feist, Gustavo Grieco and Alex Groce, 'Slither: a Static Analysis Framework for Smart Contracts,' in (2019) *IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain* 8.
107 To understand what a program will do, it must be tested and, ideally, formally verified, see generally: Joshua A Kroll and others, 'Accountable Algorithms' (2017) 165 *U Penn L Rev* 633, 645; Ye Liu and others, 'Towards Automated Verification of Smart Contract Fairness,' in (2020) *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*
108 Antonopoulos and Wood (n 33) 172, 173.

that smart contracts are viewed - they must be extensively tested and formally verified *before* deployment. While testing denotes the broader activity of examining the expression and operation of a program, formal verification concerns the program's internal consistency and the correctness of its underlying algorithm.[109] Testing can be subdivided into static analysis, which involves the inspection of code, and dynamic analysis, which involves its actual execution.[110] Transparency is predominantly associated with static analysis, which examines the smart contract in isolation and cannot predict its execution. Only dynamic analysis can, to an extent, determine whether the smart contract will operate as intended.[111] Unfortunately, dynamic analysis requires a test environment that closely resembles the final execution environment. Testnets cannot, however, replicate the complexity of the actual execution environment, particularly with regards to the interactions between many different smart contracts. In principle, testing can only provide relative certainty as to the specific inputs that have been tested.[112] It is practically infeasible to test a smart contract for every possible input or interaction – especially if it is to execute in an open environment, where it may encounter countless smart contracts written by coders with varying motivations and competence levels. To fully illustrate the practical difficulties of testing – and hence ensuring that smart contracts execute as intended - this paragraph should discuss public test-nets, blockchain emulators and simulators, as well as the challenges of developing testing strategies.[113] The technical issues are complex, but the point is simple: testing computer programs is technical skill that most users do not have and, more importantly, it is difficult to imagine the resources required to test a smart contract to the extent that would render it predictable.

4. Immutability

Tales of trust and certainty are inextricably linked to the concept of immutability. In the context of blockchains, immutability concerns the inability to change the underlying consensus algorithm and the contents of the individual blocks.[114] In the context of smart contracts, immutability concerns the inability to change their code. Once deployed on a blockchain, the smart contract cannot be modified and remains beyond the control of any single entity.[115] Immutability is also associated with smart contracts being tamper-proof, that is: resistant to any form of external interference. Immutability is supposed to ensure absolute adherence to the rules embedded in the code. Upon closer examination, however, immutability raises significant concerns, especially if smart contracts are to govern real world relationships. It is one thing to preclude third parties (e.g., hackers) from altering their code, it is yet another to preclude the possibility of modifying smart contracts by those who created them and/or by those who use them. Unauthorized alterations must be distinguished from permitted and indispensable modifications. A blank statement that any form of subsequent interference is undesirable is simply not true.

4.1 Modifiable Code

Smart contracts are computer programs - and every computer program requires subsequent or even ongoing modifications[116] to meet the changing needs of their users and/or to correct errors and vulnerabilities. Perfect code does not exist. Smart contracts are no exception. Once deployed on the blockchain, all errors and vulnerabilities in smart contracts become permanent. To address the problems resulting from immutability, methods are being developed to enable smart contract "modifications" by, for example, having their code refer to secondary smart contracts or by "migrating" from faulty smart contracts to updated ones.[117] While the idea of correcting smart contracts seems commendable, it also reneges on their main promise – that of being immune to outside interference. After all, any modifications require human involvement *after* the smart contract is deployed. Not only would "modifiable" smart contracts cease to be immutable, but they would also require governance mechanisms indicating the entities authorized to make such modifications and, ideally, the permitted scope thereof. It must be assumed that the parties themselves would be technically incapable of amending the code and would require assistance from third parties, that is - expert coders. It is worth observing that the ability to modify the smart contract does not guarantee that the modified code is correct and error-free. It also gives power to those who are authorized to perform such modifications.[118] This creates a difficult situation: the immutability of smart contracts is disadvantageous due to the statistical inevitability of coding errors and vulnerabilities but introducing the possibility of modifying smart contracts defeats their very purpose as it renders them "mutable" and hence less trustworthy.

4.2 Comprehensive rules

The code of a smart contract is an expression of certain rules and, following the popular blockchain narrative, such rules are generally synonymous with the terms of a contract. If both the code and the rules expressed thereby are to be immutable, then both must be perfect

109 See: Daniele Magazzeni and others, 'Validation and verification of smart contracts: a research agenda,' (2017) 9 *Computer* 50, 54; technically, smart contracts require verification, validation and testing. Verification aims to ensure that "we are developing the right product" (i.e. it meets the specifications), validation aims to ensure that "we have developed the product right" (i.e. it fulfills its intended purpose) and testing should reveal "the existence of errors in the product," see: Osman Balci, 'Validation, verification, and testing techniques throughout the life cycle of a simulation study' (1994) 53 *Annals of Operations Research* 121.

110 Richard E Fairley, 'Tutorial: Static Analysis and Dynamic Testing of Computer Software' (1978) Computer 14, 22; Dolores R Wallace, and Roger U Fujii, 'Software verification and validation: an overview' (1989) 6 *IEEE Software* 10.

111 Here, for the sake of simplicity, it is useful to assume that the output intended by the coder is identical to the output desired by the user of a particular smart contract.

112 In principle, no amount of testing guarantees how a program would operate in a situation that has not been tested, see: Michael Sipser, Introduction to the Theory of Computation 201 (2006), Alan M Turing, 'On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction' (1937) 43 *Proc London Math Soc'y* 544, 544-46 ; Axel Halin and othersl, 'Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack' (2019) 24 *Empirical Software Engineering* 674.

113 see: Chaimaa Benabbou, Chaimaa and Önder Gürcan, 'Survey of Verification, Validation and Testing Solutions for Smart Contracts' (2021) https://arxiv.org/abs/2112.03426 accessed 1 October 2022.

114 Walch (n 40) 735, 736; Jeffrey M. Lipshaw, 'The Persistence of "Dumb" Contracts' (2019) 2 *Stan J Blockchain L & Pol'y* 1, 24.

115 A contract can be "deleted," leaving a blank account on Ethereum.

116 Cohney and Hoffman (n 22) 325, 343.

117 Erik Daniel and Florian Tschrosch, 'Towards Verifiable Mutability for Blockchains' (2021) https://arxiv.org/pdf/2106.15935.pdf accessed 1 October 2022; B Marino and Ari Juels, 'Setting Standards for Altering and Undoing Smart Contracts,' in J Alferes and others, (eds) *Rule Technologies. Research, Tools, and Applications* (Springer 2016).

118 For a detailed discussion see: Pedro Antonino and others, 'Specification is Law: Safe Creation and Upgrade of Ethereum Smart Contracts' (2022) at https://arxiv.org/abs/2205.07529 accessed 1 October 2022; see also: Mehdi Salehi, Jeremy Clark and Mohammad Mannan, 'Not so immutable: Upgradeability of Smart Contracts on Ethereum' (2022) https://arxiv.org/abs/2206.00716?context=cs accessed 1 October 2022.

from the moment the smart contract is deployed. More specifically, for the smart contract to be of practical value, its rules embodied must be comprehensive and "future-compatible": capable of addressing a wide range of events that might affect a given smart contract throughout its lifecycle. The "certainty of outcomes," guaranteed by immutability, requires that such outcomes remain aligned with changing circumstances, including price fluctuations, pandemics, or natural disasters. The resulting practical difficulties are immediately apparent. Market conditions may change dramatically, even within short periods of time. A smart contract that looked attractive when deployed in June may become nonsensical when executed in October. Maintaining an alignment between the smart contract and any ongoing events would, however, necessitate detailed and complex code as well as an unprecedented level of foresight. A large upfront investment would be required to provide for a wide range of possible execution paths, especially for smart contracts governing long-term relationships. To re-emphasize: once deployed, the smart contract will continue executing *irrespective* of the surrounding circumstances. If a response to such changed circumstances has not been provided for in the terms, the smart contract may become commercially infeasible. Such would be the case, for example, if it continued to release payments far above the market price because its creator failed to encode an adequate price adjustment formula or if it continued to enable access to certain premises despite the fact that new regulations required that such premises be closed due to public safety measures.[119] The problems are not technical but practical and concern the inability to predict how future events will affect a smart contract's viability. The fact that the rules embedded in the code are immutable and that they will be followed "no matter what" can be a boon or a bust. Everything will depend on whether the users of a particular smart contract know *and* are willing to accept the risk that the smart contract may cease to be commercially viable in the event of a change in circumstances.

5. Integrating with the Real World

Neither blockchains nor smart contracts interface with the outside world. To remain "trustless and secure" blockchains are insulated from external input. A smart contract devoid of access to information about events in the real world is, however, of limited commercial importance as it cannot facilitate, or execute, traditional transactions that involve real-world assets and events. As indicated, smart contracts exist on-chain and "codify the conditions" of release of the crypto-assets controlled by them. If such conditions concern outside events, such as weather data, exchange rates, the arrival of a shipment or the delivery of a container, mechanisms are required to establish whether they have been fulfilled.[120] Consequently, smart contracts must obtain information about events in the real world ("off-chain information"). Entities providing such information are called "oracles."[121] Their role is to provide off-chain information automatically, without human participation. The benefits of smart contract would be lost if the fulfillment of the release conditions had to be confirmed by a fallible, biased and *un*trustworthy human. Oracles, however, come with their unique set of challenges.

5.1 The "Oracle Problem"

Oracles exist off-chain, in the real world, and are created by human coders. They do not share any attributes of blockchains. In particular, it bears emphasizing that the consensus algorithms underlying blockchains cannot determine or validate the correctness of any off-chain information.[122] Oracles obtain off-chain information from external sources and, in principle, are incapable of directly establishing its veracity. In other words, oracles "only" provide information – they do not create or verify its correctness. The resulting risks are obvious: if the off-chain information is incorrect, the smart contract will release crypto-currencies even if the release conditions were not met. Logically, even a smart contract written in perfect code cannot be trusted if it can execute based on false off-chain information. Consequently, as oracles effectively control the release of assets held by a smart contract, they must be trusted. This is known as "the oracle problem." The off-chain information may be incorrect for multiple reasons, both technical and non-technical. Technical reasons may derive from programming errors or from external interference. They may concern the data source(s), the oracle itself or, in the case of oracle networks - the individual nodes, the entire network as well as the transfer mechanisms between data sources and oracles. Non-technical reasons concern the fact that individual oracles as well as data sources may become corrupted.[123] Even if an oracle operates correctly and the data source is reliable, the entities managing the oracle and/or the data source may alter the information given sufficient commercial incentives, i.e., bribery. The likelihood of such compromise is correlated to the value of the cryptocurrencies controlled by a given smart contract. The more important the off-chain information and the more valuable the cryptocurrencies controlled by a given smart contract, the greater the level of trust that must be placed in the oracle – and the greater the incentive for its compromise.[124] Unsurprisingly, in parallel with the aforementioned vulnerabilities of smart contracts, exploits based on oracle manipulation are increasingly common.[125]

5.2 Attempted Solutions

To date, most efforts to solve the oracle problem, focus on its technical component since the socio-economic dimension is more difficult to address. To discuss such efforts, it is useful to introduce a general taxonomy of oracles. Oracles come in two broad categories: centralized and consensus-based. Centralized oracles are, as the name implies, controlled by a single entity. Centralized oracles are more suitable to furnishing information that is not publicly available, such as information concerning products moving through supply chains. Given that centralized oracles cannot be "trustless," they aim to be "provably" honest or, at least, capable of establishing the integrity and authenticity of the off-chain information.[126] As centralised oracles require that the entities operating them be trustworthy, they contradict the very purpose of smart contracts - that of eliminating the need for trust in a single entity. There is, after all, no point in constructing an elaborate ecosystem around a decentralized and distributed execution environment, such as the Ethereum blockchain, if the

119 Could smart contract creator have predicted the pandemic and the recent crypto-crash?
120 Werbach (n 7) 547.
121 See generally: Guilio Caldarelli, 'Understanding the Blockchain Oracle Problem : A Call for Action' (2020) 11 *Information* 509.
122 Marco Comuzzi, Cinzia Capiello and Giovanni Meroni, 'On the need for data quality assessment in Blockchains' (2021) *IEEE Internet Computing* 71, 75.
123 Cohney and Hoffman (n 22) 355.
124 Jesus Rodriguez, 'The Middleman of Trust: The Oracle Paradox and Five Protocols that Can Bring External Data into the . . . , HACKERNOON (July 31, 2018) https://hackernoon.com/the-middleman-of-trust-the-oracle-paradox-and-five-protocols-that-can-bring-external-data-into-the-df39b63e-92ae accessed 1 October 2022.
125 Caldarelli (n 121) 8; see also: Chainalysis, The 2022 Crypto Crime Report (2022) 72, 73, https://go.chainalysis.com/rs/503-FAP-074/images/Crypto-Crime-Report-2022.pdf accessed 1 October 2022.
126 Junhoo Park and others, 'Smart contract data feed framework for privacy-preserving oracle system on blockchain' (2021) 10 *Computers* 7, 10, 12.

success of a transaction depends on – or requires trust in – an oracle. The problem of centralization is addressed by so-called consensus oracles: networks of oracle nodes, which collectively determine which off-chain information should be provided to smart contracts.[127] Such determination relies on a combination of consensus protocols, voting procedures and/or economic incentives. In principle, the aim is to ensure co-operation between the nodes to "agree" on the correctness of off-chain information. To minimize the risk of false information, consensus oracles require a larger number of nodes and data sources. For example, Schelling oracles rely on crowdsourcing knowledge about certain events and involve entities betting their reputation or assets to prove that information about those events is true. The information is openly evaluated by other platform participants, the assumption being that if many participants confirm its accuracy, such information must be true. Unfortunately, Schelling oracles can only verify publicly available information and are of little use in situations where the smart contract requires information concerning the fulfillment of unique, transaction-specific payment conditions.[128] Establishing the price of a publicly traded stock differs from establishing the arrival of a shipment at a private location. Other oracle voting strategies are exemplified by prediction markets, such as Augur, which enable users to bet on anything (elections, sports) and be rewarded or penalized depending on the correctness of their predictions.

Another example of consensus-based oracles are decentralized oracle networks, ("DONs") which require many nodes to reach consensus on the correctness of off-chain information obtained from multiple data sources. Ideally, the number of nodes and data sources should scale in proportion to the value of a given smart contract, so that smart contracts controlling more valuable assets obtain information from a larger number of nodes, which in turn obtain information from a larger number of data sources. The problem with this scenario is that once there are many oracle nodes and many data sources, it becomes necessary to ensure their efficient co-operation in reaching consensus as to whether the information is correct. Notwithstanding the similarity of the terminology, consensus mechanisms in DONs differ from their blockchain counterparts. While the latter confirm compliance with concise technical parameters in a closed, artificial system, the former must determine the veracity of information about real-world events, with all its ambiguities, gradations, and complexities. Establishing the timestamp of a block differs from establishing whether the delivered goods arrived at a specified location and match their description. To aggravate matters, when reaching consensus on the correctness of off-chain information, the nodes must rely on external data sources and are unable to directly verify whether such information reflects reality. In other words, DONs rely on the statistical likelihood that the off-chain information reported by the majority of nodes is correct, not on the actual verification of such information. Consequently, DONs can only establish how many nodes reported certain information as true and must address situations where different nodes provide divergent reports. The larger the number of nodes, the greater the risk of discrepancies and the greater the need for mechanisms ensuring that nodes "collectively exhibit correct behaviour." Examples of such mechanisms can be found in Chainlink.[129] There, nodes are given crypto-economic incentives to report correct information: nodes providing incorrect reports forfeit the tokens they have initially deposited – or staked - to participate in the DON.[130] Such system "should render attacks such as bribery unprofitable for an adversary, even when the target is a smart contract with high monetary value."[131] Unfortunately, staking only works if the market price of the deposited crypto-currency exceeds the value of the assets controlled by a smart contract. Moreover, to address divergent reports, DONs require two tiers of nodes: the first directly accesses the data sources and generates reports, the second monitors irregularities in the first tier. Effectively, some nodes become "watchdogs"[132] reporting the incorrect behavior of other nodes.

In sum, centralized oracles are efficient but, given that a single entity controls the oracle, they remain vulnerable, both technically and economically. Consensus oracles increase the reliability of the off-chain information but require a complex, multi-layered infrastructure to coordinate the nodes to achieve an acceptable degree of "trustlessness." [133]

6. Concluding Observations

Claims regarding the transformative potential of smart contracts must be investigated, not blindly trusted. It is necessary to acknowledge the difficulties of creating reliable and predictable smart contracts. It is also necessary to to examine the immediate practical implications of such concepts as immutability or transparency.

Tales of their *future* significance notwithstanding, smart contracts are first and foremost computer programs – and it is difficult to rationalize how a computer program could eliminate trust or enhance commercial certainty. The fact that smart contracts execute on blockchains changes little in this regard. There is, after all, no blockchain-specific mechanism preventing the deployment of smart contracts that contain errors and vulnerabilities. Consensus algorithms protect against malicious nodes but not against malicious or incompetent coders. The challenges inherent in the execution environment represented by permissionless blockchains, combined with novel programming languages and novice coders, produce a dangerous combination that contradicts both certainty and trust. The transparency of the smart contract does not bring any immediate benefits to the average user. The visibility of the code does not translate into its readability or understandability. It also provides limited assurances as to the quality of the smart contract or insights as to how it will actually execute. Unless the parties have created the smart contract themselves and/or extensively tested it prior to deployment, they have no guarantees as to the outcomes it will produce. In practice, only expert programmers can evaluate the viability of a given smart contract and, at least theoretically, ensure that it will execute as intended. The immutability of smart contracts seems equally disadvantageous. It would only be beneficial if it was possible to ensure that their code was perfect

127 Shuai Wang and others, 'A Novel Blockchain Oracle Implementation Scheme Based on Application Specific Knowledge Engines' in (2019) *IEEE International Conference on Service Operations and Logistics, and Informatics* (2019) 258.

128 Jack Peterson and others, Augur: a decentralized oracle and prediction market platform (2015) https://arxiv.org/abs/1501.01042 accessed 1 October 2022.

129 Lorenz Breidenbach and others , 'Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks (2021) https://research.chain.link/whitepaper-v2.pdf?_ga=2.138147696.157341870 8.1621330901-1199595055.1621330901 accessed 1 October 2022.

130 See: J Adler and others, 'Astraea: A decentralized blockchain oracle,' in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data* 1145.

131 Breidenbach (n 129) 17, 18.

132 Breidenbach (n 129) 88.

133 Hamda Al-Breiki and others , 'Trustworthy Blockchain Oracles: Review, Comparison, Research Challenges' (2020) 8 *IEEE Access* 85678.

from the outset and that the rules embodied therein were sufficiently comprehensive to allow for changed circumstances. Setting the smart contract in stone at the time of deployment resembles fixing the course of a cruise liner before it leaves port. There may be icebergs. We all know what happened to the Titanic. Code must be capable of being modified and contracts must remain responsive to their surroundings. Commerce wants certainty but also flexibility. Smart contracts that depart from the blockchain ideology and are amenable to modifications, require mechanisms governing the extent of such modifications and a clear indication of those who are allowed to introduce them. Smart contracts that remain true to the blockchain ideology and remain immutable, require mechanisms to allocate the risks resulting from their use. Given the statistical inevitability of coding errors and vulnerabilities, such risks are non-trivial. Additional risks derive from the oracle problem. The ability to trust a smart contract depends on the trustworthiness of the oracle and the data source. Efforts to solve the oracle problem demonstrate that trust and certainty cannot be achieved solely by technological means. Even the most elaborate technical solution will contain a weak spot, be it in the form of a corruptible data source, a faulty oracle or a badly designed governance system for decentralized oracle networks.

Irrespective of their legal classification, smart contracts are technically incapable of enhancing certainty or eliminating the need to trust humans. Legal analyses of smart contracts must be based on their actual technical attributes and on an understanding of the risks involved in their use. Technology is a question of fact. Only after establishing the facts, is it possible to select the most appropriate legal response, if any.