2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)

Smarter Smart Contract Development Tools

Michael Coblenz, Joshua Sunshine, Jonathan Aldrich, Brad A. Myers
School of Computer Science
Carnegie Mellon University
Pittsburgh, USA
{mcoblenz, sunshine, aldrich, bam}@cs.cmu.edu

***Abstract*—Much recent work focuses on finding bugs and security vulnerabilities in smart contracts written in existing languages. Although this approach may be helpful, it does not address flaws in the underlying programming language, which can facilitate writing buggy code in the first place. We advocate a re-thinking of the blockchain software engineering tool set, starting with the programming language in which smart contracts are written. In this paper, we propose and justify requirements for a new generation of blockchain software development tools. New tools should (1) consider users' needs as a primary concern; (2) seek to facilitate safe development by detecting relevant classes of serious bugs at compile time; (3) as much as possible, be blockchain-agnostic, given the wide variety of different blockchain platforms available, and leverage the properties that are common among blockchain environments to improve safety and developer effectiveness.**

***Index Terms*—smart contracts, usability of programming languages, blockchain**

I. Introduction

Blockchain platforms are increasingly supporting sophisticated computation, implemented in *smart contracts*. Unfortunately, programs implemented on blockchain platforms have suffered a series of bugs and security vulnerabilities through which over $80 million worth of virtual currency has already been stolen [1], [2]. The observation that these platforms are significant targets of high-stakes software development has led to a new area of research in which researchers propose methods to improve security. We argue that the research community places too much emphasis on small changes to the development process that might identify specific bugs and vulnerabilities after they have already been introduced. Instead, we propose that the community refocus itself on a top-down agenda to rethink the problems of blockchain software development from a human perspective.

We sampled conference proceedings from 2018 and categorized each blockchain-related paper. Among 35 blockchain-related papers we identified from 2018, only one focused on the possibility of programming languages other than Solidity to implement smart contracts — and that paper compared existing languages, rather than proposing a new language design [3]. On the other hand, five conducted analyses of smart contracts written in existing languages. Although other work on language design is ongoing, most blockchain research seems to focus on building blockchain applications and platforms rather than on *what tools* software engineers will use to write applications.

The notion that programming languages affect how people think dates back to the early history of computing. Iverson quoted A. N. Whitehead, who wrote in 1911: "By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race" [4]. We expect, then, given the nascent nature of Solidity and the serious bugs regularly found in Solidity programs, that iterating on the smart contract language would significantly improve the ability of software engineers to write smart contracts effectively. Indeed, a line of research has found various impacts of language design on programmer effectiveness [5], [6]. We argue, as Stefik and Hanenberg do [7], that programming language designers have a responsibility to consider usability as a key criterion when designing and evaluating programming languages.

In this paper, we propose three requirements for languages intended for smart contract authorship, and observe that no commonly-used language meets these requirements. We are designing a new programming language, Obsidian [8], according to these requirements, but we invite other researchers to join us in exploring the space of better blockchain programming languages. We argue that languages should (1) consider users' needs as a primary concern; (2) seek to facilitate safe development by detecting relevant classes of serious bugs at compile time; and (3) as much as possible, be blockchain-agnostic, given the wide variety of different blockchain platforms available.

II. Analysis of recent blockchain papers

To assess the recent focus on smart contract languages relative to other topics in the blockchain research community, we conducted a literature review. We identified a set of categories into which we divided the papers, revising the categories as we found additional papers that did not fit in any of the established categories. The categories and paper counts are shown in Table I.

We included all papers in the proceedings of WETSEB 2018 and ICBC 2018. In addition, we searched the proceedings of ICSE 2018 and PACMPL so that we would include more general-interest papers on blockchain software engineering and programming languages; the latter searches (for the term "blockchain") resulted in only one paper each. In all, we found 35 papers on blockchain topics.

We were surprised that there was only one paper in the corpus pertaining to smart contract programming language

978-1-7281-2257-1/19/$31.00 ©2019 IEEE
DOI 10.1109/WETSEB.2019.00013

Category	Papers
Blockchain application implementations and techniques	15
Blockchain platforms: consensus, performance, security, analysis	11
Legal Regulation of blockchain systems	11
Analysis of smart contracts in existing languages	5
Alternative smart contract languages	1
Blockchain application modeling/visualization	1
Security	1

TABLE I

BLOCKCHAIN RESEARCH CATEGORIES AND PAPER COUNTS

design [3]. That paper, by Parizi and Dehghantanha, reported on a user study comparing Solidity [9], Pact [10], and Liquidity [11]. Because the vulnerabilities were analyzed by a tool designed to find particular vulnerabilities that are common among Solidity programs, it is not clear whether the programs in the other languages included serious vulnerabilities that the tool did not detect. The authors measured times for novice participants to complete a collection of programming tasks when assigned one of three different languages, arguing that faster times indicate increased usability. However, time for novices to complete small programming tasks is only one aspect of usability. A more thorough examination would consider the challenges users face; more realistic tasks and experienced users; and perhaps most importantly, a fine-grained analysis of which language design features contributed to or detracted from usability.

One additional paper proposed a programming language for analysis of the Ethereum blockchain [12], but that language is not suitable for implementing smart contracts themselves; it is limited to analyses of the whole ledger.

Several blockchain languages did not appear in our mapping study, which focused on recent academic publications in the venues we considered most likely for work of this nature. A paper on Flint was published in the Programming conference proceedings [13]. Other languages, such as Liquidity, Pact, and Vyper [14], are commercial projects. Although they may be better than Solidity, they have not achieved wide adoption and have not been examined in a peer-reviewed context.

III. REQUIREMENTS FOR SMART CONTRACT LANGUAGES

Having observed relatively little academic research that considers the possibility of improving smart contract development by creating or refining programming languages, we elicit requirements for successful smart contract languages by observing the application domain and the history of the community's experience with existing languages.

A. Strong static safety

Once a smart contract is deployed and users rely on the deployed contract, bugs in the deployed contract can be impossible to fix, since the nature of some blockchains is such that the contract cannot be modified after deployment. Furthermore, smart contracts may implement high-stakes software governing valuable resources, such as cryptocurrencies, or maintaining important business records. Bugs in smart contracts undermine the very purpose of the blockchain, which is to facilitate transactions among parties that have not established trust.

Delmolino et al. found that novices typically make several serious errors when writing smart contracts [15]; although the study was done with Serpent, which predated Solidity, popular blockchain languages have not been designed to prevent the bugs that Delmolino et al. found, such as errors in encoding state machines resulting in loss of money.

As a result, we argue for strong, compile-time mechanisms to eliminate as many classes of bugs as is practical. Recent work on static smart contract checking, such as Oyente [16] and MadMax [17], aim to find instances of particular classes of bugs. But these kinds of static analysis may not have the same kinds of deep impact on programmer's reasoning as deeper language design decisions. Indeed, correctness by construction may be a more direct means of attaining safety, since it demands that programmers structure their artifacts in a way that avoids classes of bugs. It is better, then, to combine careful language design, which shapes how people think, with static analyses that can detect other bugs that the language cannot prevent. For example, commonly-used languages offer type systems that do not facilitate reasoning about assets, which can be owned and consumed. This leads to bugs in which assets are accidentally either consumed more than once or lost forever, as has been shown to occur [15]. Likewise, traditional languages do not facilitate compile-time reasoning about the states that objects are in, even though smart contracts typically implement state machines that support different transactions depending on the state. This can lead to bugs in which transactions are invoked on contracts that are in inappropriate states.

B. User-centered design

Blockchain software development is fundamentally a *human* process, involving human developers who must create applications to meet *human* needs [18]. A smart contract language, then, is an interface by which people can specify behavior, and is subject to the principles of human-computer interaction.

Many programming language designs and tools for software engineers aim to reduce the difficulty of writing programs, but unfortunately, most of the tools in use today were not formally evaluated with users. Instead, designers guessed what would be "good" for users and provided it. But the users of the tools may have different needs than the designers, so it does not suffice for designers to create tools for themselves and hope that others will learn to use the tools effectively. We argue that software engineers, being people too, are amenable to the methods of human-computer interaction [19] and that designers of programming languages should use a wide variety of techniques in order to make their tools more usable [18].

A lot of approaches developed in the programming language research community are not adopted by developers because they are too hard to use or are impractical for real-world usage. For example, Bhargavan et al. [20] proposed formal verification of smart contracts. Though many researchers are

working on this, formal verification is still impractical for most programmers to use.

End-user programming offers the potential to empower more users to write software. Spreadsheets are a widely-used programming environment. Can we design smart contract languages that enable business analysts at a company to write code reflecting policy changes [21] according to business needs, rather than assuming that a professional software engineer will always be part of the process? Can we enable domain experts, who might already be fluent in visual languages such as BPM, to write and modify smart contracts [22], [23]? This goal might seem to be at odds with the goal to offer stronger static checking, but in many end-user systems the key is to make it very easy to accomplish common tasks safely. For example, Scratch uses the shapes of connectors to indicate how components may be connected together [24]. Each language designer should carefully consider who the target users are and whether it is possible to enable more people to write and maintain smart contracts.

C. Blockchain-agnosticism

As of this writing, there were at least 28 different blockchain platforms [25]. An author of a smart contract may need to deploy on one blockchain platform initially, and then migrate as the blockchain market matures. Even Ethereum, which is the longest-running and most popular smart contract platform, plans significant changes in Ethereum 2.0, which may cause some users to migrate to or from Ethereum. Each platform supports a particular set of languages; currently-supported languages include Java, Go, Kotlin, Solidity, C++, WebAssembly, Python, JavaScript, Liquidity, and Rust (among others). A new smart contract language that only works on one platform locks itself to an uncertain future. More importantly, programs last a long time, and implementations need to be robust to changes in underlying infrastructure.

Instead of committing to a particular platform, designers should make a small number of safe assumptions about the nature of blockchain platforms, allowing developers to create stable software against a simple abstraction. By doing so, language designers might improve safety and usability relative to general-purpose languages. Languages might rely on the following properties that most blockchain systems have in common:

Sequential execution: Blockchain platforms typically support neither parallelism nor concurrency. This simplifies reasoning relative to systems that do support parallelism or concurrency.

Deterministic evaluation: Because all peers must obtain the same result when executing transactions, smart contracts cannot depend on arbitrary external API invocations.

High cost of computation: Computation on the blockchain is substantially more expensive than off-chain computation, so although transactions are typically small, they need to be exceptionally cheap. Public blockchains require users to pay cryptocurrency according to the computational cost of their transactions, providing additional motivation to keep transactions cheap and also motivating a need to predict transaction costs in advance of execution.

Unpredictable transaction ordering: Clients submit transaction requests to blockchains without knowing what transactions will occur between their submission and the transactions' executions. If an intervening transaction violates an implicit precondition of a transaction, the results of the transaction might be surprising to the user who issued it. [16]

Cryptography needed to maintain secrecy: Blockchains permit a collection of nodes to see all the data on the blockchain. Any data that should remain secret from some of these nodes must be encrypted. This requires cryptographic techniques, such as timed commitments [26]. This difficulty comes up frequently; for example, a gambling application might need to allow players to place secret bets, but if the bets are stored naïvely, the bets will be public.

Off-chain interaction: Client software that executes off-blockchain needs to interact with deployed smart contracts. Some existing approaches, such as that in Ethereum, limit the APIs to those that only take primitive arguments, stifling the expressiveness of the APIs. New languages should facilitate well-designed, expressive APIs by allowing arbitrary data structures to be passed and returned. New languages should make it easy to develop client software that interacts with blockchains.

Storage: Some blockchain platforms, such as Hyperledger Fabric, require users to manually save and restore data from the ledger as key/value pairs. This allows fine-grained control but also requires programmers to expend significant effort and offers the potential for bugs. Languages should facilitate automatic, efficient storage of smart contract state.

IV. Obsidian: A new blockchain programming language

Our design complements some of the static analyses that have been proposed (such as in Oyente) by detecting additional bugs that existing analyses cannot detect. For example, in Obsidian, we statically rule out a class of bugs that result in *loss of assets* due to owning references to assets going out of scope. We also take advantage of the fact that contracts typically support different transactions depending on their state, and we rule out bugs in which transactions are invoked when the contract is in an inappropriate state. This kind of approach has been shown to be helpful even when only used as documentation [27]. We hope to show that programmers who use Obsidian tend to write smart contracts with fewer serious bugs.

Although we are targeting professional software engineers with Obsidian, we are integrating user-centered design into the language design process in order to maximize the chances that it will be effective for all levels of users [28]. We plan to evaluate the final language in a user study and we

encourage other language designers to substantiate their claims of usability or increased productivity with studies with target populations.

Obsidian is applicable to many different blockchain platforms, but our first target is Hyperledger Fabric. Obsidian could be made suitable for Ethereum as well with some additional work on resource estimation. Obsidian compiles to Java, which is supported natively on Fabric, and has low runtime overhead, minimizing cost of transaction execution. Obsidian automatically serializes and deserializes objects for archiving in the ledger as well as to support API invocation by clients. Obsidian leverages the assumption that the environment is sequential to simplify reasoning, although aspects of the Obsidian type system might facilitate reasoning in concurrent contexts.

V. Conclusion

The distinctive characteristics of blockchain programming environments and the applications that people want to write for them motivate the design of domain-specific languages to improve safety, security, and developer effectiveness. Recent research has focused on improving programs written in existing languages, such as Solidity, but there remains a potential for novel languages to shape how developers think and thus make developers much more productive. By leveraging properties common to many different blockchain platforms, researchers and language designers may be able to create novel cross-platform smart contract languages that make developers more effective.

Acknowledgment

This material is based upon work supported by the US Department of Defense, by NSF grants CNS-1734138 and CNS-1423054, by NSA lablet contract H98230-14-C-0140, by the Software Engineering Institute, and by AFRL and DARPA under agreement #FA8750-16-2-0042. Michael Coblenz is supported by an IBM PhD fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] E. Gün Sirer, "Thoughts on the DAO hack," 2016. [Online]. Available: http://hackingdistributed.com/2016/06/17/thoughts-on-the-dao-hack/

[2] L. Graham. (2017) $32 million worth of digital currency ether stolen by hackers. [Online]. Available: https://www.cnbc.com/2017/07/20/32-million-worth-of-digital-currency-ether-stolen-by-hackers.html

[3] R. M. Parizi and A. Dehghantanha, "Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security," vol. 10974. Springer International Publishing, 2018, pp. 75–91. [Online]. Available: http://link.springer.com/10.1007/978-3-319-94478-4

[4] K. E. Iverson, "Notation as a tool of thought," *Commun. ACM*, vol. 23, no. 8, pp. 444–465, Aug. 1980. [Online]. Available: http://doi.acm.org/10.1145/358896.358899

[5] S. Hanenberg, S. Kleinschmager, R. Robbes, É. Tanter, and A. Stefik, "An empirical study on the impact of static typing on software maintainability," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1335–1382, oct 2014.

[6] S. Endrikat, S. Hanenberg, R. Robbes, and A. Stefik, "How do API documentation and static typing affect API usability?" in *International Conference on Software Engineering*. New York, NY, USA: ACM, 2014, pp. 632–642.

[7] A. Stefik and S. Hanenberg, "The programming language wars: Questions and responsibilities for the programming language community," ser. Onward! 2014. New York, NY, USA: ACM, 2014, pp. 283–299. [Online]. Available: http://doi.acm.org/10.1145/2661136.2661156

[8] M. Coblenz, "Obsidian: a safer blockchain programming language," in *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017, pp. 97–99.

[9] Ethereum Foundation, "Solidity," https://solidity.readthedocs.io/en/develop/. Accessed Jan. 3, 2017.

[10] Kadena, "Pact," 2019. [Online]. Available: https://pact.kadena.io

[11] OCamlPRO, "Liquidity, a simple language over Michelson," 2019. [Online]. Available: https://github.com/OCamlPro/liquidity/blob/master/docs/liquidity.md

[12] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "Ethereum query language," *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, 2018.

[13] F. Schrans, S. Eisenbach, and S. Drossopoulou, "Writing safe smart contracts in Flint," in *Conference Companion of the 2Nd International Conference on Art, Science, and Engineering of Programming*, ser. Programming'18 Companion. New York, NY, USA: ACM, 2018, pp. 218–219. [Online]. Available: http://doi.acm.org/10.1145/3191697.3213790

[14] The Ethereum Foundation, "Vyper," 2019. [Online]. Available: https://github.com/ethereum/vyper

[15] K. Delmolino, M. Arnett, A. E. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," *IACR Cryptology ePrint Archive*, vol. 2015, p. 460, 2015.

[16] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," in *Proceedings of ACM CCS'16*, 2016.

[17] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "Madmax: Surviving out-of-gas conditions in ethereum smart contracts," *OOPSLA*, 2018.

[18] M. Coblenz, J. Aldrich, B. A. Myers, and J. Sunshine, "Interdisciplinary programming language design," in *Onward! 2018 Essays*, ser. SPLASH '18, 2018.

[19] B. A. Myers, A. J. Ko, T. D. LaToza, and Y. Yoon, "Programmers are users too: Human-centered methods for improving programming tools," *Computer*, vol. 49, no. 7, pp. 44–52, July 2016.

[20] K. Bhargavan, N. Swamy, S. Zanella-Béguelin, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, and T. Sibut-Pinote, "Formal Verification of Smart Contracts," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, New York, New York, USA, 2016.

[21] IBM. Blockchain for supply chain. [Online]. Available: https://www.ibm.com/blockchain/supply-chain/

[22] O. López-Pintado, L. García-Bañuelos, M. Dumas, and I. Weber, "Caterpillar: A blockchain-based business process management system," in *BPM 2017, Barcelona, Spain*, 2017.

[23] A. Tran, Q. Lu, and I. Weber, "Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management," *Demo Track at BPM*, vol. 2018, pp. 56–60, 2018.

[24] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: Programming for all." *Commun. Acm*, vol. 52, no. 11, pp. 60–67, 2009.

[25] M. Yusuf, "A comprehensive list of blockchain platforms," 2018. [Online]. Available: https://www.technoduet.com/a-comprehensive-list-of-blockchain-platforms/

[26] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts," Cryptology ePrint Archive: Report 2016/1007, https://eprint. iacr. org/2016/1007, Tech. Rep., 2016.

[27] J. Sunshine, J. D. Herbsleb, and J. Aldrich, "Structuring documentation to support state search: A laboratory experiment about protocol programming," in *European Conference on Object-Oriented Programming (ECOOP)*, 2014.

[28] C. Barnaby, M. Coblenz, T. Etzel, E. Kanal, J. Sunshine, B. Myers, and J. Aldrich, "A user study to inform the design of the obsidian blockchain dsl," in *PLATEAU '17 Workshop on Evaluation and Usability of Programming Languages and Tools*, 2017.