Requirements Engineering Challenges for Blockchain Rollups

Jan Gorzny
Zircuit
jan@zircuit.com

Martin Derka
Zircuit
martin@zircuit.com

***Abstract*—Rollups are a type of blockchain network that derives state from another blockchain network. These systems are complex pieces of middleware that provide a development platform for decentralized applications but also depend on another blockchain network to function. They involve a number of components, some of which execute off-chain (possibly on specialized hardware), and some of which are implemented via smart contracts. They are expected to handle millions of dollars in cryptocurrencies and operate at scale with strong liveness guarantees. These properties result in a unique set of requirements for these systems. In this document, we report on the requirements engineering challenges related to building such a system from first-hand experience.**

***Index Terms*—blockchain, rollup, requirements engineering, web3**

I. Introduction

Modern blockchains like Ethereum [1] support so-called *smart contracts*. A smart contract is a program deployed on the blockchain whose functions can be called via blockchain transactions [2]. Smart contracts provide the ability to build "decentralized Applications" (dApps) which use the blockchain as a back-end for logic and data storage. dApps can hold or implement digital assets such as Ether or ERC-20 tokens [3].

These smart contracts can also be used to track the state of another blockchain. The network running the smart contracts is called the *layer one* and the network anchored to these contracts is the *layer two* [4]. The smart contracts on the layer one can implement a so-called *bridge* [5] between the networks, and if the layer two state is only considered valid according to some conditions enforced by the smart contract, the layer two inherits some security from the layer one. Namely, the layer two state is only valid if the underlying layer one accepts it, which is hard to do if the layer one is hard to fool. This design allows the layer two to execute its state updates off-chain and periodically post the resulting state to the layer one. As a result, rollups (a.k.a. *commit-chains* [6] or *validating bridges* [7]) can process more transactions per second with cheaper fees than their layer one without completely sacrificing security. A more complete description of rollups is provided in Section II.

The benefits of rollups have made them popular in blockchain ecosystems. At the time of writing there are 45 active rollups and 34 more in development according to the website L2Beat [8] which tracks these systems. As blockchain systems gain popularity, the amount of layer two rollups may stabilize, but rollups on rollups (i.e., *layer three* rollups) may become popular. This is in part because they can provide the benefits of layer two rollups but for specific state transition functions. In particular, dApps may choose to implement their own "app chain" which inherits security and performance from a suitable layer two, but is tailored to the specific dApp needs; i.e., the state transition will be one specific to the dApp. In such a ecosystem, rollups are likely to be engineered and re-engineered over and over again. Requirements Engineering (RE) practices must be in place to guide this development, just as it is in other specific domains (e.g., safety-critical molecular programs [9], scientific computing [10], and artificial intelligence [11], [12]).

In this work, we aim to highlight RE challenges based on our experience building a rollup (Zircuit [13]). These challenges arise from their security-critical nature (they secure millions of dollars of cryptocurrency), their place in an ecosystem as middleware, the complex domain-specific languages used for their implementation, their legal considerations, their reliability considerations, and their (eventual) decentralized operation. These challenges arise from both first hand experience in building a zero-knowledge rollup (see Section II-A) and from anticipating the future of building and maintaining these systems.

We aim to answer the following research questions within this work:

RQ1 What are the unique RE challenges for building a rollup network?
RQ2 How do rollup RE concerns differ from other web3 projects?
RQ3 What RE lessons are important for the community around rollups?

Contributions. In answering the research questions above, we provide two main contributions.

1) We provide an industrial report of RE challenges in building a zero-knowledge rollup. Our challenges are mapped to challenges present in existing RE literature (where possible), and we highlight that

DOI 10.1109/REW61692.2024.00052

this domain has unique challenges and development processes. These challenges should be overcome or addressed in future rollup development.

2) We suggest directions for future RE research in web3 systems. Observing that there is no common systematic approach to RE practices in web3 systems despite a need for community driven requirements validation in this field, we suggest directions for additional RE studies. We point out explicit areas where the existing RE literature fails to capture the unique concerns for building complex web3 systems.

This paper is structured as follows. Section II gives a high level description of rollups and their development process to provide context for the reader. Section III outlines the methodology of the work and Section IV reports on the RE challenges faced while building a rollup. Section V concludes the paper.

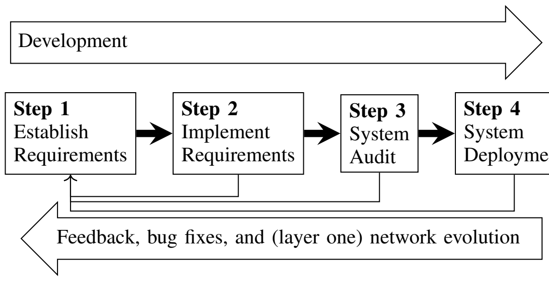

Fig. 1: Development steps in a rollup. Requirements are established and implemented, and audits are performed prior to deployment of the network. At any stage, feedback and evolution of related components can change the requirements (and therefore their implementation).

II. Preliminaries

In this section, we provide a concrete description of blockchain rollups (Section II-A) before outlining the rollup development process (Section II-B).

A. Rollups

A rollup can be broken down into several components[1]: a *sequencer*, a *state proposer*, and an (explicit or implicit) *verifier*. A sequencer is responsible for ordering layer two transactions and committing, via a transaction to layer one, to a batch of transactions to be executed. This batch is made up of layer two transactions. A state proposer executes the transactions in a batch (in the order provided by the sequencer's commitment) and computes new state roots which are written to layer one. Verifiers in a rollup ensure that state roots are (eventually) correct. Rollups come in two major types, which may change the responsibilities of these components: *optimistic* and *zero-knowledge*.

An *optimistic* rollup is one in which the state proposer is bonded and proposes new state roots. The state proposer is bonded in the sense that they stake some funds on layer one that are lost if they post an incorrect state root. A verifier in an optimistic rollup is an actor who submits a so-called *fraud proof* to challenge an incorrect state root proposed by the state proposer. In an optimistic rollup, state roots are considered correct unless a bonded verifier successfully challenges the state root with a fraud proof within some period of time (e.g., seven days). Such a verifier may need to propose an alternative state root than the one provided by the state proposer. Often, verifiers are part of the state proposer, and the combined entities are called *validators*. A verifier who successfully challenges the state proposer wins the state proposer's bond; those that lose the challenge give up their own stake to the state proposer. Arbitrum is an example of an optimistic rollup [15].

A *zero-knowledge* (ZK) rollup is one in which the state proposer generates cryptographic proofs that each new state root is correct. In a ZK rollup, the state proposer performs state transitions within a zero-knowledge proof framework (e.g., [16]), which generates a *validity proof*: an artifact that proves that a particular function was executed with particular inputs, which resulted in the new state. These validity proofs can be verified using layer one smart contracts (as a result, the verifier actor is implicit in such rollups). A state proposer for a ZK rollup may have one or more provers as a subcomponent, which generates the validity proofs given a batch of transactions and a previous state root. Note that the "zero-knowledge" aspect of these proofs are sometimes helpful for privacy (see also Section IV-D), but mostly these systems are used because the proofs are also *succinct*. This property enables the proofs to be verified in a fraction of the time required to run the computation in the first place, enabling verification directly on a layer one blockchain.

This document presents first-hand experience from developing Zircuit, a ZK rollup. Zircuit was built on top of the "OP Stack", an open-source public good, which implements the Optimism optimistic rollup [17]; Zircuit replaces the fraud proof system within that stack with a ZK proof system.

The *liveness* of a blockchain is a property where new blocks can always be added to it and transactions can be finalized; that is, the chain never halts. The liveness of a rollup is expected to match that of its layer one: as long as the layer one is not halted, the layer two should also be able to produce blocks and finalize transactions.

[1]Other work like [14], [7] use different terms for these components, but each rollup has some component that performs these actions.

B. Rollup Development

Figure 1 outlines the development process for a rollup. There are four main steps to development: requirements elicitation, requirement implementation, a system audit, and the deployment of the system. We describe these steps in more detail below.

Steps 1 and 2 are common. First, an initial set of requirements is elicited for the network (Step 1). In agile rollup development, this may not set the specifics of some requirements (e.g., the expected validity proof size) but will guide the core use cases and typically define the larger structure of the rollup. The requirements at this stage will likely dictate the virtual machine to be used on the rollup, the target layer one to build on, and the finality mechanism for the rollup. Next, the requirements are implemented (Step 2) and requirements may be added and others may be refined. These may arise because of technical difficulties (e.g., the proof system for the state transition cannot easily support a particular opcode or function) or because the requirements are not precise or complete.

Step 3, which may not be common outside of web3 system development, is an external code audit of the system. Although this is strictly optional, many web3 applications and networks will undergo this process. The audit aims to reduce the number of actual bugs in the system and may include a line-by-line review of the system (or parts of it) or a focused inspection of critical areas. Line-by-line reviews are the slowest and most expensive option, though they are the most comprehensive. To reduce these costs, only some system components may be reviewed in such a manner; often, these are the smart contracts. Smart contracts are typically immutable and public, meaning that errors in their implementation are difficult to fix. The smart contracts implementing a rollup (or any other dApp) can be placed behind a proxy contract [18] in order to effectively skirt the immutability of the code, but updates may still require a hard-fork [19] of the layer two network. In a centralized rollup, this may be technically challenging but possible; in a decentralized one, it may be difficult to get community buy-in for the suggested update. The auditors benefit greatly from a list of requirements for the network, but often have to infer them from the code and discussions with the developers. Issues found by the audit affect the requirements and in turn their implementation.

Finally, the system is deployed (Step 4). Often the system is first deployed as a layer two *test-net* on a layer one *test-net*; a *test-net* is a functionally equivalent version of the network that does not handle cyrptocurrencies of value. This is intended to provide a final check of the system, allow users to test the functionality of the network with their dApps, and to stress test the system. After some time, a *main-net* deployment occurs on a real layer one, and real funds (and risk) are present. Even after deployment requirements may need to change based on user feedback, identification of bugs, and the evolution of the underlying layer one. While most changes to the layer one are backwards compatible, layer two code is often a fork of layer one code (namely, execution clients), which may not provide backward compatible API in newer versions. In some cases, changes can be implemented immediately; in others, a network upgrade or fork may be required (see also Section IV-B).

III. Methodology

This research was performed by searching for publications that referenced the challenges we experienced first hand directly. We looked for these in top-rated conferences and journals via databases like the ACM Digital Library, Google Scholar, IEEE Xplore, and Science Direct. The focus of the search was related to "Requirements Engineering Challenges" and we added terms "Agile Development", "Blockchain", "Complex System", "Decentralized Autonomous Organizations" (DAOs) , "Distributed Ledger", "Open-Source Software" (OSS), and "Web3", along with related spellings. Initial screening removed repeated publications and those which pre-dated blockchain technology. Candidate publications were selected based on their relevance perceived on their title, keywords, and abstracts. Few publications were found to directly reference blockchain technology, and so we included those which were related to our perceived challenges (Section IV). Only a handful of publications captured our challenges precisely or referred to blockchain technologies in a meaningful capacity. No single publication entirely captured the RE challenges we faced developing a rollup.

IV. Challenges

There are a number of challenges inherit to specifying requirements for rollups. These challenges need to be addressed early in the development process in order to prevent costly changes later. The challenges arise from the fact that rollups are security-critical, serve as middleware, difficult to test, subject to legal considerations, complex online systems, and are OSS.

A. Rollups are Security-Critical

Rollups are non-trivial to develop but are expected to handle millions of dollars in cryptocurrencies. At the time of writing, L2Beat [8] reported that the cumulative value of cryptocurrencies in these systems is \$40.27 billion USD, with the Arbitrum rollup responsible for \$17.5 billion of that on its own. This makes them attractive targets for malicious actors (in the past, over \$2 billion USD was stolen from blockchain bridges [20]) and increases the risk of critical bugs. This also introduces a need for these systems to upgrade slowly, in order to prevent the operators from introducing new bugs or leveraging any of their privileged roles. Bugs may therefore persist for a while on-chain and in the open. These systems need to be open-source so that the community can verify which code is being executed, in order to avoid introducing new trust assumptions (see also Section IV-F).

Validating that a rollup supports necessary security features, cannot be upgraded quickly, and does not have privileged roles introduces RE-centric challenges.

First, it requires rollup developers to share their requirements. Unfortunately, most rollups do not explicitly report on their requirements. Validation of these requirements becomes a community challenge using only implied functionality and reviews of the code base. This presents the first challenge that must be overcome by both rollup developers and the community.

Challenge 1. There are no standards for presenting and tracing requirements through complex web3 systems for community members.

Second, there are no standards for rollup functionality, and as a result, there are no standard requirements. For example, L2Beat reports on the implementation of so-called *escape hatch* functionality of rollups, which aims to allow users to recover their assets even if the rollup is offline [14]. This functionality is challenging to implement, may be unique to each rollup, often requires compromises elsewhere, and is not standardized (indeed, there are no standards for nearly any aspect of a rollup). As a result, a rollup's set of requirements for these security-critical features may not satisfy everyone, or may dictate that a possible resolution requires expensive hardware (e.g., to generate validity proofs). A resolution to the following challenge would help to classify, compare, and understand these systems for both developers and end-users.

Challenge 2. There are no standard requirements for security-critical features. Unlike on-chain standards for tokens (like ERC-20 [3]), there are no standards for requirements of rollups, including for critical parts like bridges or escape hatches.

Both challenges can be overcome publicly sharing a rollup's requirements. Public requirements should be precisely stated and accompanied by a method to trace their implementation within the code base. These challenges are related to the "representation of requirements knowledge" challenge presented in [21]. However, while the proposed mechanisms within that paper focus on internal tools, ideal solutions are not limited to rollup developers and would also allow the community to validate requirements. This concern for public traceability is not emphasized in that work.

B. Rollups are Middleware

Rollups generally support smart contracts themselves, allowing dApps to build on them and even other rollups (i.e., layer three networks and beyond), which makes them middleware. This introduces the challenge of collecting requirements from other networks. Requirements are influenced by the choice of layer one, which in turn may add requirements to mimic many of its features, and other rollup networks.

The choice of layer one, often Ethereum, influences the requirements of a rollup. First, the smart contract capabilities of the layer one may have specific limitations (e.g., amount of so-called *gas* used to meter on-chain execution), which directly impacts engineering requirements of the rollup. For example, while there are many ZK proof systems, some may not have proofs small enough to be verified on-chain without running out of gas. Second, dApp developers often want to be able to port smart contract code onto a layer two with minimal friction. This is in part a result of dApp developers requiring costly smart contract audits (c.f. Figure 1 Step 3, which also applies to dApps), which are not necessarily accurate after updates to their code. To avoid re-audits, dApp developers prefer the ability to redeploy their smart contracts on a rollup which supports their code without additional changes. This is not always the case: the website rollup.codes [22] reports on changes the semantics of opcodes in rollup virtual machines. Most rollups on Ethereum aim to support the Ethereum Virtual Machine (EVM) to allow easy porting of Solidity [23] smart contracts popular on Ethereum itself, but subtle opcode changes may invalidate dApp developer requirements. Note that while EVM-compatibility is not true for all rollups – some rollups, like Starknet, use a different virtual machine altogether – it is common at the time of writing.

Other rollups may also influence the set of desired requirements. As middleware, rollups can expand their set of supported functionality, either through new opcodes or other requirements. Zircuit chose to add additional functionality to its sequencer in order to detect malicious transactions, while other rollups like Taiko [24] and Boba [25] have added additional opcodes to their versions of the EVM. As rollups experiment with useful middleware functionality, popular features will spread and be implemented by more and more rollups.

This results in the following challenge that will not be problematic for smaller web3 systems implemented only via smart contracts. The following challenge is also discussed in part by [26], when "a source of requirement[s] is features appearing in other software".

Challenge 3. Requirements are constantly being updated, sourced from related systems, and may be incompatible. Requirements need to be synchronized not only within development teams, but with external projects and systems.

This challenge is related to the "supporting change and evolution" area of [21]. The findings of [21] related to "synchronization of development" and "updating requirements" are partially relevant. However, the necessary synchronization was not an issue within teams of Zircuit but rather with open-source dependencies and forks. These dependencies allowed a good amount of re-use in components, but turned out to be problematic when components were tightly coupled and some needed customization. The findings that product-line engineering may be valuable to avoid re-specifying and encouraging re-use also in that paper are also of interest. This concern is unique to rollup development in the web3 space as it does not apply to smart contract based systems which can reuse libraries and standard implementations but do not need to worry about the communication between several services – most of their communication happens on-chain.

We remark that the product-line engineering felt as if it was out of scope of any particular rollup. For example, both Zircuit and Optimism use a fork of the Go Ethereum client (Geth) [27], but with different requirements. Ideally the customization required for both of these rollups (which are not the same) would be supported in a software product-line fashion by Geth itself. Given that the Geth team likely has other concerns, this may not be feasible alone, but a collaboration between Geth and various rollup teams could simplify and unify development in this space. This suggests a wider open-source RE challenge which may be solved via public grants and other incentives between cooperating teams. This also relates to Challenges 1 and 2, as such a product-line will enable consistent evolution of requirements and a core set of requirements common to all cooperating rollups.

C. Rollups are Difficult to Test

Rollups are very difficult, if not impossible, to thoroughly test. The number of paths that they should be able to execute is large, and ZK rollups are even more problematic. This is because ZK rollups transform the state transition function of the rollup (i.e., the block creation function) into a mathematical framework that generates a proof that the function was performed correctly, as well as the output of the function (e.g., the layer two block). These functions are encoded in a Domain-Specific Language (DSL) which compiles the function into a set of constraints that hold if and only if the computation is performed correctly. The number of constraints in the transition function for a blockchain may be very large (e.g., 2^{26}) and are therefore impossible to manually check. This leads to RE challenges related to requirements validation and verification, and can make some requirements difficult to implement.

First, it can be impossible to verify ZK constraints directly. Instead, users of the ZK systems can merely test these constraints or determine if the constraint compiler is trustworthy. The former approach can miss important bugs resulting from under constrained systems (see e.g. [28]), while the latter may also be difficult or require specialized knowledge. Not every rollup uses the same ZK proof system (and some do not any such system at all), and as a result, it may be some time before the community agrees on trustworthy compilers and ZK software stacks. This uncertainty makes it difficult to verify requirements and gives rise to the following challenge.

Challenge 4. Manual verification of requirements may be impossible and must rely on tooling.

Second, the use of a ZK proof system can complicate implementations of (non-functional) requirements, leading to their changes. Requirements related to system performance can be particularly complicated to implement, and may require specialized hardware (namely, GPUs), in order to be feasible. However, specialized hardware may not always be available or may be too costly to operate. This can lead to concessions in terms of proof generation performance, proof size, or reliability (see also Section IV-E). In some cases, requirements like validity proof size are not flexible and are dictated by the choice of layer one complicating things further. These complications are more common for non-functional requirements regarding performance.

Challenge 5. Non-functional requirements are hard to specify ahead of time, but system changes are costly.

These challenges fit in part to those related to building and maintaining shared understanding of the system presented in [21]. Creating and maintaining traces for requirements becomes paramount when they must be implemented using a DSL which is translated to something which is not human-readable. Challenge 4 is therefore unique to rollup development within the web3 space as such DSLs are not common for strictly on-chain dApps. Similarly, the creation of documentation to complement tests is important to convey to both developers and the community that concerning issues have been addressed during development. Challenge 5 is unique to rollups as on-chain dApps do not need to worry about many common concerns, like those related to performance or scalability, because the chain running the dApp forces their outcome.

D. Rollups may be Subject to Legal Considerations

Rollups which introduce privacy may have to consider the implication of their actions in the context of legal policy and guidelines. Although most rollups aim to simply provide a better user experience to blockchain users through higher transaction throughput and lower fees, some (e.g., Aztec [29]) aim to add functionality including privacy. This can be achieved by using ZK proof systems for the original purpose: arguments of knowledge. Proof systems used in this regard, rather than simply because they provide succinct validity proofs, can hide transaction details and result in private transactions. There may be other legal considerations as well, like those related to Know-Your-Customer (KYC) concerns or related to the jurisdiction where the system is being operated. Tackling requirements from these sources is very difficult, as the laws and guidelines are relatively new and may be inconsistent across jurisdictions. Many projects aim to steer clear of these considerations altogether, in part because there are no development recommendations for these systems. The main challenge here is to handle legal uncertainty for this relatively new technology.

Challenge 6. Requirements arising from legal considerations are not well outlined by government agencies.

To overcome this challenge, it would be beneficial to compile set of recommendations, perhaps similar to that of [30], for ZK and privacy-preserving technologies. However, we worry this may be premature as many entities may not have an established stance on these topics.

E. Rollups are Complex, Online Systems

There are operational considerations that need to be taken into account in order to satisfy rollup liveness requirements. There include running off-chain rollup components with five-nines up time, especially as they are immature and centralized. We have not found any team that shares their requirements for these operational considerations, even though they are important to the success of a project. This may be because they are rollup specific, but as rollups also move towards *shared sequencing* (multiple rollups using the same sequencer), there is a need to share these non-functional requirements. Just as an Machine Learning (ML) system is composed of more than just a ML library, a rollup is more than a single blockchain node [31]. These challenges are related to building and maintaining shared understanding of the system in [21]; namely, "system vs. component thinking". There is a need to think beyond the key differentiators of rollups (i.e., fraud proofs or validity proofs), and instead to make sure that all parts of the system operate correctly – even those that are less public or innovative than others.

Challenge 7. Rollup research and development overlooks requirements for user experience, infrastructure, and interoperability concerns.

Again, these concerns are not present for entirely on-chain dApps, which can always rely on the underlying blockchain for liveness and accessibility.

F. Decentralized Rollups are Open-Source Software

Rollups that move past a point of centralized operations – that is, have decentralized components that can be run by anyone in the world – must be OSS, in order to ensure users know what they are running. Previous work (like [26]) has studied RE challenges for OSS, and it is likely that these challenges will persist in this domain.

Current rollup development is more or less operating using the "cathedral" model to deploy and release code [32]. As a result, the work on issues arising from when "requirements are asserted by developers" in [26]. As code is open-sourced and these systems mature, a model borrowing from the bazaar is starting to appear at a high level. Projects are taking established code bases and changing core features (e.g., using a ZK proof system instead of a fraud proof system), but these just result in different rollups. That is, it is resulting in more "cathedrals" rather than resulting in changes to the original codebases.

Challenge 8. Community-sourced requirements are difficult to incorporate into existing systems as blockchains must evolve slowly.

This may be common to other dApps which aim to fork libraries and standard implementations. However, as on-chain dApps are more or less immutable, the suggestion and incorporation of new requirements is relatively unique to rollups within the space, except when on-chain dApps provide entirely new versions to satisfy these requirements.

When requirements are communicated, they often arise from informal discussions. We have found discourse on rollup requirements via Twitter (now X) threads, Telegram chats between teams, and Discord groups. RE concerns are discussed when "requirements exist informally as part of communication messages (e.g., emails, discussion forums)" in [26]. Rollups may be the primary method for scaling the Ethereum ecosystem, and there is no expectation of a single system winning out or being "best". The following challenge is therefore not unique to complex OSS, but is unique to rollups within web3 OSS.

Challenge 9. Requirements may be imprecise and spread among many sources.

It is our hope that, along with suitable resolutions to Challenges 2, 3, and 7, that a standard for rollup requirements is developed and used by the community.

Finally, challenges related to the OSS nature of rollups may also be subject to challenges that arise from protocol governance lead by a Decentralized Autonomous Organization (DAO). DAOs are a concept unique to web3 and operate protocols, produce software, or govern communities (see e.g., [33]). There are preliminary studies of the interactions of DAOs and OSS projects [34], but we anticipate that this area of web3 will continue to evolve. Potential challenges related to DAO-based governance of rollups will likely relate to Challenge 9 and will need to be considered in advance of such proliferation to ensure user funds, and trust in the ecosystem, are not at risk.

V. Conclusion

In this work, we have reported on the RE challenges of building a ZK rollup. We have demonstrated that these systems are complex and should benefit from existing RE frameworks, but are also unique and require their own considerations. We have answered RQ1 by listing 9 explicit RE-centric challenges facing rollup developers (and their users). In each case, we commented on the applicability of these challenges to other web3 dApps, answering RQ2 in the process. As an answer to RQ3, we note that the lessons of related systems should not be overlooked but are not necessarily taken to heart in practice. Furthermore, as also pointed out by [35] for blockchains in general, there are no systematic studies of RE concerns for rollups or web3 systems in general. RE-centric studies which aim to confirm the challenges in this paper, possibly through developer interviewers, will undoubtedly surface additional concerns that should also be addressed.

This work should not be considered complete: as with all parts of web3, these systems will continue to evolve over time. Moreover, we were not able to provide a comprehensive review of our challenges in light of the current RE literature, and we leave this for future work. For example, there are other challenge areas of [21] that did not fit into the unique areas highlighted in Section IV (e.g., the challenge of expressing value in user stories that can be completed in a single sprint, persisted throughout this system's design). These may not be unique concerns but should still be discussed in order to provide strong RE recommendations and frameworks for these systems. These recommendations should keep the web3 community in mind, and suggest methods by which the community can easily trace requirements within complex systems. The complexity of these systems means that this area (and web3 systems more generally) warrants further study for RE researchers. We hope that this work provides context for, and guides research directions of, future studies.

References

[1] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014. Ethereum Project Yellow Paper, https://ethereum.github.io/yellowpaper/paper.pdf.

[2] Nick Szabo. Smart contracts: Building blocks for digital markets. 1996.

[3] Fabian Vogelsteller and Vitalik Buterin. ERC-20: Token standard, Nov 2015. Available at https://eips.ethereum.org/EIPS/eip-20. Accessed 8 December 2023.

[4] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. SoK: Layer-two blockchain protocols. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*, volume 12059 of *Lecture Notes in Computer Science*, pages 201–226. Springer, 2020.

[5] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J. Knottenbelt. SoK: Communication across distributed ledgers. In Nikita Borisov and Claudia Díaz, editors, *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II*, volume 12675 of *Lecture Notes in Computer Science*, pages 3–36. Springer, 2021.

[6] Rami Khalil, Alexei Zamyatin, Guillaume Felley, Pedro Moreno-Sanchez, and Arthur Gervais. Commit-chains: Secure, scalable off-chain payments. Cryptology ePrint Archive, Paper 2018/642, 2018. https://eprint.iacr.org/2018/642.

[7] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. SoK: Validating bridges as a scaling solution for blockchains. Cryptology ePrint Archive, Paper 2021/1589, 2021. https://eprint.iacr.org/2021/1589.

[8] Bartek Kiepuszewski. L2bridge risk framework, 2022. https://gov.l2beat.com/t/l2bridge-risk-framework/ Accessed 2 Feb. 2024.

[9] Robyn R. Lutz. Requirements engineering for safety-critical molecular programs. In *30th IEEE International Requirements Engineering Conference, RE 2022, Melbourne, Australia, August 15-19, 2022*, pages 302–308. IEEE, 2022.

[10] Yang Li, Nitesh Narayan, Jonas Helming, and Maximilian Koegel. A domain specific requirements model for scientific computing. In Richard N. Taylor, Harald C. Gall, and Nenad Medvidovic, editors, *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, May 21-28, 2011*, pages 848–851. ACM, 2011.

[11] Xavier Franch, Andreas Jedlitschka, and Silverio Martínez-Fernández. A requirements engineering perspective to AI-based systems development: A vision paper. In Alessio Ferrari and Birgit Penzenstadler, editors, *Requirements Engineering: Foundation for Software Quality - 29th International Working Conference, REFSQ 2023, Barcelona, Spain, April 17-20, 2023, Proceedings*, volume 13975 of *Lecture Notes in Computer Science*, pages 223–232. Springer, 2023.

[12] Khan Mohammad Habibullah, Gregory Gay, and Jennifer Horkoff. Non-functional requirements for machine learning: understanding current use and challenges among practitioners. *Requir. Eng.*, 28(2):283–316, 2023.

[13] Zircuit. www.zircuit.com/. Accessed 2 Feb. 2024.

[14] Jan Gorzny, Po-An Lin, and Martin Derka. Ideal properties of rollup escape hatches. In Kaiwen Zhang, Abdelouahed Gherbi, and Paolo Bellavista, editors, *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good, DICG 2022, Quebec, Quebec City, Canada, 7 November 2022*, pages 7–12. ACM, 2022.

[15] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1353–1370. USENIX Association, 2018.

[16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[17] Optimism. www.optimism.io/. Accessed 2 Feb. 2024.

[18] William Edward Bodell III, Sajad Meisami, and Yue Duan. Proxy hunting: Understanding and characterizing proxy-based upgradeable smart contracts in blockchains. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 1829–1846. USENIX Association, 2023.

[19] Richard Ma, Jan Gorzny, Edward Zulkoski, Kacper Bak, and Olga V Mack. *Fundamentals of Smart Contract Security*. Momentum Press, 2019.

[20] Sung-Shine Lee, Alexandr Murashkin, Martin Derka, and Jan Gorzny. SoK: Not quite water under the bridge: Review of cross-chain bridge hacks. In *IEEE International Conference on Blockchain and Cryptocurrency, ICBC 2023, Dubai, United Arab Emirates, May 1-5, 2023*, pages 1–14. IEEE, 2023.

[21] Rashidah Kasauli, Eric Knauss, Jennifer Horkoff, Grischa Liebel, and Francisco Gomes de Oliveira Neto. Requirements engineering challenges and practices in large-scale agile system development. *J. Syst. Softw.*, 172:110851, 2021.

[22] RollupCodes. www.rollup.codes/. Accessed 2 Feb. 2024.

[23] Solidity. https://docs.soliditylang.org/en/v0.8.23/ Accessed 2 Feb. 2024.

[24] Taiko. taiko.xyz/. Accessed 2 Feb. 2024.

[25] Boba network. https://boba.network/. Accessed 26 Mar. 2024.

[26] Jaison Kuriakose and Jeffrey Parsons. How do open source software (OSS) developers practice and perceive requirements engineering? An empirical study. In *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 49–56, 2015.

[27] Go ethereum. https://geth.ethereum.org/ Accessed 2 Feb. 2024.

[28] Fatemeh Heidari Soureshjani, Mathias Hall-Andersen, MohammadMahdi Jahanara, Jeffrey Kam, Jan Gorzny, and Mohsen Ahmadvand. Automated analysis of Halo2 circuits. In Stéphane Graham-Lengrand and Mathias Preiner, editors, *Proceedings of the 21st International Workshop on Satisfiability Modulo Theories (SMT 2023) co-located with the 29th International Conference on Automated Deduction (CADE 2023), Rome, Italy, July, 5-6, 2023*, volume 3429 of *CEUR Workshop Proceedings*, pages 3–17. CEUR-WS.org, 2023.

[29] Aztec Network. https://aztec.network/. Accessed 2 Feb. 2024.

[30] Krishna Ronanki, Beatriz Cabrero Daniel, Jennifer Horkoff, and Christian Berger. RE-centric recommendations for the development of trustworthy(er) autonomous systems. In *Proceedings of the First International Symposium on Trustworthy Autonomous Systems, TAS 2023, Edinburgh, United Kingdom, July 11-12, 2023*, pages 1:1–1:8. ACM, 2023.

[31] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2503–2511, 2015.

[32] Eric S. Raymond. *The cathedral and the bazaar - musings on Linux and Open Source by an accidental revolutionary*. O'Reilly, 1999.

[33] Tanusree Sharma, Yujin Kwon, Kornrapat Pongmala, Henry Wang, Andrew Miller, Dawn Song, and Yang Wang. Unpacking how decentralized autonomous organizations (DAOs) work in practice. *CoRR*, abs/2304.09822, 2023.

[34] Paul van Vulpen, Jozef Siu, and Slinger Jansen. Governance of decentralized autonomous organizations that produce open source software. *Blockchain: Research and Applications*, 5(1):100166, 2024.

[35] Muhammad Shoaib Farooq, Mishaal Ahmed, and Muhammad Emran. A survey on blockchain acquainted software requirements engineering: Model, opportunities, challenges, and future directions. *IEEE Access*, 10:48193–48228, 2022.