Verifying and Displaying Move Smart Contract Source Code for the Sui Blockchain

Rijnard van Tonder MystenLabs, Inc. USA rijnard@mystenlabs.com

ABSTRACT

Smart contract development presents additional challenges beyond traditional software workflows, e.g., locally in IDEs. For smart contract developers to understand and trust code execution, they need to write and use software libraries with a comprehensible code representation-i.e., source code. However, blockchains do not typically store the original source code of smart contracts, but a condensed bytecode representation. Thus, when developers consult smart contract source code, they need to be sure that it corresponds to the same bytecode on the blockchain. Depending on available developer tools, this process can be ad-hoc, cumbersome, or opaque. In this paper we present our design and implementation of a new tool that serves to verify Move smart contract source code against its bytecode representation on the Sui blockchain. We demonstrate the user-facing shift where developers now benefit from seeing source code in their browser instead of bytecode. We further highlight future features and research directions that verified source availability brings to smart contract developer experience.

KEYWORDS

smart contracts, source code, bytecode, compilers, program comprehension, software development, blockchain

ACM Reference Format:

Rijnard van Tonder. 2024. Verifying and Displaying Move Smart Contract Source Code for the Sui Blockchain. In 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3639478.3640038

1 INTRODUCTION

Smart contract development presents additional challenges beyond traditional software workflows, e.g., locally in IDEs. The runtime execution and output of a smart contract is wholly dependent on a distributed system that ensures the integrity of a blockchain. For smart contract developers to understand and trust code execution, they need to write and use software libraries with a comprehensible code representation—i.e., source code. However, blockchains do not typically store the original source code of smart contracts, but a

ICSE-Companion '24, April 14-20, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 979-8-4007-0502-1/24/04...\$15.00 https://doi.org/10.1145/3639478.3640038 condensed bytecode representation. This bytecode representation strips away useful information in typical software development, like comments, docstrings, and syntax that help explain the implementation.

Just as in usual software development, smart contract developers generally need and depend on a source representation to write a smart contract. The key difference is that developers must trust that the source they are developing against corresponds to the bytecode that will be linked against their code on the blockchain. In general, a combination of trust and tooling can help ensure that such source code really is the same as the bytecode contract that is executable on the blockchain. For example, when developers decide to publish their smart contract on a blockchain, a tool could first check the integrity of the dependencies. The tool could compile every source dependency into bytecode that the developer depends on, and then check that the local output of bytecode matches the existing on-chain bytecode (that the yet-to-be-published smart contract will depend on once published). When the source code output corresponds to the expected on-chain bytecode, we say it is "verified".

We've identified that developers on the Sui blockchain¹ would largely benefit from being able to see verified Move source code online. Indeed, existing blockchains have varying support or pending requests for other languages [3, 6, 7]. However, every smart contract language poses unique challenges to providing such a service, and the constraints of a given blockchain and language will impose on the feasibility, design, and implementation of a system to provide such verified source code. This paper describes our exploration, implementation, and result of designing a source verification service for the Move language. While providing immediate value to developers, we also consider the service to be a fundamental building block in unlocking greater developer user experience. For example, source availability in the browser could allow for communicating software quality signals [11] and code navigation via the Language Server Protocol [4].

2 MOVE SOURCE VERIFICATION TOOL AND SERVICE

Overview. The Move Source Verification Service (MSVS) serves Move source code that has been verified against its on-chain bytecode representation. The service is powered by a tool implemented in Rust, comprising:

- a backend server to clone source code, orchestrate verification, and serve source code over a REST API.
- a source-to-bytecode verification routine in a standalone library (Rust crate).

¹sui.io

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.



Figure 1: The pipeline illustrating how we verify smart contract code with MSVS. First, the reference source implementation is pulled and compiled (1). The expected bytecode for the source is then pulled from the blockchain (2) and compared byte-for-byte to the compiled source code (3). If the process succeeds, the source is displayed in the Sui Explorer frontend (4).

The implementation is open-source under MIT license.² A hosted version of the tool³ integrates with suiexplorer.com, a web frontend for exploring Sui blockchain data and code (see Figure 3 for a visual example). The MSVS implementation is further independently configurable and runnable (e.g., for developers who want to serve their own source packages, or run the tool internally), and can be integrated with alternative frontends beyond suiexplorer.com. Hardware requirements are generally low, and depends on the scale and number of source packages hosted. For example, we've deployed and tested the server on a single core machine with less than 512MB of RAM over a handful of standard Move library packages.

Tool and Service Operation. Figure 1 illustrates the pipeline for the Move Source Verification Service. The service 3 represents the tool presented in this paper: a server that orchestrates and implements source verification to enable an end-to-end flow. At start up, the server clones well-known source repositories containing Move packages ①. The set of source repositories are specified in a configuration file, and target fixed branches in repositories. The configuration may be modified as needed.⁴ Source packages are expected to specify the on-chain address where the bytecode is published in a Move.toml file. The server uses this address to fetch the corresponding bytecode from the blockchain 2. The server builds the source code for each package and compares the output byte-for-byte to the expected bytecode found on-chain 3. If this comparison succeeds, we say that the source code is verified. The server will then respond to API requests for source code from a frontend client (in this case, Sui Explorer) which are rendered to users in the browser 4.

Cloning and verification is generally performed concurrently per repository and package. In its first instantiation, the server tracks four packages that roughly correspond to standard libraries,⁵ comprising roughly 18,000 lines of Move code. Verifying these packages takes around 8 seconds on our hosted service. The central operation of the service is to effectively keep source code in lockstep with on-chain bytecode. One additional challenge in the Sui blockchain (and other blockchains supporting similar operations) is the notion of smart contract upgrades. On the Sui blockchain packages may be upgradable⁶ and our MSVS must account for such changes to accurately report source. The server currently implements threads that monitor changes to the fixed addresses mentioned prior on the Sui blockchain. When a thread detects an on-chain upgrade (i.e., the bytecode changes on the blockchain), the corresponding package source code is immediately invalidated on the server, so that it no longer reports that source when requested until the new code can be reverified. The server then pulls the source code again and repeats the verification process, which typically takes less than 10 seconds.

Note that we do not derive (or decompile) the source code from the on-chain bytecode. Our service expects a repository containing a reference implementation to verify against. While decompiling is an attractive option when source code is unavailable, available source containing comments and code formatting will remain the preference of active developers. At present, a high quality decompiler for Move code does not yet exist.

3 DEVELOPER EFFORT, USER EXPERIENCE, AND OPEN QUESTIONS

Increasing Developer Experience with User-facing Changes The before-and-after of our tool is illustrated by Figures 2 and 3, and the main outcome of our approach. Previously, we displayed only the Bytecode tab for modules (shown left). The Bytecode representation shows flat addresses when referencing other modules (e.g., $0 \times 00..001: ascii$). The corresponding Source representation in Figure 3 reveals the namespaces and provides greater clarity (e.g., std::ascii instead). The source also reveals essential constants (which may be inlined during compilation) and documentation for types and functions. Users can now largely benefit from source code

 $^{^2}https://github.com/MystenLabs/sui/tree/main/crates/sui-source-validation-service <math display="inline">^3API$ hosted at source.mystenlabs.com

⁴Click here to see the default configuration file online: configuration file online.
⁵Find the packages online by clicking here.

⁶https://docs.sui.io/build/package-upgrades

Verifying and Displaying Move Smart Contract Source Code for the Sui Blockchain

ICSE-Companion '24, April 14-20, 2024, Lisbon, Portugal

odules		STACKED SIDE-BY-SIDE
Search	Bytecode Source Verified	Execute
address	1 // Move bytecode v6	burn >
authenticator_state	2 module 2.coln { 3 use 00000000000000000000000000000000000	
bag	4 use 00000000000000000000000000000000000	join >
balance	6 use 00000000000000000000000000000000000	mint and transfer
	7 use 00000000000000000000000000000000000	mint_and_transfer >
bcs	<pre>9 use 00000000000000000000000000000000000</pre>	and the standard and
bls12381	10 use 0000000000000000000000000000000000	update_description
borrow	11 use 00000000000000000000000000000000000	undate icon url
borrow	13	update_icon_un
clock	14 struct Coin <phantom ty0=""> has store, key {</phantom>	undata nama
coin	16 balance <ty0></ty0>	update_name >
	17 }	undete envelopt
display	18 struct CoinMetadata <phantom ty0=""> has store, key {</phantom>	update_symbol >
dynamic_field	20 decimals: u8,	
dynamic_object_field	21 name: String,	
	22 description: String,	
ecdsa_k1	24 icon url: Ontions[[r]>	

Figure 2: The bytecode representation of a Move smart contract on suiexplorer.com (cf. Move source code in Figure 3).

Starte 0 Starte 0 Starte 0 Starte 0 address address <th></th> <th></th> <th></th>			
Search Bytecode Source Verified address authenticator_state bag bag bag balance 6 6 7 module suit:coin { 9 use std::string; 9 9 0 9 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 1 1 1 1 1 1 1 1 1 11 11 12 13 14 15 16 17 17 18 19 19 11 11 11 12 13 14 15 16 17 17 18 19 </th <th>odules</th> <th></th> <th>STACKED SIDE-BY-SIDE</th>	odules		STACKED SIDE-BY-SIDE
address 1 // Copyright (c) Mysten Labs, Inc. burn authenticator_state 2 // SPDX-License=Identifier: Apache-2.0 authenticator_state 3 bag 5 /// tokens and coins. 'Coin' can be described as a secure wrapper around 6 /// 'Balance' type. module sui::coin { base cos 9 use std::string; 9 use std::string; borrow 11 use sui::toin { 12 use sui::toin:{Self, Option}; 12 use sui::tobject::{Self, Balance, Supply}; 13 use sui::tobject::{Self, UID); tock 14 use sui::trix_context;; 15 use std::vector; 16 17 toppamic_field 20 /// Trying to split a coin more times than its balance allows. costs_k1 21 cost ENotEnough: u64 = 2; 22 /// A coin of type 'T' worth 'value'. Transferable and storable	Search Q	Bytecode Source Verified	Execute
ubuthenticator_state 2 // SPDX-License-Identifier: Apache-2.0 a 4 /// Defines the 'Coin' type - platform wide representation of fungible bag 5 /// tokens and coins. 'Coin' can be described as a secure wrapper around 6 /// 'Balance' type. module sui::coin { rmodule sui::coin { mint_and_transfer ocs 9 use std::soti; obs12381 10 use sui::balance::{Self, 0ption}; 11 use sui::balance::{Self, Balance, Supply}; update_description idock 14 use sui::transfer; use sui::balance::{Self, UTI}; update_icon_url > idock 14 use sui::transfer; update_icon_url > 15 use sui::unt::(Self, UTI); idock 14 use sui::transfer; update_name > 16 isplay 18 /// A type passed to create_supply is not a one-time witness. 19 const EBadWitness: u64 = 0; /// Invalid arguments are passed to a function. tynamic_field 20 /// Invalid arguments are passed to a function. itedsa_k1 24 itedsa_r1 25 /// A coin of type 'T worth 'value'. Transferable and storable itedsa_r1 26 struct Coin-ophantom T> has key, store {	address	1 // Copyright (c) Mysten Labs, Inc.	burn >
bag 4 /// Defines the `Coin` type - platform wide representation of fungible join bag 5 /// tokens and coins. `Coin` can be described as a secure wrapper around imit_and_transfer balance module sul::coin { mint_and_transfer base 9 use std::string; update_description base 9 use std::scli; update_description base 11 use sul::balance: {Self, 0ption}; update_description base 11 use sul::cointance: {Self, 0ption}; update_description base 11 use sul::cointance; {Self, 0ption}; update_description base 11 use sul::chance; {Self, UID}; update_icon_url > back 14 use sul::chance; update_icon_url > back 14 use sul::chance; update_suppol > back 14 use sul::chance; update_one back 14 use sul::chance; update_one back 14 use sul::chance allows. update_suppol > chock 14 use sul::chance allows. update_suppol > thynamic_field 17 try pantic applanti a c	authenticator_state	2 // SPDX-License-Identifier: Apache-2.0 3	
balance 5 /// tokens and tother, type. palance 7 module sui::coin { base 7 module sui::coin { base 9 use std::string; 9 use std::soption::{Self, Option}; base 9 use std::option::{Self, Option}; base 10 use sui::balance:.{Self, Balance, Supply}; base 11 use sui::txContext; base 12 use sui::txContext; base 13 use sui::object::{Self, Ulb}; base 15 use sui::transfer; 16 use std::vector; 17 17 16 use std::vector; 17 17 18 /// A type passed to create_supply is not a one-time witness. 17 17 16 use std::vector; 17 17 16 18 17 19 const EEnadwitness: u64 = 0; 17 19 const EEnadwitness: u64 = 1; 17 10 const EEnadwitness: u64 = 1; 17 10 const EEnadwitness: u64 = 2; 18 /// A type passed to a function. 19 const EEnvalidArg: u64 = 1; 10 10 const EEnvalidArg: u64 = 2;	baq	4 /// Defines the `Coin` type – platform wide representation of fungible	join >
<pre>palance</pre>		5 /// tokens and coins. Coin can be described as a secure wrapper around 6 /// `Balance` type,	
bcs 8 use std::string; 9 use std::option; obs12381 10 use std::option::{Self, 0ption}; update_description borrow 12 use sui::balance::{Self, Balance, Supply}; update_icon_url > borrow 12 use sui::tx_context::TXContext; update_icon_url > borrow 13 use sui::transfer; update_icon_url > borrow 15 use sui::url::{Self, UTD}; update_name > coin 16 use std::vector; update_symbol > 17 18 /// A type passed to create_supply is not a one-time witness. update_symbol > dynamic_field 20 //// Invalid arguments are passed to a function. update_symbol > decdsa_k1 23 const ElavalidArg: u64 = 1; /// Trying o split a coin more times than its balance allows. update_symbol > secdsa_k1 24 /// A coin of type 'T' worth 'value'. Transferable and storable update_storable	palance	7 module sui::coin {	mint_and_transfer >
obsection 9 use std::soci; update_description obs12381 10 use std::socin update_description borrow 12 use sui::tx_context::TxContext; update_icon_url > 3 use sui::transfer; update_icon_url > clock 14 use sui::url::{Self, UTD}; update_name > clock 15 use sui::url::{Self, UT}; update_name > coin 16 use std::vector; update_symbol > 77 77 18 /// A type passed to create_supply is not a one-time witness. update_symbol > dynamic_field 20 //// Invalid arguments are passed to a function. update_symbol > dynamic_object_field 21 const ElnvalidArg: u64 = 1; /// Trying to split a coin more times than its balance allows. update_symbol > secdsa_k1 24 /// A coin of type 'T' worth 'value'. Transferable and storable update_storable	ocs	8 use std::string;	
<pre>bis12381 use suf::balance::{Self, Balance, Supply}; uupdate_icon_url > 11 use suf::balance::{Self, Balance, Supply}; borrow 12 use suf::tx_context::TxContext; 13 use suf::tx_context::TxContext; 14 use suf::transfer; 15 use suf::url::{Self, Url}; coin 16 use std::vector; 17 display 18 /// A type passed to create_supply is not a one-time witness. 17 dynamic_field 20 /// Invalid arguments are passed to a function. 21 const EladWitness: u64 = 0; 40namic_object_field 22 /// Trying to split a coin more times than its balance allows. 23 const ENotEnough: u64 = 2; 24 eecdsa_r1 26 struct Coin 26 struct Coin 27 // A coin of type 'T' worth 'value'. Transferable and storable 26 struct Coin 27 // A coin of type 'T' worth 'value'. Transferable and storable 26 struct Coin 27 // A coin of type 'T' worth 'value'. Transferable and storable 26 struct Coin 27 // A coin of type 'T' worth 'value'. Transferable and storable 26 struct Coin 27 // A coin of type 'T' worth 'value'. Transferable and storable 26 struct Coin 27 // A coin of type 'T' worth 'value'. Transferable and storable 27 // A coin of type 'T' worth 'value'. Transferable and storable 28 /// A coin of type 'T' worth 'value'. Transferable and storable 29 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 20 /// A coin of type 'T' worth 'value'. Transferable and storable 21 // A coin of type 'T' worth 'value'. Transferable and storable 22 /// A coin of type 'T' worth 'value'. Transferable and storable 23 /// A coin of type 'T' worth 'value'. Transferable and storable 24 /// A coin of type 'T' worth 'value'. Transferable and st</br></pre>	1.10001	9 Use std::ascil; 10 Use std::ontion::{Self. Ontion}:	update_description
borrow 12 use sui::tx_context::TxContext; update_icon_url > 13 use sui::tx_context::Self, UID); update_icon_url > 14 use sui::transfer; update_icon_url > 15 use sui::url::Self, UID); update_icon_url > 16 use sui::url::Self, Url}; update_symbol > 16 use std::vector; update_symbol > 17 17 update_symbol > 18 /// A type passed to create_supply is not a one-time witness. update_symbol > 19 const EBadWitness: u64 = 0; update_symbol > dynamic_field 20 /// Invalid arguments are passed to a function. update_symbol > dynamic_object_field 21 const EInvalidArg: u64 = 1; update_symbol > ecdsa_k1 23 const ENotEnough: u64 = 2; /// A coin of type 'T' worth 'value'. Transferable and storable ecdsa_r1 26 struct Coin <phantom t=""> has key, store { update_storable</phantom>	51512381	<pre>use sui::balance::{Self, Balance, Supply};</pre>	
13 use sui::object::{Self, UID}; 14 use sui::transfer; 15 use sui::url::{Self, UID}; 16 use sui::url::{Self, UID}; 16 use std::vector; 17 16 18 // A type passed to create_supply is not a one-time witness. 19 const EBadWitness: u64 = 0; dynamic_field 20 /// Invalid arguments are passed to a function. const EInvalidArg: u64 = 1; const EInvalidArg: u64 = 1; dynamic_object_field 22 /// Trying to split a coin more times than its balance allows. ecdsa_k1 24 ecdsa_r1 26 struct Coin <phantom t=""> has key, store {</phantom>	worrow	<pre>12 use sui::tx_context::TxContext;</pre>	update_icon_url >
clock 14 use suf::transfer; update_name isplay 15 use std::vector; update_symbol display 18 /// A type passed to create_supply is not a one-time witness. update_symbol dynamic_field 20 /// Invalid arguments are passed to a function. update_symbol dynamic_object_field 21 const EInvalidArg: u64 = 1; update_stance allows. eedsa_k1 23 const ENotEnough: u64 = 2; /// A toin of type 'T' worth 'value'. Transferable and storable eedsa_r1 26 struct Coin <phantom t=""> has key, store { struct Coin<phantom t=""> has key.</phantom></phantom>		<pre>13 use sui::object::{Self, UID};</pre>	
ise statility (if get), if get), if get), if get), if get), if get), if get) if get set statility (if get), if	clock	14 Use sul::transter;	undato namo
display 17 17 display 18 /// A type passed to create_supply is not a one-time witness. update_symbol > dynamic_field 19 const EBadWitness: u64 = 0; update_symbol > dynamic_object_field 20 /// Invalid arguments are passed to a function. update_symbol > dynamic_object_field 22 /// Trying to split a coin more times than its balance allows. update_symbol > excdsa_k1 23 const ENotEnough: u64 = 2; update_symbol > excdsa_r1 25 /// A coin of type 'T' worth `value`. Transferable and storable update_symbol >	oin	16 use std::vector:	upuate_name
display 18 /// A type passed to create_supply is not a one-time witness. 19 const EBadWitness: u64 = 0; 20 /// Invalid arguments are passed to a function. 21 const EInvalidArg: u64 = 1; 22 /// Trying to split a coin more times than its balance allows. 23 const ENotEnough: u64 = 2; 24 24 25 26 struct Coin <phantom t=""> has key, store {</phantom>		17	
dynamic_field 19 const EBadWitness: u64 = 0; 20 /// Invalid arguments are passed to a function. dynamic_object_field 21 const EInvalidArg: u64 = 1; 22 /// Trying to split a coin more times than its balance allows. acdsa_k1 23 const ENotEnough: u64 = 2; acdsa_r1 24 /// A coin of type `T` worth `value`. Transferable and storable	display	18 /// A type passed to create_supply is not a one-time witness.	update_symbol >
Aynamic_field 20 /// Invalid arguments are passed to a function. Aynamic_object_field 21 const EInvalidArg: u64 = 1; Aynamic_object_field 22 /// Trying to split a coin more times than its balance allows. Const ENotEnough: u64 = 2; 23 const ENotEnough: u64 = 2; Ardsa_r1 26 51// A coin of type 'T' worth 'value'. Transferable and storable 26 struct Coin 54 struct Coin		<pre>19 const EBadWitness: u64 = 0;</pre>	
dynamic_object_field 21 const EInvalidArg: u64 = 1; 22 /// Trying to split a coin more times than its balance allows. cedsa_k1 23 const ENotEnough: u64 = 2; 24 24 ecdsa_r1 25 /// A coin of type `T` worth `value`. Transferable and storable 26 struct Coin <phantom t=""> has key, store {</phantom>	dynamic_field	20 /// Invalid arguments are passed to a function.	
Aprilation_Object_initial 22 /// Trying to split a coin more times than its balance allows. Secdsa_k1 23 const ENotEnough: u64 = 2; Secdsa_r1 25 /// A coin of type `T` worth `value`. Transferable and storable Secdsa_r1 26 struct Coin <phantom t=""> has key, store {</phantom>	dynamic_object_field	<pre>21 const EInvalidArg: u64 = 1;</pre>	
acdsa_k1 23 const ENotEnough: u64 = 2; acdsa_r1 25 /// A coin of type `T` worth `value`. Transferable and storable acdsa_r1 26 struct Coin <phantom t=""> has key, store {</phantom>		22 /// Trying to split a coin more times than its balance allows.	
acdsa_r1 25 /// A coin of type `T` worth `value`. Transferable and storable 26 struct Coin <phantom t=""> has key, store {</phantom>	ecdsa_k1	<pre>23 const ENotEnough: u64 = 2; 24</pre>	
ecdsa_r1 26 struct Coin <phantom t=""> has key, store {</phantom>		25 /// A coin of type `T` worth `value`. Transferable and storable	
	ecdsa_r1	26 struct Coin <phantom t=""> has key, store {</phantom>	

Figure 3: The source representation of the Move contract shown in Figure 2.

availability, and in time, we are eager to build out additional features such as code navigation (jumpt-to-definition, find-references) uniquely for smart contracts in the browser.

Design of Planned Studies to Identify Developer Experience and Friction. At the time of writing, our MSVS is on the cusp of being put into production, and is not yet a mature tool. Our primary research question is grounded in practice and industrial application: Is it feasible to design and build a service that ensures the fidelity of smart contract source code for the Move language? What does a successful implementation entail, and does it meet the needs of developers? Prior launched blockchains have demonstrated various complexities that arise, and the possibility of forfeiting such a concept [3]. We have found that the exploration and design of an effective smart contract verification is not merely an engineering exercise, and will continue to entail unknown complexities. For example, we are currently evaluating how to support divergent compilers that may emit different bytecodes, further complicating the source verification process. Our main contribution with our tool is thus a demonstration of the feasibility and implementation of a system that overcomes these initial challenges for the Move language.

Upon launch, we plan to largely evaluate quantitative metrics around this service (how often active users consult the source code in the browser out of all visitors) and build out new features that specifically require trusted *source code* to be available. For example, we are interested in evaluating and surfacing software quality metrics [8–10, 12]) to help give users the signals they need to trust a given smart contract. One practical example shows that the presence of repository badges can help to assess quality assurance in open source software [11]. Likewise, we seek corresponding signals, such as badges, to display alongside smart contract source to surface software quality signals. Our MSVS is the basic building block to unlock these studies and better understand the unique needs of Move smart contract developers.

4 RELATED WORK

Blockchains and supported smart contract languages exist in various flavors [1, 2, 5], and all impose unique constraints and challenges in their respective developer ecosystems. The Sui blockchain is no different, and in general the approach and implementation of our Move Source Verification Service address these uniquely and in new ways, particularly with respect to: (1) smart contract bytecode representation, and the ability to compare locally compiled bytecode to on-chain bytecode; (2) accounting for source changes in the presence of our unique package upgrade process;⁷ and (3) identifying challenges and striving for solutions that cause developer friction for the Move language. At a more general level, existing work in practice identifies the importance of linking to source repositories, for example, the Anchor Program Registry on the Solana blockchain [7]. On Ethereum, etherscan supports rendering source code for, e.g., the Solidity language [6]. The Tezos issue tracker contains an open feature request for verifying source code similar to the approach we've implemented [3]. Overall, in the state of practice, most blockchain languages benefit from some level

of source code discovery and verification to benefit their developer ecosystem.

5 CONCLUSION

Enhancing developer productivity and ergonomics in the area of smart contracts reveals a growing need for developers to easily access and understand source code they can trust. We identified unique requirements to enable source code views for the Sui blockchain and implemented MSVS to meet those. We described its implementation and operation as an essential building block for the developer ecosystem. We anticipate that usage and subsequent expansion of source-adjacent features (like quality indicators and editor-like jump-to-definition functionality) will reveal ongoing needs to sharpen developer tools for smart contracts.

ACKNOWLEDGMENTS

The author would like to acknowledge Ashok, Jk, and the engineering team generally at Mysten Labs for help and support of this work.

REFERENCES

- 2023. Discover the languages of Tezos. https://tezos.com/developers/languages. Online; accessed 24 October 2023.
- [2] 2023. Ethereum Smart Contract Languages. https://ethereum.org/en/developers /docs/smart-contracts/languages. Online; accessed 24 October 2023.
- [3] 2023. Feature Request: Possibility to Verify Source For Tezos Smart Contracts. https://gitlab.com/tezos/tezos/-/issues/4491. Online; accessed 24 October 2023.
 [4] 2023. Language Server Protocol. https://microsoft.github.io/language-server-
- protocol. Online; accessed 24 October 2023. [5] 2023. Solana Rust Program Quickstart. https://docs.solana.com/getstarted/rust.
- Online; accessed 24 October 2023. [6] 2023. Solidity Verified Contract. https://etherscan.io/token/0x0AaCfbeC6a2475
- [6] 2023. Solidity Verhied Contract. https://energean.ob/okei/oko/activeCo224/3 6c20D41914F2caba817C0d8521#code. Online; accessed 24 October 2023.
- [7] 2023. What is Anchor and the Anchor Program Registry? https://www.alchemy. com/overviews/solana-anchor. Online; accessed 24 October 2023.
- [8] Allan J. Albrecht and John E. Gaffney Jr. 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Trans. Software Eng.* 9, 6 (1983), 639–648.
- [9] Eirini Kalliamvakou, Daniela E. Damian, Kelly Blincoe, Leif Singer, and Daniel M. Germán. 2015. Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub. In International Conference on Software Engineering (ICSE '15). 574–585.
- [10] Sandra Slaughter, Donald E. Harter, and Mayuram S. Krishnan. 1998. Evaluating the Cost of Software Quality. *Commun. ACM* 41, 8 (1998), 67–73.
- [11] Asher Trockman, Shurui Zhou, Christian Kästner, and Bogdan Vasilescu. 2018. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In *International Conference on Software Engineering (ICSE)*. ACM, 511–522.
- [12] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb, and Jonathan Streit. 2012. The Quamoco product quality modelling and assessment approach. In International Conference on Software Engineering (ICSE '12). 1133–1142.

⁷https://docs.sui.io/build/package-upgrades