Identifying Solidity Smart Contract API Documentation Errors

Chenguang Zhu\textsuperscript{1}, Ye Liu\textsuperscript{2}, Xiuheng Wu\textsuperscript{2}, Yi Li\textsuperscript{2}

\textsuperscript{1}The University of Texas at Austin
\textsuperscript{2}Nanyang Technological University, Singapore

ASE 2022
Oct 13, 2022
Decentralized Applications and Smart Contracts

Apps
(Centralized)

DApps

Traditional Contracts

1. Relying on intermediaries
2. Non-transparent
3. Slow and costly

Smart Contracts

1. Immutable
2. Self-enforcing
3. Affordable and accessible

• Relying on intermediaries
• Non-transparent
• Slow and costly
Why is DApp a big thing?

Internet is the information superhighway, blockchain is the Internet of value

- Direct peer-to-peer exchange of surplus electricity
- Reduce transaction costs

Decentralized finance
- Banking, insurance, decentralized exchange, …
- Nearly $30 billion locked inside
- 4.4 million wallets

Energy trading

Supply chain management
- Better visibility and traceability
- Improve financing, contracting, and international transactions
Ethereum Decentralized Applications (DApps)

**Key Features**
- Event driven front-end code
- Smart contract execution powered by gas
- Transaction may fail if requirements are not met

**Prevalence**
- Nearly 50M Solidity smart contracts deployed on Ethereum
- 1.96x increase within two years
- 4,056 DApps, serving 113.86K daily active users

**Events**
- Internet
- Web Server
- Front-end
  - JavaScript, HTML, CSS
- Smart Contracts
- Ethereum Virtual machine
- Ethereum Blockchain
- Browser

**Transaction revert**

**Gas**
Motivation: DApp’s complexity keeps growing

- Developers rely on third-party libraries – e.g., OpenZeppelin, Dappsys, ERC721-Ext, etc.

According to Kondo et al. (2020)
- 36.3% of the verified contracts uses code from OpenZeppelin
- ERC-20 and SafeMath are among the most frequently used APIs

API Documentation Errors
- 46% commits within the past 6 months from OpenZeppelin modified/fixed API documentations
- Domain-specific errors: event emissions, transaction requirements/reversions
Example: ECR-721 Contract Extensions

DocCon Detected API Doc Errors

- The `OfferWithdrawn` event emission is undocumented
- The event is also transitively emitted by function `_beforeTokenTransfer`

Developers’ Fixes

```solidity
1 /// @dev Revoke an active offer
2 function _cancelOffer(uint256 tokenId) private {
3     delete _offers[tokenId];
4     emit OfferWithdrawn(tokenId);
5 }

/// @dev Clear active offers on transfers
6 function _beforeTokenTransfer(address, address, uint256 tokenId) {
7     internal virtual override(ERC721) {
8         if (_offers[tokenId].price > 0) {
9             _cancelOffer(tokenId);
10         }
11     }
```

- /// @dev Revoke an active offer
  + /// @dev Revoke an active offer.
  + /// Emits an {OfferWithdrawn} event.
  ...

- /// @dev clear active offers on transfers
  + /// @dev Clear active offers on transfers.
  + /// Emits an {OfferWithdrawn} event if an active offer exists.

We helped save some gas!

jwahdatehagh commented on Aug 3

Sales and Transfers shouldn't result in OfferWithdrawn events as mentioned in #13.
That can be inferred off chain and we can save the bit of gas.
Outline

1. Introduction
2. Existing approaches
3. DocCon
   • Code fact extraction
   • Doc fact extraction
   • Inconsistency queries
4. Evaluation
5. Summary
Limitations of the Existing Solutions

- No existing techniques for Solidity smart contracts yet
- Solutions for other languages (e.g., Java) do not fit
  - Grammatical errors
  - Incorrect code names
  - Parameter properties: nullness, type, range limitation

Different features matter in Solidity DApp documentations

- Events: emissions
- Transactions: requirements, reversions, …
- Language-Specific Elements: contracts, modifiers, events, addresses, …

```java
/* If button is less than zero or greater than the number of button masks reserved for buttons */
if (button <= 0 || button > BUTTON DOWN MASK.length) {
    ...
}
```
Overview of DocCon

1. Smart Contract Solidity Code
   - Extracting Code Facts
     - Code Facts
     - Code Fact Extractor

2. Smart Contract API Documentation
   - Extracting Document Facts
     - Document Facts
     - Document Fact Extractor

3. Error Detector
   - Fact Query Engine
     - API Documentation Errors
Differential Factbase

Wu, Zhu & Li (FSE 2020)
Step 1: Code Facts Extraction

- Traverse Solidity ASTs to extract code names & relations
  - Build ASTs from source code
  - Code entity names: contracts, functions, events, etc.
  - Code entity relations: calls, event emissions, transaction reversions, etc.

```solidity
/* ... Emits a {TokensReleased} event. */
function release(address token) public virtual {
    uint256 releasable = vestedAmount(token,
    uint64(block.timestamp)) - released(token);
    _erc20Released[token] += releasable;
    emit ERC20Released(token, releasable);
    SafeERC20.safeTransfer(IERC20(token), beneficiary(),
    releasable);
}
```

- HasFn("VestingWallet", "release")
- HasParam("VestingWallet", "release", "token")
- Emit("VestingWallet", "release", "ERC20Released", "true")
- Call("VestingWallet", "release", ["IERC20(token)",
  "beneficiary()", "releasable"], "SafeERC20",
  "safeTransfer", ["token", "to", "value")"
Fact Schema – A Partial List

<table>
<thead>
<tr>
<th>Predicates</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Override(\texttt{ca}:Ct, \texttt{fa}:Fn, \texttt{cb}:Ct, \texttt{fb}:Fn)</td>
<td>Function \texttt{cb.fb} overrides \texttt{ca.fa}</td>
</tr>
<tr>
<td>HasFn(\texttt{c}:Ct, \texttt{f}:Fn)</td>
<td>Contract \texttt{c} has a function \texttt{f}</td>
</tr>
<tr>
<td>FnHasMod(\texttt{c}:Ct, \texttt{f}:Fn, \texttt{m}:Mod)</td>
<td>Function \texttt{c.f} has a modifier \texttt{m}</td>
</tr>
<tr>
<td>Require(\texttt{c}:Ct, \texttt{f}:Fn, \texttt{e}:Expr)</td>
<td>\texttt{c.f} requires condition \texttt{e} to be true</td>
</tr>
<tr>
<td>Revert(\texttt{c}:Ct, \texttt{f}:Fn, \texttt{e}:Expr)</td>
<td>\texttt{c.f} reverts under condition \texttt{e}</td>
</tr>
<tr>
<td>Emit(\texttt{c}:Ct, \texttt{f}:Fn, \texttt{ev}:Event, \texttt{e}:Expr)</td>
<td>\texttt{c.f} emits event \texttt{ev} under condition \texttt{e}</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Same schema is used for both the code and doc facts
Step 2: Doc Facts Extraction

- Use custom document templates
  - Each template is a rule for extracting a fact from a sentence
  - We designed 37 templates based on our observation of Solidity library documentations

- A partial list of document templates

<table>
<thead>
<tr>
<th>Document Templates</th>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;In c.f: &quot;Requirements: - 'va' must be strictly less than 'vb'&quot;&gt;</td>
<td>Require(c, f, va &lt; vb)</td>
</tr>
<tr>
<td>&lt;In c.f: &quot;Reverts with ... if 'va' is at least 'vb'&quot;&gt;</td>
<td>Revert(c, f, va &gt;= vb)</td>
</tr>
<tr>
<td>&lt;In c.f: &quot;Emits an {e} event&quot;&gt;</td>
<td>Emit(c, f, e, &quot;true&quot;)</td>
</tr>
</tbody>
</table>

```plaintext
1 /** ... Emits a TokensReleased event. */
2 function release(address token) public virtual {
    Emit("VestingWallet", "release", "TokensReleased", "true")
```
Step 3: Error Detection through Factbase Queries

**Incorrectness:**

- **External incompleteness:**
  - Level-1 Error: \( \{ x \mid x \in D \land x \notin C \} \)
  - Level-2 Error: \( \{ x \mid x \in C \land x \notin D \land x.\text{name} \in \{ \text{Emit, Revert, Require} \} \} \)
  - Level-3 Error: \( \{ x \mid x \in C \land x \notin D \land x.\text{name} \notin \{ \text{Emit, Revert, Require} \} \} \)

- **Internal incompleteness:**

  - L1 \( \{ \text{Require}(..., \_\text{minted}(\text{tokenId})) \} \text{ in } D \text{ but not in } C \)
  - L2 \( \{ \text{HasParam}(..., \text{recipient}) \} \text{ in } D \text{ but not in } C \)
  - L3 \( \{ \text{Emit}(..., \text{OfferWithdrawn}) \} \text{ in } C \text{ but not in } D \)
  - L3 \( \{ \text{HasParam}(..., \text{to}) \} \text{ in } C \text{ but not in } D \)
Inconsistency Queries: Inferring Additional Facts

- Facts about one function also apply to another function, if there are sentences such as "@dev see ..." in documentation
  - E.g., function `ca.fa` reverts under condition `e` if its documentation contains "See `cb.fb`" and `cb.fb` reverts under `e`.
  - \( \text{Revert}(ca, fa, e) \leftarrow \text{SeeFn}(ca, fa, cb, fb), \text{Revert}(cb, fb, e) \).

- Facts propagate through the call chain
  - \( \text{Revert}(ca, caller, e) \leftarrow \text{Revert}(cb, callee, e), \text{Call}(ca, caller, cb, callee) \).
Examples

Wrong Event Names (L1)
- The `ERC20Released` event is incorrectly documented as `TokensReleased`

Wrong Transaction Requirements (L1)
- The transaction requirement of `tokenId` is spurious
More Examples

```solidity
1 /** @dev Stores the sent amount as credit to be withdrawn. */
2 * @param payee The destination address of the funds. */
3 function deposit(address payee) public payable virtual onlyOwner {
4     uint256 amount = msg.value;
5     _deposits[payee] += amount;
6     emit Deposited(payee, amount);
7 }
8
9 /** @dev Called by the payer to store the sent amount as credit to be pulled ... */
10 * @param dest The destination address of the funds.
11 * @param amount The amount to transfer. */
12 function _asyncTransfer(address dest, uint256 amount) internal virtual
13     _escrow.deposit{value: amount}(dest); }
```

### Missing Events (L2)
- The **Deposited** event emission is undocumented
- Transitivity affect another function

```solidity
1 /** @dev Returns the item at the beginning of the queue. */
2 function front(Bytes32Deque storage deque) internal view returns (bytes32 value) {
3     if (empty(deque)) revert Empty();
4     int128 frontIndex = deque._begin;
5     return deque._data[frontIndex];
6 }
7
8 /** @dev Returns the item at the end of the queue. */
9 function back(Bytes32Deque storage deque) internal view returns (bytes32 value) {
10    if (empty(deque)) revert Empty();
11    int128 backIndex;
12    unchecked { backIndex = deque._end - 1;}
13    return deque._data[backIndex];
```

### Missing Transaction Reversions (L2)
- The transaction reversions are undocumented
RQ1: How precise is DocCon in detecting errors in Solidity smart contract API documentations?

RQ2: How relevant are the smart contract API documentation errors detected by DocCon?

RQ3: What are the categories of the smart contract API documentation errors detected by DocCon?
Evaluation: Subjects

- Three popular Solidity smart contract libraries
  - OpenZeppelin
  - Dappsys
  - ERC721 Contract Extensions

- >18K stars in total on GitHub
RQ1: DocCon’s Precision

<table>
<thead>
<tr>
<th>Library</th>
<th>#Detected</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Level-1</td>
<td>Level-2</td>
</tr>
<tr>
<td>OpenZeppelin</td>
<td>49</td>
<td>567</td>
</tr>
<tr>
<td>Dappsys</td>
<td>4</td>
<td>141</td>
</tr>
<tr>
<td>ERC721 Contract Extensions</td>
<td>3</td>
<td>79</td>
</tr>
<tr>
<td>Overall</td>
<td>56</td>
<td>787</td>
</tr>
</tbody>
</table>

**Precision: Manual Inspection**
- Level-1: Inspected all
- Level-2: Inspected 449 errors
- Level-3: Did not inspect

**Answer to RQ1**
DocCon detected 56 level-1 and 787 level-2 API documentation errors in all the three libraries, with the level-1 and level-2 precision of 76% and 66%, respectively.
RQ2: DocCon’s Practical Relevance

- Reported 40 errors to developers
  - Developers confirmed 29 (72.5%)
  - Developers fixed 22 (55%)
  - All our bug reports are publicly available: https://sites.google.com/view/doccon-tool

- Developer reacted positively
  - “Thank you for pointing that out. We definitely need more consistency or at least clearer guidelines on how we approach that matter.” [1]
  - “You’re welcome to submit pull requests as well next time.” [2]

Answer to RQ2
DocCon’s detection results are useful to developers in practice

RQ3: Categorization of Smart Contract API Documentation Errors

**Error Categories**
- Event Emission
- Transaction Requirement/Reversion
- Element Containment
- Element Reference

**Answer to RQ3**
DocCon detected four categories of errors, two of which have no counterparts in general-purpose programming languages
Contribution and Summary

**Problem Highlight**

We show that many errors exist in smart contract library API documentations.

**Evaluation**

Reported 40 errors to library developers, who confirmed 29 and fixed 22.

**DocCon**

Novel fact-based technique for detecting errors in Solidity smart contract API documentations.

**Publicly Available**

https://sites.google.com/view/doccon-tool