Identifying Solidity Smart Contract API Documentation Errors

Chenguang Zhu[1], Ye Liu[2], Xiuheng Wu[2], Yi Li[2]

1. The University of Texas at Austin

2. Nanyang Technological University, Singapore

ASE 2022

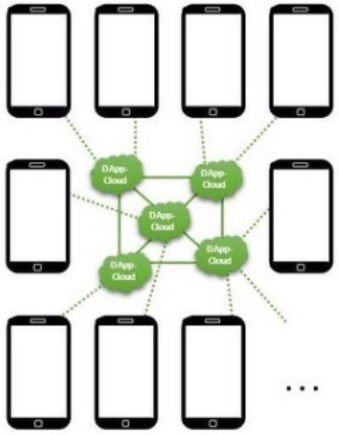
Oct 13, 2022

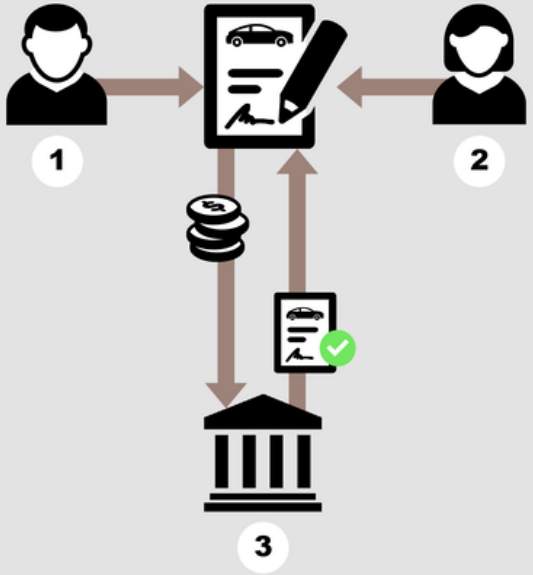
Decentralized Applications and Smart Contracts

Apps

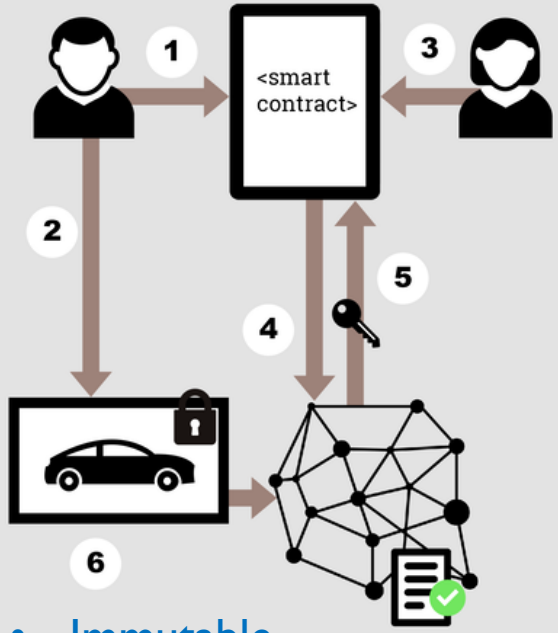
(Centralized)

DApps



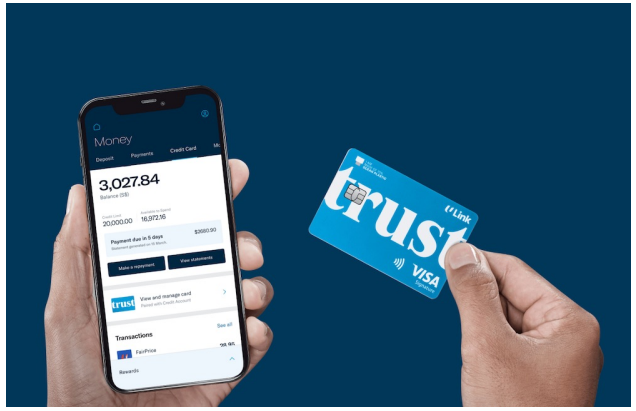


- Relying on intermediaries
- Non-transparent
- Slow and costly



- Immutable
- Self-enforcing
- Affordable and accessible

Why is DApp a big thing?



Decentralized finance

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

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

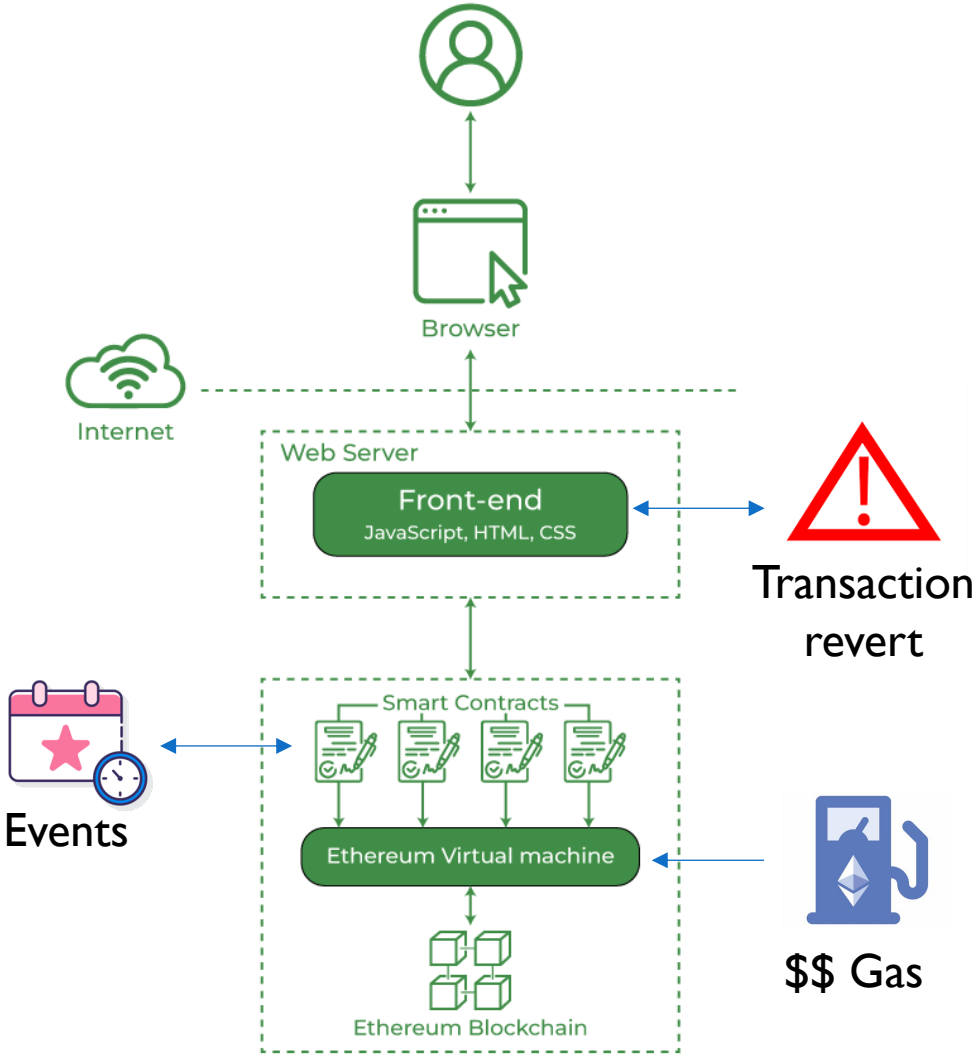
Energy trading

Supply chain management

- Better visibility and traceability
- Improve financing, contracting, and international transactions

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

Ethereum Decentralized Applications (DApps)



Prevalence

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

Key Features

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

Solidity Smart Contract Libraries

- 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

OpenZeppelin

- **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


```
1  /// @dev Revoke an active offer
2  function _cancelOffer(uint256 tokenId) private {
3      delete _offers[tokenId];
4      emit OfferWithdrawn(tokenId);
5  }
6  /// @dev Clear active offers on transfers
7  function _beforeTokenTransfer(address, address, uint256 tokenId)
   ↪  internal virtual override(ERC721) {
8      if (_offers[tokenId].price > 0) {
9          _cancelOffer(tokenId);
10     }
11 }
```

Developers' Fixes

```
-     /// @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.
```

DocCon Detected API Doc Errors

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

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

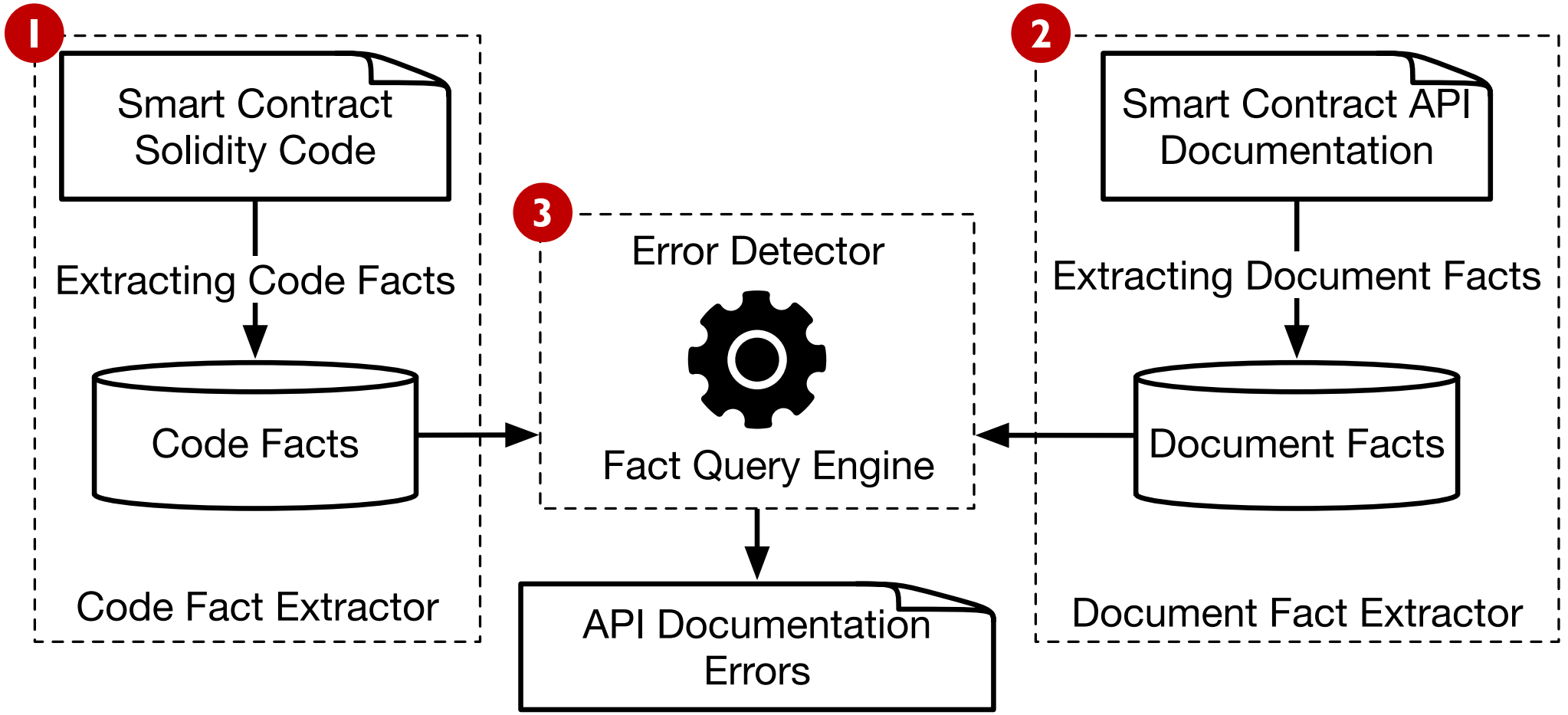
Limitations of the Existing Solutions

- 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

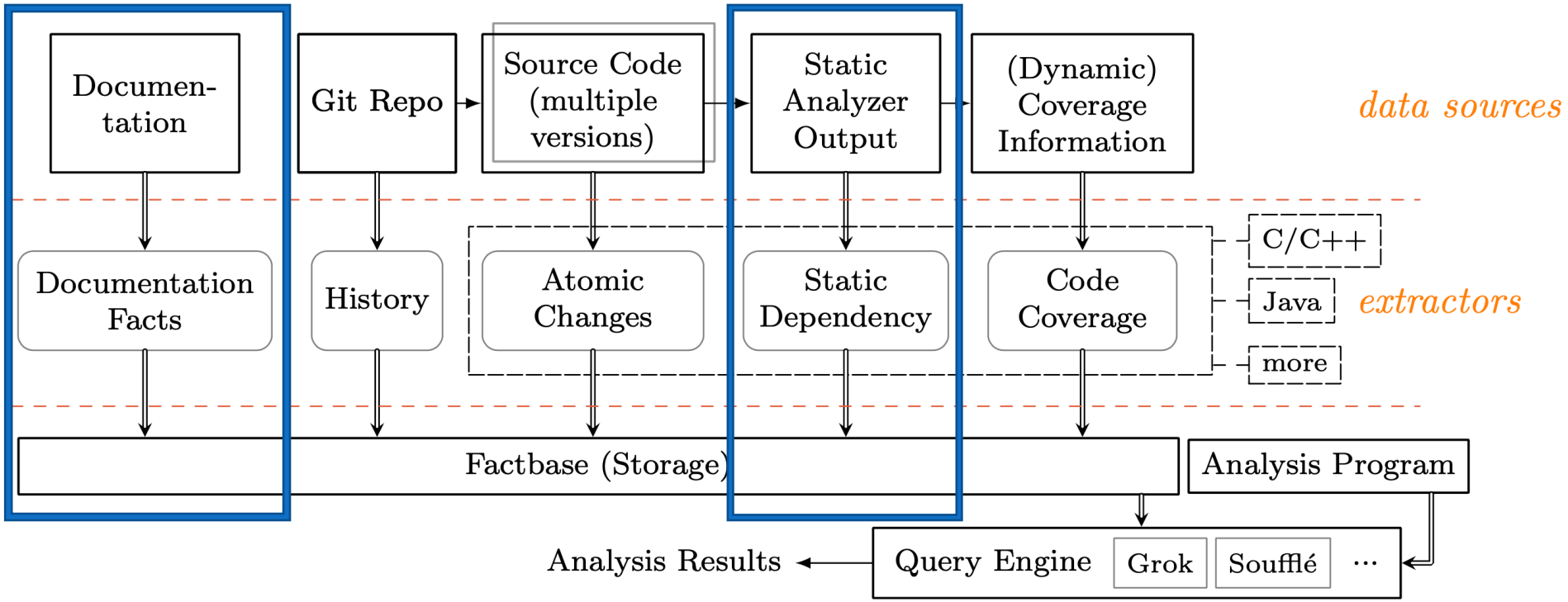
```
/* If button is less
than zero or greater
than the number of
button masks reserved
for buttons */

if (button <= 0 || button > BUTTON
DOWN MASK.length) {
    …
}
```

- Different features matter in Solidity DApp documentations
  - Events: emissions
  - Transactions: requirements, reversions, …
  - Language-Specific Elements: contracts, modifiers, events, addresses, …

Overview of DocCon



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.

```
1 /** ...  Emits a {TokensReleased} event. */
2 function release(address token) public virtual {
3     uint256 releasable = vestedAmount(token,
  ↪ uint64(block.timestamp)) - released(token);
4     _erc20Released[token] += releasable;
5     emit ERC20Released(token, releasable);
6     SafeERC20.safeTransfer(IERC20(token), beneficiary(),
  ↪ releasable);
7 }
```

- 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

Predicates	Descriptions
Override(ca:Ct, fa:Fn, cb:Ct, fb:Fn)	Function cb.fb **overrides** ca.fa
HasFn(c:Ct, f:Fn)	Contract c **has** a function f
FnHasMod(c:Ct, f:Fn, m:Mod)	Function c.f **has** a modifier m
Require(c:Ct, f:Fn, e:Expr)	c.f **requires** condition e to be true
Revert(c:Ct, f:Fn, e:Expr)	c.f **reverts** under condition e
Emit(c:Ct, f:Fn, ev:Event, e:Expr)	c.f **emits** event ev under condition e
…	…

- Same schema is used for both the code and doc facts

Step 2: Doc Facts Extraction

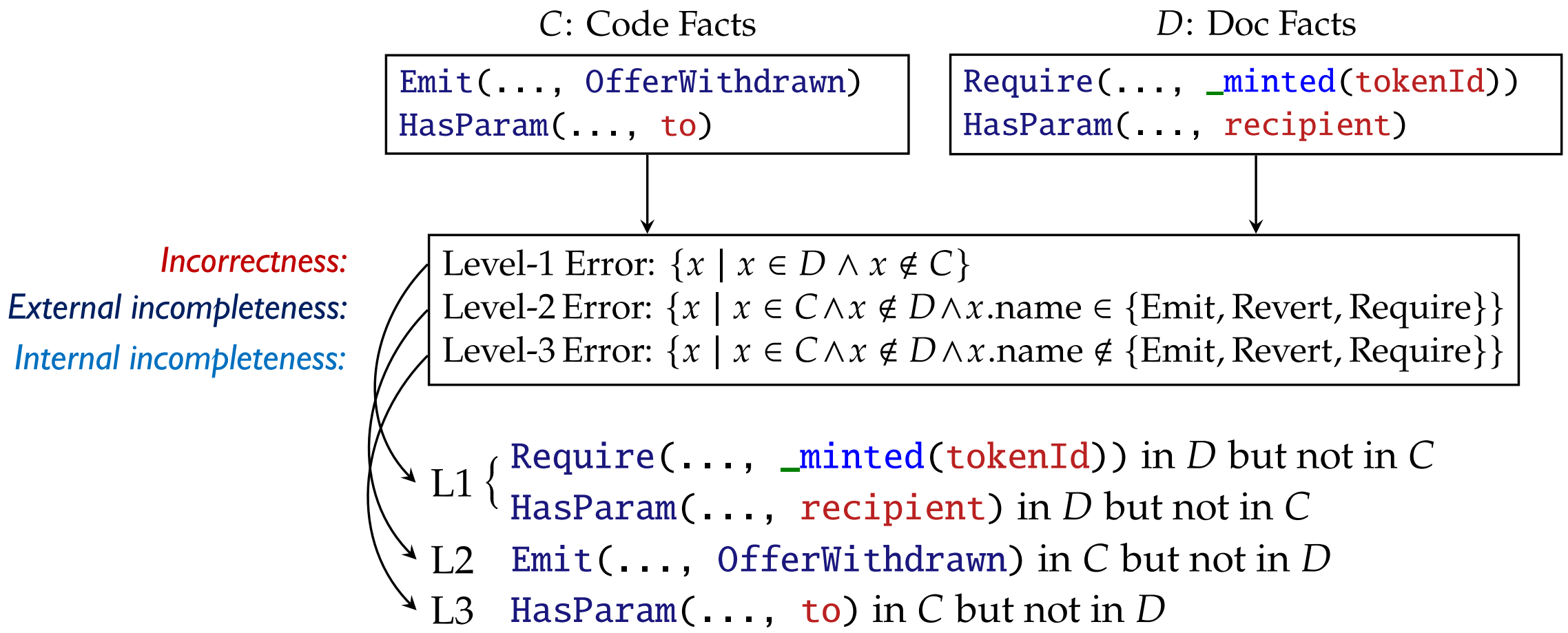
- Use custom document templates
  - Each template is a rule for extracting a fact from a sentence
  - We designed 37 templates based on our observation of Solidity library documentations
- A partial list of document templates

Document Templates	Facts
<In c.f: "Requirements: - 'va' must be strictly less than 'vb'">	Require(c, f, va < vb)
<In c.f: "Reverts with ... if 'va' is at least 'vb' ">	Revert(c, f, va >= vb)
<In c.f: "Emits an {e} event">	Emit(c, f, e, "true")

```
1 /** ...  Emits a {TokensReleased} event. */
2 function release(address token) public virtual {
```

Emit("VestingWallet", "release", "TokensReleased", "true")

Step 3: Error Detection through Factbase Queries



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`.**
  - `Revert(ca, fa, e) <- SeeFn(ca, fa, cb, fb), Revert(cb, fb, e).`
- **Facts propagate through the call chain**
  - `Revert(ca, caller, e) <- Revert(cb, callee, e), Call(ca, caller, cb, callee).`

```
/**
 * @dev See {IERC721-approve}.
 */
```

Examples

```
1 /** ...  Emits a {TokensReleased} event. */
2 function release(address token) public virtual {
3     uint256 releasable = vestedAmount(token,
  ↪  uint64(block.timestamp)) - released(token);
4     _erc20Released[token] += releasable;
5     emit ERC20Released(token, releasable);
6     SafeERC20.safeTransfer(IERC20(token), beneficiary(),
  ↪  releasable);
7 }
```

Wrong Event Names (L1)

- The *ERC20Released* event is incorrectly documented as *TokensReleased*

```
1 /** ...  Requirements:
2 * - `tokenId` must be already minted.
3 * - `receiver` cannot be the zero address.
4 * - `feeNumerator` cannot be greater than the fee denominator. */
5 function _setTokenRoyalty(uint256 tokenId, address receiver,
6     uint96 feeNumerator) internal virtual {
7     require(feeNumerator<=_feeDenominator(), "ERC2981: ...");
8     require(receiver!=address(0),"ERC2981: Invalid parameters");
9     ... }
```

Wrong Transaction Requirements (L1)

- The transaction requirement of *tokenId* is spurious

More Examples

```
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  /** @dev Called by the payer to store the sent amount as credit
   ↪  to be pulled ...
9  *   @param dest The destination address of the funds.
10 *   @param amount The amount to transfer. */
11 function _asyncTransfer(address dest, uint256 amount) internal
   ↪  virtual {
12     _escrow.deposit{value: amount}(dest); }
```

```
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  /** @dev Returns the item at the end of the queue. */
8  function back(Bytes32Deque storage deque) internal view returns
   ↪  (bytes32 value) {
9      if (empty(deque)) revert Empty();
10     int128 backIndex;
11     unchecked { backIndex = deque._end - 1;}
12     return deque._data[backIndex]; }
```

Missing Events (L2)

- The *Deposited* event emission is undocumented
- Transitively affect another function

Missing Transaction Reversions (L2)

- The transaction reversions are undocumented

Evaluation: Research Questions

- 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

Library	#Detected			Precision	
	Level-1	Level-2	Level-3	Level-1	Level-2
OpenZeppelin	49	567	3741	78%	72%
Dappsys	4	141	448	50%	53%
ERC721 Contract Extensions	3	79	377	100%	73%
Overall	56	787	4566	76%	66%

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

[1] [2] https://github.com/OpenZeppelin/openzeppelin-contracts/issues/3374

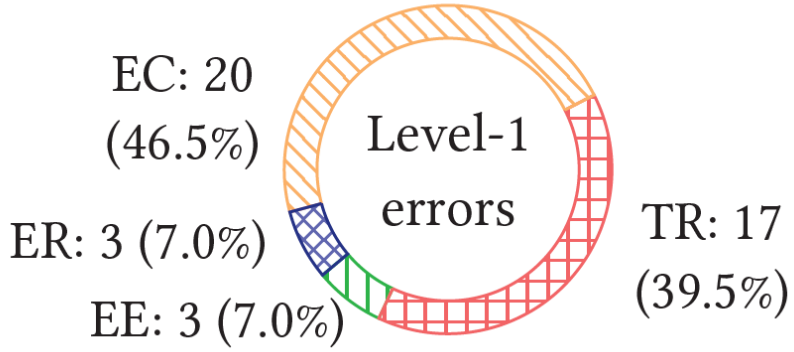
RQ3: Categorization of Smart Contract API Documentation Errors

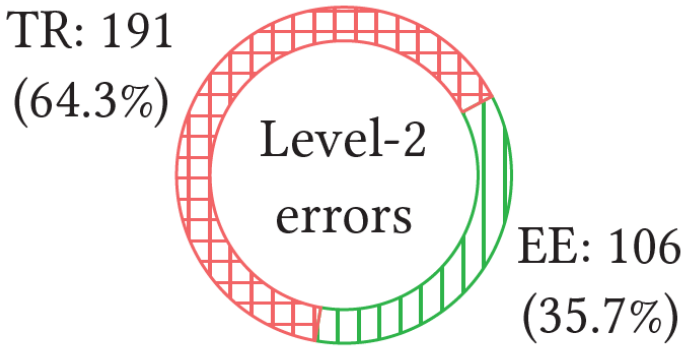
Error Categories

- 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

yi_li@ntu.edu.sg
@liyistc

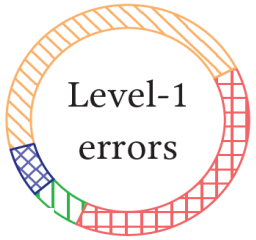
Problem Highlight

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

DocCon

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

Evaluation



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

Publicly Available

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