

0.7

Belt Finance Process Quality Review

Score: 56%

Overview

This is a [Belt Finance](#) Process Quality Review completed on June 30th 2021. It was performed using the Process Review process (version 0.7.2) and is documented [here](#). The review was performed by Nic of DeFiSafety. Check out our [Telegram](#).

The final score of the review is 56%, a fail. The breakdown of the scoring is in [Scoring Appendix](#). For our purposes, a pass is 70%.

Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

- **Here are my smart contracts on the blockchain**
- **Here is the documentation that explains what my smart contracts do**
- **Here are the tests I ran to verify my smart contract**
- **Here are the audit(s) performed on my code by third party experts**
- **Here are the admin controls and strategies**

Disclaimer

This report is for informational purposes only and does not constitute investment advice of any kind, nor does it constitute an offer to provide investment advisory or other services. Nothing in this report shall be considered a solicitation or offer to buy or sell any security, token, future, option or other financial instrument or to offer or provide any investment advice or service to any person in any jurisdiction. Nothing contained in this report constitutes investment advice or offers any opinion with respect to the suitability of any security, and the views expressed in this report should not be taken as advice to buy, sell or hold any security. The information in this report should not be relied upon for the purpose of investing. In preparing the information contained in this report, we have not taken into account the investment needs, objectives and financial circumstances of any particular investor. This information has no regard to the specific investment objectives, financial situation and particular needs of any specific recipient of this information and investments discussed may not be suitable for all investors.

Any views expressed in this report by us were prepared based upon the information available to us at the time such views were written. The views expressed within this report are limited to DeFiSafety and the author and do not reflect those of any additional or third party and are strictly based upon DeFiSafety, its authors, interpretations and evaluation of relevant data. Changed or additional information could cause such

views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

This completed report is copyright (c) DeFiSafety 2021. Permission is given to copy in whole, retaining this copyright label.

Chain

This section indicates the blockchain used by this protocol.

Chain: Binance Smart Chain

Guidance:

Ethereum
Binance Smart Chain
Polygon

Code and Team

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is [here](#). This review will answer the questions;

- 1) Are the executing code addresses readily available? (%)
- 2) Is the code actively being used? (%)
- 3) Is there a public software repository? (Y/N)
- 4) Is there a development history visible? (%)
- 5) Is the team public (not anonymous)? (Y/N)

1) Are the executing code addresses readily available? (%)

Answer: 100%

They are available at website <https://docs.belt.fi/contracts/contract-deployed-info> as indicated in the [Appendix](#).

Guidance:

- | | |
|------|--|
| 100% | Clearly labelled and on website, docs or repo, quick to find |
| 70% | Clearly labelled and on website, docs or repo but takes a bit of looking |
| 40% | Addresses in mainnet.json, in discord or sub graph, etc |
| 20% | Address found but labelling not clear or easy to find |
| 0% | Executing addresses could not be found |

2) Is the code actively being used? (%)

 Answer: 100%

Activity is 7000 transactions a day on contract *MasterBelt.sol*, as indicated in the [Appendix](#).

Percentage Score Guidance

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

3) Is there a public software repository? (Y/N)

 Answer: Yes

GitHub: <https://github.com/BeltFi/>

Is there a public software repository with the code at a minimum, but normally test and scripts also (Y/N). Even if the repo was created just to hold the files and has just 1 transaction, it gets a Yes. For teams with private repos, this answer is No.

4) Is there a development history visible? (%)

 Answer: 30%

With 39 commits and 1 branch, this is an underdeveloped software repository.

This checks if the software repository demonstrates a strong steady history. This is normally demonstrated by commits, branches and releases in a software repository. A healthy history demonstrates a history of more than a month (at a minimum).

Guidance:

100%	Any one of 100+ commits, 10+branches
70%	Any one of 70+ commits, 7+branches
50%	Any one of 50+ commits, 5+branches
30%	Any one of 30+ commits, 3+branches
0%	Less than 2 branches or less than 10 commits

How to improve this score

Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools. A public development history indicates clearly to

the public the level of continued investment and activity by the developers on the application. This gives a level of security and faith in the application.

5) Is the team public (not anonymous)? (Y/N)

 Answer: Yes

Team info can be found at <https://www.notion.so/About-OZYS-3d0a4167386f43c88d6b16f97f6d3b3e>.

For a yes in this question the real names of some team members must be public on the website or other documentation. If the team is anonymous and then this question is a No.

Documentation

This section looks at the software documentation. The document explaining these questions is [here](#).

Required questions are;

- 6) Is there a whitepaper? (Y/N)
- 7) Are the basic software functions documented? (Y/N)
- 8) Does the software function documentation fully (100%) cover the deployed contracts? (%)
- 9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)
- 10) Is it possible to trace from software documentation to the implementation in code (%)

6) Is there a whitepaper? (Y/N)

 Answer: Yes

Location: <https://docs.belt.fi/>

7) Are the basic software functions documented? (Y/N)

 Answer: Yes

Basic software function documentation found at <https://docs.belt.fi/contracts/contract-operation>.

8) Does the software function documentation fully (100%) cover the deployed contracts? (%)

 Answer: 80%

Contracts overview
Belt.Fi Documentation

Contract operation & security
Belt.Fi Documentation

Belt Finance has documented their functions in their documentation, but has not elaborated on the usage of the functions they defined. For some of the functions, they document them well.

<https://docs.belt.fi/understanding-belt/fees>

Guidance:

- 100% All contracts and functions documented
- 80% Only the major functions documented
- 79-1% Estimate of the level of software documentation
- 0% No software documentation

How to improve this score

This score can improve by adding content to the requirements document such that it comprehensively covers the requirements. For guidance, refer to the [SecurEth System Description Document](#). Using tools that aid traceability detection will help.

9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)

 Answer: 0%

Code examples are in the [Appendix](#). As per the [SLOC](#), there is 15% commenting to code (CtC).

The Comments to Code (CtC) ratio is the primary metric for this score.

Guidance:

- 100% CtC > 100 Useful comments consistently on all code
- 90-70% CtC > 70 Useful comment on most code
- 60-20% CtC > 20 Some useful commenting
- 0% CtC < 20 No useful commenting

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the [SecurEth Software Requirements](#).

10) Is it possible to trace from software documentation to the implementation in code (%)

 Answer: 20%

Their documentation lists the functions and describes what some of the functions do.

Guidance:

- 100% Clear explicit traceability between code and documentation at a requirement level for all code
- 60% Clear association between code and documents via non explicit traceability
- 40% Documentation lists all the functions and describes their functions
- 0% No connection between documentation and code

How to improve this score

This score can improve by adding traceability from requirements to code such that it is clear where each requirement is coded. For reference, check the SecurEth guidelines on [traceability](#).

Testing

This section looks at the software testing available. It is explained in this [document](#). This section answers the following questions;

- 11) Full test suite (Covers all the deployed code) (%)
- 12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)
- 13) Scripts and instructions to run the tests (Y/N)
- 14) Report of the results (%)
- 15) Formal Verification test done (%)
- 16) Stress Testing environment (%)

11) Is there a Full test suite? (%)

 Answer: 0%

No testing suite available in their public GitHub repository.

This score is guided by the Test to Code ratio (TtC). Generally a good test to code ratio is over 100%. However the reviewers best judgement is the final deciding factor.

Guidance:

- 100% TtC > 120% Both unit and system test visible
- 80% TtC > 80% Both unit and system test visible
- 40% TtC < 80% Some tests visible
- 0% No tests obvious

How to improve this score

This score can improve by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)

 Answer: 100%

Documented full coverage in their Haechi audit at https://github.com/BeltFi/belt-contract/blob/main/audit/HAECHI_AUDIT_Smart_contract_audit_report_for_BeltFi_earnV2_additional.pdf

Guidance:

100% Documented full coverage

99-51% Value of test coverage from documented results

50% No indication of code coverage but clearly there is a reasonably complete set of tests

30% Some tests evident but not complete

0% No test for coverage seen

13) Scripts and instructions to run the tests (Y/N)

 Answer: No

There are no scripts and instructions to run the non-existent test suite in their public GitHub repository.

How to improve this score

Add the scripts to the repository and ensure they work. Ask an outsider to create the environment and run the tests. Improve the scripts and docs based on their feedback.

14) Report of the results (%)

 Answer: 0%

Belt Finance does not have their own published coverage report in their public GitHub repository.

Guidance:

100% Detailed test report as described below

70% GitHub Code coverage report visible

0% No test report evident

How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

15) Formal Verification test done (%)

 Answer: 0%

No evidence of a Formal Verification test was found in their documentation or on the web.

16) Stress Testing environment (%)

 Answer: 0%

No evidence of Belt Finance test-net smart contract usage.

Security

This section looks at the 3rd party software audits done. It is explained in this [document](#). This section answers the following questions;

17) Did 3rd Party audits take place? (%)

18) Is the bounty value acceptably high?

17) Did 3rd Party audits take place? (%)

 Answer: 70%

[SOOHO published a Belt Finance security assessment on June 15th 2021.](#)

[SOOHO published a Belt Finance security assessment on March 9th 2021.](#)

[HAECHI published a Belt Finance smart contract audit on May 6th 2021.](#)

Belt Finance was released at the end of February 2021.

Some recommended fixes have been implemented, and others are planned to be implemented with Belt V2.

Guidance:

100% Multiple Audits performed before deployment and results public and implemented or not required

90% Single audit performed before deployment and results public and implemented

- or not required
- 70% Audit(s) performed after deployment and no changes required. Audit report is public
- 50% Audit(s) performed after deployment and changes needed but not implemented
- 20% No audit performed
- 0% Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, question

18) Is the bounty value acceptably high (%)

 Answer: 0%

There is no Belt Finance Bug Bounty program.

Guidance:

- 100% Bounty is 10% TVL or at least \$1M AND active program (see below)
- 90% Bounty is 5% TVL or at least 500k AND active program
- 80% Bounty is 5% TVL or at least 500k
- 70% Bounty is 100k or over AND active program
- 60% Bounty is 100k or over
- 50% Bounty is 50k or over AND active program
- 40% Bounty is 50k or over
- 20% Bug bounty program bounty is less than 50k
- 0% No bug bounty program offered

Active program means a third party actively driving hackers to the site. Inactive program would be static mention on the docs.

Access Controls

This section covers the documentation of special access controls for a DeFi protocol. The admin access controls are the contracts that allow updating contracts or coefficients in the protocol. Since these contracts can allow the protocol admins to "change the rules", complete disclosure of capabilities is vital for user's transparency. It is explained in this [document](#). The questions this section asks are as follow;

- 19) Can a user clearly and quickly find the status of the admin controls?
- 20) Is the information clear and complete?
- 21) Is the information in non-technical terms that pertain to the investments?
- 22) Is there Pause Control documentation including records of tests?

19) Can a user clearly and quickly find the status of the access controls (%)

 Answer: 40%

Found at <https://docs.belt.fi/contracts/contract-operation>

and <https://docs.belt.fi/tokenomics/belt#belt-inflation-distribution>

Note: Gave them a 40% because they labelled their access controls as "Contract Operation", which is a term that inexperienced users would have trouble identifying as access controls. In addition, governance info is found under "BELT Token", so access control is in multiple places and not very well labelled.

Guidance:

- 100% Clearly labelled and on website, docs or repo, quick to find
- 70% Clearly labelled and on website, docs or repo but takes a bit of looking
- 40% Access control docs in multiple places and not well labelled
- 20% Access control docs in multiple places and not labelled
- 0% Admin Control information could not be found

20) Is the information clear and complete (%)

 Answer: 60%

a) They list the functions that the team can upgrade or can not.

c) They list what specific functions can added/removed/changed.

Source: <https://docs.belt.fi/contracts/contract-operation>

Guidance:

All the contracts are immutable -- 100% OR

- a) All contracts are clearly labelled as upgradeable (or not) -- 30% AND
- b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND
- c) The capabilities for change in the contracts are described -- 30%

How to improve this score

Create a document that covers the items described above. An [example](#) is enclosed.

21) Is the information in non-technical terms that pertain to the investments (%)

 Answer: 90%

All information is written in a way that is easy and comprehensive for investors to understand. Source: <https://docs.belt.fi/contracts/contract-operation>

Guidance:

- 100% All the contracts are immutable

90%	Description relates to investments safety and updates in clear, complete non-software language
30%	Description all in software specific language
0%	No admin control information could not be found

How to improve this score

Create a document that covers the items described above in plain language that investors can understand. An [example](#) is enclosed.

22) Is there Pause Control documentation including records of tests (%)

 Answer: 0%

No evidence of a Pause Control function was found in their documentation, and no Pause Control tests were found in their GitHub repository.

Guidance:

100%	All the contracts are immutable or no pause control needed and this is explained OR
100%	Pause control(s) are clearly documented and there is records of at least one test within 3 months
80%	Pause control(s) explained clearly but no evidence of regular tests
40%	Pause controls mentioned with no detail on capability or tests
0%	Pause control not documented or explained

How to improve this score

Create a document that covers the items described above in plain language that investors can understand. An [example](#) is enclosed.

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : [@defisafety](#)

I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 and there I started [SecuEth.org](#) with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got [EthFoundation funding](#) to assist in their development.

Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

Scoring Appendix

	Total	Belt Finance	
	Points	Answer	Points
PQ Audit Scoring Matrix (v0.7)			
Code and Team	Total	260	146.5
1) Are the executing code addresses readily available? (%)	20	100%	20
2) Is the code actively being used? (%)	5	100%	5
3) Is there a public software repository? (Y/N)	5	Y	5
4) Is there a development history visible? (%)	5	30%	1.5
5) Is the team public (not anonymous)? (Y/N)	15	Y	15
Code Documentation			
6) Is there a whitepaper? (Y/N)	5	Y	5
7) Are the basic software functions documented? (Y/N)	10	Y	10
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	15	80%	12
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	5	0%	0
10) Is it possible to trace from software documentation to the implementation in code (%)	10	20%	2
Testing			
11) Full test suite (Covers all the deployed code) (%)	20	0%	0
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	5	100%	5
13) Scripts and instructions to run the tests? (Y/N)	5	N	0
14) Report of the results (%)	10	0%	0
15) Formal Verification test done (%)	5	0%	0
16) Stress Testing environment (%)	5	0%	0
Security			
17) Did 3rd Party audits take place? (%)	70	70%	49
18) Is the bug bounty acceptable high? (%)	10	0%	0
Access Controls			
19) Can a user clearly and quickly find the status of the admin controls	5	40%	2
20) Is the information clear and complete	10	60%	6
21) Is the information in non-technical terms	10	90%	9
22) Is there Pause Control documentation including records of tests	10	0%	0
Section Scoring			
Code and Team	50	93%	
Documentation	45	64%	
Testing	50	10%	
Security	80	61%	
Access Controls	35	49%	

Executing Code Appendix

[BETA (v1)]

Contract

Address

BELT	0xE0e514c71282b6f4e823703a39374Cf58dc3eA4f
MasterBelt	0xD4BbC80b9B102b77B21A06cb77E954049605E6c1
bDAIStratVLEV	0xd49CB5B097E9F0B51B3C61C5127A9c35BDeC7051
bDAI	0xFD 22e3bF935C1C94254F050BB e093563f533534
bDAIGov	0x224BF96bd2E506FA59B0D67d20AaB39a9574efA1
bUSDCStratVLEV	0xeD77Ce44feFE9D90b61e23c36250E9A7AD440a07
bUSDC	0x08BED6851CADc4EFc91147E3Ca63C39406B31a2D
bUSDCGov	0x6C1e403240D11e9514AD6C40cfA6Ee88a8a10739
bUSDTStratVLEV	0x8c680d7eC5C8B980bF8cD73001865B80eA7C629b
bUSDT	0x56A9452024AE2dEdB01e1179AcB1c152d50C0145
bUSDTGov	0xa6464E891FfD3A3a46ECC4BCEDf5EA8f6A90F76A
bBUSDStratVLEV	0xC31cf50C3559329ed83D87f09af3884E935f2873
bBUSD	0x7c8Dd1e39cD8142414f24f0bA80638b2E2fa5234
bBUSDGov	0x3F52620092CFBA45aE9a303eFEe0de8fa81E1A6D
BeltLPToken	0x86aFa7ff694Ab8C985b79733745662760e454169
StableSwapB	0xF16D312d119c13dD27fD0dC814b0bCdcaAa62dfD
DepositB	0xf157A4799bE445e3808592eDd7E7f72150a7B050
VaultBPool	0xeff8B733f12Ae6902409047c44AD3ee0Bf58f201
CakeVaultBPool	0x224172206bd6d089E6B16fFCD77876B1D092E5AF
ViewContract	0x9137a703756a931db7d2598Cb00E8A69B324D319
BeltRouter	0x70897189b10b5F145e9CF3384146a4bbA9914a72
BeltView	0x7B5c5Da87aF373F7382e59eEbee0d550D276de2c

v2 Contracts

4Belt Pool

Contract	Address
BeltLPToken	0x06b72E2016A0200059261a280C6550916f1D1404

DEIILP TOKEN

0x9CD73F20104E599930201C209ED3F904014D1404

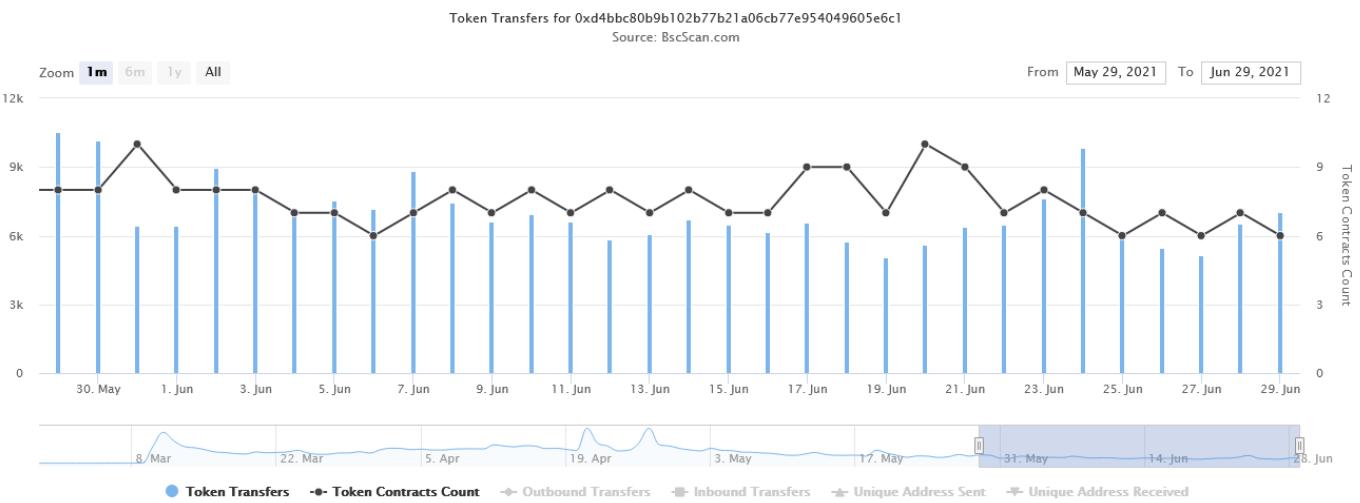
StableSwapB

0xAEA4f7dcd172997947809CE6F12018a6D5c1E8b6

DepositB

0xF6e65B33370Ee6A49eB0dbCaA9f43839C1AC04d5

Code Used Appendix



Example Code Appendix

```

1 pragma solidity ^0.6.12;
2
3 // import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/tol
4 abstract contract Context {
5     function _msgSender() internal view virtual returns (address payable) {
6         return msg.sender;
7     }
8
9     function _msgData() internal view virtual returns (bytes memory) {
10        this; // silence state mutability warning without generating bytecode - see https://
11        return msg.data;
12    }
13 }
14
15 library SafeMath {
16     /**
17      * @dev Returns the addition of two unsigned integers, reverting on
18      * overflow.
19      *
20      * Counterpart to Solidity's `+` operator.
21      *
22      * Requirements:
23      *
24      * - Addition cannot overflow.
25     */
26 }
```

```
--  
function add(uint256 a, uint256 b) internal pure returns (uint256) {  
    uint256 c = a + b;  
    require(c >= a, "SafeMath: addition overflow");  
    return c;  
}  
  
/**  
 * @dev Returns the subtraction of two unsigned integers, reverting on  
 * overflow (when the result is negative).  
 *  
 * Counterpart to Solidity's '-' operator.  
 *  
 * Requirements:  
 * - Subtraction cannot overflow.  
 */  
function sub(uint256 a, uint256 b) internal pure returns (uint256) {  
    return sub(a, b, "SafeMath: subtraction overflow");  
}  
  
/**  
 * @dev Returns the subtraction of two unsigned integers, reverting with custom message  
 * overflow (when the result is negative).  
 *  
 * Counterpart to Solidity's '-' operator.  
 *  
 * Requirements:  
 * - Subtraction cannot overflow.  
 */  
function sub(  
    uint256 a,  
    uint256 b,  
    string memory errorMessage  
) internal pure returns (uint256) {  
    require(b <= a, errorMessage);  
    uint256 c = a - b;  
  
    return c;  
}  
  
/**  
 * @dev Returns the multiplication of two unsigned integers, reverting on  
 * overflow.  
 *  
 * Counterpart to Solidity's '*' operator.  
 *  
 * Requirements:  
 * - Multiplication cannot overflow.  
 */  
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
```

```

79     // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
80     // benefit is lost if 'b' is also tested.
81     // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
82     if (a == 0) {
83         return 0;
84     }
85
86     uint256 c = a * b;
87     require(c / a == b, "SafeMath: multiplication overflow");
88
89     return c;
90 }
91
92 /**
93 * @dev Returns the integer division of two unsigned integers. Reverts on
94 * division by zero. The result is rounded towards zero.
95 *
96 * Counterpart to Solidity's `/` operator. Note: this function uses a
97 * `revert` opcode (which leaves remaining gas untouched) while Solidity
98 * uses an invalid opcode to revert (consuming all remaining gas).
99 *
100 * Requirements:
101 *
102 * - The divisor cannot be zero.
103 */
104 function div(uint256 a, uint256 b) internal pure returns (uint256) {
105     return div(a, b, "SafeMath: division by zero");
106 }
107
108 /**
109 * @dev Returns the integer division of two unsigned integers. Reverts with custom mes-
110 * sage if the result is negative or if the divisor is zero. The result is rounded towards zero.
111 *
112 * Counterpart to Solidity's `/` operator. Note: this function uses a
113 * `revert` opcode (which leaves remaining gas untouched) while Solidity
114 * uses an invalid opcode to revert (consuming all remaining gas).
115 *
116 * Requirements:
117 *
118 * - The divisor cannot be zero.
119 */
120 function div(
121     uint256 a,
122     uint256 b,
123     string memory errorMessage
124 ) internal pure returns (uint256) {
125     require(b > 0, errorMessage);
126     uint256 c = a / b;
127     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
128
129     return c;
130 }
```

```
131
132
133     /**
134      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer mod
135      * Reverts when dividing by zero.
136      *
137      * Counterpart to Solidity's `%` operator. This function uses a `revert`
138      * opcode (which leaves remaining gas untouched) while Solidity uses an
139      * invalid opcode to revert (consuming all remaining gas).
140      *
141      * Requirements:
142      *
143      * - The divisor cannot be zero.
144      */
145      function mod(uint256 a, uint256 b) internal pure returns (uint256) {
146          return mod(a, b, "SafeMath: modulo by zero");
147      }
148
149     /**
150      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer mod
151      * Reverts with custom message when dividing by zero.
152      *
153      * Counterpart to Solidity's `%` operator. This function uses a `revert`
154      * opcode (which leaves remaining gas untouched) while Solidity uses an
155      * invalid opcode to revert (consuming all remaining gas).
156      *
157      * Requirements:
158      *
159      * - The divisor cannot be zero.
160      */
161      function mod(
162          uint256 a,
163          uint256 b,
164          string memory errorMessage
165      ) internal pure returns (uint256) {
166          require(b != 0, errorMessage);
167          return a % b;
168      }
169
170  interface IERC20 {
171      /**
172      * @dev Returns the amount of tokens in existence.
173      */
174      function totalSupply() external view returns (uint256);
175
176      /**
177      * @dev Returns the amount of tokens owned by `account`.
178      */
179      function balanceOf(address account) external view returns (uint256);
180
181      /**
182      * @dev Moves `amount` tokens from the caller's account to `recipient`.
183      *
```

```

184     * Returns a boolean value indicating whether the operation succeeded.
185     *
186     * Emits a {Transfer} event.
187     */
188     function transfer(address recipient, uint256 amount)
189     external
190     returns (bool);
191
192    /**
193     * @dev Returns the remaining number of tokens that `spender` will be
194     * allowed to spend on behalf of `owner` through {transferFrom}. This is
195     * zero by default.
196     *
197     * This value changes when {approve} or {transferFrom} are called.
198     */
199     function allowance(address owner, address spender)
200     external
201     view
202     returns (uint256);

```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	70	19179	3420	2028	13731	1257

Comments to Code 2028/13731 = 15%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	0	0	0	0	0	0

Tests to Code 0/0 = 0%