Augur Process Quality Documentation

Score: 74%

This is a Process Quality Review on Augur completed on 8 January, 2021. It was performed using the Process Review process (version 0.6) and is documented here. The review was performed by ShinkaRex of Caliburn Consulting. Check out our Telegram.

The final score of the review is 74%, a pass. The breakdown of the scoring is in Scoring Appendix.

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**

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, 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. 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.

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? (Y/N)
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)

Are the executing code addresses readily available? (Y/N)

> Answer: Yes

The V2 addresses are not really published but we were were pointed to this internal deployment file ; https://github.com/AugurProject/augur/tree/dev/packages/augur-artifacts/src/environments. The addresses should be properly published. After discussion they are available at https://www.augur.net/blog/augur-v2-launch/. However neither location is clear or readily available.

How to improve this score

Make the Ethereum addresses of the smart contract utilized by your application available on either your website or your GitHub (in the README for instance). Ensure the addresses is up to

date. This is a very important question wrt to the final score.

Is the code actively being used? (%)

> Answer: 100%

There are many internal transactions on the AugurTrading contract 0x63A1eed178323C5eE0aD72fbD8a8cF1a7902881e.

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

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

> Answer: Yes

Location: https://github.com/AugurProject

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.

How to improve this score

Maintain a public repo, at least for deployed code. Public repo's are in line with the vision of Ethereum where development is shared and public.

Is there a development history visible? (%)

> Answer: 100%

with 42,826 commits and 688 branches, this is a healthy 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

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

> Answer: Yes

The names of the team members are in the WhitePaper.

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 seems a No.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

1. Is there a whitepaper? (Y/N)
2. Are the basic software functions documented? (Y/N)
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)

4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)
5. Is it possible to trace from software documentation to the implementation in codee (%)

Is there a whitepaper? (Y/N)

> Answer: Yes

Location: https://augur.net/whitepaper.pdf

How to improve this score

Ensure the white paper is available for download from your website or at least the software repository. Ideally update the whitepaper to meet the capabilities of your present application.

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

> Answer: No

There was clearly some software function documentation done for V1 of Augur, but this has been removed or broken during the V2 upgrade.

A good, but incomplete docs exist at: https://docs.augur.sh/docs/contracts/overview#dispute-bond however these are not available from the Augur website, yet, so they do not yet count.

How to improve this score

Write the document based on the deployed code. For guidance, refer to the SecurEth System Description Document.

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

⚠ Answer: 0%

There was clearly some software function documentation done for V1 of Augur, but this has been removed or broken during the V2 upgrade.

Given the documentation as of writing a score of 70% is valid using the information in https://docs.augur.sh/docs/contracts. However these are not available from the Augur website, yet, they do not yet count

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.

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

ⓘ Answer: 30%

There are not many useful comments on their code. their SLOC is a low value, with only 17% of the code having significant comments.

Code examples are in the Appendix. As per the SLOC, there is 17% 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.

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

> ⚠ Answer: 0%

There was clearly some software function documentation done for V1 of Augur, but this has been removed or broken during the V2 upgrade.

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;

1. Full test suite (Covers all the deployed code) (%)
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)
3. Scripts and instructions to run the tests (Y/N)
4. Packaged with the deployed code (Y/N)
5. Report of the results (%)
6. Formal Verification test done (%)
7. Stress Testing environment (%)

Is there a Full test suite? (%)

> Answer: 100%

There is clearly a full set of tests, the TtC ratio is 175%.

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.

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

> Answer: 50%

Augur V1 featured a coverage report, but this has been removed or broken during the V2 upgrade.

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

How to improve this score

This score can improve by adding tests achieving full code coverage. A clear report and scripts in the software repository will guarantee a high score.

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

> Answer: Yes

Location: https://github.com/AugurProject/augur/tree/master/packages/augur-core

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.

Packaged with the deployed code (Y/N)

> Answer: Yes

Location: https://github.com/AugurProject/augur/tree/master/packages/augur-core

How to improve this score

Improving this score requires redeployment of the code, with the tests. This score gives credit to those who test their code before deployment and release them together. If a developer adds tests after deployment they can gain full points for all test elements except this one.

Report of the results (%)

Answer: 0%

There is no evident report of the results.

How to improve this score

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

Formal Verification test done (%)

Answer: 0%

There is no evidence of formal verification testing having been done.

Stress Testing environment (%)

Answer: 100%

https://github.com/AugurProject/augur/blob/dev/packages/augur-artifacts/src/environments/staging.json

The deployment of the kovan test net is defined in the enclosed file.

Audits

Answer: 90%

After discussion with the developers, we found two audits. One is public, but not obvious. The other is public but not accessible without prior knowledge.

The issues list in GitHub has an audit and security label set. When filtering as per these labels and audit by Micah Zoltu (a well known author) and others clearly took place. This drives the score to 90%.

Micah is quite public with the opinion that audits, since they are written for a private developer audience should not be made public and used to justify trust and security in a protocol. This opinion has merits but we at DeFiSafety do not agree. Given Micah's opinions we are not surprised that these audit results are only found in the issues of the GitHub.

The second audit by open zeppelin is on the open zeppelin website, and therefore public, but is not referenced in the Augur documentation. This is possibly for the reasons stated previously. As the audit cannot be found by any third party (unless they start searching all of the auditor's databases) we will not add to the score. In addition, the audit listed a high number of critical and high severity items. Almost all were fixed. The audit also mentioned the limited documentation, which we also found.

Guidance:

1. Multiple Audits performed before deployment and results public and implemented or not required (100%)
2. Single audit performed before deployment and results public and implemented or not required (90%)
3. Audit(s) performed after deployment and no changes required. Audit report is public. (70%)
4. No audit performed (20%)
5. Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, question 1 (0%)

Appendices

Author Details

The author of this review is Rex of Caliburn Consulting.

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.

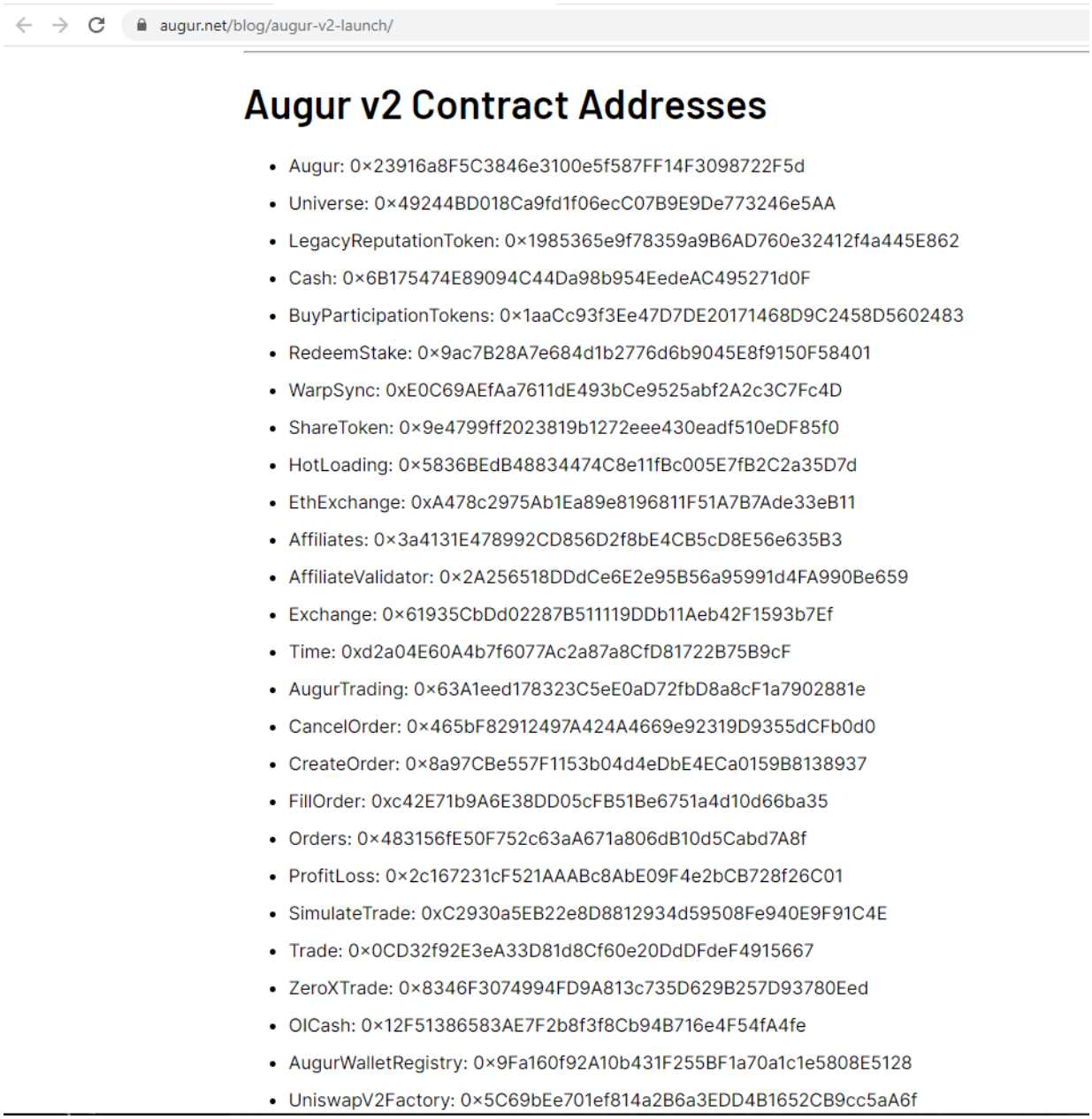
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

Career wise I am a business development manager for an avionics supplier.

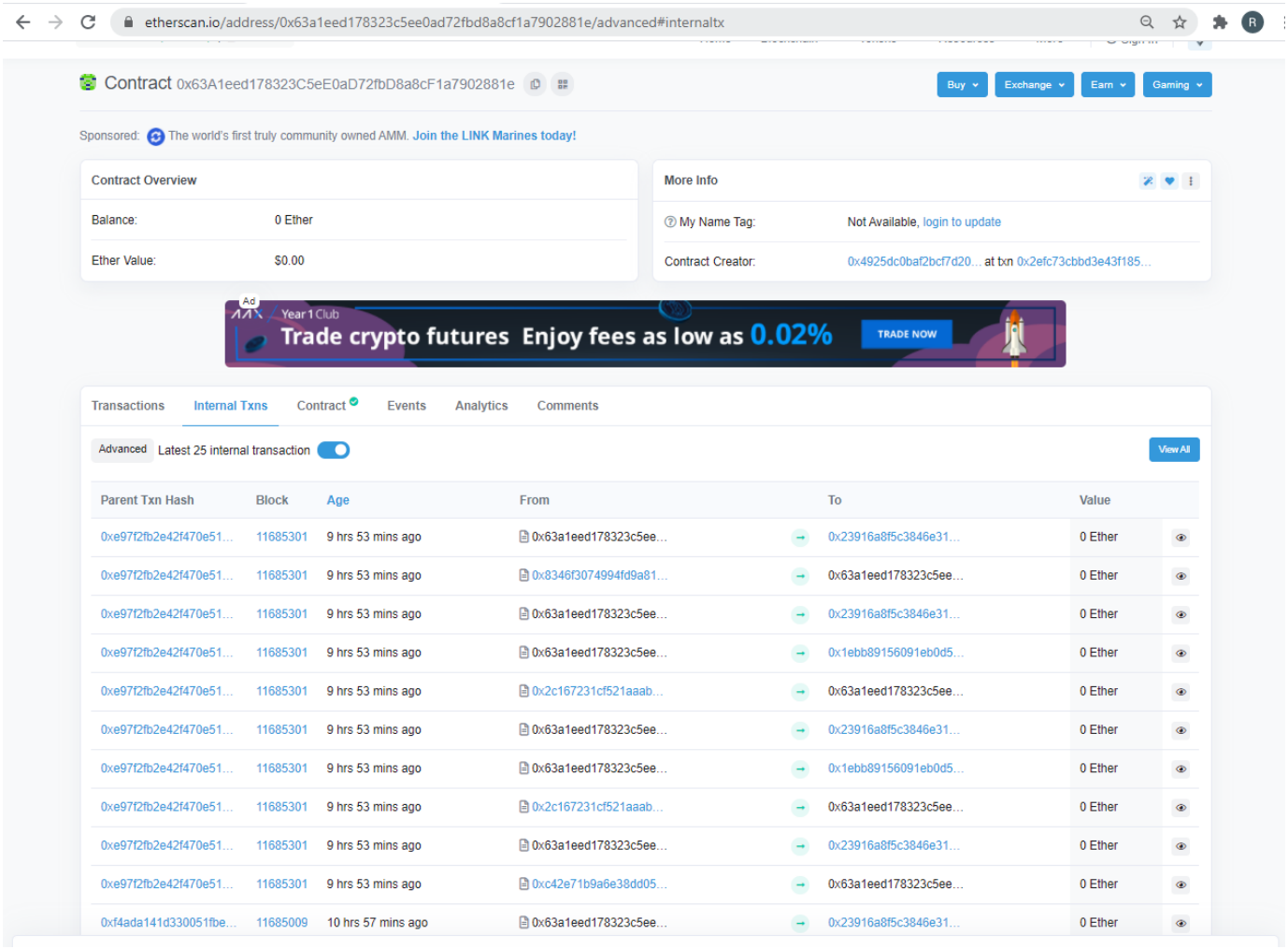
Scoring Appendix

	Total	Augur	
PQ Audit Scoring Matrix (v0.6)	**Points**	**Answer**	**Points**
Total	**240**		178.5
Code and Team			**74%**
2. Are the basic software functions documented? (Y/N)	**10**	N	0
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	0%	0
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**10**	30%	3
5 Is it possible to trace from software documentation to the implementation in code (%)	**5**	0%	0
Testing			
1. Full test suite (Covers all the deployed code) (%)	**20**	100%	20
2. Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	50%	2.5
3. Scripts and instructions to run the tests? (Y/N)	**5**	Y	5
4. Packaged with the deployed code (Y/N)	**5**	Y	5
5. Report of the results (%)	**10**	0%	0
6. Formal Verification test done (%)	**5**	0%	0
7. Stress Testing environment (%)	**5**	100%	5
Audits			
Audit done	**70**	90%	63
Section Scoring			
Code and Team	70	100%	
Documentation	45	18%	
Testing	55	68%	
Audits	70	90%	
Audit Number		58	
Date			
Private Repo		Y	
Version		0.6	

Executing Code Appendix



Code Used Appendix



Example Code Appendix

```
// Copyright (C) 2015 Forecast Foundation OU, full GPL notice in LICENSE

// Bid / Ask actions: puts orders on the book
// price is denominated by the specific market's numTicks
// amount is the number of attoshares the order is for (either to buy or to
// price is the exact price you want to buy/sell at [which may not be the c

pragma solidity 0.5.15;


import 'ROOT/IAugur.sol';
import "ROOT/ICash.sol";
import 'ROOT/libraries/math/SafeMathUint256.sol';
import 'ROOT/reporting/IMarket.sol';
import 'ROOT/trading/IAugurTrading.sol';
import 'ROOT/trading/IOrders.sol';
import 'ROOT/reporting/IShareToken.sol';
import 'ROOT/libraries/token/IERC20.sol';


// CONSIDER: Is `price` the most appropriate name for the value being used?
library Order {
```

```
23      using SafeMathUint256 for uint256;
24
25      enum Types {
26          Bid, Ask
27      }
28
29      enum TradeDirections {
30          Long, Short
31      }
32
33      struct Data {
34          // Contracts
35          IMarket market;
36          IAugur augur;
37          IAugurTrading augurTrading;
38          IShareToken shareToken;
39          ICash cash;
40
41          // Order
42          bytes32 id;
43          address creator;
44          uint256 outcome;
45          Order.Types orderType;
46          uint256 amount;
47          uint256 price;
48          uint256 sharesEscrowed;
49          uint256 moneyEscrowed;
50          bytes32 betterOrderId;
51          bytes32 worseOrderId;
52      }
53
54      function create(IAugur _augur, IAugurTrading _augurTrading, address _cre
55          require(_outcome < _market.getNumberOfOutcomes(), "Order.create: Ou
56          require(_price != 0, "Order.create: Price may not be 0");
57          require(_price < _market.getNumTicks(), "Order.create: Price is out
58          require(_attoshares > 0, "Order.create: Cannot use amount of 0");
59          require(_creator != address(0), "Order.create: Creator is 0x0");
60
61          IShareToken _shareToken = IShareToken(_augur.lookup("ShareToken"));
62
63          return Data({
64              market: _market,
65              augur: _augur,
66              augurTrading: _augurTrading,
67              shareToken: _shareToken,
68              cash: ICash(_augur.lookup("Cash")),
69              id: 0,
70              creator: _creator,
71              outcome: _outcome,
72              orderType: _type,
73              amount: _attoshares,
74              price: _price,
75              sharesEscrowed: 0,
76              moneyEscrowed: 0,
77              betterOrderId: _betterOrderId,
```

```
            worseOrderId: _worseOrderId
        });
    }

    //
    // "public" functions
    //

    function getOrderId(Order.Data memory _orderData, IOrders _orders) inte
        if (_orderData.id == bytes32(0)) {
            bytes32 _orderId = calculateOrderId(_orderData.orderType, _orde
            require(_orders.getAmount(_orderId) == 0, "Order.getOrderId: Ne
            _orderData.id = _orderId;
        }
        return _orderData.id;
    }

    function calculateOrderId(Order.Types _type, IMarket _market, uint256 _
        return sha256(abi.encodePacked(_type, _market, _amount, _price, _se
    }

    function getOrderTradingTypeFromMakerDirection(Order.TradeDirections _c
        return (_creatorDirection == Order.TradeDirections.Long) ? Order.Ty
    }

    function getOrderTradingTypeFromFillerDirection(Order.TradeDirections _
        return (_fillerDirection == Order.TradeDirections.Long) ? Order.Type
    }

    function saveOrder(Order.Data memory _orderData, bytes32 _tradeGroupId,
        getOrderId(_orderData, _orders);
        uint256[] memory _uints = new uint256[](5);
        _uints[0] = _orderData.amount;
        _uints[1] = _orderData.price;
        _uints[2] = _orderData.outcome;
        _uints[3] = _orderData.moneyEscrowed;
        _uints[4] = _orderData.sharesEscrowed;
        bytes32[] memory _bytes32s = new bytes32[](4);
        _bytes32s[0] = _orderData.betterOrderId;
        _bytes32s[1] = _orderData.worseOrderId;
        _bytes32s[2] = _tradeGroupId;
        _bytes32s[3] = _orderData.id;
        return _orders.saveOrder(_uints, _bytes32s, _orderData.orderType, _
    }
}
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	104	11099	1696	1412	7991	1141

Comments to Code 1412/ 7991 = 17%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
Python	73	16627	1894	724	14006	528

Tests to Code 14006 / 7991 = 175%