Process Quality Review Process

Version 0.7 May 2021

Differences between previous versions

Overview

This document describes the process for a PQ Review (Process Quality) on a deployed smart contract, version 0.7.3 The focus of this process is on DeFi contracts but it could be applied to any smart contract.

The intent is generation of a simple quality score for the smart contract application being reviewed. It will indicate the overall quality of the development process and the documents that the process generated. The readers can dig into the details of the review in order to learn how the score was generated. Every step will be documented and available for review.

The basic premise of these reviews is that developers following and documenting a good software development process should have secure code and maintain security that users can trust. For blockchain developments a good process should have public readily auditable documents and other traces, making the process clear.

These reviews are developed and focused towards released operating smart contract applications. The initial focus is on DeFi applications as these bring in many new users investing significant sums that are trusted by the smart contracts.

PQ reviews are initially done *without* the organization’s permission or support using exclusively documents publicly available such as the website, the software repository of the code (GitHub, etc), and the code available from Etherscan. After the initial audit report is generated, it will be presented to the developers of the reviews organization for correction or improvement. The results of these corrections will be clear and publicly documented in a new version of the report.

The author has no commercial arrangement with the organizations being audited. **These reviews are not funded by the organization being reviewed.** The author is aiming for community support to review as a public good.

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 access controls and strategies**

Aerospace

The author of this process has a long history in the avionics industry. Aerospace software development has always maintained that a rigorous and frequently audited software development process leads to safe and secure software that can be supported for many decades. The avionics process is DO-178C. It is significantly more rigorous than the expectation of this review process, however the steps used an overarching philosophy has guided the specification of this review process.

Aerospace Requirements

For more detail, this presents the software and system requirements for aerospace code certification in an extremely simplified format. It makes a useful comparison.

1. All System Requirements documented in proper requirements diction
2. There is documented traceability from each system requirement to software

requirements or low level requirements
3. All software requirements (or low-level requirements/comments) exist for each part of software code
4. There is documented traceability from the low level requirements to the software code they cover
5. Every piece of software code is covered by both unit tests and system tests. If a unit test covers the requirements of the system test, the single test suffices.
6. There is documented traceability from the software to the tests
7. They were requirement review meetings held according to a documented process to review every requirement
8. There are documented software requirement reviews held according to a documented process for each software requirement
9. There are test reviews held according to a documented process for each test
10. When there is a requirement to change software code for which already reviewed requirements, code or test must change than a change impact analysis document must review the changes and recommend which reviews must be held again on the requirements, software and tests
11. During the audit any test can be chosen by the auditor and the auditor can review traceability from the test through the code to the requirements with reviews for each step in the process

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.

Sections of the Review Process

The process breaks the scores into the following sections;

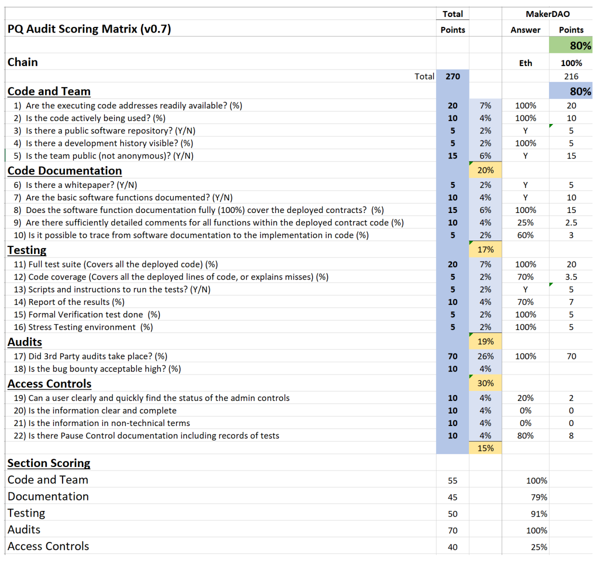
- Code and Team -- Executing Code Verification
- Documentation -- Software Documentation for the Executing Code
- Test --Overall test strategy for Executing Code
- Security -- Review of Software Security Audits and Bug Bounty
- Access Controls -- Review of the public information about admin access controls

Scoring

The final review score is indicated as a percentage. The percentage is calculated as (Achieved Points / (Total Possible Points). For each element the answer can be either Yes/No or a percentage. For each element there is a "Scoring Weight". The element points achieved is the scoring weight times the answer. The Achieved Points is the sum of every element's points.

Scoring Weight: xx

Example Scoring Matrix:



	Total		MakerDAO	
PQ Audit Scoring Matrix (v0.7)	**Points**		**Answer**	**Points**
				80%
Chain			**Eth**	**100%**
Total	**270**			216
Code and Team				**80%**
1) Are the executing code addresses readily available? (%)	**20**	7%	100%	20
2) Is the code actively being used? (%)	**10**	4%	100%	10
3) Is there a public software repository? (Y/N)	**5**	2%	Y	5
4) Is there a development history visible? (%)	**5**	2%	100%	5
5) Is the team public (not anonymous)? (Y/N)	**15**	6%	Y	15
Code Documentation		20%		
6) Is there a whitepaper? (Y/N)	**5**	2%	Y	5
7) Are the basic software functions documented? (Y/N)	**10**	4%	Y	10
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	6%	100%	15
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**10**	4%	25%	2.5
10) Is it possible to trace from software documentation to the implementation in code (%)	**5**	2%	60%	3
Testing		17%		
11) Full test suite (Covers all the deployed code) (%)	**20**	7%	100%	20
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)	**5**	2%	70%	3.5
13) Scripts and instructions to run the tests? (Y/N)	**5**	2%	Y	5
14) Report of the results (%)	**10**	4%	70%	7
15) Formal Verification test done (%)	**5**	2%	100%	5
16) Stress Testing environment (%)	**5**	2%	100%	5
Audits		19%		
17) Did 3rd Party audits take place? (%)	**70**	26%	100%	70
18) Is the bug bounty acceptable high? (%)	**10**	4%		
Access Controls		30%		
19) Can a user clearly and quickly find the status of the admin controls	**10**	4%	20%	2
20) Is the information clear and complete	**10**	4%	0%	0
21) Is the information in non-technical terms	**10**	4%	0%	0
22) Is there Pause Control documentation including records of tests	**10**	4%	80%	8
		15%		
Section Scoring				
Code and Team	55		100%	
Documentation	45		79%	
Testing	50		91%	
Audits	70		100%	
Access Controls	40		25%	

Private Software Repos

Some development teams, especially in DeFi prefer a private repo to reduce the ability for an easy fork of their development. We clearly understand the business incentive for this and want to allow developers to keep the repo private and still get a good score.

Normally the software repo also holds the tests. If the repo is private, it is up to the developers to publicly show that they have a full test suite in order to get the score.

The developers will be penalized by not having a public repo, but not significantly. There are two questions "Is there a private repo?" and "Is there evidence of steady development?" which without a public repo you cannot score well.

Audits are also a special case. With a public repo, anyone can check the differences between the audited and deployed code because all the information is publicly available. Assuming the audit takes place on code that cannot be seen publicly, then 25% is deducted from the usual score. So a 100% become 75% and 90% become 65%. If the auditing firm indicates that there audit report is relevant to the deployed code, then full marks are regained.

Chain

This section indicates the blockchain used by this protocol. The chains are explained in this document. You must pick one of the Chains in the Multi Blockchain Description Document and fill in the blank.

Chain: _____

Code and Team

Any PQ review starts with the code being executed and used by the application. It answers and generates a score on the following 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? (%)

Scoring weight: 20

Are the addresses of the deployed contract on the mainnet *clearly* available on the public documents for the service? **This is a very important question as it impacts the audits score also. It is virtually impossible to have a passing score with a zero score.** All the contract addresses must be visible, not just the token address. This includes AMM strategies that may update regularly. The addresses can be over multiple pages as long as one page has the latest addresses or links to their locations. **Each contract address must have the software visible in Etherscan. Bytecode only for the contract is equivalent to no address.**

Why? Because if a user cannot easily see the contracts he is using (even if he does not understand Solidity) then how can he trust that protocol?

An image of the web page with the addresses will be recorded as part of the review. Readily available means clearly labelled on the website or on a page on the GitBook or in the readme of the software repository.

If the contract address is found through alternate, unclear means (such as in an audit report), then this score will be 20% but the rest of the review will take place using the addresses. Audits will keep their score.

If the addresses are not found, the rest of the audit takes place without code. **The audit results will be zero, even if an audit is available as there is no confidence on the code that is executed wrt to the code audited.** The test and documentation sections will receive scores based on the software repository contents as these indicate the care of the developers.

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

How to improve this score

Make the ethereum addresses of the smart contract utilized by your application available on either your website, Gitbook or your github (in the README for instance). Ensure the addresses are up to date.

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

Scoring weight: 5

This looks at the transaction history of the contract through Etherscan. The results are scored as indicated below. The proof is documented in an appendix. The reviewer should choose an important contract, not the token, that is executed regularly.

The chart for the appendix is generated by searching the address, clicking Analytics, then Transactions (for the Transaction chart) and scale it to one month. Display internal transactions where appropriate.

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)

Scoring weight: 5

This checks if there is a public software repository. If it is visible, even one just made for deployment, then this scores as Yes. For teams with not public software repository, then No.

How to improve this score

Ensure you contracts and tests are available for viewing on a public software repository (like GitHub). The link can be from the website or GitBook documentation.

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

Scoring Weight: 5

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

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

Scoring Weight: 15

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

The documentation section describes the quality of the system and software documentation and commenting. This version of the Process Audit standard requests only basic documentation. For perspective, the aerospace software requirements are mentioned below.

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)

Scoring weight: 5

This is simple. Is there a white paper or other basic description of what the project is doing referenced by either the web site or the GitHub. Yes/No. We accept Medium articles explaining the application.

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.

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

Scoring weight: 10

This is also simple but very important. Are the basic software functions of the smart contracts documented in either the website, GitBook or the GitHub? Yes/No. This document says what the application does in some technical detail. The information may be in the white paper but this requires more specific technical detail than the white paper score requires.

How to improve this score

Write the document based on the deployed code. For guidance, refer to the SecurEth System Description Document for guidance. This document can be written after deployment.

8) Does the software function documentation fully (100%)

cover the deployed contracts? (%)

Scoring weight: 15

This score requires documentation specifically on the contract code. Generalized math formulas or state diagrams without directly referencing the code do not count. The goal for a 100% score is for documentation (in either the website, GitBook or the GitHub) that cover all of the developed source code. It does not have to cover public libraries. The requirements do not need rigid requirement diction (such as a "shall" in every requirement). For guidance, refer to the SecurEth System Description Document.

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? (%)

Scoring weight: 5

Software requirements are not the same as system requirements (that are referenced in the previous questions). Software requirements refer specifically to the code implementation. It is an additional layer of documentation within the code and must exclusively be in comments in the source file. Ideally the comments have fixed formatting giving consistent comments for all authors in all source files for the project. For guidance refer to the SecurEth Software Requirements.

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

For this initial version of the Process Audit process the auditor qualitatively determines the number between 0 and 100 and include example source in the appendices to justify the score.

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 (%)

Scoring weight: 10

Traceability is a documented link between the software and the code. It can be code snippets withing the docs, a simple identifier (such as a shortened content hash) that connects a text requirement to the code it implements. Since requirements are usually in separate documents from the software requirements/comments traceability links each requirement to the code that implements it. Ideally there are requirements for every piece of code, 100% requirement traceability. For reference, check the SecurEth guidelines on traceability.

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 covers the test suite that the deployed passed before its deployment on the mainnet. Ideally the suite is in the same repository, completely covers all code as the deployed contracts and has a report indicating a successful run. Additional testing can include Formal Verification and active test/stress environments.

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) Full test suite (Covers all the deployed code) (%)

Scoring weight: 20

Does the deployed code have a suite of tests and scripts to run them? Do the tests allow for comprehensive testing of the code? Are there both system and unit tests? It is better to test after deployment than never test at all. Testing is described in the SecurEth guidelines. Unit tests are written on a file by file basis and generally are used for code coverage. System tests are for functionality, testing usage of the

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) (%)

Scoring weight: 5

Here we consider if the unit tests fully cover the code. Ideally there is a report of a code coverage run with results in the repository. The normal goal is 100% code coverage. Without a report, the author determines a percentage based on the test suite percentage, artifacts in the test scripts and qualitative estimation. If there are any departures, there should be documentation on why certain pieces of code cannot be tested. This should be minimal and well-argued.

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.

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

Scoring weight: 5

Can 3rd parties run the full test suite using well documented scripts in the deployed software repository? Yes/No. If the scripts and tests are in a similar repository with the same code this is acceptable (but not ideal). This is a difficult score to get with a private repo.

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 (%)

Scoring weight: 10

Guidance: 100% - Detailed test report as described below 70% - GitHub Code coverage report visible 0% - No test report evident

Ideally there is a test report in the repository of the code that was deployed. Surprisingly most repositories don't have one. Most testing tools make tests that only report errors. No output means a pass. When auditing deployed code this is not informative. A test report also indicates what is not covered and why. Often there are corner cases of code that are difficult or impossible to test. A report indicates this with a brief explanation.

If you have a private repo (which means your tests are not public) a report becomes vital. A good test report can show that you have a full test suite (Q1), have code coverage (Q2). If it shows the script information, then you will get marks for Q3 and of course Q5 on the report. This stops the testing being very low.

Code coverage proof actually needs a report. The output should indicate the successful coverage, on what code and using what options.

The output is a percentage. This allows the author to "grade" the report, if it exists.

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 (%)

Scoring weight: 5

Formal verification is a specific process of testing software processes that is well suited to some blockchain software. Its use is still limited, but it adds value in some cases. For this reason the weight is lower than other elements. The output is a percentage. This allows the author to "grade" the report, if it exists.

16) Stress Testing environment (%)

Scoring weight: 5

A Stress testing environment is an active test environment (not the active mainnet) that is used for testing limits of the application. This would run on a testnet with simulated inputs. By maintaining an up to date test environment, the developers can test possible limit cases as they develop. It adds another layer of protection. The output is a percentage determined by the author.

Security

This section looks at the security aspects including 3rd party software audits and bug bounties. It is explained in this document. This section answers the following questions;

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

Scoring weight: 70

Smart contract audits are typically an indicator of quality in the blockchain world. The intent is to have 3rd party blockchain software experts review your code and tests to look for overall quality and the subtle blockchain specific weaknesses that could be utilized by an attacker.

If the smart contract address on the mainnet are not found or if the addresses are found but the code is hidden as byte code, the **audit results will be zero**, even if an audit is available as there is no confidence on the code that is executed wrt to the code audited.

If the quality of the report does not reflect an intelligent audit obviously done on the code then the authors reserve the right to reduce the score up to 0%. This attempts to cover valueless documents that say Audit and PASS but are not real audits.

The authors also reserve the right to reduce audit scores if they do not cover some economic issues. A smart contract audit that covers solidity well but ignores financial risks specific to DeFi has limited value (but not zero).

With a public repo, anyone can check the differences between the audited and deployed code because all the information is publicly available. Assuming the audit takes place on code that cannot be seen publicly, then 25% is deducted from the usual score. So a 100% become 75% and 90% become 65%. If the auditing firm indicates that there audit report is relevant to the deployed code, then full marks are regained.

For this reason the score multiplier is high; 70.

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 are needed but not implemented
20% No audit performed
0% Audit Performed after deployment, existence is public, report is not public OR smart contract address' not found, question

Deduct 25% if audited code not available for comparison.

18) Is the bounty value acceptably high (%)

Scoring Weight: 10

This section checks the value of the bug bounty program, if it exists. First, a score of 0% results if there is no bug bounty program. Based on discussions with Immunifi we have given a bias towards very high bug bounty programs and active programs as it will improve safety of the system. Active program means a third party (such as Immunifi) actively driving hackers to the site. Inactive program would be static mention on the docs.

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

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;

1. Can a user clearly and quickly find the status of the admin controls?
2. Is the information clear and complete?
3. Is the information in non-technical terms?
4. Is there Pause Control documentation including records of tests?

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

Scoring weight: 5

This question answers the ease of which a user can find the Access control documentation. Is it clearly labeled? Is it where it user expects it to be?

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 (%)

Scoring weight: 10

This question looks in the information supply about the access controls. Is it clear which parts of the protocol are upgradable or immutable. The ownership of the upgradable contracts should be specified and finally the capabilities for upgrade or changes in variables should be described. The only way to get 100% in this question is to have immutable contracts. Implicitly this means that upgradable contracts are less than ideal for DeFi safety.

Guidance: All the contracts are immutable -- 100% OR

All contracts are clearly labelled as upgradeable (or not) -- 30% AND The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND 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 (%)

Scoring weight: 10

Upgradable contracts allow the admins to make changes that will impact the investments of the protocol users. Users should understand how these changes can impact their investments. This question grades the quality of the explanations with higher scores given to plain language descriptions relevant to the investments that require no special software background. Of course, as in the question above the only way to get a perfect score is to have immutable contracts.

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 (%)

Scoring weight: 10

A Pause control allows the admin to freeze aspects of a protocol. If a hack is underway, this pause can protect the investors.

If no pause control exists, then the documentation should state this, preferably with reasoning.

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.