PieDAO Process Quality Review

Score 77%

This is a PIEDAO Process Quality Review completed on 3 May 2021. It was performed using the Process Review process (version 0.6.2) and is documented here. The review was performed by ShinkaRex of DeFiSafety. Check out our Telegram.

The final score of the review is 85%, a strong pass. 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**

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

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

They are available at website https://docs.piedao.org/technical/deployed-smart-contracts as indicated in the Appendix.

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: 70%

Activity is *3* transactions a day on contract DEFI+S Pie, 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

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

✓ Answer: Yes

GitHub: https://github.com/pie-dao

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.

Is there a development history visible? (%)

> ✓ Answer: 100%

With 217 commits 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

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.

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

> ✓ Answer: Yes

The team names are on the GitHub clearly.

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;

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://docs.piedao.org/

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: Yes

In the Dev Docs there are well-documented software functions.

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: 80%

All of the major functions are documented.

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: 29%

Code examples are in the Appendix. As per the SLOC, there is 29% 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: 60%

There is a clear association between the code and the documentation via non-explicit 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 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%

With a TtC of 215%, there is clearly a robust test suite

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: 98%

There is code coverage indication in the Quantstamp audit.

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: No

There are no apparent scripts and installations to run the tests.

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

The tests are packaged with the deployed code.

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.

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.

Formal Verification test done (%)

⚠ Answer: 0%

thereis no evident formal verification testing having been done

Stress Testing environment (%)

✓ Answer: 100%

They have contracts deployed on the Gorli testnet.

Audits

✓ Answer: 100%

PieDAO has had 4 audits preformed on the protocol, as seen on their GitHub. There were concerns about too much assembler making the code difficult to maintain and many issues found (though largely resolved).

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 DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

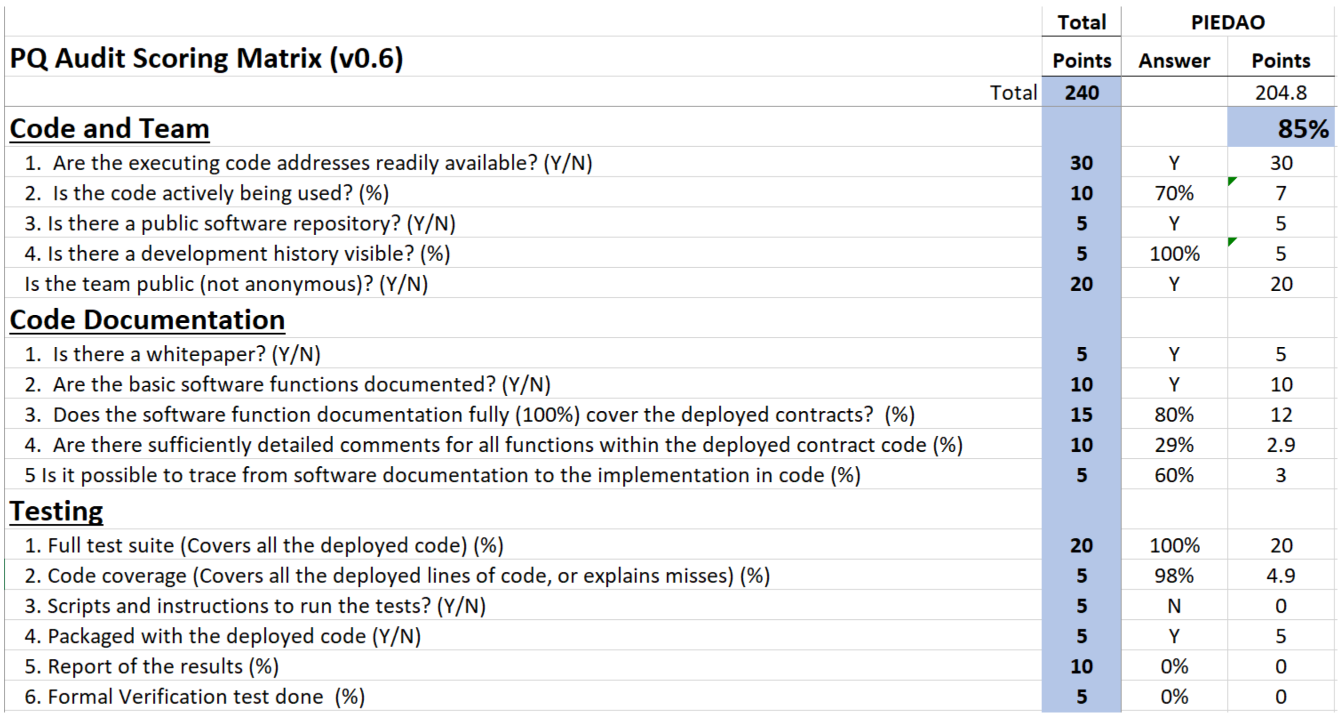
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

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.

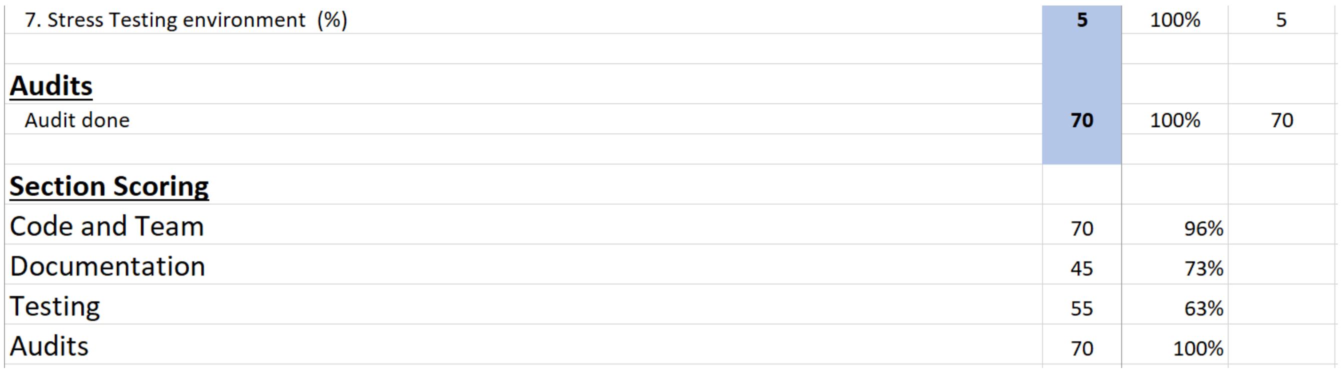
Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

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

Scoring Appendix

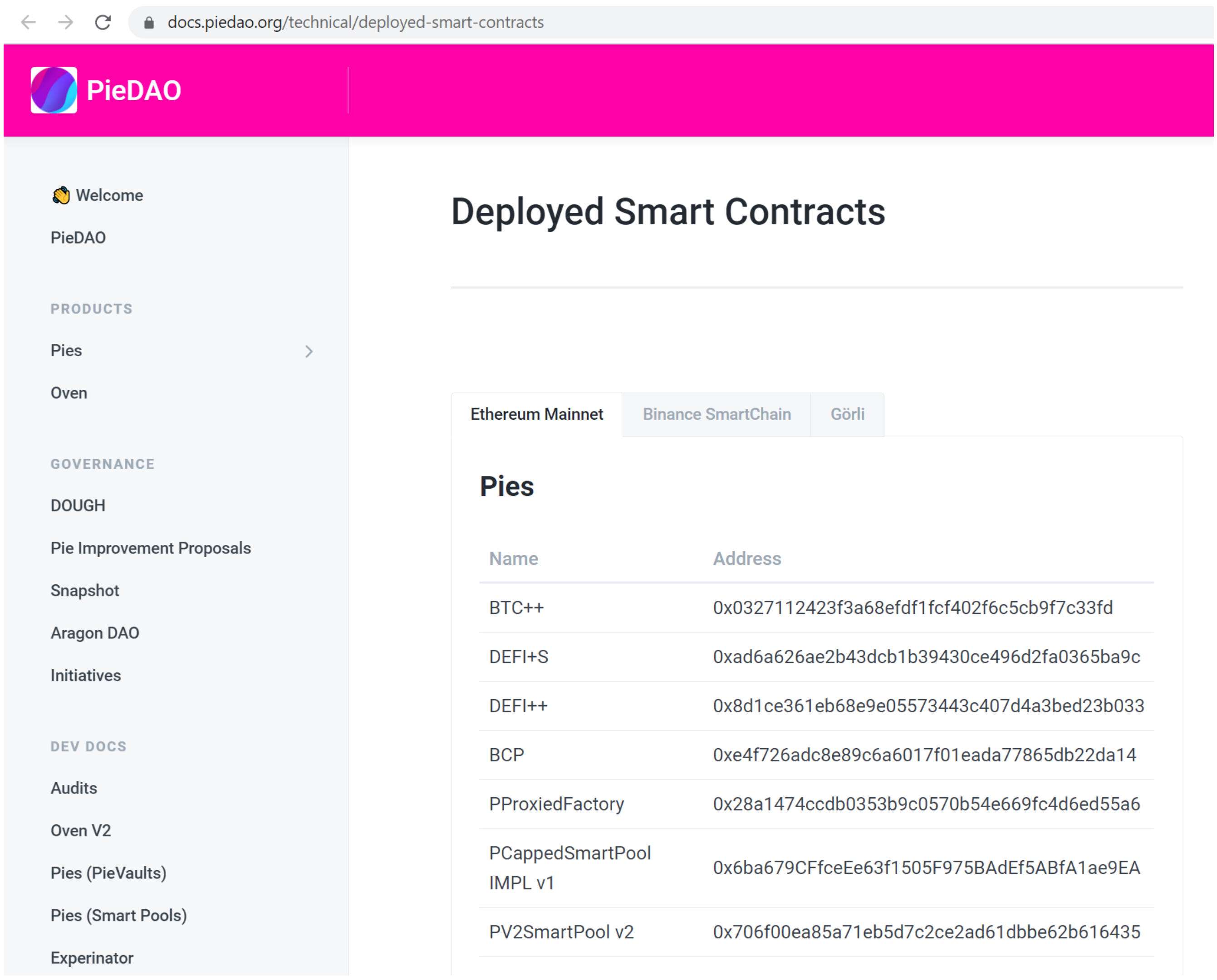


	Total	PIEDAO	
PQ Audit Scoring Matrix (v0.6)	**Points**	**Answer**	**Points**
Total	**240**		204.8
Code and Team			**85%**
1. Are the executing code addresses readily available? (Y/N)	**30**	Y	30
2. Is the code actively being used? (%)	**10**	70%	7
3. Is there a public software repository? (Y/N)	**5**	Y	5
4. Is there a development history visible? (%)	**5**	100%	5
Is the team public (not anonymous)? (Y/N)	**20**	Y	20
Code Documentation			
1. Is there a whitepaper? (Y/N)	**5**	Y	5
2. Are the basic software functions documented? (Y/N)	**10**	Y	10
3. Does the software function documentation fully (100%) cover the deployed contracts? (%)	**15**	80%	12
4. Are there sufficiently detailed comments for all functions within the deployed contract code (%)	**10**	29%	2.9
5 Is it possible to trace from software documentation to the implementation in code (%)	**5**	60%	3
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**	98%	4.9
3. Scripts and instructions to run the tests? (Y/N)	**5**	N	0
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**	100%	70
Section Scoring			
Code and Team	70	96%	
Documentation	45	73%	
Testing	55	63%	
Audits	70	100%	

Executing Code Appendix



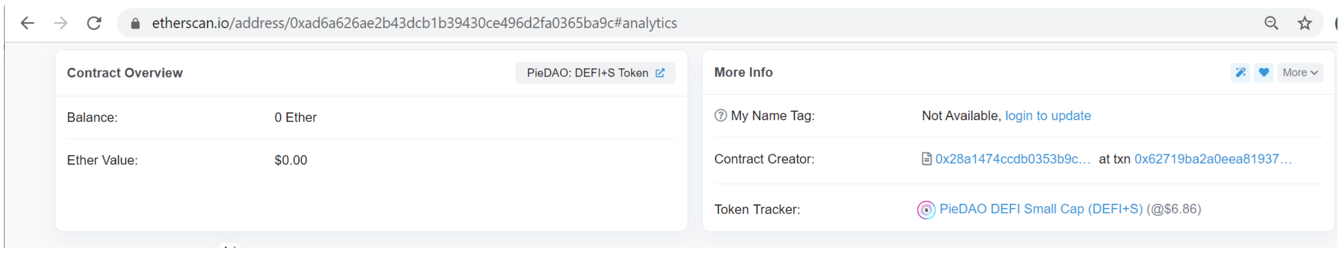
docs.piedao.org/technical/deployed-smart-contracts

Deployed Smart Contracts

Pies

Name	Address
BTC++	0x0327112423f3a68efdf1fcf402f6c5cb9f7c33fd
DEFI+S	0xad6a626ae2b43dcb1b39430ce496d2fa0365ba9c
DEFI++	0x8d1ce361eb68e9e05573443c407d4a3bed23b033
BCP	0xe4f726adc8e89c6a6017f01eada77865db22da14
PProxiedFactory	0x28a1474ccdb0353b9c0570b54e669fc4d6ed55a6
PCappedSmartPool IMPL v1	0x6ba679CFfceEe63f1505F975BAdEf5ABfA1ae9EA
PV2SmartPool v2	0x706f00ea85a71eb5d7c2ce2ad61dbbe62b616435

Code Used Appendix



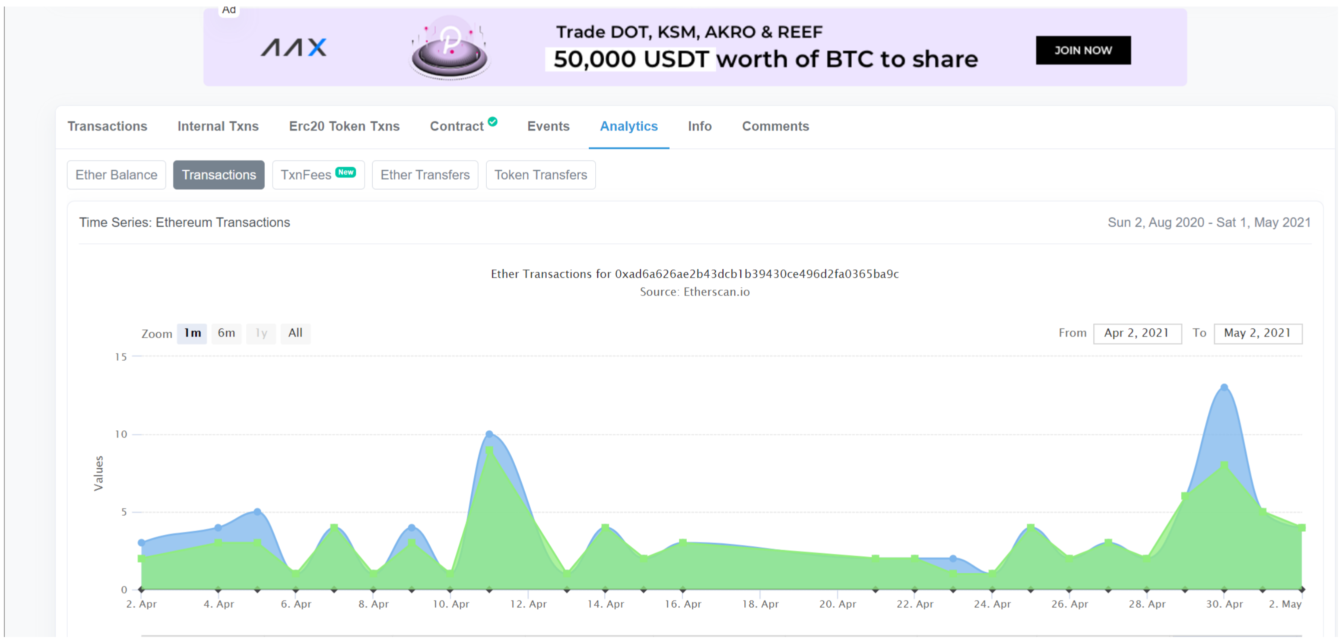
etherscan.io/address/0xad6a626ae2b43dcb1b39430ce496d2fa0365ba9c#analytics

Contract Overview — PieDAO: DEFI+S Token

Balance:	0 Ether
Ether Value:	$0.00

More Info

My Name Tag:	Not Available, login to update
Contract Creator:	0x28a1474ccdb0353b9c... at txn 0x62719ba2a0eea81937...
Token Tracker:	PieDAO DEFI Small Cap (DEFI+S) (@$6.86)



Example Code Appendix

```
/ SPDX-License-Identifier: MIT
pragma solidity ^0.7.1;
pragma experimental ABIEncoderV2;

import "@pie-dao/diamond/contracts/Diamond.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import "@pie-dao/proxy/contracts/PProxy.sol";

import "../interfaces/IExperiPie.sol";

contract PieFactoryContract is Ownable {
    using SafeERC20 for IERC20;

    address[] public pies;
    mapping(address => bool) public isPie;
    address public defaultController;
    address public diamondImplementation;

    IDiamondCut.FacetCut[] public defaultCut;

    event PieCreated(
        address indexed pieAddress,
        address indexed deployer,
        uint256 indexed index
    );

    event DefaultControllerSet(address indexed controller);
    event FacetAdded(IDiamondCut.FacetCut);
    event FacetRemoved(IDiamondCut.FacetCut);

    constructor() {
        defaultController = msg.sender;
    }

    function setDefaultController(address _controller) external onlyOwner
        defaultController = _controller;
        emit DefaultControllerSet(_controller);
    }

    function removeFacet(uint256 _index) external onlyOwner {
        require(_index < defaultCut.length, "INVALID_INDEX");
        emit FacetRemoved(defaultCut[_index]);
        defaultCut[_index] = defaultCut[defaultCut.length - 1];
        defaultCut.pop();
    }

    function addFacet(IDiamondCut.FacetCut memory _facet) external onlyOw
        defaultCut.push(_facet);
        emit FacetAdded(_facet);
    }

    // Diamond should be Initialized to prevent it from being selfdestruc
    function setDiamondImplementation(address _diamondImplementation) ext
        diamondImplementation = _diamondImplementation;
    }

    function bakePie(
        address[] memory _tokens,
        uint256[] memory _amounts,
        uint256 _initialSupply,
        string memory _symbol,
        string memory _name
    ) external {
        PProxy proxy = new PProxy();
        Diamond d = Diamond(address(proxy));

        proxy.setImplementation(diamondImplementation);

        d.initialize(defaultCut, address(this));

        pies.push(address(d));
        isPie[address(d)] = true;

        // emit DiamondCreated(address(d));
        require(_tokens.length != 0, "CANNOT_CREATE_ZERO_TOKEN_LENGTH_PIE
        require(_tokens.length == _amounts.length, "ARRAY_LENGTH_MISMATCH

        IExperiPie pie = IExperiPie(address(d));

        // Init erc20 facet
        pie.initialize(_initialSupply, _name, _symbol);

        // Transfer and add tokens
        for (uint256 i = 0; i < _tokens.length; i++) {
            IERC20 token = IERC20(_tokens[i]);
            token.safeTransferFrom(msg.sender, address(pie), _amounts[i])
            pie.addToken(_tokens[i]);
        }

        // Unlock pool
        pie.setLock(1);

        // Uncap pool
        pie.setCap(uint256(-1));

        // Send minted pie to msg.sender
        pie.transfer(msg.sender, _initialSupply);
        pie.transferOwnership(defaultController);
        proxy.setProxyOwner(defaultController);

        emit PieCreated(address(d), msg.sender, pies.length - 1);
    }

    function getDefaultCut()
        external
        view
        returns (IDiamondCut.FacetCut[] memory)
    {
        return defaultCut;
    }

    function getDefaultCutCount() external view returns (uint256) {
        return defaultCut.length;
    }
}
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complexity
Solidity	26	2414	460	442	1512	107

Comments to Code 442/1512 = 29%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complexity
JavaScript	28	4209	825	119	3265	32

Tests to Code 3265/1512 = 215%