INTERSECT / DApp Testing Strategy

Draft, v0.1

In attn GovTool / March 13th, 2024

This document defines the testing strategy for decentralized applications (DApps) under Intersect..

Mission	To deliver reliable and functional decentralized applications through automated testing practices and community collaboration.
Vision	To set the industry benchmark for the quality and reliability of decentralized applications, ensuring they meet the evolving needs and expectations of users in the Cardano ecosystem
Core values	Quality, Security, Privacy and Trust, Transparency, Team Collaboration, Innovation, Community Engagement
Strategic areas	Automated Testing Continuous Integration/Continuous Deployment (CI/CD) Security, Privacy and Trust Performance and Scalability User-Centric Testing Cross-Network Compatibility Risk minimization
Objectives	G1. Achieve 90% Automated Test Coverage G2. Automate 90% of Testing Processes G3. Regular Security Audits -> Zero High Risk Vulnerabilities G4. Scalability and Performance Benchmarks G5. Community Testing Initiatives
Initiatives	xxx
North Star metric	Automated Error Reports per week
Metrics	Test Coverage / Code quality metrics / Community Engagement Score / User Engagement and Feedback Metrics / Performance Metrics / ...

1. Mission

To deliver reliable and functional decentralized applications through automated testing practices and community collaboration.

2. Vision

To set the industry benchmark for the quality and reliability of decentralized applications, ensuring they meet the evolving needs and expectations of users in the Cardano ecosystem.

3. Values

- **Quality**:
  Commitment to the highest standards of software quality. With attention to precision and details, meticulously examining every facet of our DApps to ensure flawless functionality.
- **Security, Privacy and Trust**:
  Security First mentality. Losing a user's trust on a project regarding security or privacy issues is a disaster. No corners should be cut on security or privacy.
- **Transparency**:
  Open sharing of our testing processes, results, and improvements fosters transparency and community involvement
- **Team Collaboration**:
  Working together within and across teams to achieve our common goals.
- **Innovation**: Continuously improving our testing strategies to support latest development.

- **Community Engagement**: Engaging with the community to identify, report, fix and test bugs, issues and enhancements with the collective goal of building quality software and a vibrant ecosystem.

4. Strategic Areas

Automated Testing

Automated testing is a cornerstone of Intersect projects, ensuring that code contributions are rigorously tested for functionality, performance, and security.

This strategic area involves implementing and enhancing automated testing frameworks for efficiency and thoroughness to cover all possible aspects of DApp functionality. Leveraging automated tests to cover unit, integration, functional, security, and performance testing.

Continuous Integration/Continuous Deployment (CI/CD)

CI/CD pipelines are integral to Intersect projects, enabling automated testing, building, and deployment of code changes. These pipelines help in identifying and addressing issues early in the development cycle. Tools like GitHub Actions, Jenkins, and Travis CI are commonly used for implementing CI/CD workflows.

Security, Privacy and Trust

Prioritizing security in our testing to protect users and meet security and privacy standards.

Conducting thorough security audits and incorporating ongoing security testing in the development lifecycle. Conducting regular security audits and compliance checks to preemptively address potential vulnerabilities.

Emphasizing the identification and mitigation of security risks in smart contracts (if involved) and DApp interfaces.

Performance and Scalability

Ensuring the projects are tested for performance under various loads and scales. Assessing the DApp's performance, particularly in terms of usability and responsiveness helps improve the user experience.

User-Centric Testing

Incorporating user feedback and real-world scenarios into our testing processes.

- Encouraging and facilitating community contributions to testing efforts.
- Including usability testing to ensure intuitive user interfaces and workflows.

Cross-Network Compatibility

Testing DApps across multiple network chains to ensure functionality is working as intended on all supported networks. Testing on testnet only is not always sufficient to ensure mainnet compatibility and omitting the tests on mainnet can lead to surprises.

Risk minimization

Several things can affect the performance of a DApp including the network performance so an important strategic area is exploring the minimization of unknown or uncovered risks.

5. Goals

Here are the proposed goals:

- G1. Achieve 90% Automated Test Coverage
- G2. Automate 90% of Testing Processes
- G3. Regular Security Audits → Zero High Risk Vulnerabilities
- G4. Scalability and Performance Benchmarks
- G5. Community Testing Initiatives

G1. Achieve 90% Automated Test Coverage

Every project should strive for achieving and maintaining high automated test coverage so that new features can be developed and deployed rapidly with confidence.

Strategic Areas covered

- Automated Testing
- Continuous Integration/Continuous Deployment

Initiatives

- Write tests for GovTool and related DApps
- Establish CI/CD workflows to automate running tests
- Measure and record coverage reports

G2. Automate 90% of Testing Processes

Emphasis should be given on automated testing to ensure reliability and speed up development. Integrating testing fully into CI/CD pipelines allows for swift feedback and deployment.

Strategic Areas covered

- Automated Testing
- Continuous Integration/Continuous Deployment

Initiatives

- Write GH workflows GovTool and related DApps to automate running testing
- Use test results for decision making. For eg. Approve/Reject PR.

G3. Regular Security Audits -> Zero High Risk Vulnerabilities

Projects should conduct continuous vulnerability scanning and identify and resolve any high-risk vulnerabilities before release.

Based on the feasibility, at least one external comprehensive security audits per year for critical projects would help boost community confidence in the project.

Strategic Areas covered

- Security, Privacy and Trust
- Risk minimization

Initiatives

- Implement various security testing methodologies, eg. Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST) and Dependency Scanning like DependaBot for checking for vulnerabilities in third-party dependencies.
- Employ external independent security audit

G4. Scalability and Performance Benchmarks

Projects should establish and meet scalability and performance benchmarks. These benchmarks could be different for different projects so the project maintainers should decide what kind of benchmarks are suitable and establish measurement and reporting processes.

Establishing and meeting performance benchmarks, including load testing for high user volumes improves predictability.

Strategic Areas covered

- Performance and Scalability

- Risk minimization
- Cross-Network Compatibility

Initiatives
- Define performance and scalability requirements
- Establish measurement testbed for the benchmarks
- Keep records of the benchmarks

G5. Community Testing Initiatives

Launch and maintain an active community testing initiative, including bug bounties and test contribution programs. Foster a Testing Culture: Double community engagement in testing efforts within a year (???)

Strategic Areas covered
- User-Centric Testing
- Risk minimization
- Cross-Network Compatibility

Initiatives
- Define contribution guidelines and disclosure forms
- Establish reward and recognition mechanisms for contributors

6. Measuring and Monitoring Progress. Success Definition.

Success is defined as meeting our predefined goals within their timelines while adhering to our values.

Progress will be measured through:
- **Test Coverage Metrics**: Aim for above 90% coverage across projects.

- **Test Coverage and Pass Rates**: Monitoring the breadth and success of our automated tests.
- **Automated Error reports**: Tracked across releases.
- **Defect Detection Rate**: Track the rate at which testing identifies defects pre-release.
- **Performance Benchmarks Achievement**, differs between projects. Includes Performance Metrics under normal and peak loads.
- **Community Contributions**: Monitor the volume and quality of community testing contributions.
- **Security Audit Outcomes and Vulnerability Resolution Times:** Including automated resolution of automated vulnerability reports from scanners like SonarCloud and DependaBot.
- **User Feedback on Usability and Experience:** Measured through user reports on GitHub
- User Transaction Feedback Time: Measuring the responsiveness of our DApps to user actions.
- Load Testing Results: Evaluating how our DApps perform under simulated peak usage conditions.
- **Blockchain Operation Success Rates**: Tracking the reliability of transactions and data retrieval processes. Focus on wallet interaction; required client side analytics.

7. North Star Metric

NS metric is the single indicator that guides the Open Source activities, and measures their success. It emerges directly, and is fully aligned with the Long-Term Strategy (mission, vision, and values).

Here are a few candidates:

- **Automated Error Reports per week**, which measures the number of error reports received in the reporting system per week. Lower is better.

- **Issue Resolution Time**, which measures the average time taken from identifying an issue to deploying a fix. A reduction in this metric indicates more efficient testing and development processes, aligning with our mission of high-quality and reliable software. Lower is better.

- **User Satisfaction Score**, derived from user feedback on usability, performance, and security. This reflects the effectiveness of testing strategy in delivering quality DApps.

- **User Transaction Success Rate**, indicating the percentage of transactions that are successfully executed and confirmed without issues, as a direct measure of DApp functionality and reliability. This can be measured either via blockchain/txn data or via analytics on wallet transaction operations on the frontend side of DApp.

- **User Engagement Time**, For some DApps, it may make sense in measuring how long users interact with the DApp.

8. Metrics

Here are some useful metrics.

- **Test Coverage**
    - Coverage percentage
    - Number of test suites
    - Automated Test Success Rate: The percentage of automated tests that pass on the first run.
- **Other Code quality metrics**
    - Clean code attributes,
    - Code smells
    - Security hotspots, vulnerabilities
    - Duplications
- **Community Engagement Score**

  Based on the number of community contributions to testing
    - Pull requests,

    - Bug reports, test cases
    - Discussions
- **Security**
    - Security Vulnerability Reports
    - Security Vulnerability Response Time;
    - Security Audit Findings:
        - The number and severity of vulnerabilities identified
        - The number of resolved vulnerabilities
- **User Engagement and Feedback Metrics**
    - Usability test results
    - GitHub issues with usability label
    - User satisfaction and usability scores based on direct feedback, surveys, and usability studies.
- **Automated Error reports**
    - Number of automated reports received in error reporting system
    - Tracked per release
- **Performance Metrics**
    - Transaction throughput
    - User interface responsiveness.
    - API response times
- **Bug reports**
    - Outstanding bugs by severity
    - Monthly/weekly user bug reports (manual)
    - Monthly/weekly user bug reports (automated)
    - Bug Fix Rate: The rate at which identified bugs are resolved.
- **Transaction Completion Rate**: The percentage of successful blockchain transactions initiated by the application.
- **User Engagement with Core Features**: User interaction levels with the primary functionalities of the application. This is useful in conjunction with error reports. It can be used to relate if the no or low error reports is (or not) linked to low usage of the feature.

9. Key Components of Testing Strategies

1. Unit Testing

Purpose: To test individual components or functions in isolation to ensure they work as expected.
Common Tools: Jest/Mocha etc. for Javascript-based projects, JUnit for Java-based projects, PyTest for Python projects, and other language-specific frameworks.
Responsible: Dev Team

2. Integration Testing

Purpose: To verify that different modules or services work together as intended.
Approach: This involves testing APIs, database interactions, and other integration points.
Responsible: Dev Team and QA Team

3. Functional Testing

Purpose:

- To check that the software meets the specified requirements and behaves as expected in scenarios close to real-world use.
- *DApp Front-End Testing*: Ensuring compatibility across different browsers and devices, and testing the integration with various blockchain wallets and extensions.

Approach: Automated scripts are used to perform these tests, simulating user interactions with the DApp. The testing of the user stories are covered here. Playwright and Selenium like tools are useful for these tests.
Responsible: Dev Team and QA Team

4. Performance and Scalability Testing

Purpose: To ensure that the software performs well under expected and peak load conditions.
Approach: Tools like Gatling and Locust are often for performance testing.
Responsible: Dev Team and QA Team

5. Security Testing

Purpose: To identify vulnerabilities and ensure that the software is secure against attacks.
Approach: This includes regular security audits, using tools for static and dynamic analysis, and adhering to security best practices in development.
Responsible: Dev Team, Code Quality Analysis Tool and QA/Security Team

6. Regression Testing

Purpose: To make sure that new changes don't adversely affect existing functionality.
Approach: Automated test suites are run to detect regressions as new code is integrated.
Responsible: First, CI/CD workflow should detect this then QA Team

7. Cross-DApp and Interoperability Testing

Purpose:

- Ensuring seamless operation across related DApps and services. Making sure the implementation is based on the standards rather than DApp specific implementation, thus avoiding interoperability issues.
- Testing the DApp's ability to interact with other DApps, smart contracts, and services across blockchain ecosystems.

Approach: Testing with multiple implementations if available. Making sure that wallet interoperability is working.

8. Cross-chain Integration Testing

Purpose:

- Verifying the DApp's interaction with blockchain data and transactions, including testing across different networks (mainnet, testnets), gas usage optimization, transaction processing, and error handling.

Approach: Plan, prepare and select tests to run on different network chains. It might not be possible to run all tests on all networks.
Responsible: QA Team + DevOps Team

10. Strategy Implementation

Every project onboarding to follow this strategy should start with alignment of the goals and the metrics for the tests.

Step 1: Planning: Test Selection and Execution Strategy

1. Select which tests make sense for your project.
2. Define when to run which tests.
   For eg. on PR open, merge, scheduled nightly etc.
3. Define which environment to run those tests on:
   a. Developers local machine
   b. Deployed dev environment
   c. Deployed staging environment
   d. Deployed production environment

Step 2: Test Development

1. Define scope of tests and metrics
2. Write test or scaffolds to expand upon later
3. Measure the intended metrics
4. Repeat until all the tests are developed

Step 3: Test execution and Automation

1. Automate the test execution
    a. Using CI/CD - Github Actions
    b. Scheduled cron jobs
    c. Other scripts
2. Measure the intended metrics
3. Report the measured metrics to Reporting Server
4. Repeat until all the tests are automated

Step 4: Reporting Dashboard

1. Make sure your test results and metrics are visible in the reporting dashboard

Step 5: Community Testing Initiatives

1. Define contribution guidelines and disclosure forms
2. Establish reward and recognition mechanisms for contributors

Appendix A - Testing Infrastructure

PR Jobs
The infrastructure for running the tests is **primarily GitHub actions**. This covers the tests that need to be executed during the PR cycle, before and after merge, Tag creation process. Different branches could have different tests.

Scheduled Jobs
Some tests are required to be executed nightly or in a scheduled fashion. For that, it might still be possible to use GitHub actions. If not, any other tool or script used for that purpose should have proper documentation (and source if available) on the repository.

Appendix B - Code Quality Analysis Tools

To have an objective measurement on the code quality, we propose to use existing software available in the market, preferably open source for software quality checks.

Here are some known tools:

- SonarQube / SonarLint / SonarCloud
- Snyk
- Codacy

Appendix C - Automated error monitoring system

Not all users report bugs even if they find them. Therefore, having an automated error monitoring system built into the DApp is important in finding out the hidden/unreported errors.

Depending on the tool itself and the implementation within the DApp, it might be necessary to ask for user consent and might require data anonymization before collection. DApp developers should be aware of that.

Here is a good example of such tool:
- Sentry (https://sentry.io, https://github.com/getsentry/sentry)

Others:
- Prometheus (https://prometheus.io)

Appendix D - Reporting Dashboard

Reporting Dashboard is an important tool for obtaining the overview of all tests that are conducted across the projects.

Here are the considered requirements for a Reporting Dashboard Tool:

1. Language and Framework Support: The tool should support a wide range of programming languages (e.g., Python, JavaScript) and testing frameworks (e.g., JUnit, NUnit, pytest, Mocha) to ensure compatibility with your project's technology stack.

2. Test Execution Data: Ability to capture comprehensive test execution data, including test steps, start and end times, statuses (passed, failed, skipped), and environment details.

3. Rich Reporting Features: The tool should generate clear and detailed reports that include:

- Graphical representation of test execution results (charts, graphs).
- Historical trend analysis to track test execution over time.
- Filtering and search capabilities to easily navigate through test results.
- Attachments and Screenshots: Support for attaching logs, screenshots, and videos to test reports to provide additional context for debugging and analysis.

4. Integration Capabilities: Seamless integration with continuous integration (CI) tools (e.g., GitLab CI, Travis CI) and issue tracking systems (e.g., JIRA, GitHub).

5. Customization and Extensibility: The ability to customize the look and feel of reports, add custom fields or sections, and extend functionality through plugins or APIs.

6. Community and Documentation: Active community support and comprehensive documentation to assist with installation, configuration, and troubleshooting.

7. License and Cost: Open-source, Free (Preferably)

8. Security and Privacy: Commitment to security practices to protect sensitive test data, with features like role-based access control (RBAC) and data encryption.

Recommendation

Allure Reports is an open-source test reporting tool designed to support a wide range of programming languages, including Java, Python, C#, and JavaScript, as well as various testing frameworks like JUnit, NUnit, pytest, and Mocha. It captures extensive test execution data such as test steps, durations, outcomes, and environmental contexts to provide a comprehensive overview of testing activities.

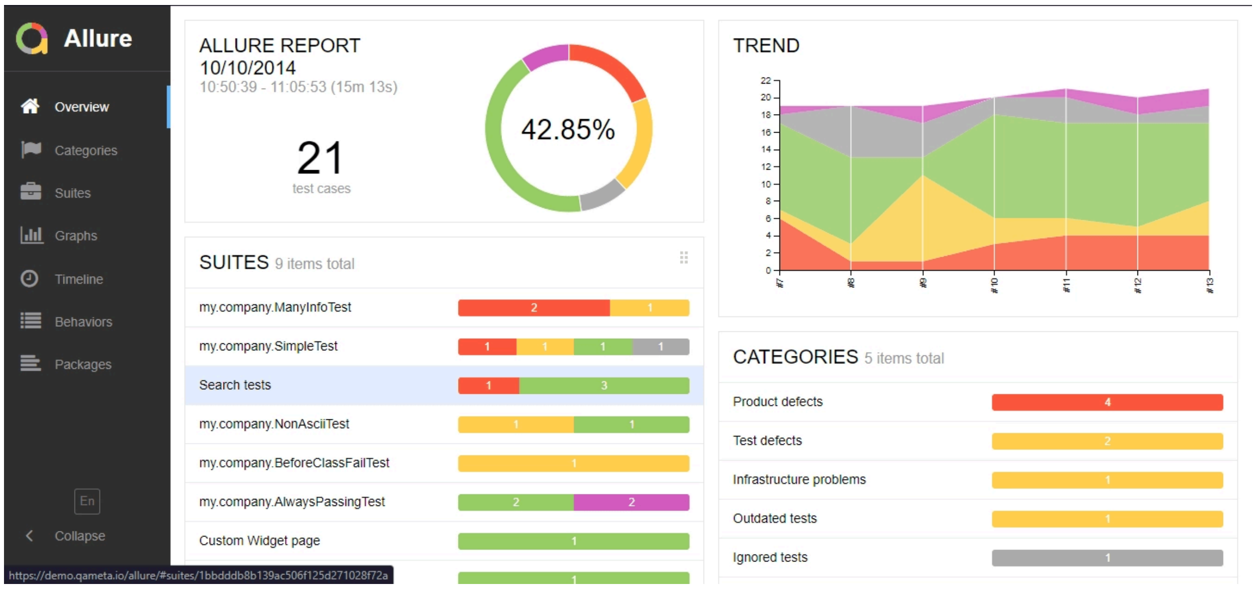
This tool offers rich reporting features, including graphical representations of test outcomes, trend analysis over time, and advanced filtering/search functionalities for efficient navigation through test results. It enhances test reports with attachments like logs, screenshots, and videos for deeper insight and easier debugging.

Allure Reports integrates smoothly with continuous integration (CI) tools and issue tracking systems, facilitating a seamless workflow within development pipelines. It allows for extensive customization and extensibility through plugins or APIs, ensuring that reports can be tailored to specific project requirements.

With a focus on community support and thorough documentation, Allure Reports seems user-friendly for both technical and non-technical team members, ensuring a low maintenance burden. It adheres to strict security practices, including role-based access control and data encryption, to safeguard sensitive test information.

This tool is licensed under a permissive open-source license, allowing free use, modification, and distribution within any organization. It is built to scale with project growth and operates across multiple platforms (Windows, Linux, macOS), making it a versatile choice for diverse development environments.

Repo: https://github.com/allure-framework/allure2
Website: https://allurereport.org/
Blog: https://qameta.io/blog/allure-report-hands-on/



ALLURE REPORT
10/10/2014
10:50:39 - 11:05:53 (15m 13s)

21
test cases

42.85%

SUITES 9 items total

my.company.ManyInfoTest	2 1
my.company.SimpleTest	1 1 1 1
Search tests	1 3
my.company.NonAsciiTest	1 1
my.company.BeforeClassFailTest	1
my.company.AlwaysPassingTest	2 2
Custom Widget page	1
	1

CATEGORIES 5 items total

Product defects	4
Test defects	2
Infrastructure problems	1
Outdated tests	1
Ignored tests	1

https://demo.qameta.io/allure/#suites/1bbdddb8b139ac506f125d271028f72a