How we test Cardano

The importance of testing our cryptocurrency systems

AUGUST 30, 2017

Testing is of course critically important to a cryptocurrency because the correctness and robustness of the system are what you rely on to keep your money safe, and ensure that you can spend it when you need to. So as you would expect, as our development team is getting ready for the Cardano mainnet release, testing is one of the main things that is on our minds.

There are many different ways in which we test Cardano and in this post we will talk about several.

Testing can be divided into two main kinds: functional and non-functional:

- Functional testing is about checking that all the system's components meet their specifications.

  Functional testing is done with components on their own, in which case we call it unit testing or component testing. It is also done with all the components together, in which case we call it integration testing or system testing.

  These kinds of tests are typically of the form: given some scenario, and certain inputs, the component or system produces the correct output or takes the correct next action.

- Non-functional testing reveals "how" the system behaves, including the performance of the system, the resources it uses and how the system behaves when under great load or attack.

We have a few major parts of the Cardano system: the core, the wallet backend and the Daedulus frontend. Different parts of the system are appropriate to test in different ways.

Public testnets

The most visible form of testing is of course the public testnets where we ask users to try the system out. This is a kind of beta testing. This is just the tip of the iceberg compared to all the testing we do internally, but it is still very useful because it covers a different set of problems compared to our internal tests.

Users have a huge variety of desktop computers, both in hardware and configuration. It is impossible for us to test all the combinations that our users have. So having lots of real users try out the system really helps to find those strange combinations where something does not work well, and gives us the confidence that we will not bump into similar problems for mainnet.

A testnet release helps us test usability of the system: our websites, the installers, the Daedulus interface and how many cryptocurrency concepts people need to know to use the system.

There is no escaping the fact that a public testnet release is in some ways more "real" than any test situations we can construct artificially. Though we can certainly push the system to breaking point using our internal stress tests, there are complexities of a real world deployment that are hard to replicate in an artificial test.

Finally, it also helps our team practice making public releases, which helps us work out the kinks in our processes so that we can avoid problems during the mainnet release or later updates. And it's not just our developers and technical operations teams, a successful launch also depends on our communications and support teams. The very process of getting questions, feedback and problem reports from users during the testnet phases helps us to make sure that our support teams have the right procedures in place so that we can be confident that they can help everyone effectively during and after the mainnet launch.

Automatic unit and component testing

We have an increasing collection of fully automatic functional tests that cover various important parts of the logic in the core and wallet backend. These are functional tests in that they check that each component meets its specification.

These tests are run automatically by our continuous integration system, which means they are run before any change to the code is accepted into our master branch. This helps to protect us against introducing regressions.

Wherever possible we make use of property based testing, rather than simple individual unit tests. Classic unit tests for a component tend to simply use a specific set of inputs and check that a specific output is produced. To comprehensively test a component in this style often requires a large number of specific pairs of input and expected output. This is laborious and tends to miss corner cases that programmers do not think of. By contrast, property based testing involves taking the component's high level specification and reformulating the specification as an executable property. That means that for any specific inputs the property can actually be executed to check that the property is true for those inputs. These properties are expressed in the same programming language as the code being tested. The technique then involves checking the property on hundreds or thousands of test inputs. The technique is to use systematic random generation to produce test inputs. This means that programmers do not have to think of lots of test inputs and it avoids human bias. So it tends to give much better test coverage with less effort.

Specifically, we use the QuickCheck system for property based testing. Perhaps the greatest advantage is that it makes developers *think* in terms of the specification and properties of their code, rather than individual inputs and outputs. This is a much higher level way of thinking about code and helps to produce simpler more reliable code.

System level tests and performance tests

While all the unit and component testing gives us confidence that each part of the system works ok on its own, system level tests are to check if the parts all work together as a whole.

For this we have to set up a cluster of machines and configure them to run the blockchain protocol together. The main functional test that we use works like this: we have a special transaction generator program that constructs tens of thousands of transactions and submits them to nodes in the cluster. The code is instrumented to record certain key events in a log file, such as when each transaction reaches each node. We let this run for around an hour. At the end of the run we have a tool that analyses the blockchain and the log files from all the nodes. This checks that all the transactions that were submitted did make it into the blockchain. It also checks if there were any unexpected forks in the blockchain or missing blocks. In normal conditions there will be no forks or missing blocks.

We can use the same basic approach to test the system when we deliberately attack it, such as taking out nodes, or preventing nodes from talking to each other for a while. In this case we expect temporary forks or missing blocks, but we can check that the system recovers properly.

We use the same basic approach for non-functional performance tests. We adjust the transaction generator to submit transactions at a higher rate to stress the system and see how high we can push the throughput before it hits a bottleneck. We can also check that even though the system has hit its maximum capacity it continues to function in a stable way.

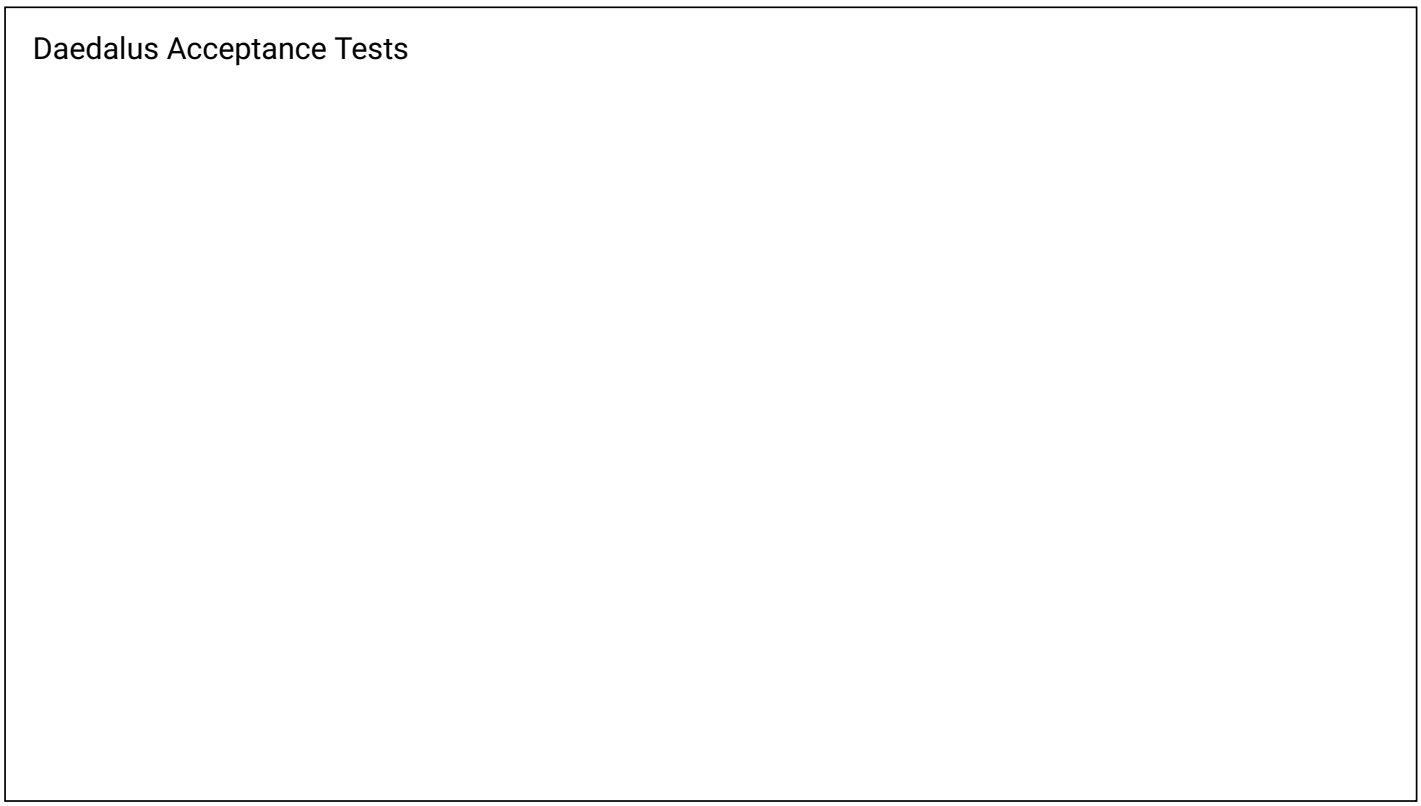
Throughput, meaning transactions per second, is important but so is latency. By latency we mean how long it takes for a transaction to get into the blockchain. Our analysis tool can also determine the distribution of latency. A low latency with little variance shows us that transactions are flowing smoothly to the nodes that create blocks and that those nodes are creating blocks on time.

Frontend testing

Our Deadulus frontend team have a fully automated set of tests that cover every function of the user interface. In turn this also tests every interaction between the wallet frontend and wallet backend. So this also gives us an automatic integration test for the combination of the wallet frontend and backend.

Frontend testing is a bit different from most other testing. Most testing works by a test program directly using a program interface, whereas frontend testing requires interacting with an actual human interface. User interface testing frameworks simulate what a real user does: clicking buttons and typing in web form boxes.

The result is actually rather fascinating to watch: it's as if an invisible robot is sitting at the computer typing and clicking very quickly to set up accounts, send transactions and all the other things.



Daedalus acceptance tests – a fully automated set of tests that cover every function of the user interface.

Security auditing

Counterintuitively, when it comes to cryptography and security – which cryptocurrencies of course rely on completely – testing is in fact not very effective. Testing usually shows us that the expected things do work, but it's hard to use normal testing to show that unexpected things cannot happen. And showing that some hacker cannot subvert the system is just the kind of thing that is hard to test for.

The solution is not testing but auditing by experts in cryptography and security. This means experts carefully reviewing the designs to check that the arguments for why the system should be safe are sound, and also reviewing the code to make sure the code matches up with the design.

Of course, the basic design for the proof of stake blockchain used in Cardano has already been peer reviewed by academic cryptographers. There are other parts of the system that we have had to develop in the last year – beyond just the blockchain – and the most security critical parts of those have been reviewed by our research team, and also by an external security audit team. Additionally, the security audit team have reviewed many of the most important parts of the code to check that the code matches the design.

Conclusion

A cryptocurrency system is a surprisingly complex piece of software and it has to work correctly, be robust to deliberate attacks and have good performance. Of course Cardano is a new from-scratch cryptocurrency, not based on any existing system, so all of it has to be carefully tested or reviewed.

Hopefully this post has given you some insight into how much is involved in testing Cardano, and how serious we are about security, robustness and performance.