LimeChain

Telerik Academy

Unit testing of smart contracts

Lime Academy - Season 4

Telerik Academy

Why do we need to pay even higher attention of unit testing our smart contracts ?

- lots of money involved
- bugs affects accounts holding real money
- once a bug is introduced, not easy to fix (if it is possible at all)
- soft / hard forks

Telerik Academy

What do we need to cover with unit tests ?

- positive test cases (what should happen)
- negative test cases (what should NOT happen)

Telerik Academy

When do we need to execute unit tests ?

- updating old functionality
- adding new features
- before each deployment

Telerik Academy

Available development / testing frameworks

- Truffle: Javascript & Solidity (Mocha/Chai)
- Embark: Javascript (Mocha)
- Etherlime: Javascript (Mocha/Chai asserts)
- Dapple: Solidity
- Populus: Python
- Go-Ethereum: Go

Telerik Academy

Writing sample unit test with Etherlime

- unit tests location
- basic unit test structure
  - require('etherlime')
  - .describe()
  - .it()
  - asserts
- running the tests
  - etherlime test

Telerik Academy

Hooks

- .before()
- .after()
- .beforeEach()
- .afterEach()

Telerik Academy

Pending unit tests

Telerik Academy

Exclusive unit tests

- .skip()
- .xdescribe()
- .xit()

Telerik Academy

Inclusive unit tests

- .only()

Telerik Academy

Events

- utils.hasEvent(txReceipt, contract, eventName)
- utils.parseLogs(txReceipt, contract, eventName)

Telerik Academy

Exceptions

- require
- function modifiers

- assert.revert(promiseOfFailingTransaction)

Telerik Academy

Time travel

- utils.timeTravel(provider, seconds)
- utils.setTimeTo(provider, timestamp)

Telerik Academy

Code coverage

- etherlime coverage

Telerik Academy

Best practices & guidelines

- readable code & comments
- positive, negative cases, events
- overflows / underflows
- for each function
  - who can execute it
  - who can NOT execute it
  - when could be executed
- write as many unit tests as possible

Telerik Academy

Best practices & tools

- use automated code review tools
  - significantly decreases chances of mistakes
  - developers save up to 30-50% of the reviewing time
- external audit
- code coverage
  - highest coverage, the best (> 90%)

Telerik Academy

Further reading

- Mocha framework documentation

  https://mochajs.org/
- etherlime testing & coverage

  https://etherlime.readthedocs.io/en/latest/cli/test.html

Telerik Academy

Homework

- add (as much as you can) unit tests to your crypto cars app
- get at least 90% code coverage

Telerik Academy

Q & A

Telerik Academy

Thanks !

vlado@limechain.tech