Testing in DLT projects studied

Petya Cvitic
Department of Computers and Systems Sciences
Mid Sweden University
Östersund, Sweden
{peya1400}@student.miun.se

Quorum (Go) The project is a fork of the Go Ethereum client meaning a part of the libraries are actually inherited from Ethereum.

Judging by the documentation and the repository of the project, it seem to include:

- unit testing - files ending with *_test.go* in most of the subdirectories of the main directory [1] such as for example the file *state_test.go* [1] (some of those files focus on unit testing, others on integration testing)
- integration testing - there was a dedicated repository providing tools for running integration tests but that has been deprecated [2]; smoke tests – in the file *accountcmd_test.go* [2]; some of the files ending with *_test.go* in most of the subdirectories of the main directory [1] such as for example the file *peer_test.go* [3] handle integration testing;
- system testing – in *tests* directory [3] which focuses on fuzzing among others; some system testing seems to have been done by external parties as well (as reported by Pavlovics [4]for example)
- acceptance testing in *quorum-acceptance-tests* directory [5].

Hyperledger Fabric (Go) According to the Hyperledger Foundation1 activities ensuring quality assurance have been performed. Moreover, the project has a documentation with guidelines on how to contribute to the project2. According to it unit and integration tests are a necessary requirement for a good pull request.

Judging by the repository and the documentation of the project, it seems to include:

- unit testing (files ending with *_test.go* found in some of the subdirectories of the project repository such as for example in the *ledgermgmt* directory [6], some of those files focus on unit testing, others on integration testing)
- integration testing - the *integration* directory [7]; some mocking seems to have been done as well [8];)
- system testing – the *fabric-test* directory [9]; fuzz testing method for Hyperledger Fabric has been designed by Chang [10].

Hyperledger Indy (Rust) According to Hyperledger Indy's website, the project has undergone unit testing as well as integration and system testing1.

Judging by the repository and the documentation of the project, it seems to include:

- unit testing - within the files in *src* directories, the module is annotated *#[cfg(test)]* (for example line 569 of the file *mod.rs* [4]), and the functions are marked with the *#[test]* attribute (for example line 570 of the same file);
- system/integration testing - according to the documentation it is available at *tests* directories (libindy tests) [11], (python wrapper tests) [12] , *sdk* directory (java wrapper tests) [13], *libindy-demoTests* directories (iOS wrapper tests) [14], *indy-test-automation* [15].

Hyperledger Iroha (C++) In the README of the scripts directory ([16]) creators suggest running automated tests when getting started with the project. All tests seem to be gathered in one *test* directory [17].

Judging by the repository and the documentation of the project, it seems to include:

- unit testing - a few of the test files in the directory above focus on unit testing (for example the files *transaction_cache_test.cpp* [5], the file *hash_test.cpp* [6].
- integration testing – for example in the folders dedicated to integration and regression tests as a subcategory of the *test* directory [17];
- system testing – for example in the subdirectories of *test* directory [17] called system and fuzzing.

Google Test has been the Testing Framework (since some test files start with #include gtest/gtest.h such as for example the file *port_guard_test.cpp* [7]). Google test

[1]https://github.com/ConsenSys/quorum/blob/master/core/state/state_test.go
[2]https://github.com/ConsenSys/quorum/blob/master/cmd/geth/accountcmd_test.go
[3]https://github.com/ConsenSys/quorum/blob/master/p2p/peer_test.go
[4]https://github.com/hyperledger/indy-sdk/blob/master/libindy/src/services/ledger/mod.rs
[5]https://github.com/hyperledger/iroha/blob/main/test/module/libs/cache/transaction_cache_test.cpp
[6]https://github.com/hyperledger/iroha/blob/main/test/module/libs/crypto/hash_test.cpp
[7]https://github.com/hyperledger/iroha/blob/main/test/module/test/framework/integration_framework/port_guard_test.cpp

includes even mocking, which seems to be used in that project (as implied by #include ¡gmock/gmock.h¿ in the file *regression_test.cpp* [8].

MultiChain (C++) As the project is written mainly in C++, a separate folder with tests is a natural solution. The repository of the project does contain a directory with that name ([18]) but it is empty. No other content with focus on testing seems to be available in the project's repository.

R3 Corda (Kotlin) Corda has a suite of tests that any contributing developers must maintain and extend when adding new code [19]. Those include:

- unit testing - for example *test* directories [20], [21], etc;
- integration tests - for example *core-tests* directory [22], *integration-test* directory [23], *webserver* directory [24], etc;
- smoke/system tests - for example *testing* directory [25], *smoke-test-utils* directory [26]) – according to the documentation [19] those could be classified as system tests.

Judging by the file *build.gradle* [9], line 43 and downwards, Corda uses the frameworks Junit, Mockito, AssertJ3, Hamcrest4, OpenTest4j.

Dragonchain (Python) According to the documentation of the project, a developer should run full tests (found in ./tools.sh) before starting to develop locally.

Judging by the repository and the documentation of the project, it seems to include:

- unit testing - separated in own files spread throughout the directories of the project (the names of those files contain the word test) such as for example the file *crypto_utest.py* [10], some of those files focus on unit testing, others on integration testing;
- integration testing – some mocking seems to have been used (as implied by the import statement of the file *contract_job_utest.py* [11];
- no system testing - there is not a folder dedicated to system testing as in other projects. That implies there might not have been much system testing done.

Hyperledger Besu (Java) Judging by the repository and the documentation of the project, it seem to include:

- unit testing - in most of the */src/* directories in a separate subdirectory such as for example in the *test* directory [27];
- integration testing – according to the *build.gradle* file [12] the Mockito framework is used for integration testing, this is handled by all files that import the Mockito classes or/and the folders named *controller*, such as for example the file *BesuControllerBuilderTest.java* [13]; container tests - the *container-tests* directory [28] (container tests are usually used in integration testing [29]);
- acceptance testing - the *acceptance-tests* directory [30].

The existing tests could also have been used with regards to system testing but that is not very clear from the repository of the project.

Hyperledger Burrow (Go) Judging by the repository and the documentation of the project, it seem to include:

- unit testing - files ending with *_test.go* found in some of the subdirectories of the project repository such as for example the file *perm_flag_test.go* [14], (some of those files focus on unit testing, others on integration testing);
- integration testing – the *integration* directory [31]; the *testnet* directory [32]; files ending with *_test.go* found in some of the subdirectories of the project repository (some of those files focus on unit testing, others on integration testing);
- system testing – the *tests* directory [33].

Hyperledger Sawtooth (Python) In the file *BUILD.md* [15] creators suggest running automated tests when getting started with the project. Judging by the repository and the documentation of the project, it seems to include:

- unit testing - separated in own directories for each module such as for example the *tests* directory [34];
- integration testing – there seems to be a dedicated *tests* directory [35];
- system testing - some system testing seems to be covered - in the *tests* directory [36] (those are classified as system tests according to Sawtooth [37]).

Algorand (Go, C) Many of the libraries and code for cryptography of the project are written in C. This might be due to the huge amount of C code already existing for blockchain purposes, but also for performance reasons. The code wrapping these libraries is Go [38]. Hence, most of the tests are in Go. According to the documentation of the project ([39] under the category *test*) unit and integration tests could be run automatically. Judging by the repository and the documentation of the project, it seems to include:

- unit testing - many files ending with *_test.go* in the repository such as, for example, the file

[8] https://github.com/hyperledger/iroha/blob/main/test/regression/regression_test.cpp

[9] https://github.com/corda/corda/blob/release/os/4.10/build.gradle

[10] https://github.com/dragonchain/dragonchain/blob/master/dragonchain/lib/crypto_utest.py

[11] https://github.com/dragonchain/dragonchain/blob/master/dragonchain/job_processor/contract_job_utest.py

[12] https://github.com/hyperledger/besu/blob/main/besu/build.gradle

[13] https://github.com/hyperledger/besu/blob/main/besu/src/test/java/org/hyperledger/besu/controller/BesuControllerBuilderTest.java

[14] https://github.com/hyperledger/burrow/blob/main/permission/perm_flag_test.go

[15] https://github.com/hyperledger/sawtooth-core/blob/main/BUILD.md

hashes_test.go [16] (some of the files ending with *_test.go* in the repository focus on unit testing, others on integration testing);
- integration testing – for example mocking (as implied by the file *mockNetwork.go* [17]), end-to-end testing found in the *test* directory [40];
- system testing - a fuzzing solution for testing the package is available - in the the *go-fuzz* directory [41].

Bitcoin (C++, Python) The documentation of the project (in the *bitcoin* directory [42]) has a whole section dedicated to testing. The testing framework Boost is mentioned there among else. Judging by the repository and the documentation of the project, it seems to include:

- unit testing - a whole subdirectory focusing on that: the *test* directory ([43]), the file *README.md* in that subdirectory implies that the testing framework of Boost is used;
- integration testing – for example regression testing (as mentioned in the the file *README.md* [18]); in *test* subdirectories within the *bitcoin/src* directory [44], [45], etc;
- system testing – for example fuzzing (as documented in the file *fuzzing.md* [19]).

Cardano (Haskell) According to the documentation of the project (in the *cardano-node* directory [46]) cardano-node goes through integration and release testing by Devops/QA while automated CLI tests are ongoing alongside development. Furthermore, developers can launch their own testnets or run the chairman tests (those run locally some testnets and perform some basic tests to ensure they are working). Moreover, contributors are advised to do some testing according to the documentation of the project [47]. Some of the test files import Test.Tasty (such as for ex. the file *Main.hs* [20]) implying the framework Tasty is used for testing the project. Some of the test files import Genesis (such as for ex. the file *Ledger.hs* [21]) implying the framework Genesis is also used for testing the project.

Judging by the repository and the documentation of the project, it seem to include:

- unit testing - some of the test files in directories named Test such as for example the file *Ledger.hs* [22] (some of the test files in directories named Test focus on unit testing, others on integration testing), even an issue has been opened in 2020 demanding more unit testing ([48]) but that issue has been closed implying some unit tests have been added since then;
- integration testing – for example testnet in the *cardano-testnet* directory [49], as well as some of the test files in directories named *test* such as for example the *test* directory [50]; (some of the test files in directories named *test* focus on unit testing, others on integration testing);
- system testing – the *e2e* directory [51].

Dfinity (Rust) Judging by the repository and the documentation of of the project, it seem to include:

- unit testing - within the files in *src* directories, the module is annotated *#[cfg(test)]* (for example line 675 of the file *consensus.rs* [23], and the functions are marked with the *#[test]* attribute (for example line 772 of the file *consensus.rs* [24]); mocking is used among others (for example line 30 of the same file);
- integration testing – testnet in the *testnet* folder ([52]); in their own *test directories*, placed next to the *src* directories (for example the *tests* directory [53]);
- system testing - some system testing has also been covered according to the projects repository, found in the dedicated *tests* directory [54].

EOSIO (C++) As the project is written mainly in C++, a separate folder with tests is a natural solution. The testing frameworks Boost and Genesis are mentioned among else.

Judging by the repository and the documentation of the project, it seem to include:

- unit testing - handled by files placed in their own *unittests* directory [55];
- integration testing – testnet according to the documentation of the project [56], found in the *eosio-wasm-spec-tests* directory [57];
- system testing – in the *tests* directories [58], [59].

Ethereum (Go) Judging by the repository and the documentation of the project, it seems to include:

- unit testing - files ending with *_test.go* in most of the subdirectories of the *go-ethereum* directory [60] such as for example the file *block_validator_test.go* [25] (some of the files ending with *_test.go* focus on unit testing, others on integration testing);
- integration testing - for example the *tests* directory [61], which focuses on fuzzing among others (some of the files ending with *_test.go* in most of the subdirectories of the *go-ethereum* directory [60] handle integration testing, others handle unit testing);
- system testing - a dedicated *system-testing* repository [62]; various fuzz-testing tools have been created for

[16] https://github.com/algorand/go-algorand/blob/master/crypto/hashes_test.go

[17] https://github.com/algorand/go-algorand/blob/master/components/mocks/mockNetwork.go

[18] https://github.com/bitcoin/bitcoin/blob/master/README.md

[19] https://github.com/bitcoin/bitcoin/blob/master/doc/fuzzing.md

[20] https://github.com/input-output-hk/cardano-node/blob/master/cardano-testnet/test/Main.hs

[21] https://github.com/input-output-hk/cardano-node/blob/master/cardano-api/test/Test/Cardano/Api/Ledger.hs

[22] https://github.com/input-output-hk/cardano-node/blob/master/cardano-api/test/Test/Cardano/Api/Ledger.hs

[23] https://github.com/dfinity/ic/blob/master/rs/consensus/src/consensus.rs

[24] https://github.com/dfinity/ic/blob/master/rs/consensus/src/consensus.rs

[25] https://github.com/ethereum/go-ethereum/blob/master/core/block_validator_test.go

testing for example the EVM [63], the consensus protocol [64], or the smart contracts in Ethereum but those are not a part of the project on GitHub.

IOTA (Rust) Judging by the repository and the documentation of the project, it seem to include:

- unit testing - within the files in *src* directories, the module is annotated *#[cfg(test)]* (for example line 318 of the file *config.rs* [26], and the functions are marked with the *#[test]* attribute (for example line 447 of the same file);
- integration testing – in their own test directories, placed next to (or in some cases within) the *src* directories (for example the *tests* directories [65] or [66])
- no system testing - there is not convincing evidence on system testing according to the repository or the documentation of the project (the only directory somehow relevant seems to be the *bee-test* directory [67] but most of the files in that repository are almost, the key word fuzzing is mentioned in one of the files, namely the file *buf.rs* [27] but it does not seem like real fuzzing solution has been implemented.

Solana (Rust) Judging by the repository and the documentation of the project, it seem to include:

- unit testing - within the files in *src* directories, the module is annotated *#[cfg(test)]* (for example line 1369 of the file *consensus.rs* [28]), and the functions are marked with the *#[test]* attribute (for example line 1427 of the same file);
- integration testing – in their own *test* directories, placed next to (or in some cases within) the *src* directories (for example the *tests* directory [68]);
- system testing - some system testing has also been covered (found in the *system-test* directory [69] and the *program-test* directory [70]).

Stellar (C, C++) Judging by the repository and the documentation of the project, it seem to include:

- unit testing - in the *test* directory within each subdirectory of the *src* directory [71] such as for example the *test* directory [72];
- integration testing – the *integration-tests* repository [73] (testing different components of the network); moreover according to the *Readme* file of the *supercluster* repository ([74]) there is a package (Stellar Supercluster, SSC) provided externally and used for automated integration testing of stellar-core;
- system testing - in *test* directory situated next to the *src* directory ([75]) which seems to include fuzzing among else; moreover, according to the documentation of the project [76], once an application is fully functional, testing different scenarios and edge cases is recommended to make sure the system is behaving as expected (i.e stress testing is recommended). An anchor validation test suite [77] is offered, which aims to assess the services's compliance with the protocol (found in the *anchor-transfer-server-validator-ui* repository [78]). That could probably be considered some kind of system testing for the project as well.

[26] https://github.com/iotaledger/bee/blob/mainnet-develop/bee-network/bee-gossip/src/config.rs

[27] https://github.com/iotaledger/bee/blob/mainnet-develop/bee-test/tests/ternary/buf.rs

[28] https://github.com/solana-labs/solana/blob/master/core/src/consensus.rs

References

[1] ConsenSys, "Quorum," 2022. [Online]. Available: https://github.com/ConsenSys/quorum

[2] ——, "Quorum," 2022. [Online]. Available: https://github.com/ConsenSys/quorum-tools

[3] ——, "Quorum," 2022. [Online]. Available: https://github.com/ConsenSys/quorum/tree/master/tests

[4] A. Pavlovics, "Jpmorgan's quorum blockchain performance testing," 2018. [Online]. Available: https://scandiweb.com/blog/jpmorgan-s-quorum-blockchain-performance-testing/

[5] ConsenSys, "quorum-acceptance-tests," 2022. [Online]. Available: https://github.com/ConsenSys/quorum-acceptance-tests

[6] Hyperledger, "Fabric," 2022. [Online]. Available: https://github.com/hyperledger/fabric/tree/main/core/ledger/ledgermgmt

[7] ——, "Fabric," 2022. [Online]. Available: https://github.com/hyperledger/fabric/tree/main/integration

[8] ——, "Fabric," 2022. [Online]. Available: https://github.com/hyperledger/fabric/tree/main/core/ledger/mock

[9] ——, "Fabric," 2022. [Online]. Available: https://github.com/hyperledger/fabric-test

[10] J. Chang, *Software Quality Control Through Formal Method*. Western Michigan University, 2020.

[11] Hyperledger, "Indy-sdk," 2022. [Online]. Available: https://github.com/hyperledger/indy-sdk/libindy/tests

[12] ——, "Indy-sdk," 2022. [Online]. Available: https://github.com/hyperledger/indy-sdk/wrappers/python/tests

[13] ——, "Indy-sdk," 2022. [Online]. Available: https://github.com/hyperledger/indy-sdk/twrappers/java/src/test/java/org/hyperledger/indy/sdk

[14] ——, "Indy-sdk," 2022. [Online]. Available: https://github.com/hyperledger/indy-sdk/twrappers/ios/libindy-pod/libindy-demoTests

[15] ——, "Indy-sdk," 2022. [Online]. Available: https://github.com/hyperledger/indy-test-automation

[16] ——, "Iroha," 2022. [Online]. Available: https://github.com/hyperledger/iroha/tree/main/scripts

[17] ——, "Iroha," 2022. [Online]. Available: https://github.com/hyperledger/iroha/tree/main/test

[18] MultiChain, "multichain," 2022. [Online]. Available: https://github.com/MultiChain/multichain/tree/master/src/obj-test

[19] R. Corda, "Testing your changes," 2022. [Online]. Available: https://docs.r3.com/en/platform/corda/4.8/open-source/testing.html

[20] Corda, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/core/src/test

[21] ——, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/node/src/test

[22] ——, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/core-tests

[23] ——, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/node/src/integration-test

[24] ——, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/testing/testserver/src/integration-test/kotlin/net/corda/webserver

[25] ——, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/testing

[26] ——, "Corda," 2022. [Online]. Available: https://github.com/corda/corda/tree/release/os/4.10/testing/smoke-test-utils

[27] Hyperledger, "Besu," 2022. [Online]. Available: https://github.com/hyperledger/besu/tree/main/besu/src/test

[28] ——, "Besu," 2022. [Online]. Available: https://github.com/hyperledger/besu/tree/main/container-tests

[29] Baeldung, "Docker test containers in java tests," 2019. [Online]. Available: https://www.baeldung.com/docker-test-containers
[30] Hyperledger, "Besu," 2022. [Online]. Available: https://github.com/hyperledger/besu/tree/main/acceptance-tests
[31] ——, "Burrow," 2022. [Online]. Available: https://github.com/hyperledger/burrow/tree/main/integration
[32] ——, "Burrow," 2022. [Online]. Available: https://github.com/hyperledger/burrow/tree/main/testnet
[33] ——, "Burrow," 2022. [Online]. Available: https://github.com/hyperledger/burrow/tree/main/tests
[34] ——, "Sawtooth-core," 2022. [Online]. Available: https://github.com/hyperledger/sawtooth-core/tree/main/cli/tests
[35] ——, "Sawtooth-core," 2022. [Online]. Available: https://github.com/hyperledger/sawtooth-core/tree/main/integration/sawtooth_integration/tests
[36] ——, "Sawtooth-core," 2022. [Online]. Available: https://github.com/hyperledger/sawtooth-core/tree/main/validator/tests
[37] Sawtooth, "Sawtooth faq: Validator," 2022. [Online]. Available: https://sawtooth.hyperledger.org/faq/validator.html
[38] L. Petrik, "4 famous blockchains and the programming languages they're built with," 2021. [Online]. Available: https://betterprogramming.pub/blockchain-programming-language-680e5e8fc1c7
[39] Algorand, "Go-algorand," 2022. [Online]. Available: https://github.com/algorand/go-algorand
[40] ——, "Go-algorand," 2022. [Online]. Available: https://github.com/algorand/go-algorand/tree/master/test
[41] ——, "Go-algorand," 2022. [Online]. Available: https://github.com/algorand/go-fuzz
[42] Bitcoin, "Bitcoin," 2022. [Online]. Available: https://github.com/bitcoin/bitcoin
[43] ——, "Bitcoin," 2022. [Online]. Available: https://github.com/bitcoin/bitcoin/blob/master/src/test
[44] ——, "Bitcoin," 2022. [Online]. Available: https://github.com/bitcoin/bitcoin/tree/master/src/test
[45] ——, "Bitcoin," 2022. [Online]. Available: https://github.com/bitcoin/bitcoin/tree/master/src/wallet/test
[46] I. Output, "Cardano node," 2022. [Online]. Available: https://github.com/input-output-hk/cardano-node
[47] ——, "Testing," 2022. [Online]. Available: https://input-output-hk.github.io/cardano-wallet/contributing/Testing
[48] ——, "Cardano node," 2022. [Online]. Available: https://github.com/input-output-hk/cardano-node/issues/604
[49] ——, "Cardano node," 2022. [Online]. Available: https://github.com/input-output-hk/cardano-node/tree/master/cardano-testnet
[50] ——, "Cardano node," 2022. [Online]. Available: https://github.com/input-output-hk/cardano-node/tree/master/cardano-cli/test
[51] ——, "Cardano node," 2022. [Online]. Available: https://github.com/input-output-hk/cardano-wallet/blob/master/test/e2e
[52] Dfinity, "Ic," 2022. [Online]. Available: https://github.com/dfinity/ic/tree/master/testnet
[53] ——, "Ic," 2022. [Online]. Available: https://github.com/dfinity/ic/tree/master/rs/consensus/tests
[54] ——, "Ic," 2022. [Online]. Available: https://github.com/dfinity/ic/tree/master/tests
[55] EOSIO, "Eos," 2022. [Online]. Available: https://github.com/EOSIO/eos/tree/master/unittests
[56] ——, "Eosio testnet quick start guide," 2021. [Online]. Available: https://developers.eos.io/welcome/latest/quick-start-guides/testnet-quick-start-guide/index
[57] ——, "Eos," 2022. [Online]. Available: https://github.com/EOSIO/eosio-wasm-spec-tests/tree/22f7f62d5451ee57f14b2c3b9f62e35da50560f1
[58] ——, "Eos," 2022. [Online]. Available: https://github.com/EOSIO/eosio.system/tree/main/tests
[59] ——, "Eos," 2022. [Online]. Available: https://github.com/EOSIO/eos/tree/master/tests
[60] Ethereum, "Go-ethereum," 2022. [Online]. Available: https://github.com/ethereum/go-ethereum
[61] ——, "Go-ethereum," 2022. [Online]. Available: https://github.com/ethereum/go-ethereum/tree/master/tests
[62] ——, "Go-ethereum," 2022. [Online]. Available: https://github.com/ethereum/system-testing
[63] Y. Fu, M. Ren, F. Ma, H. Shi, X. Yang, Y. Jiang, H. Li, and X. Shi, "Evmfuzzer: detect evm vulnerabilities via fuzz testing," in *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 1110–1114.
[64] Y. Yang, T. Kim, and B.-G. Chun, "Finding consensus bugs in ethereum via multi-transaction differential fuzzing," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 349–365. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/yang
[65] Iota, "Bee," 2022. [Online]. Available: https://github.com/iotaledger/bee/tree/mainnet-develop/bee-crypto/tests
[66] ——, "Bee," 2022. [Online]. Available: https://github.com/iotaledger/bee/tree/mainnet-develop/bee-network/bee-gossip/src/tests
[67] ——, "Bee," 2022. [Online]. Available: https://github.com/iotaledger/bee/tree/mainnet-develop/bee-test
[68] Solana-labs, "Solana," 2022. [Online]. Available: https://github.com/solana-labs/solana/tree/master/core/tests
[69] ——, "Solana," 2022. [Online]. Available: https://github.com/solana-labs/solana/tree/master/system-test
[70] ——, "Solana," 2022. [Online]. Available: https://github.com/solana-labs/solana/tree/master/program-test
[71] Stellar, "Stellar-core," 2022. [Online]. Available: https://github.com/stellar/stellar-core/tree/master/src
[72] ——, "Stellar-core," 2022. [Online]. Available: https://github.com/stellar/stellar-core/tree/master/src/ledger/test
[73] ——, "integration tests," 2022. [Online]. Available: https://github.com/stellar/integration-tests
[74] ——, "Supercluster," 2022. [Online]. Available: https://github.com/stellar/supercluster
[75] ——, "Stellar-core," 2022. [Online]. Available: https://github.com/stellar/stellar-core/tree/master/src/test
[76] ——, "Deploy a production version on mainnet," 2022. [Online]. Available: https://developers.stellar.org/docs/anchoring-assets/enabling-cross-border-payments/setting-up-production-server/#general-tests
[77] ——, "Anchor validator," 2019. [Online]. Available: https://anchor-tests.stellar.org/
[78] ——, "anchor-transfer-server-validator-ui," 2022. [Online]. Available: https://github.com/stellar/anchor-transfer-server-validator-ui