Designing test cases

This project contains a lot of different parts to focus on. Examples are Uri testing, reading, writing, server states, client topology, behavior of separate classes and the behavior as a whole.

However, not all functions should be tested when a user is installing this software. This is because for example, several tests are designed to follow server behavior when shutting down or starting up while also reading or writing to that same server, amongst other things. Some of the edge cases might fail caused by race conditions. These cases might never be encountered under

normal use and therefore not necessary to test while installing.

The tests done on the user system are always as simple as possible with only one server setup. On the test servers the are always four servers. One for the simple cases, two for standalone combinations, three to test replica sets and four to test arbiters and sharding.

Test locations

The test situations where software is tested are the following.

1. My developer system (a RedHat Fedora linux system on a 4 core with 8 threads)
2. Test server Travis-CI (an Ubuntu linux)
3. Test server Appveyor (a windows system). For the moment it is not yet possible for me to get the tests running...
4. Users installation. There is in theory no feedback for this sort unless there are any issues created on github.

Test servers

I want to test the software against two server versions which are the minimum and maximum version of a server range I want to test. E.g. 3.6.9 and 4.0.5.

- On test locations 2 and 3 above, both versions are tested because there is plenty of time to test. Also a broad spectrum of tests is tried. However, only the the installation tests are required to be successful are the ones from the basic tests and the smoke tests. The others may fail whithout overall test fail implication. The mongodb server versions are downloaded in specific directories and used from there.
- On the developer system I can choose whatever version I want to test and which tests.

On the developer system, a Fedora 24+, 4-core, 8 threads

Mostly **Smoke and sanity testing** are done to see if installing is without problems. Depending a

what is changed, one, some or all of the other tests are executed.

User install from ecosystem or Pause/CPAN

Only **Smoke and sanity testing** are done. The installation tests on the user system must be minimized to test only one server version and a minimum test set. The server executable will be provided in the resources of the package and taken from a downloaded archive to save space. Test against oldest server in the supported range.

The sandbox

A sandbox is created to run the tests. Its purpose is to install the servers configurations there so the tests do not have to run on existing user servers. This makes it also possible to run commands which involve shutting down a server or starting it again. Also, creating a replica set will be possible. All kind of things you don't want to happen on your server. The addresses are always **localhost** and a free port number of which the search for it starts at 65010.

The configuration is for 8 servers except the user installation where there will only be one server configuration. Servers 1 to 4 have the lowest version and 5 to 8 have the highest of the current test range.

Server	Purpose
1	Standalone alone or in a mix or replicate server with 2 and 3
2	Standalone alone or in a mix or replicate server with 1 and 3
3	replicate server with 1 and 2
4	shard or arbiter server in a replica set of servers 1 to 3
5	same as 1 but on max server version
6	same as 2 but on max server version
7	same as 3 but on max server version
8	same as 4 but on max server version

Test software

The **Smoke and sanity tests** are the tests placed in directory **./t**. The other tests are found in directories **./xt**. A wrapper is made to run the tests on the test servers and return the results. It has the option to return success even when some tests had failed.

Test catagories

There are several types of tests according to the wikipedia article software testing. Here are some types of tests to say something about;

1. **Installation test:** *To assure that the system is installed correctly and working at actual customer's hardware.* This part is mainly the task of the **Perl6** installation manager **zef**. It will download the software from a repository and if done, starts testing. When successful the software is installed at the proper locations. On the test servers **zef** is instructed to behave a bit differently so more tests can run and with different **MongoDB** servers.
2. **Compatibility testing**. These test are designed to see if there are differences in behavior when server versions are changed.
3. **Smoke and sanity testing**. The tests are started by the perl6 installation manager **zef**.
4. **Functional testing**. Execution of all available tests, except for the **Compatibility testing**, **Destructive testing** and **Software performance testing**.
5. **Destructive testing**. There are tests to test exceptions when parts fail.
6. **Software performance testing**. These are benchmark programs of parts of the system to be compared with later tests. This is not a part of the tests to install or tests on the Travis-CI and AppVeyor systems.
7. **Security testing**. Sometime later there might come some tests to see if the driver is hackable. At this moment I'm quite a noob in this. The International Organization for Standardization (ISO) defines this as: *type of testing conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorized persons or systems cannot use, read or modify them, and authorized persons or systems are not denied access to them.*

Tests

The test programs **t/099-start-servers.t** and **t/998-stop-servers.t** are used to start and stop every test and are not included in the information below and the results table at the end.

Installation test

- Installation on user system by **zef**. Only download on test servers.

Compatibility testing

- Deprecation cycle tests on version of servers.
- Tests of behavior of the software when newer mongodb servers are installed.

t/[2-5]*.t: Smoke and sanity testing

- CRUD testing and some additional tests.
- Software behavior tests. These tests are designed to see how server status and client topology are settled.

Functional testing

- Basic tests of simple classes like **MongoDB::URI**

xt/Basic/020-config.t: Server config tests

1. Config test
2. Control test

xt/Basic/060-header.t: Wire protocol header tests

1. Header encoding and decoding
2. Query and reply
3. Get more operand

xt/Basic/070-uri.t: Uri string tests

1. Server names
2. URI key value options
3. Default option settings
4. Username and password
5. Reading any of the stored values
6. Failure testing on faulty URI strings

xt/Replica/6*.t: Replica server set tests

1. Replica set server in pre-init state is rejected when replicaSet option is not used
2. Replica set server in pre-init state is not a master nor secondary server, read and write denied
3. Replica set pre-init initialization to master server and update master info
4. Convert pre-init replica server to master
5. Replica set server rejected when there is no replica option in uri
6. Standalone server rejected when used in mix with replica option defined
7. Add servers to replicaset
8. Replicaset server master in uri, must search for secondaries and add them
9. Replicaset server secondary or arbiter, must get master server and then search for secondary servers

Destructive testing

- Test exceptions when no servers are found or are down
- Test exceptions when server topology is wrong
- Behavior stress tests. These tests are designed to see how server status and client topology changes when changes take place in the server configuration.
    - Driver behavior when a server goes down, starts up or changes state.
    - Topology and server states a driver can be in. These are held in the MongoDB::Client and MongoDB::Server objects.
    - Behavior tests are done against servers.
        - Standalone servers
        - Replica server tests.
        - Sharding using mongos server.

        - Accessing other server types such as arbiter.

t/Behavior/110-client.t Client object interaction tests

1. Unknown server which fails DNS lookup.
2. Down server, no connection.
3. Standalone server, not in replica set.
4. Two standalone servers.

t/Behavior/111-client.t: Client server interaction tests

1. Standalone server brought down and revived, Client object must follow.
2. Shutdown and restart server while inserting records.

Software performance testing

This greatly depends where the servers are for the user but for these tests the servers are on the same system as the test software.

- Tests how fast a MongoDB::Client object can connect
    - Standalone server
    - Replica server set
    - Shard server
- Tests how fast small and large documents are processed

Security testing

- Secure server connection tests
- Accounting tests.
- Authentication tests.

t/Accounting/112-client.t: Client authentication tests

1. Account preparation.
    - Create an account on a database.
    - Create account with wrong data
    - Create admin account to shutdown --auth server

2. Restart to authenticate
3. Test CRUD
    - on authorized database
    - on non-authorized database
4. Shutdown by user with no privileges or wrong database

Test table.

Test Type	Test date	Server#	Topology	Server Version	Softw. Version	Note	Result*
Installation	20190325	1	Single	3.6.9	0.43.6	zef install	SL
Compatibility							
Smoke							
	20190328	1	Single	3.6.9	0.43.6		SL
	20190328	5	Single	4.0.5	0.43.6		SL
Functional							
	30032019	-	-	-	0.43.7	wire prot.	SL
	30032019	-	-	-	0.43.7	config	F
	31032019	-	-	-	0.43.8	config	SL
	30032019	-	-	-	0.43.7	uri	SL
		1,2,3	ReplicaSet*	3.6.9			
		5,6,7	ReplicaSet*	4.0.5			
		1,2,3,4	Sharded	3.6.9			
		5,6,7,8	Sharded	4.0.5			
Destructive							
	31032019	1,2	Single	3.6.9	0.43.7	mix tests	F
	31032019	5,6	Single	4.0.5	0.43.7	mix tests	F
Performance							

Test Type	Test date	Server#	Topology	Server Version	Softw. Version	Note	Result*
Security							
	20190321	1	Single	3.6.9	0.43.6	authentication	F
	20190329	1	Single	3.6.9	0.43.7	authentication	SL
	20190329	5	Single	4.0.5	0.43.7	authentication	SL

*) Results are coded like (S)uccess, (F)ail, (L)inux, (W)indows. When W or L is not mentioned, it fails or is successful on both systems.