Project NAR: Combined SRS-SDD-Test Document

1. System Requirements

a) Functional Requirements

ID	Requirement	Status
1	Users will be able to register to the system.	OK
2	Users will be able to associate several computers with their users.	OK
3	Users will be able to upload files to the system.	OK
4	Users will be able to download previously uploaded files from the system.	OK
5	Users will be able to delete previously uploaded files from the system.	OK
6	Users will be able to create directory structures in the system.	OK
7	Users will be able to traverse directory structures in the system.	OK
8	Users will be able to list files and directories in the system.	OK

b) Non-functional Requirements

ID	Requirement	Status
9	UI will be easy to use and understand.	OK
10	System will provide %99 expected availability for files.	IN PROGRESS
11	System will be fair to users that the users that provide more will get more service.	IN PROGRESS
12	System will not corrupt files.	OK
13	System will not interrupt user’s regular workflow and use minimal resources.	OK

c) Project Plan and Scope

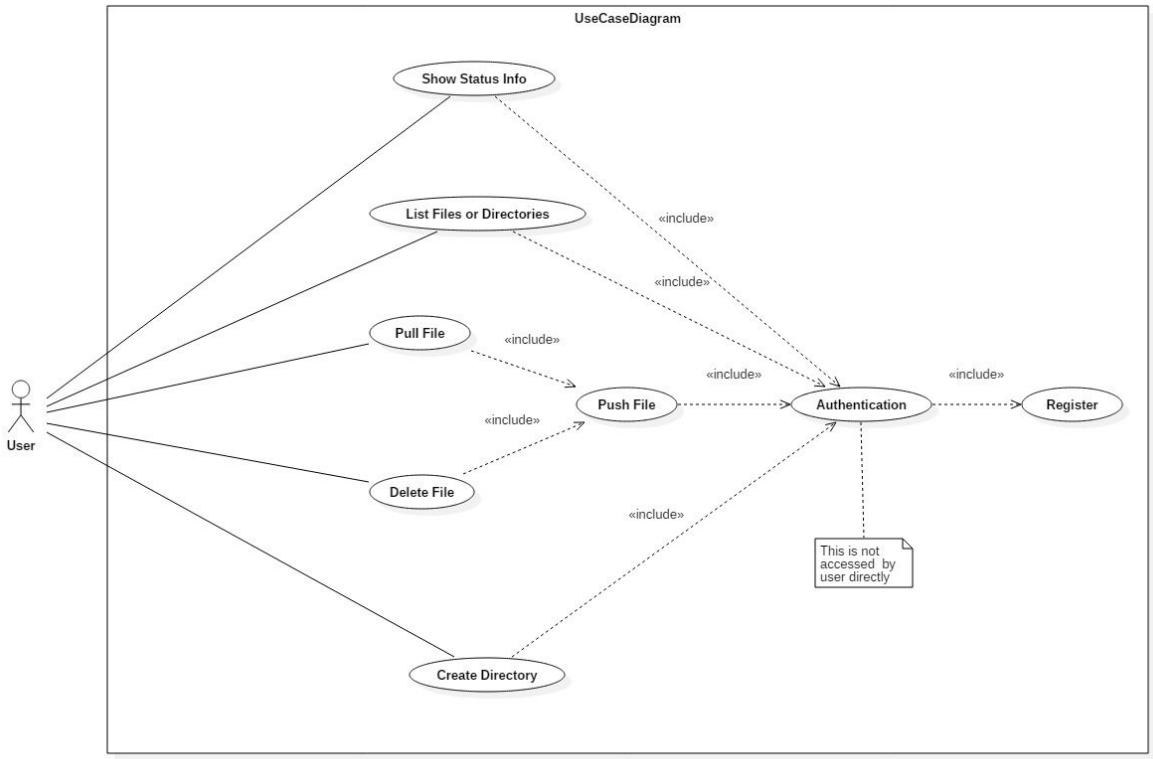
Our product will provide free, secure, private and limited-by-contribution file cloud storage system to its users. Main problem that we want to solve is centralization of file clouds. We want to create a system that is not traceable by any company or group and secure and unlimited by design. System will create a collective storage by using each user's own hard disk area.

Software will consist of following functionalities:

1. Register to system : This functionality will register user to the system. User will be entering a username and password (this password will not be stored anywhere). After entering username and password system will be asking monthly bandwidth and quota limit to registered machine.
2. Check Status : User will be able to check his/her current machine status which will contain Machine id, server id, server port, nar folder and file folder.
3. Push file : Upon push request, the file will be encrypted by a key that is known only to the user and will be fragmented into little pieces. The user will ask for a list of peers from the central entity to push his pieces to them. The central entity will coordinate the connections between peers. To provide high availability of files, the central entity will create redundancy by instructing peers to share pieces with other peers
4. Pull file : When a user wants to get his files back, he will connect to the central entity with his proof of identity and requested file and piece id. The central entity will provide a connection with a peer that has the requested piece of file. After connection, the piece will be downloaded directly from the peer. When the user collects all pieces of a file, the pieces will be defragmented and decrypted. The original file will be available to the user.
5. Delete file : If user decides to delete their files from system, central entity will search machines for which they have the parts of the file. If those machines are active, central entity will send delete request to them and remove chunk's status with that machine in database. If machine is not active, deletion operation will be kept at machine's database information and chunk's connection to machine will be removed. When machine becomes active, it will check database for deletion operations. If there are, it will proceed them.
6. Create directory : User will be able to create directories inside the nar system. User first will ask central entity for new directory and central entity will create it for that user and update database accordingly.

7. List directories and files : When a user wants to list the files that he has in the system, he will ask the central entity to give a list of his files providing proof-of-identity. Then, the central entity will return with the list of files that user has in the cloud.

d) Use Cases



2. System Design

a. Module Structure

Project is composed of following modules:

1. Cryption: Contains classes and functions for RSA encryption/decryption and AES encryption/decryption. *This is a base module.*
2. Logging: Contains a simple logger class. *This is a base module.*
3. Messaging: Divided into two submodules:
    a. IPC Messaging: This module contains message type classes and related functions that are used for IPC communication between UIs and local daemons. *This is a super-module that uses Low Level Messaging module.*
    b. Daemon-Server Messaging: This module contains message type classes and related functions that are used

for daemon-server communication. *This is a super-mdule that uses Low Level Messaging module.*

4. Socket: Divided into two submodules:
    a. Socket: TCP with SSL wrapper over boost ssl sockets. Provides simple interface to exchange data. *This is a base module.*
    b. USocket: RDT over UDP with NAT hole punching capability is implemented on this module. *This is a base module.*
5. File: Wrapper for C++ standard fstream interface. Provides simpler functions to interact with files. *This is a base module.*
6. Low Level Messaging: Lowest protocol that interacts with sockets. Messaging modules use this module as a utility module. *This is a super-module that uses Socket, JSON and Base64 modules.*
7. Database: Built over mysqlcppcon. All database interaction is done over this bridge module. *This is a base module.*
8. Base64: Provides base64 encode/decode services. *This is a base module.*
9. JSON: Provides functions to interact with JSON data. *This is a base module.*
10. Server Global Module: Composed of a single class that stores global information about the server status. *This is a base module.*
11. Server Actions: The main module of server that runs other modules to accomplish tasks. Every request is handled by a function defined in this module. *This is a super-module that uses Database, Daemon-Server Messaging, Logging, Cryption modules.*
12. Daemon Global Module: Composed of a single class that stores global information about the daemon status. *This is a base module.*
13. Daemon Actions: The main module of daemon that run other modules to accomplish tasks. Every request from UI is handled by a class defined in this module. *This is a super-module that uses File, IPC Messaging, Daemon-Server Messaging, Logging, Cryption modules.*
14. Daemon Reactive Module: Provides ways to follow server's directions when things need to done. Server sends some requests to this module when peer is in passive mode.*This is a super-module that uses File, Daemon-Server Messaging, Logging, Cryption modules.*

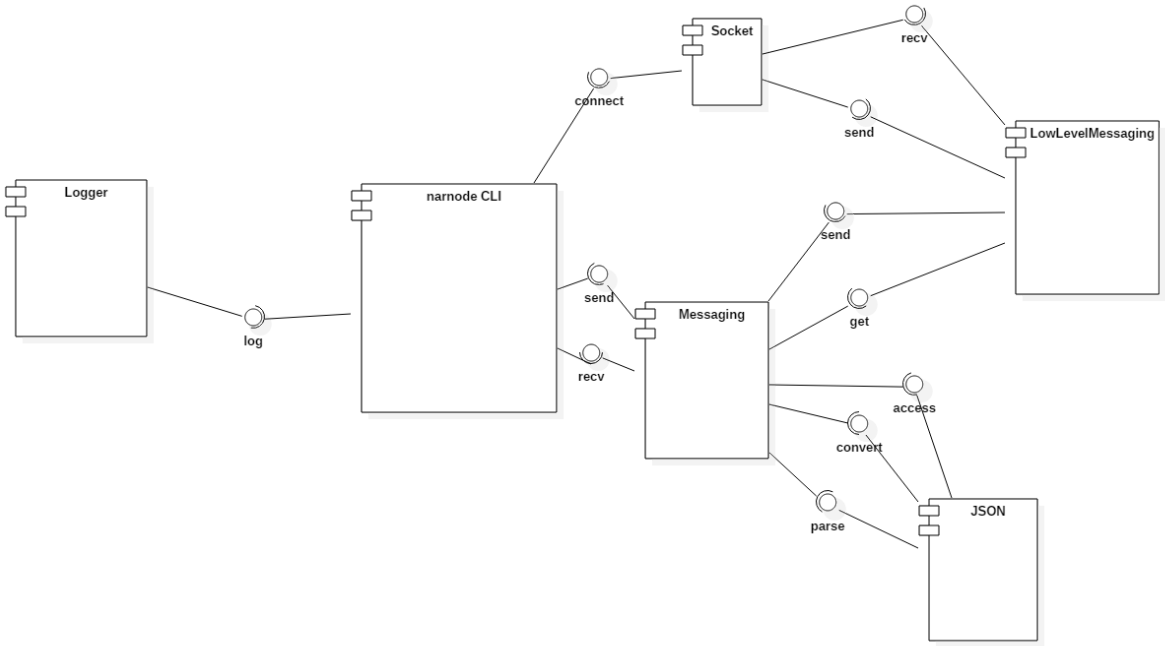
15. CLI Module: A simple command line interface to interact with the system. *This is a super-module that uses IPC Messaging, Logging modules.*

b. Source Code Structure

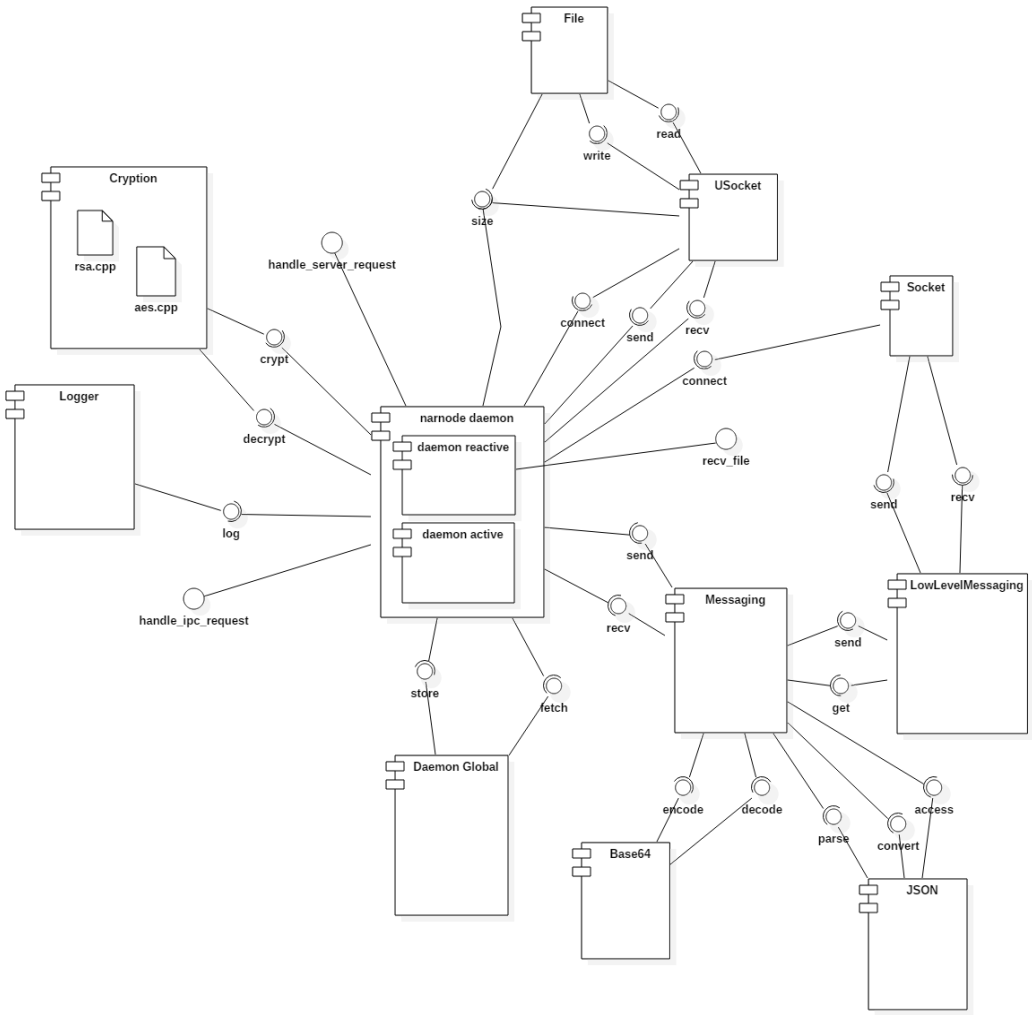
```
nar/
    lib/
        Cryption/                          #Cryption Module
            aes.cpp, aes.h                 #AES Cryptor
            rsa.cpp, rsa.h                 #RSA Cryptor
        Exception/                         #Exceptions
            Exception.h
        Logging/                           #Logging Module
            logger.cpp, logger.h
        Messaging/                         #Messaging Module
            messaging_utility.cpp,messaging_utility.h
            MessageTypes/                  #Message Classes
        Socket/                            #Socket Module
            Socket.cpp,Socket.h            #Socket Submodule
            USocket.cpp,USocket.h          #USocket Submodule
        base64/                            #Base64 Module
            base64.h, base64.cpp
        json.hpp                           #JSON Module
        cptl.hpp                           #ThreadPool Library
    narnode/                               #Node related mods
        cli.cpp/h, clitasks.cpp/h          #CLI Module
        global.cpp, global.h               #Daemon Global Mod.
        reactive.cpp, reactive.h           #Daemon React. Mod.
        utility.cpp, utility.h             #Low Level Messaging
        File/                              #File Module
            File.cpp, File.h
        ActiveTask/                        #Daemon Actions Mod.
            …                              #Task Class&Functions
    narserver/                             #Server related mods
        Database.cpp/h, dbstructs.h        #Database Module
        ServerGlobal.cpp/h                 #Server Global Module
        peers.cpp/h, sockinfo.cpp/h        #Server utilities
    Actions/                               #Server Actions Mod.
        …                                  # Task Functions
```

c. Component Diagram

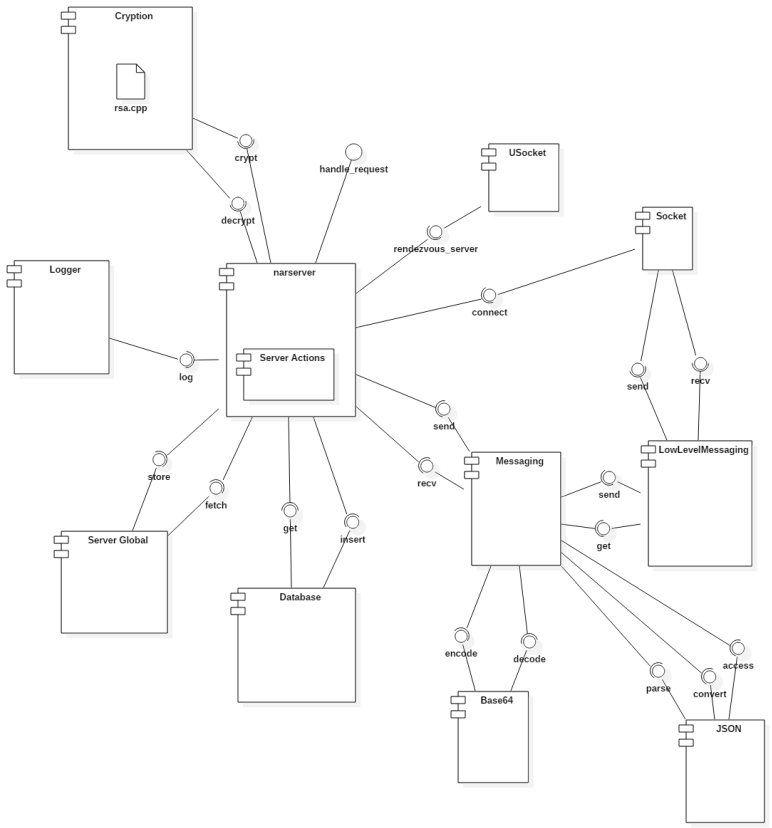
Components of narnode CLI:



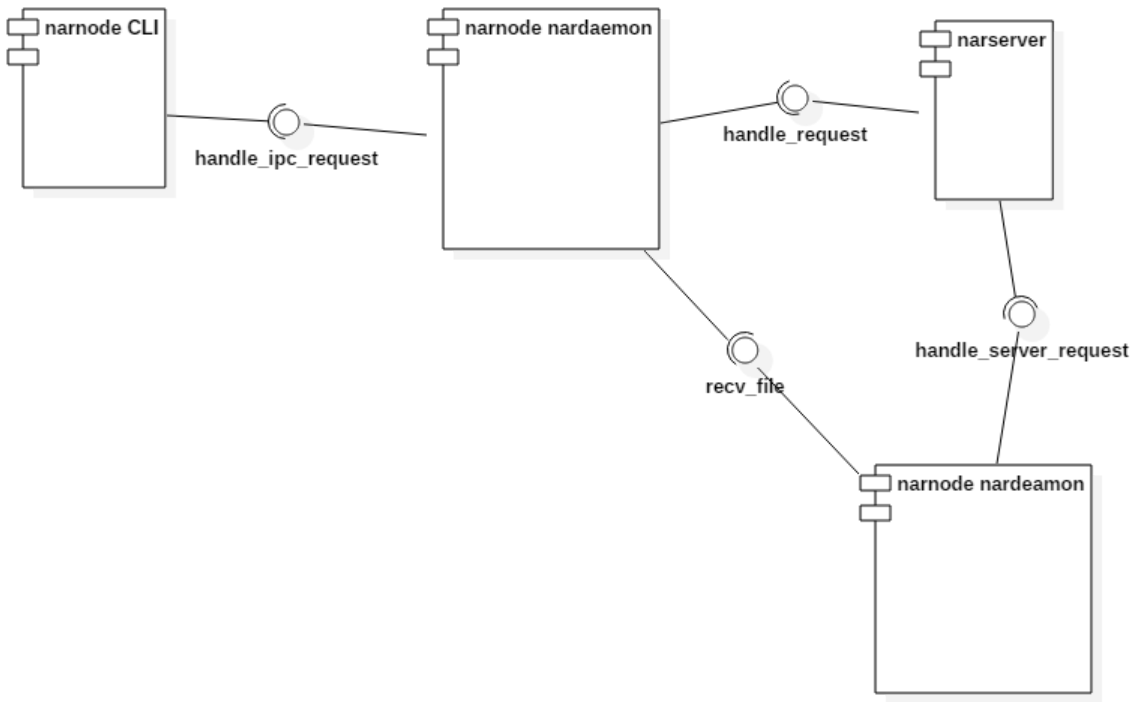
Components of narnode daemon:



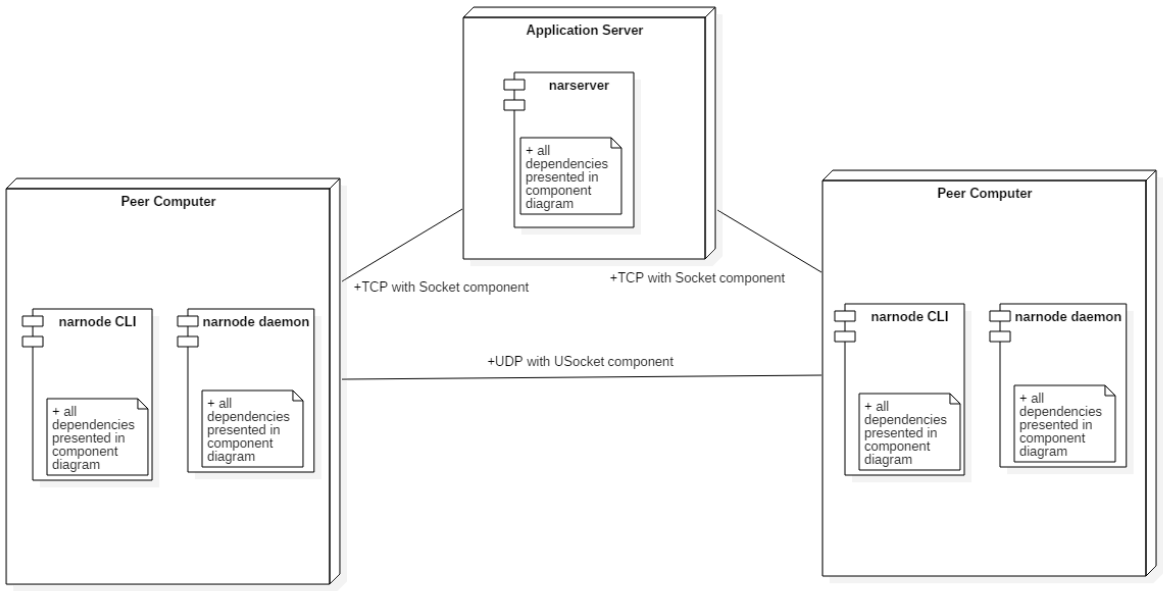
Components of narserver:



All components together:



d. Deployment Diagram



3. Testing

a) Test Plan & Scenarios

Testing has 3 main phases: **F**unctionality testing(**U**nit & **S**ystem tests), **S**tress testing and **P**enetration testing.

ID	Test Description	Status
Functionality Tests		
FU1	AesCryptor class public functions testing. [nar/lib/Cryption/aes.h, nar/lib/Cryption/aes.cpp]	PASS
FU2	RsaCryptor class public functions testing. [nar/lib/Cryption/rsa.h, nar/lib/Cryption/rsa.cpp]	PASS
FU3	base64 functions testing. [nar/lib/base64/base64.cpp, nar/lib/base64/base64.h]	PASS
FU4	Socket class public functions testing. [nar/lib/Socket.h, nar/lib/Socket.cpp]	PASS
FU5	USocket class public functions testing. [nar/lib/USocket.h, nar/lib/USocket.cpp]	PASS

FU6	File class public functions testing. [nar/narnode/File/File.h, nar/narnode/File/File.cpp]	PASS
FU7	Low Level Messaging testing. [nar/narnode/utility.h, nar/narnode/utility.cpp] (send_message, get_message)	PASS
FS1	Authentication end-to-end testing [DaemonActive, DAS Messaging, Server]	PASS
FS2	KeepAlive end-to-end-testing [DaemonReactive, DAS Messaging, Server]	PASS
FS3	First Setup Register end-to-end testing [DaemonActive, DAS Messaging, Server]	PASS
FS4	Register end-to-end testing [CLI, CDA Messaging, DaemonActive, DAS Messaging, Server]	PASS
FS5	Mkdir end-to-end testing [CLI, CDA Messaging, DaemonActive, DAS Messaging, Server]	PASS
FS6	PushFile end-to-end testing [CLI, CDA Messaging, DaemonActive, DAS Messaging, Server, SDR Messaging, DaemonReactive]	PASS
FS7	PullFile end-to-end testing [CLI, CDA Messaging, DaemonActive, DAS Messaging, Server, SDR Messaging, DaemonReactive]	PASS
FS8	LS end-to-end testing [CLI, CDA Messaging, DaemonActive, DAS Messaging, Server]	PASS
FS9	DeleteFile end-to-end testing. [CLI, CDA Messaging, DaemonActive, DAS Messaging, Server, SDR Messaging, DaemonReactive]	PASS
PENETRATION TESTS		
P1	Low Level messaging attack: Make test for every step. For each request flow, break the flow in every step.	PASS
P2	SSL Handshake attack: Disturb SSL with handcrafted packets.	PASS
P3	High Level messaging attacks: Disturb high level protocols in all levels.	PASS
P4	Slowloris Attack	PASS

b) Testing Code Structure

```
tests/                          #Test main directory
    unit/                       #Contains unit tests
        unit.py                 #Runs all unit tests
        fu1/
        fu2/
        ...
    system/                     #Contains system tests
        system.py               #Runs all system tests
        fs1/
        fs2/
        ...
    stress/                     #Contains stress test
        stress.py               #Runs all stress tests
        s1/
        s2/
        ...
    penetration/                #Contains penetration tests
        penetration.py          #Runs all penetration tests
        p1/
        p2/
        ...
```