SASKEN

Python Test Framework for Automated Testing of Network Protocols

Whitepaper

Abstract:

This article will provide the best approach towards building realistic automation frameworks for network enabled devices, based on higher level abstraction philosophy. A test automation framework provides an execution environment for the automation of test scripts. These frameworks can be utilized in a widespread manner regardless of the underlying technology. Testing a special skill in isolation before release could also potentially save millions of dollars for organizations. It enables the release of high-quality products, avoiding any post-sale quality resolving issues such as broken features or functions.

Author:
Allappagoud Biradar, Architect-Product Testing,
Product Engineering Services, Sasken

Table of Content

Why Opt for Black-box Automated Testing

The intent of black-box testing could be questioned, as typical testing focuses on providing inputs and analyzing the outputs. The extra effort of diving into the internal details may seem like unnecessary effort. However the following reasons highlight the necessity for its adoption:

a) Regressions

Most engineers work in an agile environment which is a common issue with manual black box testing. If there is a sprint scheduled for every two weeks and your iteration is also for two weeks. Now the team responsible for testing must re-evaluate the sprint every two weeks and which would lead to regression problems. Therefore, automation would play a significant role solving this issue.

b) Effective Leveraging of Code

Black-box automated tests can evaluate voluminous production code with limited lines of automated scripts. Approximately five to ten lines of unit testing code is necessary to test one line of production code. Therefore, the quantity of test code increases drastically with that of production code. For black-box automated testing, the ratio is flipped over and proved more beneficial. One line of black-box automated script could easily test 1,000 lines of production code.

The following are few examples of black-box test cases which are executed between server and client in end-to-end networking domain measuring the TCP throughput:

- Max TCP Throughput – Downstream
- Max TCP Throughput – Upstream
- Max TCP Throughput – Bidirectional

Common Failure Points to Avoid in Black-box Testing

When it comes to adopting black-box testing there are some common mistakes made which are as follows:

a) Overreliance on Recording and Playback of Tests

The first thing observed in automation testing is the capability of recording and playback of testing. It may seem like a good idea, but developers may change their system code to accommodate the same. This ends up breaking thousands of automated tests and therefore never works effectively in the testing industry.

b) Lack of Framework

A tester can record a test, but if they learn the language underlying the recording tool, then they have more control over test scripts. This proves beneficial especially when the overall flow of the application changes after introducing new workflows. Without a proper framework, all testing efforts would fail. The framework would also support scalability as and when testing efforts increase.

c) Developing Tests Mimicking Production Code

Once the necessary framework is available, test teams typically develop tests similar to production code. Automation test code cannot mimic the nature of production code. If complex scripts are written, then test team will face challenges in interpreting and maintaining them.

Effective Test Automation Methodology

The objective with test automation is the utilizing of standard testing techniques to identify manual tests that can be automated. Agile methodology is seeing widespread popularity when it comes to automation methods and techniques. The time taken for design and coding, validation of the design scripts with pre-existing test data, and adoption of the same for functional and regression testing need to be considered. Test Automation specialists need to implement automated test suites with a focus on long term utilization of the suites. Some key agile-based test automation techniques are mentioned below:

- Small & lean tests to lower maintenance
- Loose coupling of tests and data. Build smart configuration files
- Periodic reviews to ensure test suites are valid
- Supporting execution of multiple regression tests during each sprint
- Maintenance of an automation backlog and prioritizing candidate test cases

Separation of Automation Framework

It is important to isolate the automation framework and the set of tests that it will use. The framework will have minimal interaction with the application being tested.

a) Simple Tests

The goal of designing a good framework is to help test teams write easily understood test code. Automation efforts are usually hindered when the test breaks. A closer inspection of the code would lead to confusion if it is needlessly complex.

b) Ensure Tests Drive Framework Creation

Apart from avoiding the risk of over engineering, tests should also drive the framework development process. This can be done by initially writing the test objectives in simple terms and then develop a framework that helps achieve these objectives.

c) Test Configuration & Test Data

Separating out the configuration parameters and test data for a set of tests or test suites is the main goal in testing. The best practice to follow would be the creation of two configuration files with one being for network configuration and the other for test specific configuration. Care must be taken to ensure no hard-coded configuration parameters are present in the test script. A YAML-based configuration file can be used to ensure simplicity and easy readability.

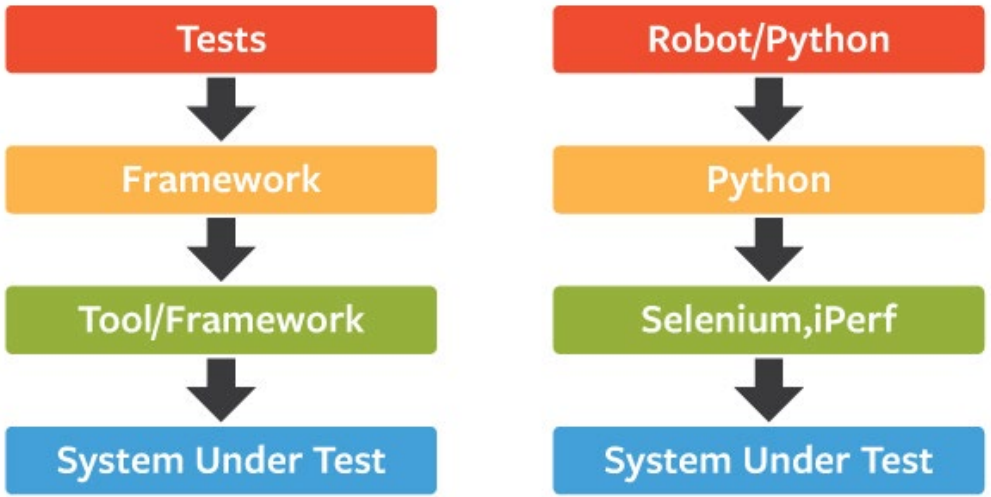
d) Overview of Folder Structure

A simple and easy to navigate folder structure for the test framework would be ideal for better efficiency. Here is one sample folder structure which can be used while creating a test framework:

- Test Automation – Home Folder
  - Config.
    - Contains configuration files. Sub folders can be created based on project requirements
  - Libraries
    - Contains Python libraries. Sub folders can also be created for Python packages library like Telnet, SSH, REST API etc.
  - Tools
    - To store any external tools such as Chrome driver to launch Google Chrome, or any shell script file for device initialization
  - Steps
    - This contains the steps taken in the testing process. Ideally small functions are developed using Python as per the input from the test case
  - Tests
    - This would contain the exact test keywords which are developed using a robotic test framework

Test Automation Architecture

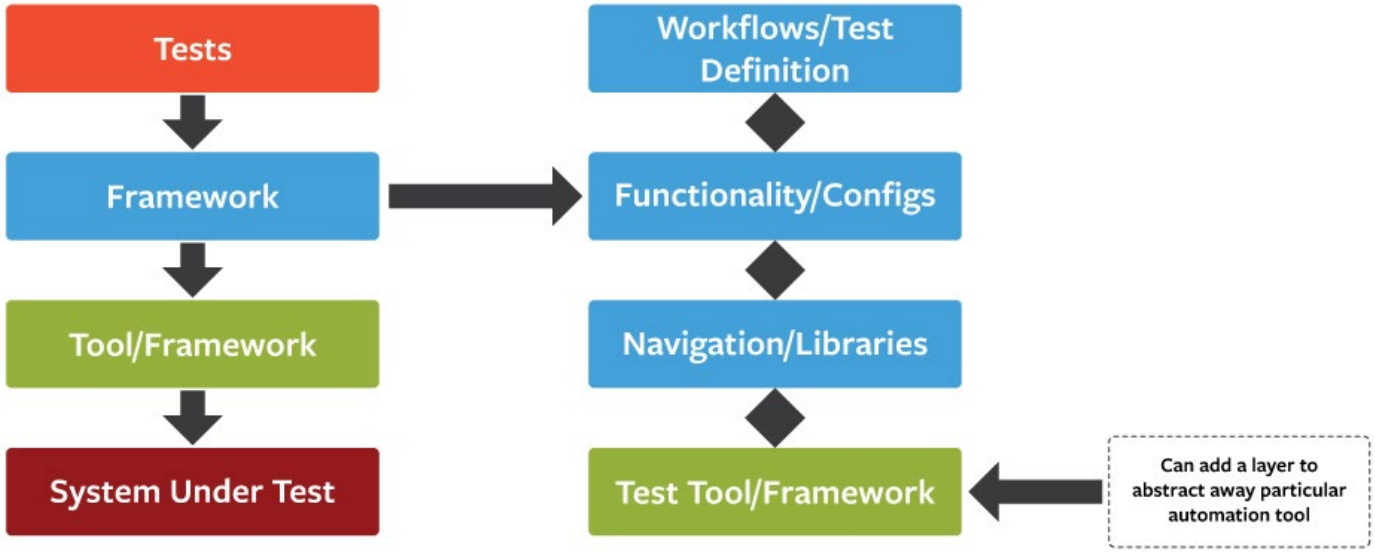
The appropriate automation architecture flow that test teams can develop based on project requirements is depicted in the following diagram:



Tests developed need to utilize the entire capability of this framework. The framework internally could use any other standard testing tool to interact with system under test. Tests will use the framework which will interact with the systems under evaluation. The reason for this is because all the testing reference will be limited to the framework and what the framework knows about the system. If there are any workflow changes in the system under test, the tests don't necessarily have to change just because they are using the framework. We just need to update those workflow changes in the framework to ensure the continuous working of the tests.

Layered Approach for Automation Architecture

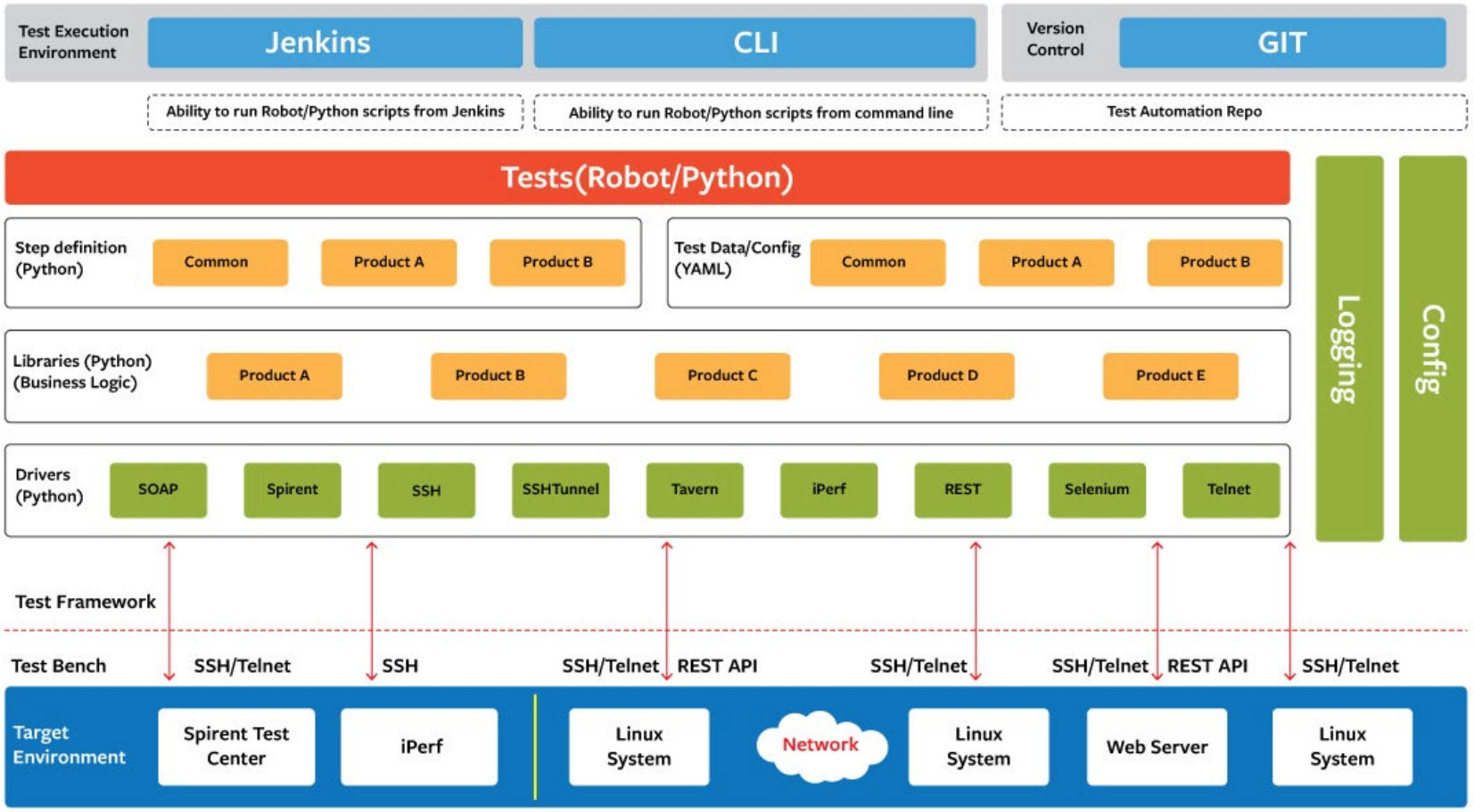
In this approach, the framework is divided into layers of functionality as depicted in the following diagram:



The workflows are built based on test scenarios. By doing so, the objective is to test the entire application based on end-to-end workflows and not just individual components of the application. All applications have some form of navigation, so there's no point in repeating the frameworks for every functionality. Test teams needs to write navigation components which deal with various navigation options of the system under test. If multiple test tools are being used to test the system, it is better to create a separate layer to handle multiple tools.

Test Automation Architecture for Network Protocol Testing

The following diagram details the Python-based test automation framework for data path testing of network enabled systems. The robot test framework can be used to develop tests as per the keyword driven style.



Overview of Test Automation Tool Chain

The table below details various tools and utilities required for developing a Python test framework. Configuration files can be written in YAML format to enable the reading of variables and parameters from the configuration file by the robot or python tests.

Jenkin jobs and pipeline management routines could also be written with and Apache Groovy script which are easy to develop and maintain.

#	Tool Name	Objective	Comments
1	Python	Test Script Development and Test Library/Driver Development	Python is popular scripting language to write automated tests
2	Robot	Test Script Development (Keyword Driven)	Robot is keyword-driven testing framework that uses tabular test data syntax.
3	Selenium	Web page Automation	Test Framework for Web application
4	JSON	Configuration File	The json file for software product configuration. JSON file is used to store network configuration information.
5	XML	Configuration File	Automated Test Reporting
6	YAML	Configuration File	Test Data & Configuration file
7	Groovy	CI Test Tool	Java like scripts, Jenkins pipeline management
8	Jenkins	CI Test Tool	
9	Shell scripting	Test Script Development	Jenkin jobs can be written in shell scripts
10	Git	Source Code Management	Test Automation
11	Spirent Test Center	Packet Generator & Analyzer	Low level protocol testing
12	Ixia iXLoad	Performance testing for multiple services & network security appliances	This tool can be used for high level applications testing (Facebook)
13	iPerf	The TCP, UDP and SCTP network bandwidth measurement tool	Traffic Generation Tool
14	Tavern	API testing on Terminal	Tavern supports testing RESTful APIs as well as MQTT based APIs

Essential Features of Test Automation Framework

We must keep in mind the key objectives of test framework development which are extensibility, portability, forward integration, and value. This leads to significant cost savings in automation design, execution cycles and regular maintenance. The following features are necessary while developing test framework based on project requirements:

- Smart grouping of test cases to ease selection for execution based on release objectives
- Enable continuous integration of testing CI tools with multiple test data sources
- Compatibility with tools, packaged solutions, web, mobile, and third-party applications
- Develop large repository of keywords towards a common, scalable knowledge base enabling reuse of functional libraries
- Dashboards with clear visibility about project status, test execution results, and analysis reports
- A single point maintenance approach reducing complex maintenance costs and testing time
- The ability to integrate with various open and commercial tools for enhancing test maintenance

Python Libraries

Python supports large sets of built-in libraries to carry out the automation of network-based systems. Some of the libraries have been provided in the following section:

- **iPerf:** Can be used to interact with iPerf3 for the running of UDP and TCP traffic
- **REST API:** Provides connectivity to REST API sessions to perform get, patch, post and delete operations
- **Selenium:** This library interacts with various web applications
- **SOAP API:** Responsible for the creation of a SOAP connection utilizing a Zeep library with the objective of creating a SOAP connection to the necessary WSDL
- **Spirent API:** Used to interact with the Spirent Test center and run TCP & UDP traffic.
- **SSH:** Enables SSH connection for the given IP and login with a given username and password
- **SSHTunnel:** Starts and stops the SSHtunnel. SSHTunnel objects can be used to create tunnel connections through jump servers for DB, API or SSH connections
- **Tavern API:** This library is used for testing RESTful APIs as well as MQTT based APIs

Effective Rules for Framework Design

It is best advised for test teams to takes over the automation work without any prior knowledge of the development history. The complexity should be in the framework and not in tests. The following rules help determine the best way in designing a framework

- Never declare variables as part of the testing process
- Never use the “new” keyword or create new objects
- Tests should not manage state on their own
- Ideally tests should not directly manipulate or interact with systems under test
- Always reduce number of parameters for API calls when possible
- Always use default values instead of providing parameters whenever possible.
- Make the API easier to use by making the internal API less complex

Best Practice for Writing Test Cases

Test case writing is one of the important steps before proceeding to writing automated test scripts. The QA team needs to make appropriate use of test management tools to document manual test procedures along with a proper breakdown of requirements. based on features and sub features. The following guidelines are some important points to be considered while designing test cases:

- Ensure test cases are simple and transparent
- Create test cases keeping end-users in mind
- Follow a consistent naming convention
- Don’t make assumptions and state preconditions
- Ensure test cases meet acceptance criteria
- Test Cases should be easily identifiable, modular and reusable. Make sure one particular functionality is tested
- Avoid too many steps by keeping it short
- Guarantee easy maintenance and scalability
- Should execute well in a test suite either sequentially or in a parallel manner
- Assert expected results and posting of conditions
- Clean up in the tear down
- Avoid the usage hard coded test data
- Peer Review

Conclusion

The idea of automating difficult tasks is possible, although it may not always be practical. However, this should not hinder the test team's ability to think out of the box. There is a clear distinction between errors in the framework and failures of the test. When designing the framework, the only thing that should be tested for is failure. If any other aspect does not function in the framework, then that is the error. Competent tests and frameworks need to be continuously integrated and automated throughout the execution processes.

About Author

Allappagoud has a decade of experience in designing and building test automation framework across industries like Automotive, Devices, Storage, and Network.
At Sasken, he works for the Product Engineering Practice and is responsible for developing and making new test automation solutions for the automotive and networks industries.

SASKEN

Python Test Framework for Automated Testing of Network Protocols

marketing@sasken.com | www.sasken.com

USA | UK | FINLAND | GERMANY | JAPAN | INDIA

Apr 2020