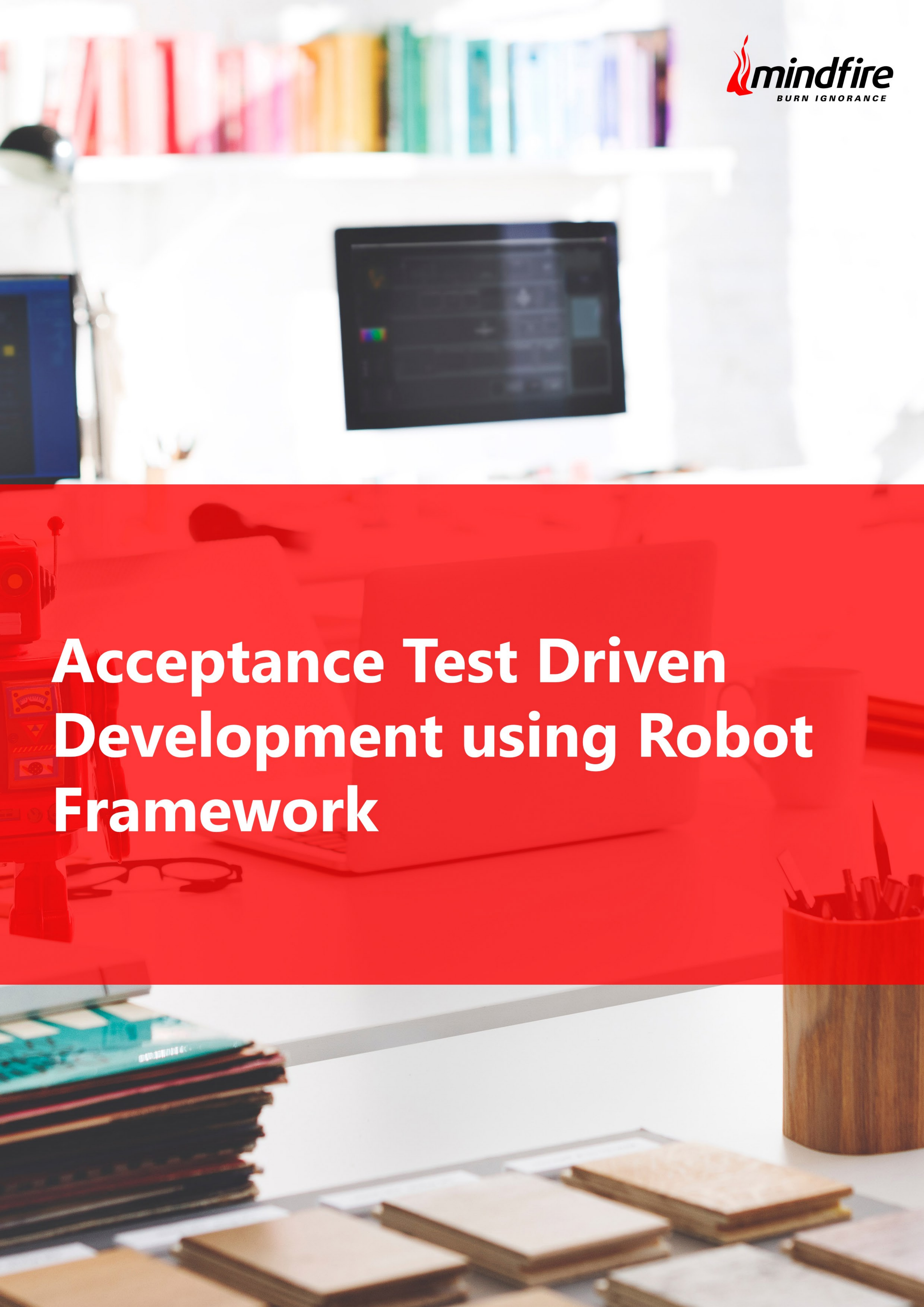


Acceptance Test Driven Development using Robot Framework

ATDD : The Approach

Problem Statement:

The effort required for Test automation preparation and the number of hours spent by developers to build the test scripts was significantly high for the applications that were developed for a US-based home security and intelligent automation service provider. The framework that was getting used was a hybrid one built with Junit; Java being the programming language. Though it was keyword driven, considerable amount of effort was getting required in reality to write the actual test methods/class courtesy the application of object oriented concepts.

The Relevance of the Approach:

The approach suggested was for implementing an Acceptance Test Driven Development using Robot Framework. Applying the new approach needed

- Identifying changes in terms of which tasks to perform on and how to perform
- Identifying what the likely consequences of the changes would be with regards to stakeholders time, resources, capabilities, values, status and satisfaction
- Determining how the stakeholders will perceive the change and whether they will treat this as an opportunity to develop new skills in automation

The older framework was a hybrid framework built with Junit and java as a programming language. Though it was a keyword driven, lot of effort was being put in to write the actual test methods/class itself as it involved the object oriented concepts.

Client Details:

Name: Confidential | **Type:** Home Security System | **Location:** U.S.

Technologies:

ASP.NET, Ember API

What is ATDD?

Acceptance test driven development (ATDD) is an agile method of merging requirement gathering with acceptance testing. The product manager works with the software engineers and QA engineers to go over the requirements and validate it. The QA engineers then write the acceptance test feature files in behavior driven development (BDD) style in given-when-then format.

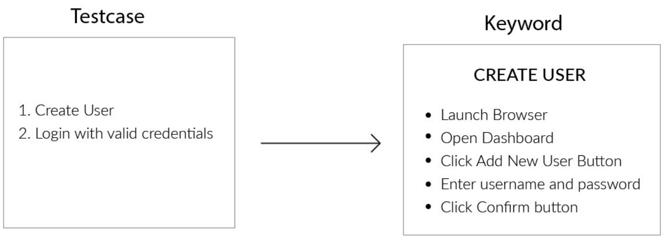
What is Robot Framework (RFW) and how it assures ATDD?

Robot Framework is an open source test automation framework for acceptance testing and ATDD. It is operating system and application independent. The core framework is developed using Python but can also run using Jython (JVM) and IronPython (.Net). RFW follows a keyword driven approach. It has many built-in libraries along with third party library. Tests can be written in BDD styles which makes easy to understand the test cases. It has simple tabular syntax using Robot IDE (RIDE) and has built-in tools to generate clear reports (both HTML and XML format) and detailed logs.

The Technique of Keyword-driven Testing approach

- Keyword-driven testing is a software testing technique that separates the test programming work of test automation from the actual test design. This technique works by specifying keywords for basic operations, like click button, open, close, enter, etc. These keywords are then converted into executable scripts by the automation tool

- Basically, these keywords (action words) simulate real user actions on the tested application.



Testcase

1. Create User
2. Login with valid credentials

Keyword

CREATE USER

- Launch Browser
- Open Dashboard
- Click Add New User Button
- Enter username and password
- Click Confirm button

- In the above Create User keyword consists of a set of other action words, the implementation of which can be stored in a separate library

- Keyword driven approach allows automation team to plan test automation at the development stage

- With RFW, you can develop tests in Python, Java, and .NET. The syntax for describing test cases is based on a keyword driven testing approach and a table format.

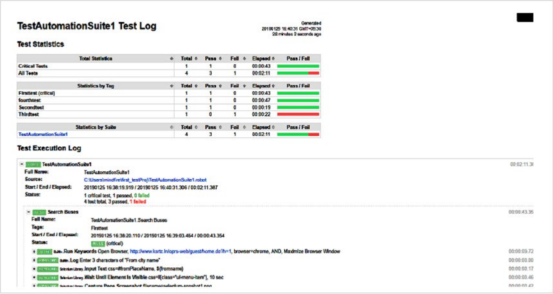
Existing Testing Framework

- Users design a set of pre-defined keywords from steps and scenarios that come from requirements and business processes. RFW utilizes this approach and has a wide variety of internal and external libraries. Apart from that user can create many re -usable keywords and libraries.

- RFW is not an object oriented framework. Users can focus on creating more functional test coverage rather than spending time in learning oops concept which is required in other custom frameworks

- RFW utilize selenium library to generate automated tests for both UI, functional and API testing. Easy to write test cases using tabular format syntax in RIDE and easily readable by non-technical person since, tests are written in BDD format.

- No extra effort to write logs and reports, capturing screenshots, integration with CI tools. Etc. It has all built in., which allows users to focus more on product testing instead of investing time in building the framework itself



TestAutomationSuite1 Test Log

Generated
20190125 16:40:31 GMT+05:30
28 minutes 2 seconds ago

Test Statistics

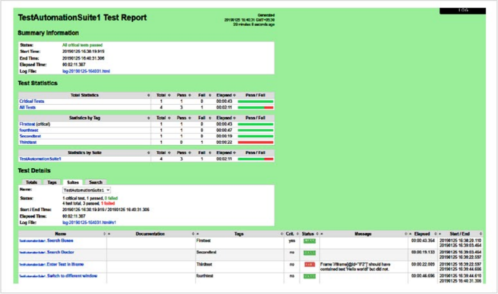
Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:43	
All Tests	4	3	1	00:02:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
Firsttest (critical)	1	1	0	00:00:43	
fourthtest	1	1	0	00:00:47	
Secondtest	1	1	0	00:00:19	
Thirdtest	1	0	1	00:00:22	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
TestAutomationSuite1	4	3	1	00:02:11	

Test Execution Log

```
- SUITE TestAutomationSuite1                                                       00:02:11.3
  Full Name:            TestAutomationSuite1
  Source:               C:\Users\mindfire\first_testProj\TestAutomationSuite1.robot
  Start / End / Elapsed: 20190125 16:38:19.919 / 20190125 16:40:31.306 / 00:02:11.387
  Status:               1 critical test, 1 passed, 0 failed
                        4 test total, 3 passed, 1 failed
  - TEST Search Buses                                                              00:00:43.35
    Full Name:          TestAutomationSuite1.Search Buses
    Tags:               Firsttest
    Start / End / Elapsed: 20190125 16:38:20.110 / 20190125 16:39:03.464 / 00:00:43.354
    Status:             PASS (critical)
    + KEYWORD BuiltIn.Run Keywords Open Browser, http://www.ksrtc.in/oprs-web/guest/home.do?h=1, browser=chrome, AND, Maximize Browser Window   00:00:09.72
    + KEYWORD BuiltIn.Log Enter 3 characters of "From city name"                  00:00:00.00
    + KEYWORD SeleniumLibrary.Input Text css=#fromPlaceName, ${fromname}          00:00:00.17
    + KEYWORD SeleniumLibrary.Wait Until Element Is Visible css=li[class="ui-menu-item"], 10 sec   00:00:00.46
    + KEYWORD SeleniumLibrary.Capture Page Screenshot filename=selenium-screenshot1.png   00:00:00.42
```



TestAutomationSuite1 Test Report

Generated
20190125 16:40:31 GMT+05:30
29 minutes 8 seconds ago

Summary Information

Status:	All critical tests passed
Start Time:	20190125 16:38:19.919
End Time:	20190125 16:40:31.306
Elapsed Time:	00:02:11.387
Log File:	log-20190125-164031.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:43	
All Tests	4	3	1	00:02:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
Firsttest (critical)	1	1	0	00:00:43	
fourthtest	1	1	0	00:00:47	
Secondtest	1	1	0	00:00:19	
Thirdtest	1	0	1	00:00:22	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
TestAutomationSuite1	4	3	1	00:02:11	

Test Details

Name:	TestAutomationSuite1
Status:	1 critical test, 1 passed, 0 failed 4 test total, 3 passed, 1 failed
Start / End Time:	20190125 16:38:19.919 / 20190125 16:40:31.306
Elapsed Time:	00:02:11.387
Log File:	log-20190125-164031.html#s1

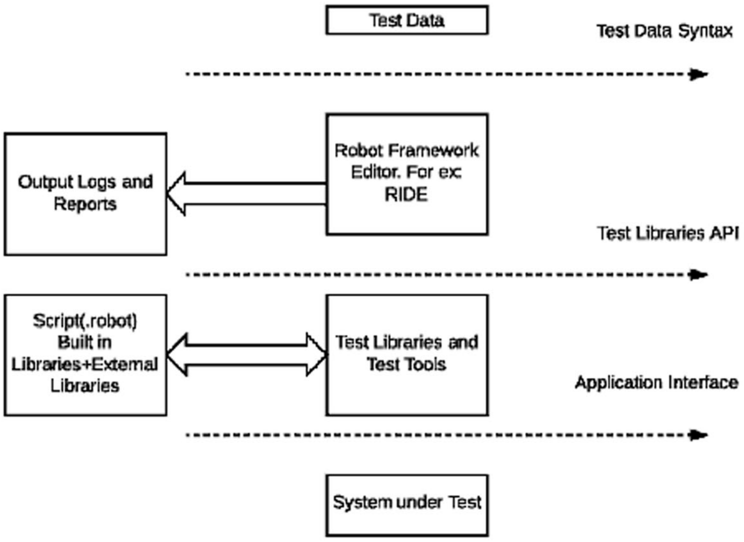
Name	Documentation	Tags	Crit.	Status	Message	Elapsed	Start / End
TestAutomationSuite1.Search Buses		Firsttest	yes	PASS		00:00:43.354	20190125 16:38:20.110 20190125 16:39:03.464
TestAutomationSuite1.Search Doctor		Secondtest	no	PASS		00:00:19.133	20190125 16:39:03.464 20190125 16:39:22.597
TestAutomationSuite1.Enter Text in Iframe		Thirdtest	no	FAIL	Frame '//iframe[@id="IF2"]' should have contained text 'Hello world' but did not.	00:00:22.009	20190125 16:39:22.597 20190125 16:39:44.606
TestAutomationSuite1.Switch to different window		fourthtest	no	PASS		00:00:46.696	20190125 16:39:44.610 20190125 16:40:31.306

- Quick and fast implementation of tests. Can execute parallel tests using pabot library and selenium grid.

- Wide online support and forums along with official documentation for Robot framework under https://robotframework.org/

Robot Framework Testing Architecture

- RFW has a modular architecture that can be extended with bundled and user defined self-made libraries.
- Test data is defined in files using the syntax. A file containing test cases creates a test suite.
- When test execution is started, the framework first parses the test data. It then utilizes keywords provided by the test libraries to interact with the system under test.
- In RFW, tests are sets of tables that contain test data, variables, keywords, and advanced settings (such as library imports and metadata). Each table begins with its name. Robot Framework tables: Settings, Variables, Test Cases and Keywords


Test Data
Test Data Syntax
Output Logs and Reports
Robot Framework Editor. For ex: RIDE
Test Libraries API
Script(.robot) Built in Libraries+External Libraries
Test Libraries and Test Tools
Application Interface
System under Test

Migration Effort, Cost and Challenges

- By automating the test cases, companies have been able to attain maximum coverage and reduced human effort and errors, thus the testing process has been streamlined.
- Companies are increasingly automating their test efforts for code reusability, maximum test coverage, low maintenance cost, least Cost: manual intervention, and ease of reporting.
- Teams are increasingly moving towards open source frameworks instead of investing in commercial licensed tool or building their own framework with a view to cut cost and bring efficiency. RFW is one such open source tool, which is ready to be used with least effort on building the framework itself.
- Easy to learn the tool and some knowledge of Python would suffice.
- Considering migration of existing test suites to RFW, you need not re-develop the scripts from ground level.
- You already have the page objects, keywords and functionality in place, thus, it just needs to be re-written in the tabular format and certain effort needs to be made to refine the keywords and build user keywords for better readability instead of focusing on building the framework architecture.
- Facilitates easier maintenance of test cases and test scripts.

Project Description

We used NRobot server to host application specific keywords. NRobot Server is a .Net based Robot Framework remote server. It can be used to host and expose .Net keywords to Robot Framework, thus bringing the full capabilities of Robot Framework keyword based automation to automation developers.

While we started migrating to RFW, it actually improved the status of the test cases and improved satisfaction of work. Test suites became easy to maintain and more readable by the stakeholders and developers. Number of codes reduced and became easy to maintain and update. The smoke test time were reduced by 50% due to fast execution by RFW.

Better integration with AI powered visual testing tool like Applitools and inbuilt support to upload results to Jira as the test execution progress.

Technical developers at the client side also benefited considerably from the migration as they were involved in system support activities like creating keywords and their counterparts in QA use those keywords to convert the test-case documents to RFW scripts.