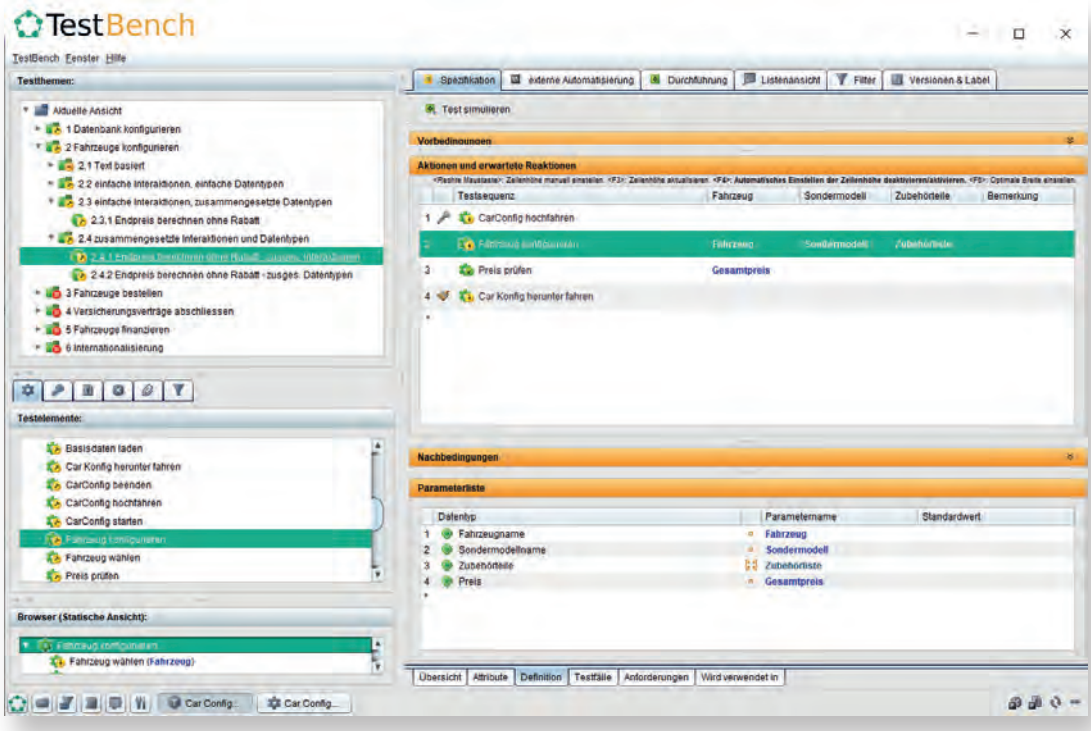
TestBench

Das Smarte Testmanagement und Testdesign Tool

Whitepaper



Whitepaper

TestBench

No-Code-Testautomatisierung mit Low-Code Integration via Robot Framework und TestBench

Autor: Dierk Engelhardt
Datum: 05.05.2025

imbus AG
Kleinseebacher Str. 9
91096 Möhrendorf
Deutschland
Tel. +49 9131/7518-0
Fax +49 9131/7518-50
info@testbench.com
https://www.testbench.com/

Problemstellung

In der agilen und kontinuierlichen Softwareentwicklung steigt die Notwendigkeit, Software schnell und mit hoher Qualität zur Verfügung zu stellen. Die Testautomatisierung spielt hierbei eine entscheidende Rolle, um Qualität effizient und kostengünstig sicherzustellen. Bei Bereitstellung und Wartung einer Testautomatisierungslösung müssen spezielle Herausforderungen gemeistert werden, die es immer wieder verhindern, dass ein hoher Anteil von Tests dauerhaft automatisiert durchgeführt werden können:

1. Es ist ausreichend tiefes Fachwissen über das Testobjekt und die zu testenden Funktionalitäten und ihr Zusammenspiel notwendig, damit die richtigen Tests entstehen. Gleichzeitig werden eingehende Kenntnisse in der Nutzung eines Testautomatisierungs-Tools oder -Frameworks mit dazu passendem Entwicklungs-Knowhow benötigt, damit die Tests implementiert und auf Dauer gewartet werden können.
   In der Regel besitzen die sogenannten Fachtester:innen das fachliche Wissen, welche Tests notwendig sind und Testautomatisierer:innen das technische Wissen, wie diese Tests zu implementieren sind. Selten sind diese Fähigkeiten in einer Person vereint.

2. Sollen Tests automatisiert werden, muss die Testspezifikation exakt sein, da sie als Vorlage für die Implementierung gilt. Interpretationsspielräume, wie sie vielleicht im manuellen Test noch akzeptabel sind, sollen nicht vorkommen. Die Testspezifikation muss also zum Zeitpunkt der Implementierung vorliegen, die dann als nachfolgender Schritt realisiert wird. Dies gilt auch für Anpassungen: Erst muss die Testspezifikation angepasst werden, dann folgt die Anpassung des Testautomatisierungs-Codes. Dabei geht wertvolle Zeit verloren, so dass das neben der Entwicklung einer Software gleichzeitige Erstellen der zugehörigen automatisierten Tests meist nicht möglich ist.

Zielstellung: Spezifikation = Testautomatisierung

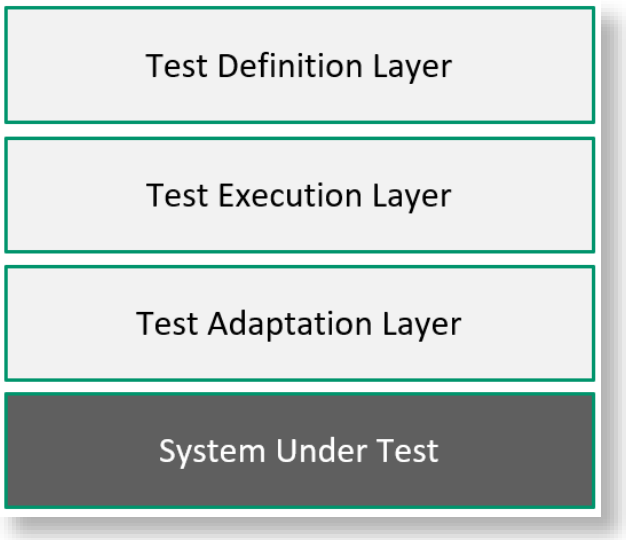
Gesucht ist also eine Vorgehensweise, die es ermöglicht, Tests gleichzeitig zu spezifizieren und zu automatisieren ohne, dass Fachtester:innen technisches Knowhow und Testautomatisierer:innen fachliches Knowhow benötigen.

Die beste Lösung für diese Herausforderungen ist ein kombinierter No-Code- und Low-Code-Ansatz, der es sowohl technisch als auch fachlich versierten Mitarbeiter:innen ermöglicht, sich aktiv an der Testautomatisierung zu beteiligen. Dieses WhitePaper beschreibt eine Lösung, bei der der Testdesigneditor von TestBench mit dem Testframework Robot Framework kombiniert wird.

Konzeptionelle Grundlage

Generische Testautomatisierungsarchitektur

Die Grundlage für das reibungslose Zusammenspiel der fachlichen und technischen Anteile einer Testautomatisierung ist die generische Testautomatisierungsarchitektur. Darin werden die fachlichen und technischen Anteile der Lösung über drei Ebenen bis zum zu testenden System miteinander verbunden:



Schichten der generischen Testautomatisierungsarchitektur

Die Ebenen bedeuten im Detail:

1. **Test Definition Layer**
   Auf dieser Ebene wird die fachliche Testspezifikation erstellt. Die beinhaltet den Ablauf der Tests und die dafür notwendigen Daten.
2. **Test Execution Layer**
   Auf dieser Ebene werden die Tests automatisiert ausgeführt, die Testprotokolle erstellt und die Testergebnisse bereitgestellt.
3. **Test Adaptation Layer**
   Auf dieser Ebene werden die zum zu testenden System passenden implementierten Programme oder Skripte bereitgestellt, die vom Test Execution Layer aufgerufen werden.

4. **System Under Test**
   Diese Ebene stellt das zu testende System (System under Test) mit seinen Technologien und Schnittstellen dar, die vom Test Adaptation Layer genutzt werden.

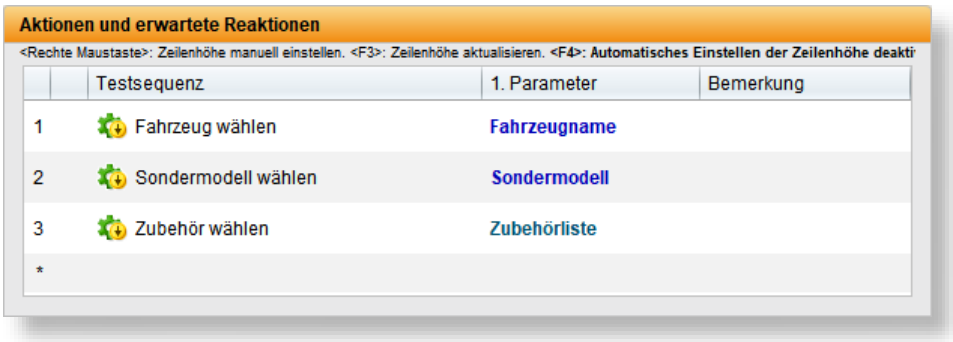
Zwischen den Ebenen sind Schnittstellen definiert, über die die Ebenen miteinander Daten austauschen und kommunizieren. Jede Ebene kann über den Einsatz von dafür spezialisierten Werkzeugen realisiert werden.
Im Fall unserer Lösung stellt der Testdesigneditor von TestBench den Test Definition Layer dar, das Robot Framework den Test Execution Layer und die Robot Framework Libraries den Test Adaptation Layer. TestBench und Robot Framework nutzen als Schnittstelle untereinander die Syntax des Keyword-Driven Test.

Keyword-Driven Testing mit TestBench

TestBench ermöglicht es Testfälle mittels Keyword-Driven Test zu erstellen. Der integrierte Testdesigneditor ermöglicht es, Testabläufe über eine benutzerfreundliche Oberfläche per Drag-and-Drop aus Keywords zu erstellen. Die notwendigen Testdaten können ebenfalls definiert und als Parameter oder Datentabellen genutzt werden. Keywords und Testdaten werden in Bibliotheken verwaltet, die einen schnellen und einfachen Zugriff ermöglichen.

Ein Keyword kann wiederum aus Keywords zusammengesetzt werden, so dass detaillierte Abläufe durch Aggregation als abstraktere, besser verständliche Abläufe abgebildet werden können. Damit entstehen Tests in verständlicher Sprache, die gleichzeitig dem notwendigen Detaillierungsgrad Rechnung tragen. Die Nutzung der Keywords erzeugt klare Testpunkte, da später während der Testdurchführung für jedes Keyword ein Testergebnis ermittelt wird. Die Nutzung von Keyword-Driven Test bedeutet auch einen hohen Grad an Wiederverwendung, da einmal definierte Keywords immer wieder aufgerufen werden. Bei Änderungen müssen auch nur einzelne Keywords angepasst werden, so dass sich auch der Wartungs- und Pflegeaufwand deutlich reduziert.
Ein Beispiel verdeutlicht die Vorteile und Möglichkeiten dieser Methode:

Es soll ein Fahrzeug aus Grundmodell, Sondermodell und mehreren Zubehörteilen konfiguriert werden. Die dafür notwendige Testsequenz wird über drei Keywords abgebildet:



Die Parameterliste der Testsequenz bildet die Datenschnittstelle der Testsequenz ab:

Die Werte der Parameter der Testsequenz werden in Datentypen abgelegt und können in der zugehörigen Datentabelle mit Werten belegt werden:

Jede Zeile dieser Tabelle repräsentiert einen Durchlauf der Testsequenz mit den jeweiligen Werten und stellt damit einen konkreten Testfall dar.Diese Testsequenz mit ihren drei Keywords kann jetzt über die gleiche Art und Weise in einem neuen Keyword abgebildet werden, welches das Fahrzeug konfigurieren abstrahiert.

Dieses neue Keyword kann wiederum in eine neue Testsequenz zum Bestellen eines Fahrzeugs eingefügt werden:

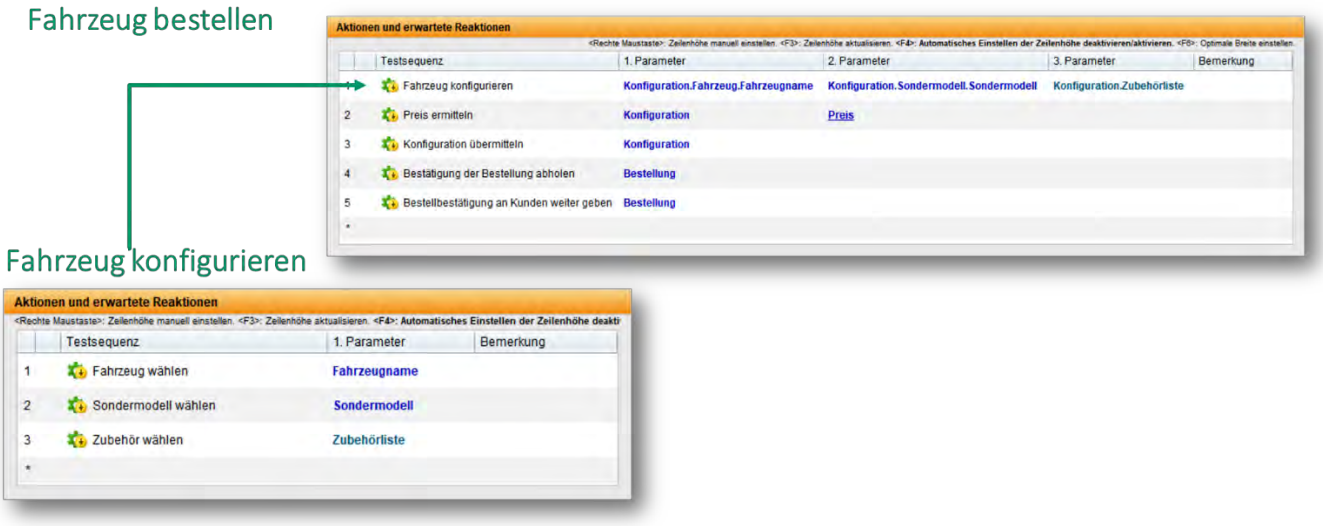
Aktionen und erwartete Reaktionen

<Rechte Maustaste>: Zeilenhöhe manuell einstellen. <F3>: Zeilenhöhe aktualisieren. <F4>: Automatisches Einstellen der Zeilenhöhe deaktivieren/aktivieren. <F6>: Optimale Breite einstellen.

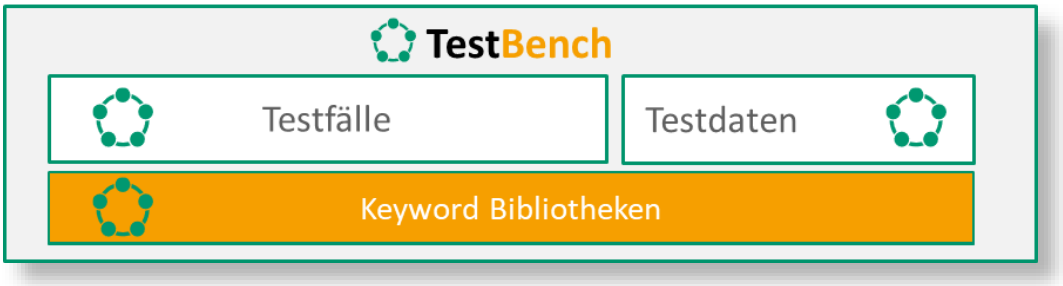
	Testsequenz	1. Parameter	2. Parameter	3. Parameter	Bemerkung
1	Fahrzeug konfigurieren	Konfiguration.Fahrzeug.Fahrzeugname	Konfiguration.Sondermodell.Sondermodell	Konfiguration.Zubehörliste	
2	Preis ermitteln	Konfiguration	Preis		
3	Konfiguration übermitteln	Konfiguration			
4	Bestätigung der Bestellung abholen	Bestellung			
5	Bestellbestätigung an Kunden weiter geben	Bestellung			
*					

Die drei Parameter können, ebenso, wie die Keywords, in Strukturen zusammengefasst werden. In der Abbildung oben zeigt das Beispiel, wie mit der Struktur eine vollständige Fahrzeugkonfiguration abgebildet wird, die nach einmaliger Konfiguration an nachgelagerte Keywords übergeben wird.

Es ergibt sich das folgende Bild einer sehr verständlichen, aber ausreichend detaillierten Testspezifikation durch Nutzung von Keyword-Driven und Data-Driven Test:



Damit stellt sich der Test Definition Layer wie folgt dar:



TestBench stellt somit den No-Code-Anteil der Lösung dar.

Keyword-Driven Testing mit Robot Framework

Das Robot Framework basiert wie TestBench auf einem Keyword-Driven Testansatz, bei dem einzelne Testaktionen ebenfalls durch Keywords definiert werden. Es ermöglicht, Tests mit einer leicht lesbaren, tabellarischen Syntax zu schreiben, die auch für Nicht-Techniker:innen gut verständlich ist. Bei der Erstellung von Test sind grundlegenden Programmierkenntnisse von Vorteil.
Robot Framework bietet bereits eingebaute Standardbibliotheken (z. B. BuiltIn, OperatingSystem, Collections) und kann durch externe Bibliotheken erweitert werden. Es stehen mehrere Hundert Keyword Bibliotheken zur Verfügung, die fertige Implementierungen für zu testende Technologien enthalten, so z.B. die Selenium Library oder die Browser Library zum Testen von Web-Oberflächen oder die Robosapiens Library zum Testen von SAP. Technische Aktionen wie UI-Interaktionen, API-Abfragen oder Datenbankoperationen können so durch bereits definierte, einfache Befehle ausgeführt werden. Erst wenn diese Bibliotheken nicht mehr ausreichend sind, ist es notwendig eigene Bibliotheken oder Ergänzungen von Bibliotheken zu programmieren und erst dann sind tiefgehende Programmierkenntnisse notwendig.
Für Robot Framework wird in der Regel mittels Entwicklungsumgebungen, wie Microsoft Visual Studio Code oder JetBrains IntelliJ Idea entwickelt. Diese Entwicklungsumgebungen werden meist durch Extensions, wie z.B. Robotcode für VS Code, erweitert, so dass auch für die Robot Framework Syntax Funktionalitäten, wie Code Completion zur Verfügung stehen. Damit werden diese Umgebungen zu vollwertigen Entwicklungswerkzeugen für Robot

Framework, die Entwicklungsumgebungen für Programmiersprachen, wie C# oder Python in nichts nachstehen.

Ein Testfall zum Konfigurieren eines Fahrzeugs könnte in Robot Framework (hier mittels VS Code und Extension RobotCode) wie folgt aussehen:

```
Fahrzeug konfigurieren
    Select Base Model    Rolo
    Select Special Model    Luxus
    Select Accessory    Sportfelgen
```

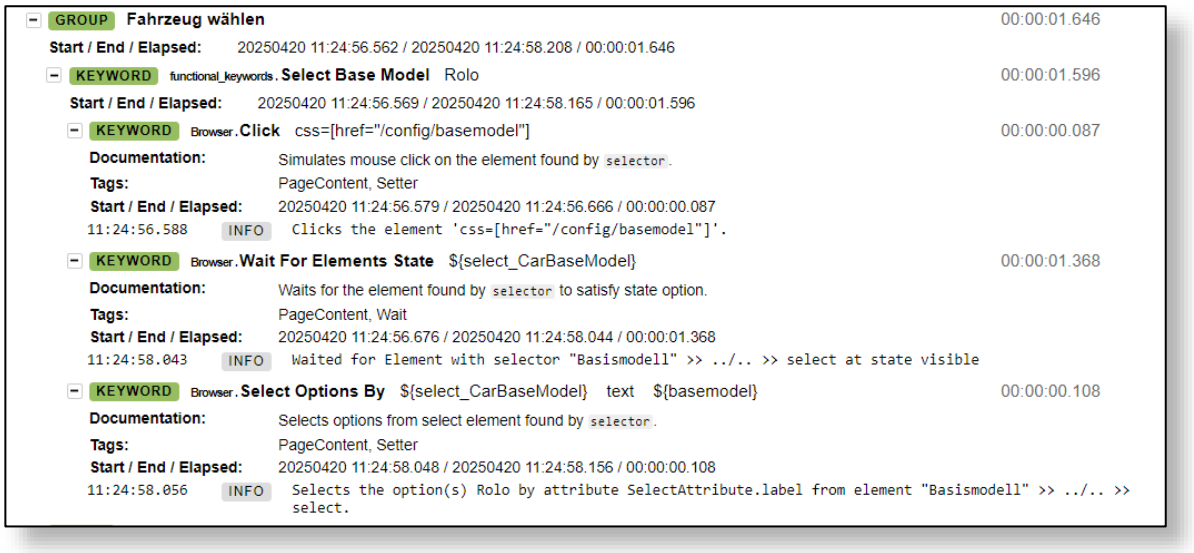
In der tabellarischen Darstellung lassen sich die Testabläufe gut lesen und die Parameter schnell zuordnen. Hinter einem Keyword, wie „Select Base Model“ steht wiederum eine für das System under Test spezifische Implementierung, die Testautomatisierer:innen vornehmen:

```
Select Base Model
    [Arguments]    ${basemodel}
    Click    css=[href="/config/basemodel"]
    Wait For Elements State    ${select_CarBaseModel}
    Select Options By    ${select_CarBaseModel}    text    ${basemodel}
```

Auf dieser Ebene werden in unserem Beispiel fertige Keywords (z.B. „Click“ oder „Select Options By“) aus der Open Source Browser Library für Robot Framework verwendet, so dass bis zu diesem Schritt die Robot Framework Syntax genutzt wird.

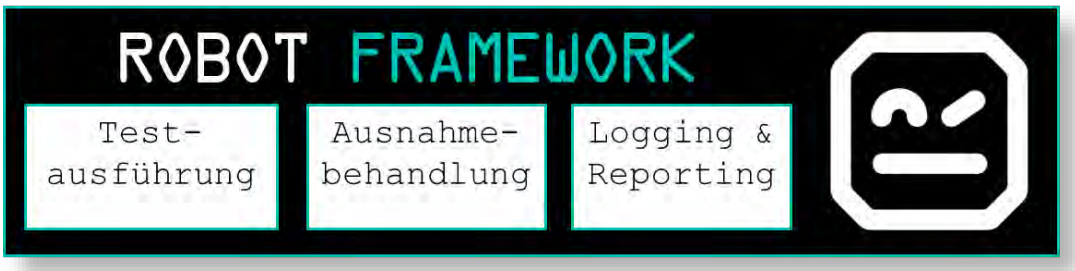
Ein weiterer, sehr entscheidender Teil des Robot Frameworks sind dessen Fähigkeiten, bei der automatisierten Testdurchführung die Ergebnisse zu ermitteln, Ausnahmen zu behandeln und in sehr aussagekräftigen Protokollen darzustellen. Bei jeder Ausführung werden übersichtliche HTML-Berichte erstellt, welche Statistiken zu Erfolg, Fehlschlag, Dauer usw. enthalten.

Ein Protokoll unseres Beispiels sieht nach einer erfolgreichen Ausführung wie folgt aus:



Für jeden Schritt des Testablaufs können die Ergebnisse bis auf die unterste Ebene der Keywords eingesehen werden.
Im Falle eines Fehlers, kann das Problem ebenfalls bis auf diese untersten Ebenen eingegrenzt werden, was ein entscheidender Vorteil bei der Ursachenanalyse darstellt.

Damit stellt sich der Test Execution Layer wie folgt dar:



Robot Framework stellt somit den Low-Code-Anteil unserer Lösung dar.

Die Implementierung des Keywords „Click“ in unserem Beispiel in der Browser Library sieht nun wie folgt aus:

```
@keyword(tags=("Setter", "PageContent"))
def click(self, selector: str, button: MouseButton = MouseButton.left):
    """Simulates mouse click on the element found by ``selector``.

    This keyword clicks an element matching ``selector`` by performing the following steps:
    - Find an element matches selector. If there is none, wait until a matching element is attached to the DOM.
    - Wait for actionability checks on the matched element, unless ``force`` option is set.
      If the element is detached during the checks, the whole action is retried.
    - Scroll the element into view if needed.
    - Use `Mouse Button` to click in the center of the element, or the specified position.
    - Wait for initiated navigation to either succeed or fail.


    | =Arguments= | =Description= |
    | ``selector`` | Selector element to click. See the `Finding elements` section for details about the selectors. |
    | ``button`` | Defaults to ``left`` if invalid. |

    Keyword uses strict mode, see `Finding elements` for more details about strict mode.

    Example:
    | `Click`    id=button_location
    | `Click`    id=button_location    left
    | `Click`    id=button_location    right

    [https://forum.robotframework.org/t//4238|Comment >>]
    """
    self.click_with_options(selector, button)
```

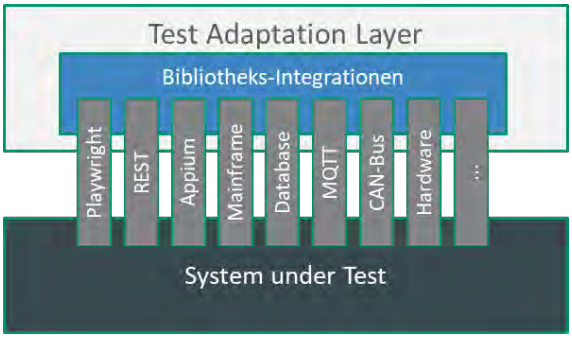
Die Browser Library stellt eine fertige Bibliothek für Robot Framework zum Testen von Web-Oberflächen auf der Basis von Playwright zur Verfügung. Playwright wiederum ist ein Open Source Testautomatisierungsframework, das von Microsoft entwickelt wird. Es ermöglicht Tests für Webanwendungen über verschiedene Browser hinweg. Playwright unterstützt die Automatisierung von Chromium (z. B. Chrome, Edge), Firefox und WebKit (z. B. Safari) mit einer einzigen, konsistenten API.
In der Bibliothek sind alle Keywords sehr gut dokumentiert, so dass sie mit grundlegenden Programmierkenntnissen gut in eigenen Tests verwendet werden können. Sollen diese Keywords erweitert werden, sind ab hier tiefgehenden Programmierkenntnisse notwendig.

Die Aufrufstruktur ähnelt der Struktur in TestBench, deckt aber nicht den fachlichen Teil der Tests ab, sondern bildet die fachlichen Tests auf die technische Ebene des System under Test ab:

[Robot Framework Testfälle]
|
|→ [Keywords des System under Test]
|
|→ [Robot Framework Libraries]
|
|→ Browser Library

Durch die Möglichkeit der Einbindung vieler vorgefertigter Bibliotheken und die Möglichkeit diese an die Eigenheiten des System under Test anzupassen oder sogar eigene Bibliotheken zu erstellen, ist auch nahezu jedes technische Problem lösbar. Auf dieser unteren Ebene ist es auch sehr einfach möglich verschiedene Technologien innerhalb eines Testfalls anzusprechen, denn über die Abbildung der Fachlichkeit auf die Technik können verschiedene Bibliotheken für diverse Technologien gleichzeitig genutzt werden. End-to-End-Tests, startend im Browser, über SAP und Mobile Devices wieder zurück zum Browser werden plötzlich ganz einfach realisierbar.

Damit ist auch der Test Adaptation Layer in unserer Lösung enthalten:



Testdesign trifft Automatisierung

Die Kernidee der Lösung besteht darin, dass Keywords aus dem Robot Framework direkt in TestBench verfügbar gemacht und dort fachlich abstrahiert werden. Fachliche Testdesigner:innen können so Testszenarien auf einer höheren Ebene erstellen, während technische Expert:innen gezielt technische Keywords definieren und pflegen. Neben diesem Bottom-up-Vorgehen kann natürlich auch top-down vorgegangen werden und zuerst die fachlichen Anteile erstellt werden, die dann auf der technischen Ebene realisiert werden.
In der Realität werden Fachtester:innen und Testautomatisierer:innen gleichzeitig und parallel arbeiten, was auch genau der Ansatz der Integration von TestBench mit Robot Framework verfolgt. Es erfolgt kein ein- oder mehrmaliger Import bzw. Export in das jeweils andere System, sondern die Systeme werden laufend miteinander synchronisiert. TestBench dient hierbei als führendes System, das Testfälle generiert und voraussetzt, dass Keywords technisch bereits umgesetzt sind.
Die Fachtester:innen erstellen neue Keywords, die die Testautomatisierer:innen sofort implementieren können, genauso, wie die Fachtester:innen sofort erkennen, dass neue Keywords implementiert wurden, die sie in ihren Tests verwenden können.
Damit diese reibungslose Zusammenarbeit optimal funktioniert, existiert eine TestBench Extension für Robot Framework, die über das API von TestBench alle nötigen Informationen zwischen den beiden Werkzeugen austauscht.
Damit können beide Gruppen in ihrer jeweiligen Domäne bleiben, müssen keinen Werkzeugwechsel vornehmen und sind trotzdem ständig auf dem Laufenden. So ist eine hohe Parallelität beim Erstellen von automatisierten Tests gewährleistet und eine ebenso hohe Reaktionsgeschwindigkeit beider Seiten bei Neuerungen und Änderungen sichergestellt.

Bei der Durchführung der Testfälle ist TestBench ebenfalls das führende System. Wenn Testfälle ausgeführt werden sollen, generiert das System die Testfälle für Robot Framework und geht davon aus, dass die darin genutzten Keywords in Robot Framework implementiert sind.

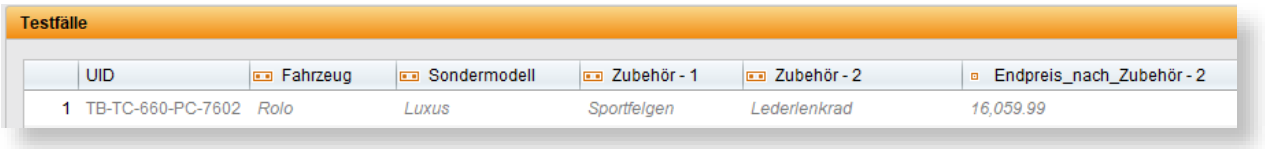
Die Testsequenz aus TestBench wird somit 1:1 abgebildet, wodurch es nicht erforderlich ist, dass die Testautomatisierer:innen über fachliches Know-how verfügen.

Aktionen und erwartete Reaktionen

<Rechte Maustaste>: Zeilenhöh

	Testsequenz	1. Parameter
1	CarConfig starten	
2	Fahrzeug wählen	Fahrzeug
3	Sondermodell wählen	Sondermodell
4	Zubehör wählen	Zubehör - 1
5	Zubehör wählen	Zubehör - 2
6	Endpreis bitte prüfen	Endpreis_nach_Zubehör - 2
7	CarConfig beenden	

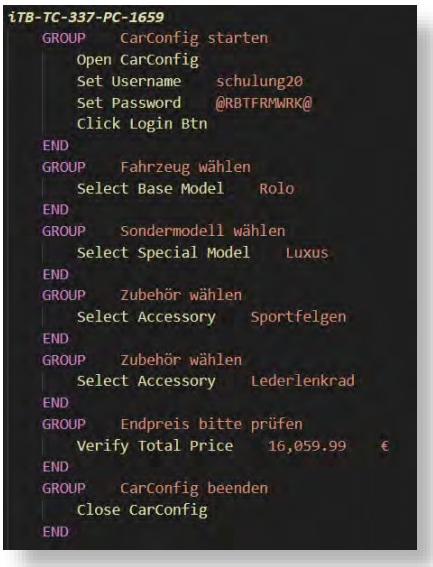
Die zugehörigen Testdaten sind in TestBench wie folgt definiert:



Testfälle

	UID	Fahrzeug	Sondermodell	Zubehör - 1	Zubehör - 2	Endpreis_nach_Zubehör - 2
1	TB-TC-660-PC-7602	*Rolo*	*Luxus*	*Sportfelgen*	*Lederlenkrad*	*16,059.99*

Die Testsequenz und der Datensatz des Testfalls aus TestBench werden als Robot Framework Testfall generiert:

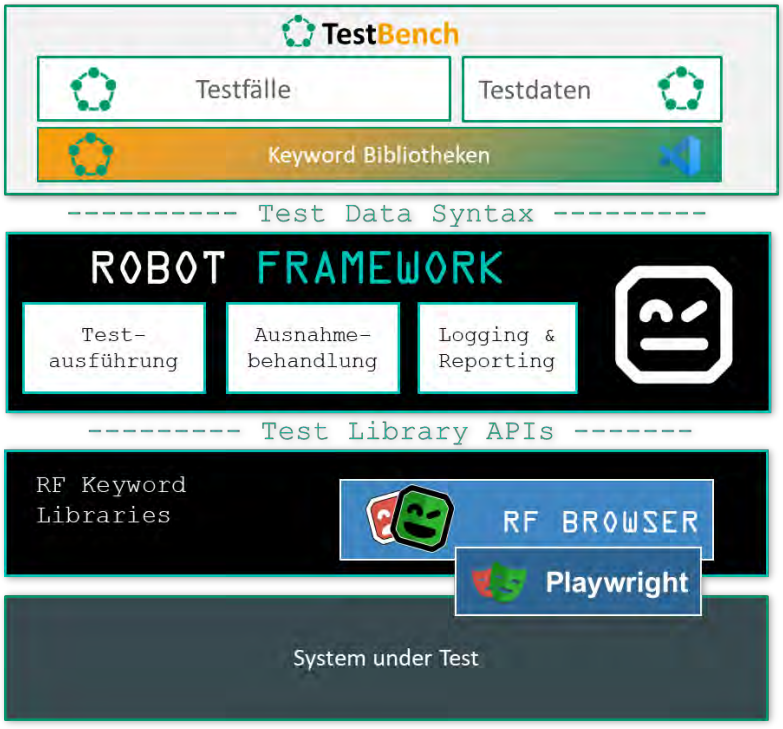


```
iTB-TC-337-PC-1659
    GROUP    CarConfig starten
        Open CarConfig
        Set Username    schulung20
        Set Password    @RBTFRMWRK@
        Click Login Btn
    END
    GROUP    Fahrzeug wählen
        Select Base Model    Rolo
    END
    GROUP    Sondermodell wählen
        Select Special Model    Luxus
    END
    GROUP    Zubehör wählen
        Select Accessory    Sportfelgen
    END
    GROUP    Zubehör wählen
        Select Accessory    Lederlenkrad
    END
    GROUP    Endpreis bitte prüfen
        Verify Total Price    16,059.99    €
    END
    GROUP    CarConfig beenden
        Close CarConfig
    END
```

Am obigen Ausschnitt sieht man, dass unter der fachlichen Ebene (erkennbar an den GROUP-Elementen) noch eine weitere Ebene in TestBench existiert, die sogenannte Navigationsebene. In unserem Fall bildet sie die fachliche Ebene auf die Bediensequenz im System under Test ab. Diese Navigationsebene stellt den Übergang zum Robot Framework dar. Sie könnte auch im Robot Framework abgebildet werden, dann jedoch müssen die Testautomatisierer:innen grundlegende Kenntnisse über das System under Test haben. Auf welcher Ebene der Übergang stattfindet, kann von Fall zu Fall festgelegt werden, die Vorgehensweise aus dem Beispiel hat sich jedoch vielfach bewährt.

Zusätzlich werden beim Generieren der Tests die Testdaten als Übergabeparameter der einzelnen Keywords als Konstanten eingesetzt. In Robot Framework muss somit keine Verwaltung für Testdaten aufgebaut werden, da diese ebenfalls aus TestBench übernommen werden. Auch bei den Testdaten müssen Testautomatisierer:innen keine fachlichen Kenntnisse besitzen, sondern nur Kenntnisse über den Typ der Daten und auf welche Art und Weise sie dem System under Test übergeben bzw. von dort ausgelesen werden müssen.

Die Kombination aus TestBench und Robot Framework bildet alle Ebenen der generischen Testautomationsarchitektur ab:



Damit ist das Ziel erreicht: Spezifikation = Testautomatisierung!

Vorteile einer mehrstufigen No-Code/Low-Code Lösung

Die Kombination aus No-Code- und Low-Code-Werkzeug zu einer integrierten Lösung bietet viele Vorteile:

- **Vereinfachtes Testdesign**: Fachanwender können eigenständig Testfälle erstellen, ohne tiefgehende technische Kenntnisse.
- **Zentralisierte Keyword-Bibliothek**: Fachliche und technische Keywords werden in zentralen, übersichtlichen Keyword-Bibliotheken verwaltet. Jede Ebene in dem für sie optimalen Werkzeug.
- **Verbesserte Zusammenarbeit**: Klare Trennung und dennoch enge Verzahnung zwischen technischer Implementierung und fachlicher Testgestaltung.
- **Effizienzsteigerung und Wartbarkeit**: Leichtere Anpassungen durch zentrale Verwaltung der Keywords und einfache Pflege der Testfälle.
- **Paralleles Arbeiten**: Fachliche Tests können gleichzeitig zur technischen Realisierung entstehen und werden in den jeweiligen Werkzeugen zusammengeführt.
- **Austauschbarkeit der Ebenen**: Die Ebenen der Testautomationsarchitektur können jeweils für sich ausgetauscht, modernisiert oder erweitert werden. Das ist vor allem für den Test Adaption Layer relevant, da der technologische Fortschritt am sprunghaftesten und schnellsten voranschreitet.

Mit der Lösung kann den beiden zu Beginn benannten Herausforderungen optimal begegnet werden:

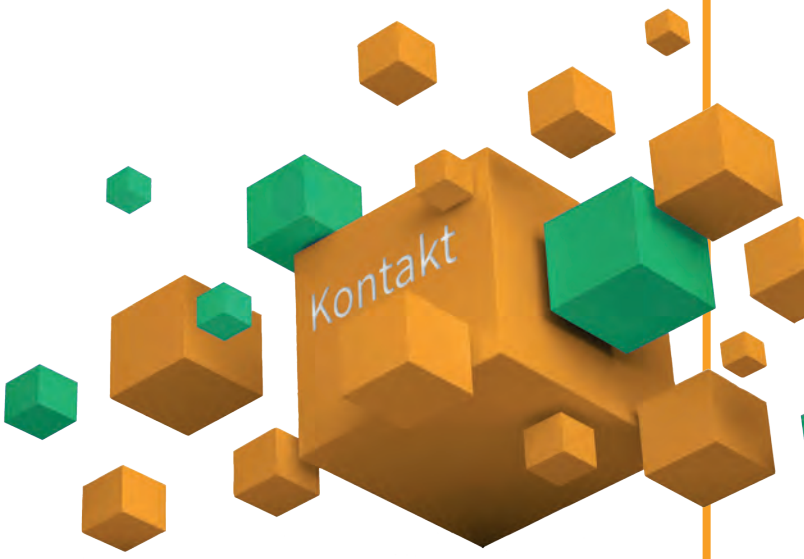
1. Personen mit Fachwissen und Personen mit technischem Wissen können reibungslos zusammenarbeiten und gemeinsam sehr schnell und effizient eine Testautomatisierung erstellen.
2. Paralleles Arbeiten der Experten ermöglicht schnellen Fortschritt, so dass die Realisierung automatisierter Tests mit der Entwicklung der zu testenden Funktionalitäten Schritt halten kann.

Im Vergleich zu hoch spezialisierten No-Code-Testautomatisierungs-Werkzeugen ist die vorgestellte Lösung wesentlich flexibler, kann einfacher auf allen Ebenen modifiziert werden und nutzt eine Art der Darstellung und Dokumentation, die sowohl für geringe Anzahlen von Tests, aber auch für große Testsuiten sehr gut eingesetzt werden kann.

Kontakt

imbus AG

Kleinseebacher Str. 9
91096 Möhrendorf
DEUTSCHLAND



Tel. +49 9131 7518-0

info@testbench.com

www.testbench.com