ESP IJAST
ESP International Journal of Advancements in Science & Technology
ISSN: 2583-9233 / Volume 4 Issue 1 January 2026 / Page No: 1-6
Paper Id: IJAST-V4I1P101 / Doi: 10.56472/25839233/IJAST-V4I1P101

Original Article

Test Automation Care Package

Aravindan Subramaniyam
Techno-Functional Specialist in Eagle Product Suite, Philadelphia, USA.

Received Date: 20 November 2025 Revised Date: 12 December 2025 Accepted Date: 05 January 2026

Abstract*: To make the test automation process go faster, we needed to harness innovation. So how do we do this? So the first strategy was to figure out how we could leverage AI to speed up the test development cycle. The second strategy involved developing a test automation care package. So combined with AI, this would not only accelerate the development process but would also assist those with limited test automation knowledge to get started quickly. Designed to jump-start your test automation, a care package offers sample test scenarios, comprehensive comments, and guidance on customization, all enhanced by AI tools such as Microsoft/GitHub Copilot. It's a technical package or bundle that was put together to help kick-start your test automation journey. They are designed around specific use cases. The paper also dwells on some of the changing tactics, which are being incorporated to reach young engineers particularly by employing the use of technology. Finally, the study can aid in the need to make prudent testing choices, and the necessity to have better literacy levels in making different use case choices.*

Keywords*: IT Quality Assurance, Software Testing, Test Automation, Python BDD, AI Code Generation, Python Automation.*

I. INTRODUCTION

Test Automation is a framework that helps to run test cases consistently and repeatedly on different versions of the same system. It helps with optimization of speed, efficiency, quality and the decrease of costs. Automated tests can run fast and frequently, enabling higher test coverage.

Automated tests can run without human intervention (overnight executions). The automation of tests is initially associated with increased effort, but the related benefits will quickly pay off. It also serves as documentation - automated tests document code in such a way that anybody can check that it continues to behave in same way after some changes are made in the code.

A. Objectives:

- Improve test efficiency, by reducing number of manual tests (can not eliminate all manual tests)
- Enhance software quality and reliability
- Provide wider test coverage
- Shorten test period and cost reduction
- Increase test frequency

B. When and What to Automate - Test Automation Opportunities:

- Functionality that introduces high risk conditions - business critical test cases
- Unit tests, component tests, and integration tests
- Recommended code coverage for unit tests around 75% - 80%
- Automate functions that are 80% stable and unchanging
- Repetitive tests that run for multiple builds (regression, smoke testing)
- Tests that tend to cause human error
- Test cases that are tedious and take a lot of effort and time to perform manually
- Tests that require multiple data sets (data driven testing)
- Frequently used functionality that introduces high-risk conditions
- Tests that run on several different hardware or software platforms and configurations
- Tests that take a lot of effort and time when manual testing
- Load and performance testing, cross-browser & cross platform/device testing

II. BENEFITS AND CHALLENGES OF TEST AUTOMATION:

A. Benefits of Test Automation:

Implementing test automation is vital for maintaining software quality. Test automation is crucial for its quality and it helps to reduce testing efforts, enabling faster and more affordable delivery of capabilities.

B. Challenges in Test Automation:

- Not all manual test cases can be automated
- The automation can only check machine interpret-able results
- Automation tests can only check actual results that can be verified by tester Time Consuming, particularly adding test automation to the projects.
- Lack of Knowledge and Skill to adapt the test automation.

III. SOLUTION

When I was looking at this challenge,realized that in order to make this process go faster, we needed to harness innovation. So how do we do this? So the first strategy was to figure out how do we leverage Al to speed up the test development cycle.

So the second strategy involved developing a test automation care package. So combined with Al, this would not only accelerate the development process but would also assist those with limited test automation knowledge to get started quickly.

Designed to jump-start your test automation, a care package offers sample test scenarios, comprehensive comments, and guidance on customization, all enhanced by AI tools such as Microsoft/GitHub Copilot. It's a technical package or bundle that was put together to help kick-start your test automation journey. They are designed around specific use cases.

A. What is Included in the Care Package?

- Fully functional parameterized code package
- Reusable test functions
- Helpful AI prompts
- Supportive documentation that provides step by step instructions

B. Available Care Packages:

a) *UI Validation:*

- Validate UI page components and flow UI to DB Data Validation
- Validate value in UI against the source DB field

b) *UI to API Data Validation:*

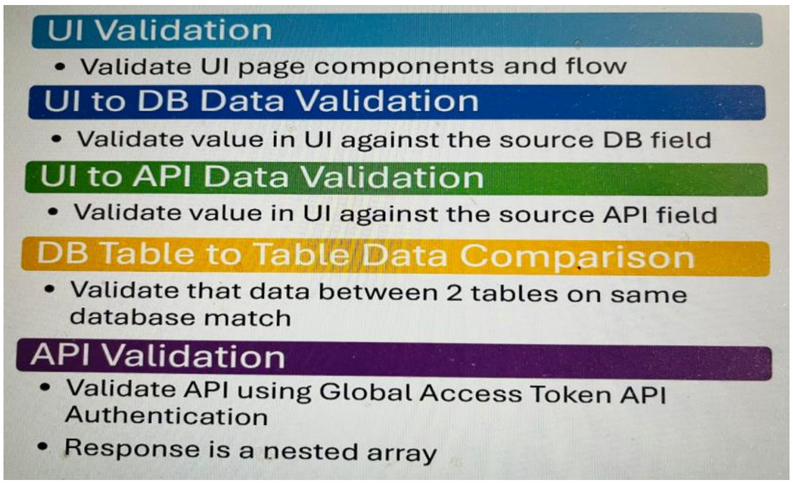
- Validate value in UI against the source API field

c) *DB Table to Table Data Comparison:*

- Validate that data between 2 tables on same database match

d) *API Validation:*

- Validate API using Global Access Token API Authentication
- Response is a nested array



UI Validation
- Validate UI page components and flow

UI to DB Data Validation
- Validate value in UI against the source DB field

UI to API Data Validation
- Validate value in UI against the source API field

DB Table to Table Data Comparison
- Validate that data between 2 tables on same database match

API Validation
- Validate API using Global Access Token API Authentication
- Response is a nested array

Figure 1 : Available Care Packages

C. Why Python?

- Python is a high-level object-oriented and user-friendly scripting language. Simple syntax - easy to learn.
- Rich libraries are available to support any test type and can be used for any purpose (UI, API, Lambda, Integration etc)
- Large community for support

D. Introduction to BDD:

BDD stands for Behavior-Driven Development, a software development methodology that focuses on the behavior of an application from the user's perspective. To implement code using BDD we use the Python library "pytest-bdd". To get started ensure you have this installed. You can install it using pip: pip install pytest-bdd

a) Feature File:

A feature file contains test scenarios written in English language (GWT) with an extension of "feature".
GWT = Given, When, Then

b) Step Definition:

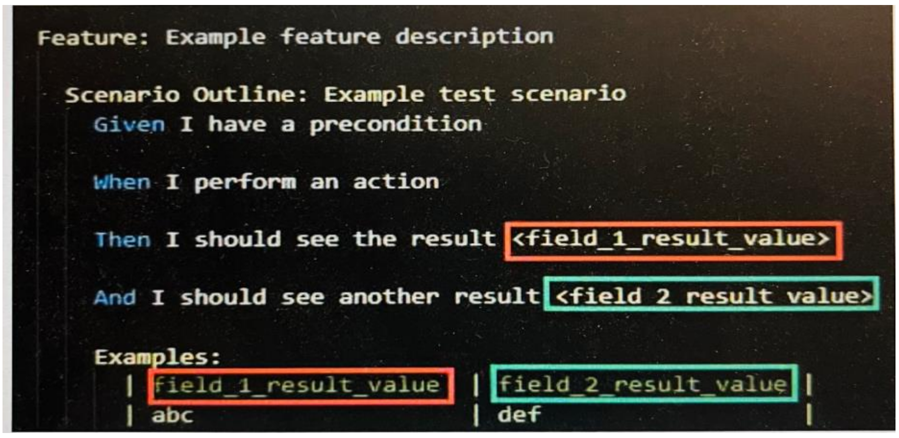
A step definition is a method that's associated with one or more GWT steps. This file contains the glue code that is associated with the steps in the feature file.

E. What is Included in the Care Packages:

a) Fuature File:

Creating a feature file in pytest-bdd involves writing scenarios in GWT syntax, which describes the behavior of your application in plain language.

To get started, you will find a feature file with_feature extension, under the /features folder in the care package. Follow the sample template feature file in the care package and modify it according to your test scenario.



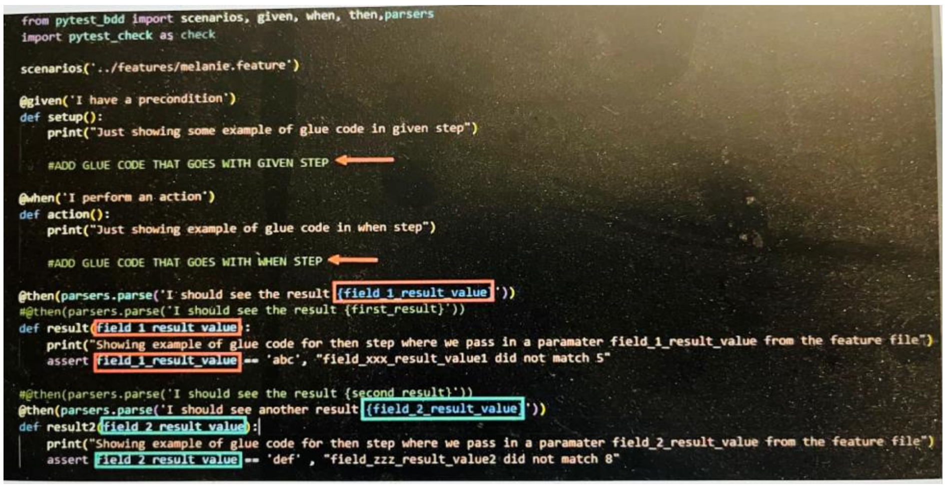
```
Feature: Example feature description

  Scenario Outline: Example test scenario
    Given I have a precondition

    When I perform an action

    Then I should see the result <field_1_result_value>

    And I should see another result <field_2_result_value>

    Examples:
      | field_1_result_value | field_2_result_value |
      | abc                  | def                  |
```

Figure 2 : Sample Feature File

b) Step Definition File:

Step definition file contains ALL the glue code that will run the steps in the feature file. In your step definition file, you write the code that performs the action described by the Gherkin step.

In each package you will find a template step definition file. Some packages have full glue code or you can create your own step definition file under /step_defs folder in the care package.



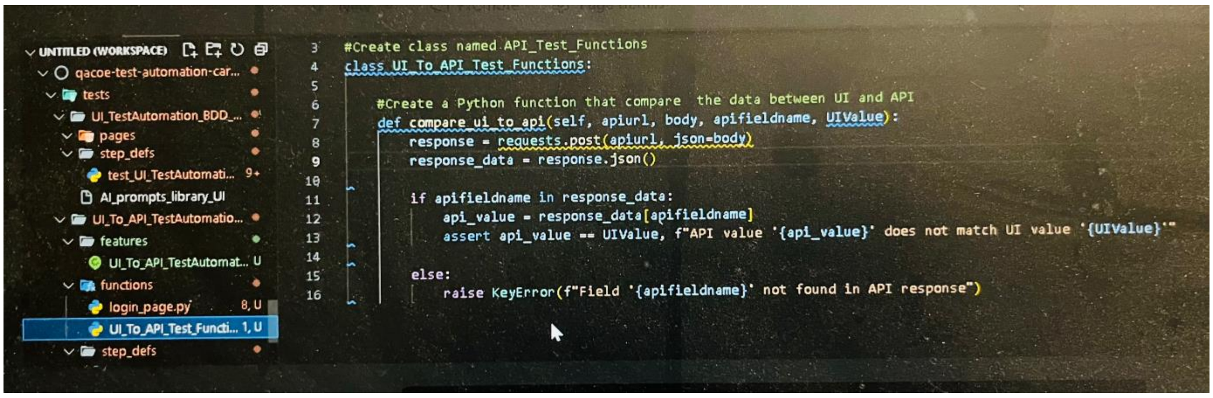
```
from pytest_bdd import scenarios, given, when, then,parsers
import pytest_check as check

scenarios('../features/melanie.feature')

@given('I have a precondition')
def setup():
    print("Just showing some example of glue code in given step")

    #ADD GLUE CODE THAT GOES WITH GIVEN STEP

@when('I perform an action')
def action():
    print("Just showing example of glue code in when step")

    #ADD GLUE CODE THAT GOES WITH WHEN STEP

@then(parsers.parse('I should see the result {field_1_result_value}'))
#@then(parsers.parse('I should see the result {first_result}'))
def result(field_1_result_value):
    print("Showing example of glue code for then step where we pass in a paramater field_1_result_value from the feature file")
    assert field_1_result_value == 'abc', "field_xxx_result_value1 did not match 5"

#@then(parsers.parse('I should see the result {second_result}'))
@then(parsers.parse('I should see another result {field_2_result_value}'))
def result2(field_2_result_value):
    print("Showing example of glue code for then step where we pass in a paramater field_2_result_value from the feature file")
    assert field_2_result_value == 'def' , "field_zzz_result_value2 did not match 8"
```

Figure 3 : Sample Step Definition File

c) Test Functions File:

To enhance test maintenance and reduce code duplication, it is a best practice to create test functions in a separate class (py) file instead of writing the complete test-script for a step within the step-definition file itself. These test functions can be accessed by the step definition files by creating an object of this class file. test_functions.py files should be created under the functions folder.

Similarly, if you are working on UI test automation following the page-object model is a best practice. Each web page in the application should be represented as a class file, and the elements on that page has to be defined as variables within the class. These class files should be places under pages folder(./tests/pages/)


UNTITLED (WORKSPACE)
qacoe-test-automation-car...
tests
UI_TestAutomation_BDD_...
pages
step_defs
test_UI_TestAutomati... 9+
AI_prompts_library_UI
UI_To_API_TestAutomatio...
features
UI_To_API_TestAutomat... U
functions
login_page.py 8, U
UI_To_API_Test_Functi... 1, U
step_defs

```
#Create class named API_Test_Functions
class UI_To_API_Test_Functions:

    #Create a Python function that compare  the data between UI and API
    def compare_ui_to_api(self, apiurl, body, apifieldname, UIValue):
        response = requests.post(apiurl, json=body)
        response_data = response.json()

        if apifieldname in response_data:
            api_value = response_data[apifieldname]
            assert api_value == UIValue, f"API value '{api_value}' does not match UI value '{UIValue}'"

        else:
            raise KeyError(f"Field '{apifieldname}' not found in API response")
```

Figure 4 : Sample Test Functions File

F. What Additional Files are Included in this Care Package Repository that are Required for Your to Run?

a) Global Configuration File:

You can find this file at:

"tests/conftest.py

This file contains python fixtures and functions to initialize browser with preferred host, environment and URL etc., t you global access from your step definition script files. This is helpful because you then do not have to include this every script you create. You have it one place and then they are made available.

b) Python Library Install File:

You can find this file at: requirements. txt (root folder level)

This file contains libraries that are needed to be installed to utilize this care package. Some are common across all care packages, and some are care packages specific.

c) Utility Functions File:

You can find this file at: tests/utilityFunctions.py

This file contains a few functions that could be used across all of your scripts. Includes functions needed to pull credentials from CyberArk. It also contains functions that interact with excel spreadsheets (to create, update and compare).

G. Managing the Credentials in Test Script:

As a quick start you can create a "local.env" file and store the credentials that you use in scripts locally. This file is stored locally only.

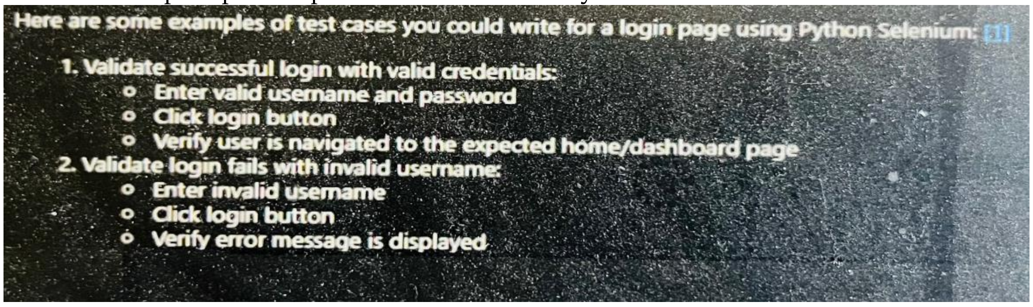
Add this file to your "gitignore" file to ensure it is never checked into your repository. Our care package code demonstrates using the "local.env" to pull credentials into the script but again this is just a quick start option. In the long term, you should aim to integrate code into your script that retrieves credentials from CyberArk.

H. AI Assisted BDD:

Al Code Assistant tools can expand upon and greatly aid in creating new test cases using Behavior Driven Development (BDD) written in Given When Then (GWT) format. With simple or detailed prompts, the tool can generate test cases, convert them to BDD format and also create test script code. This capability can help team members without technical expertise to contribute to test case creation. This can be done using MS Copilot or GitHub Copilot chat.

Below is an example where AI can greatly assist and expedite your current processes with all team members and product stakeholders - (Example is from GitHub Copilot Chat but MS Copilot can also be used):

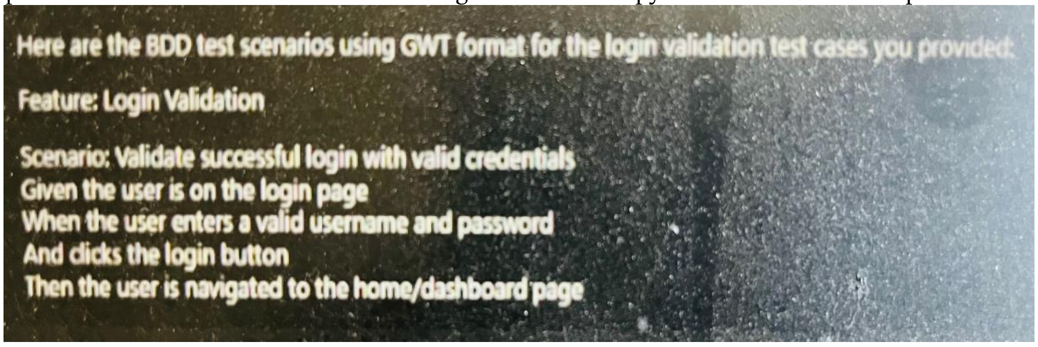
a) Can Initially use Al to Create or Expand on a List of Tests Cases to Validate an Example Piece of Code.
Example Prompt: List the test cases to validate < example functionality>.
Can also generate additional prompts to explain selected functionality


Here are some examples of test cases you could write for a login page using Python Selenium: [1]

1. Validate successful login with valid credentials:
   - Enter valid username and password
   - Click login button
   - Verify user is navigated to the expected home/dashboard page
2. Validate login fails with invalid username:
   - Enter invalid username
   - Click login button
   - Verify error message is displayed

Figure 5 : Sample Test Cases 1

b) Can Then Select Those Required Test Cases and Transform to BDD GWT Format.
Eample Prompt: Create BDD GWT based on the following Test Cases <Copy Selected tests from Step 1>

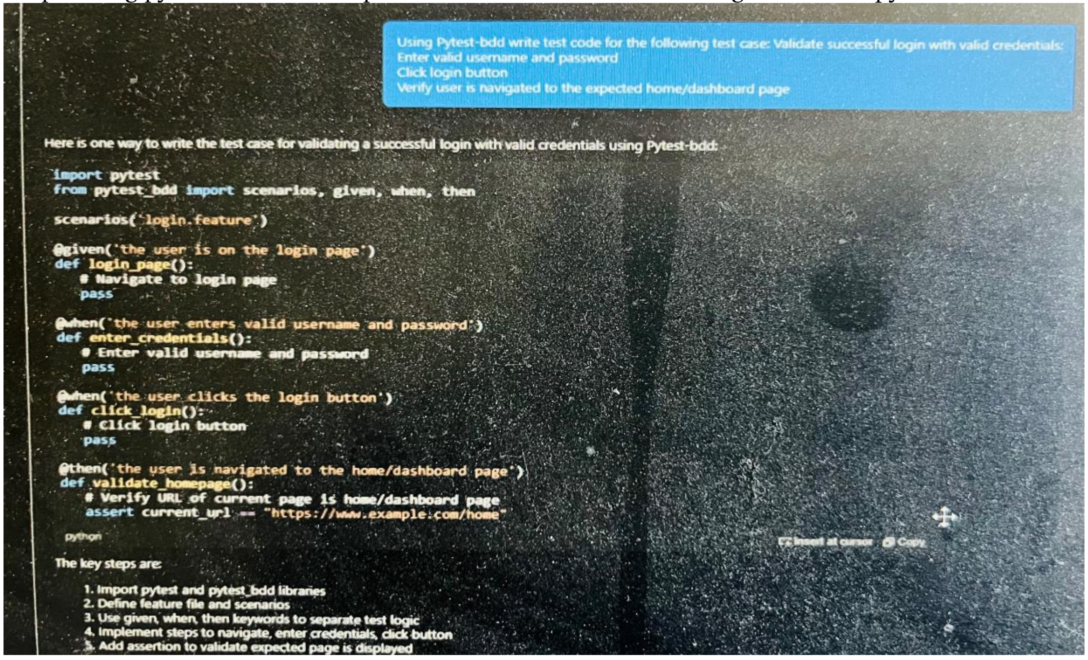

Here are the BDD test scenarios using GWT format for the login validation test cases you provided:

Feature: Login Validation

Scenario: Validate successful login with valid credentials
Given the user is on the login page
When the user enters a valid username and password
And clicks the login button
Then the user is navigated to the home/dashboard page

Figure 6 : Sample Test Cases 2

c) Can Then use that BDD Example and Ask it to Use Pytest-Bdd to Create Test Script Code (Glue Code)
Example Prompt: Using pytest-bdd create step definition file to validate following GWT's: <copy GWT statements ›


Using Pytest-bdd write test code for the following test case: Validate successful login with valid credentials:
Enter valid username and password
Click login button
Verify user is navigated to the expected home/dashboard page

Here is one way to write the test case for validating a successful login with valid credentials using Pytest-bdd:

```python
import pytest
from pytest_bdd import scenarios, given, when, then

scenarios('login.feature')

@given('the user is on the login page')
def login_page():
    # Navigate to login page
    pass

@when('the user enters valid username and password')
def enter_credentials():
    # Enter valid username and password
    pass

@when('the user clicks the login button')
def click_login():
    # Click login button
    pass

@then('the user is navigated to the home/dashboard page')
def validate_homepage():
    # Verify URL of current page is home/dashboard page
    assert current_url == "https://www.example.com/home"
```

The key steps are:

1. Import pytest and pytest_bdd libraries
2. Define feature file and scenarios
3. Use given, when, then keywords to separate test logic
4. Implement steps to navigate, enter credentials, click button
5. Add assertion to validate expected page is displayed

Figure 7 : Sample Test Cases 3

IV. CONCLUSION

This paper concludes by giving a decent overview of all the possibilities to do the Test Automation by Care Packages. Based on this proven analysis, the Test Automation Care Package can be implemented on the different use cases. the study can aid in the need to make prudent testing choices, and the necessity to have better literacy levels in making different use case choices. The conclusions made on the basis of this analysis would stress the significance of making informed decisions, diversification and clarity about one Use Cases.

In conclusion, this article reiterates that Test Automation Care Package, when supplemented with quality knowledge, technological means and testing principles, can become a revolutionary force when it comes to building automation system as a whole. The results are supposed to prompt more people to be proactive when it comes to investing and stimulate further development of innovation in the field of Quality Control and Assurance.

A. Acknowledgement:

I acknowledge the use of Copilot to prepare this assessment which is mentioned on the AI assisted BDD section.

B. Conflicts of Interest

The authors have no conflicts of interest to declare.

V. REFERENCES

[1] Jim Hazen - Before the Code:First Steps to Automation in Testing - 2020

[2] Saeed Parsa - Software Testing automation - 2023

[3] Arnon Axelord - Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects - 2018

[4] Python Automation Testing with Pytest - https://www.udemy.com/course/python-automation-pytest/ - Instructor: Kumar S. Accessed Sep 2nd 2025

[5] Learn API Automation Testing with Python & BDD Framework https://www.udemy.com/course/python-automation-pytest/ - Instructor: Kumar S. Accessed Oct 26th 2025

[6] Ashwin Pajankar - Python Unit Test Automation - 2022