

# Automation Framework - Introduction

## Automation Framework

- It provides a **systematic approach** to writing, executing, and maintaining test scripts, promoting reusability, scalability, and efficiency.
- We organize automation project files and folders in structured manner.

## Objectives of an Automation Framework

- **Reusability:** Enable reuse of test scripts and components across different test cases.
- **Scalability:** Facilitate the addition of new test cases without extensive rework.
- **Maintainability:** Simplify updates and maintenance of scripts due to application changes.
- **Efficiency:** Minimize manual effort by streamlining test execution and reducing execution time.
- **Reporting:** Provide comprehensive test reports for tracking test outcomes and failures.

## Types of Automation Frameworks

- **Built-in Frameworks:**
  - Ready-to-use frameworks.
  - Example: TestNG, Junit, Cucumber etc.
- **Customized Frameworks:**
  - Customized frameworks meet specific project or organization requirements by integrating tools and libraries.
  - Examples: Hybrid-driven framework combining POM, keyword-driven, and data-driven approaches.

## Phases in Developing an Automation Framework

1. **Requirement Analysis:** Identify testing needs, tools, and project requirements.
2. **Tool Selection:** Choose appropriate tools and libraries based on application type and project needs.
3. **Framework Design:** Decide on the structure, architecture, and approach (e.g., modular, hybrid, etc.).

4. **Implementation:** Develop reusable components, utilities, and test scripts.
5. **Integration:** Configure CI/CD pipelines for automated execution and reporting.
6. **Testing:** Validate the framework by executing sample test cases.
7. **Maintenance:** Continuously improve the framework to accommodate application changes.

## **How to Select an Automation Testing Tool**

### **1. Know Your Requirements**

- Identify the application type (static, dynamic, responsive).
- Understand the technology stack (e.g., HTML, Angular, React).
- Ensure the tool supports all target browsers and platforms.

### **2. Check Tool Features**

- **Cross-browser compatibility** (e.g., Chrome, Edge, Firefox, Safari).
- **Ease of integration** with CI/CD tools and frameworks.
- **Dynamic content handling** (e.g., pop-ups, AJAX, animations).

### **3. Evaluate Usability**

- Simple setup and easy for the team to learn.
- Availability of tutorials, documentation, and community support.

### **4. Consider Costs**

- Compare free tools (e.g., Selenium) with paid ones (e.g., UFT).
- Account for hidden costs like training and maintenance.

### **5. Test the Tool**

- Run a small trial (POC) to check compatibility with your application.
- POC stands for Proof of Concept, which is a process used to determine if a project is feasible.

### **6. Future-Readiness**

- Ensure regular updates to support new technologies.

### **7. Make a Decision**

- Compare shortlisted tools.

- Choose one that fits your budget, team skills, and application needs.

### **How to Choose Test Cases for Automation**

- **High Priority Tests:** Test cases for critical functionalities of the application.
- **Repetitive Tests:** Regression and smoke tests that are run frequently.
- **Data-Driven Tests:** Scenarios requiring multiple input combinations.
- **Time-Consuming Tests:** Tests that take significant time to execute manually.
- **Stable Tests:** Scenarios with a stable application feature.

### **Avoid automating:**

- Exploratory or ad-hoc tests.
- Tests for features still under development.
- Low-priority or rarely used functionalities.

### **Is 100% Automation Possible? Why?**

**No**, 100% automation is not practically possible.