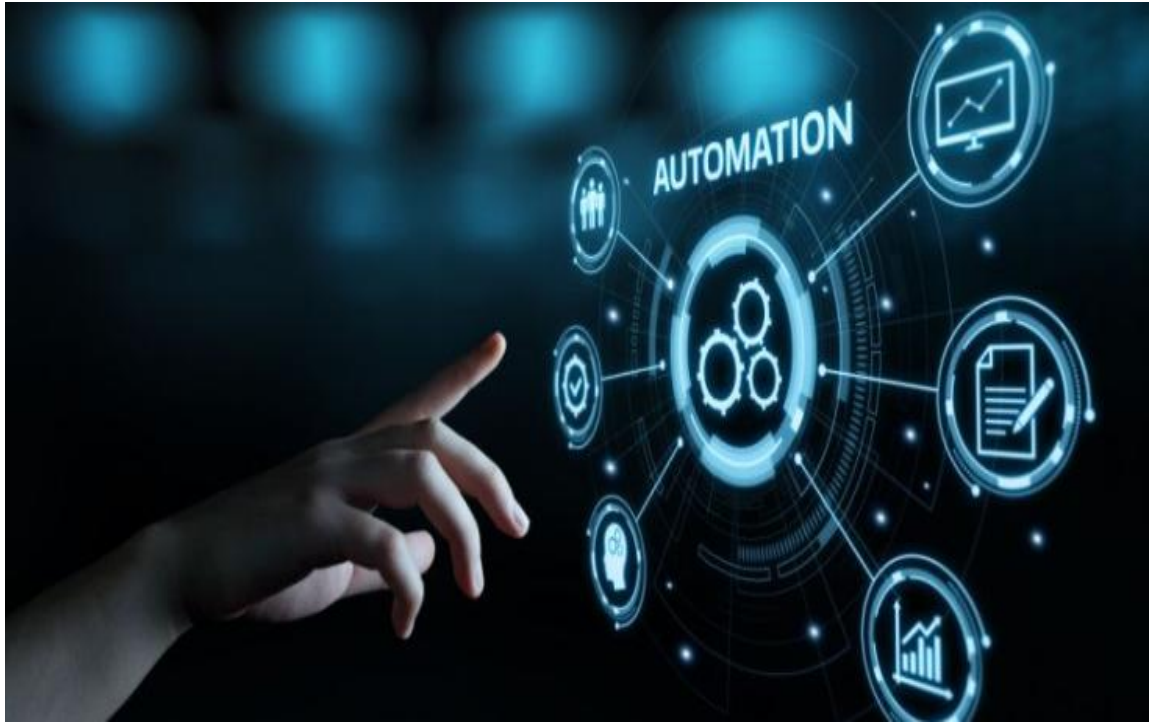
19th April 2025

Mr. Suresh Fernando

Test Automation - Overview

- Introduction to Test Automation
- Manual Vs Automated Testing
- Principles of Test Automation
- Test Automation Lifecycle
- Test Automation Frameworks
- Challenges & Myths in Test Automation
- Assignment 2

What is Automation?



Making an apparatus, a process, or a system operate automatically.

What is Test Automaton?

Test automation is the use of software tools to automatically run and validate test cases, enhancing testing efficiency...

What/When to Automate?

Regression Testing
Run **frequently** to ensure that new code hasn't broken existing functionality (e-com app).

Smoke Testing / Sanity Checks
Quick checks to see if the basic functionalities work before deeper testing

High Volume / Repetitive Tests
Saves time and reduces human error (load testing)

Data-Driven Testing
Same test logic with multiple input data sets (500 combinations of inputs like names, emails, and contact number.)

Cross-Browser / Cross-Device Testing
Tools can quickly test web apps on different browsers/device

Stable Features
Automate tests for features that don't change often (Login/Logout)

What/When **Not** to Automate?

Exploratory Testing
Requires human intuition, creativity, and observation (to find visual bugs)

Short-Lived Features
Not worth it for features that will be removed soon (a holiday sale)

Unstable or Frequently Changing UI
Tests will break often, causing more maintenance than value (Promotional page)

Usability Testing
Needs human feedback on look, feel, and ease of use (New color scheme)

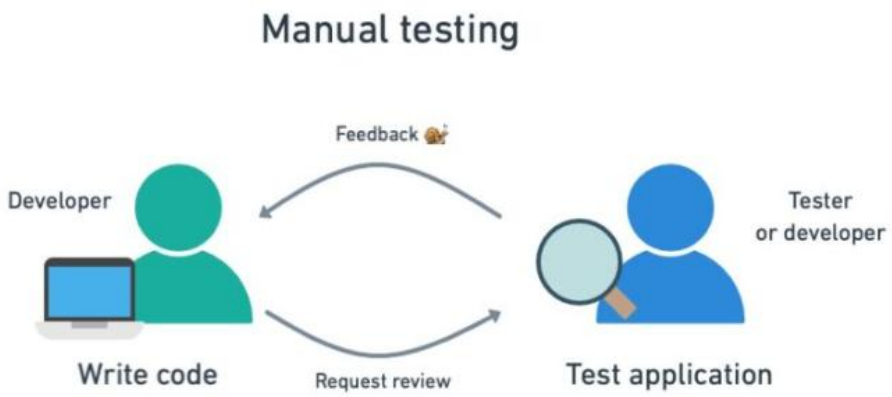
Tests That Run Only Once
The effort to automate it outweighs the benefit (One-time data migration)

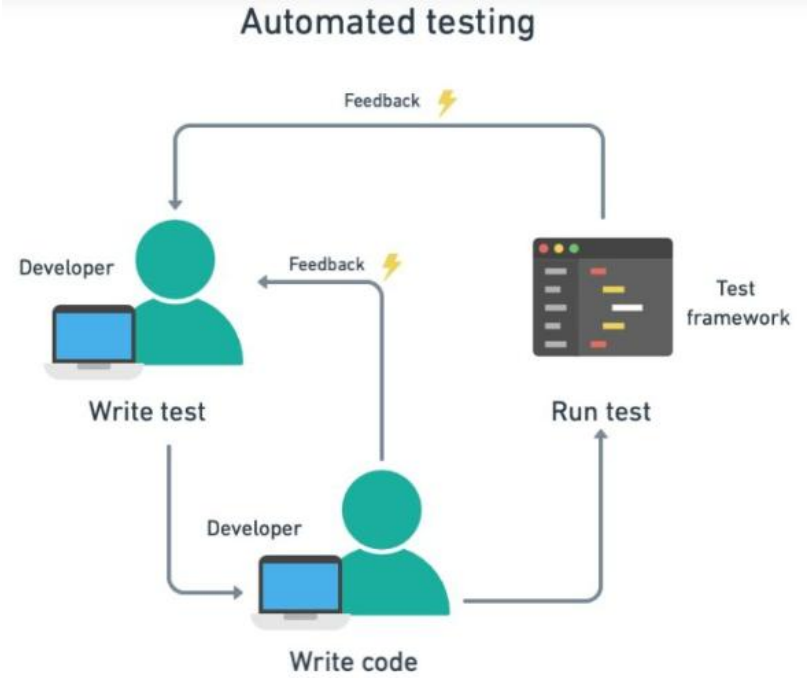
Complex Logics Involved
Some things are just too delicate to automate easily (Image comparison, CAPTCHA)

Benefits of Test Automation

- Application-wise
  - Improved Quality
  - Improved Accuracy
  - Enhanced Test Coverage
  - Early Bug Detection
  - Reusable Test Scripts
- Cost-wise

Manual vs Automated Testing





Manual vs Automated Testing

Key Differences	Manual Testing	Automated Testing
Execution Time	Requires more time as tests are performed sequentially, increasing resource usage.	Faster execution due to automated tools, leading to quicker product delivery with less resource consumption.
Inital setup	Less effort is needed in the initial setup as tests are executed manually.	Initially, more effort is required to create and maintain scripts and tools.
Reliability	Less reliable due to potential human error, impacting precision.	Higher reliability as tests are consistently executed the same way each time.
Programming	Non-programmable, limiting the creation of complex tests for defect detection.	Allows for programming complex tests to uncover hidden defects.

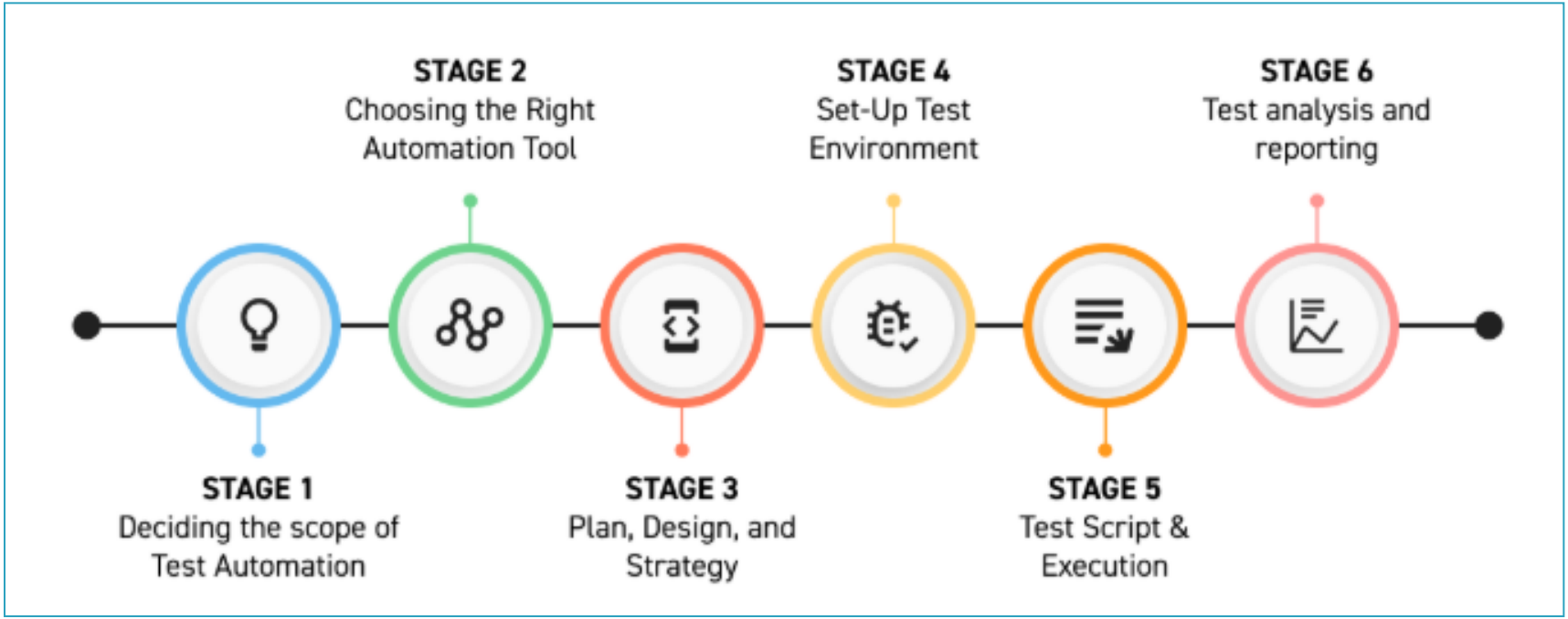
Manual vs Automated Testing

Key Differences	Manual Testing	Automated Testing
Reusability	Often requires new test cases for each function.	Test scripts can be reused across different software cycles, enhancing efficiency.
Reporting	Reporting can vary as it is manually implemented.	Ensures standardized, consistent tracking and reporting.
Flexibility	More adaptable to changing requirements and testing scenarios.	Less flexible to unexpected changes due to its pre-programmed nature.

Principles of Test Automation

- Tests should improve quality.
- Tests should reduce the risk of introducing failures.
- Testing helps to understand the code.
- Tests must be easy to write.
- A test suite must be easy to run.
- A test suite should need minimal maintenance.

Test Automation Lifecycle



Scope of Test Automation

- The first step in automation is figuring out what to automate
- Decide *what* and *why* to automate
- Find which can be suitable for automation based on:
  - how complex they may be
  - how often they get run
  - and how vital they may be.
- Focus on automating tests for important functions or matters that humans are not good at.
- Work with designers, developers, QA engineers, and others to decide what to automate.

Choosing Right Automation Tools

- Choose the right automation tool(s) for your tech stack and team skills
- Must do a comparative study of automation tools before making a decision.
- Factors to consider
  - Technology Support: Web, Desktop, Mobile
  - Scripting or Programming Requirements: Code, No-Code, Low-Code
  - Open-source vs. Licensed
  - Ease of Use
  - Adoption Time
  - Customer Support
  - End-to-End Testing

Automation Tools

Automation Tools

Tool	Scripting Languages Supported	Platforms Supported	Pricing
Selenium	Java, C#, Python, Ruby, JavaScript, Kotlin	Web (Desktop/Mobile via Appium)	Free (Open Source)
Appium	Java, Python, Ruby, JavaScript, Kotlin	Mobile (iOS, Android), Web (Mobile)	Free (Open Source)
Playwright	JavaScript, TypeScript, Python, C#, Java	Web (Desktop, Mobile)	Free (Open Source)
Cypress	JavaScript, TypeScript	Web (mostly Chrome-based browsers)	Free, Paid for dashboard
TestComplete	JavaScript, Python, VBScript, DelphiScript	Web, Desktop, Mobile	Paid
Ranorex	C#, VB.NET	Web, Desktop, Mobile	Paid
Katalon Studio	Groovy (based on Java), supports low-code	Web, Mobile, API, Desktop	Free basic, Paid plans
Robot Framework	Python, also supports Java and others via libs	Web, Mobile (via Appium), APIs	Free (Open Source)
TestCafe	JavaScript, TypeScript	Web (Desktop)	Free (Open Source)
QTP/UFT One	VBScript	Web, Desktop, Mobile	Paid

Automation Tools

Tool	Web (Desktop)	Web (Mobile)	Mobile Apps	Desktop Apps
Selenium	✔	⚠ (via Appium)	⚠ (via Appium)	✘
Appium	✘	✔	✔	✘
Playwright	✔	✔ (emulated)	✘	✘
Cypress	✔	✔ (emulated)	✘	✘
TestComplete	✔	✔	✔	✔
Ranorex	✔	✔	✔	✔
Katalon Studio	✔	✔	✔	✔
Robot Framework	✔	✔ (via libs)	✔ (via Appium)	✘
TestCafe	✔	✔ (limited)	✘	✘
UFT One	✔	✔	✔	✔

Automation Tools

- **Best for Web Automation**: Playwright, Selenium, Cypress
- **Best for Mobile Testing**: Appium, Katalon Studio
- **Beginner Friendly**: Katalon, TestCafe, Cypress
- **Enterprise-Grade Tools**: TestComplete, UFT One, Ranorex
- **Open-Source Leaders**: Selenium, Appium, Robot Framework, Playwright

Automation Tools

Front-end

***TestSigma**: Easy-to-use tool for each tech and non-tech users.*

***Selenium**: Popular open-sourse tool for web browsers.*

***TestComplete**: A commercial tool supporting multiple platforms.*

Performance

***JMeter**: Open-source tool for trying out web apps with big person hundreds.*

***LoadRunner**: Commercial tool for web / mobile apps.*

***Gatling**: Open-source tool for web apps with real-time reporting.*

Database

***dbForge Studio**: Commercial tool for SQL databases.*

***SQLTest**: Commercial tool for SQL databases*

Database Benchmark: An open-source .NET tool designed to stress test databases with large data flows.

Plan, Design and Strategy

Activity	Description	Example
Define Automation Objectives	Clarify what you want to achieve with automation	Speed up regression testing; improve test coverage
Finalize Scope	Decide **what to automate** and what to leave out	Automate login, checkout flow; skip UI animations
Select Tools & Frameworks	Choose suitable tools for tech stack & platforms	Selenium + TestNG for web app; Appium for mobile
Assess Skills & Resources	Understand team capability, skill gaps, and training needs	QA team needs Python training; hire automation engineer
Develop Automation Strategy	High-level strategy: test levels, CI/CD, data management, reporting, test maintenance plan	Run tests nightly in CI, Use page object model for maintainability, Report via Allure in GitHub Actions
Estimate Timeline & Cost	Time and cost estimates for implementation	Initial setup = 2 weeks; monthly maintenance = 8 hrs
Define KPIs & Metrics	How success will be measured	Execution time, pass rate, failure rate, test coverage

Setup Test Environment

- **Application Under Test (AUT):** Deployed version of the app being tested
- **Test Tools/Frameworks:** Tools for writing/executing tests
- **Test Data:** Structured data needed for test scenarios
- **Browsers/Devices:** Configured devices, browsers, or emulators
- **Environment Configs:** Variables, credentials, API keys, database info
- **CI/CD Integration:** Connection with CI to run tests automatically
- **Monitoring & Logs:** Logs and dashboards for debugging test results

Test Script & Execution

- Write test scripts based on test cases using the chosen framework and tools.
- **Example:**

Function Under Testing

```python
# math_utils.py

def add(a, b):
    return a + b
```



```python
# test_math_utils.py

from math_utils import add

def test_add_positive_numbers():
    assert add(2, 3) == 5

def test_add_negative_numbers():
    assert add(-2, -3) == -5

def test_add_mixed_numbers():
    assert add(-2, 3) == 1

def test_add_zero():
    assert add(0, 0) == 0
```

Test Script

Test Script & Execution

- **Example:** A sample test script using Selenium + Python:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

def test_login_success():
    driver = webdriver.Chrome()
    driver.get("https://example.com/login")

    driver.find_element(By.ID, "username").send_keys("valid_user")
    driver.find_element(By.ID, "password").send_keys("secure_password")
    driver.find_element(By.ID, "loginBtn").click()

    # Assertion
    assert "Dashboard" in driver.page_source

    driver.quit()
```

Test Analysis & Reporting

- Final and insight-driven phase of the Test Automation Lifecycle, where all the hard work of automation pays off.
- To evaluate **test outcomes**, identify issues or trends, and communicate the **health of the application** and automation efforts.
- **Key Activities:**

Activity	Description
Collect Test Results	Gather execution data: passed, failed, skipped tests
Analyze Failures	Investigate root causes: app bug, flaky test, environment issue
Generate Reports	Use tools to generate human-readable reports (HTML, PDFs, Dashboards)
Visualize Metrics	Show trends in execution time, pass rate, test coverage
Share Insights	Communicate findings with QA, Dev, PMs via dashboards, emails, standups
Improve Test Quality	Refactor or stabilize flaky tests; add missing coverage

Test Analysis & Reporting

Useful Metrics to Track

Metric	Why It Matters
Pass/Fail Rate	Overall test stability
Test Flakiness Rate	Helps flag unstable or unreliable tests
Time to Execute	Indicates efficiency of the suite
Test Coverage	Ensures all critical paths are tested
Defects Detected Early	Measures value of automation in early QA

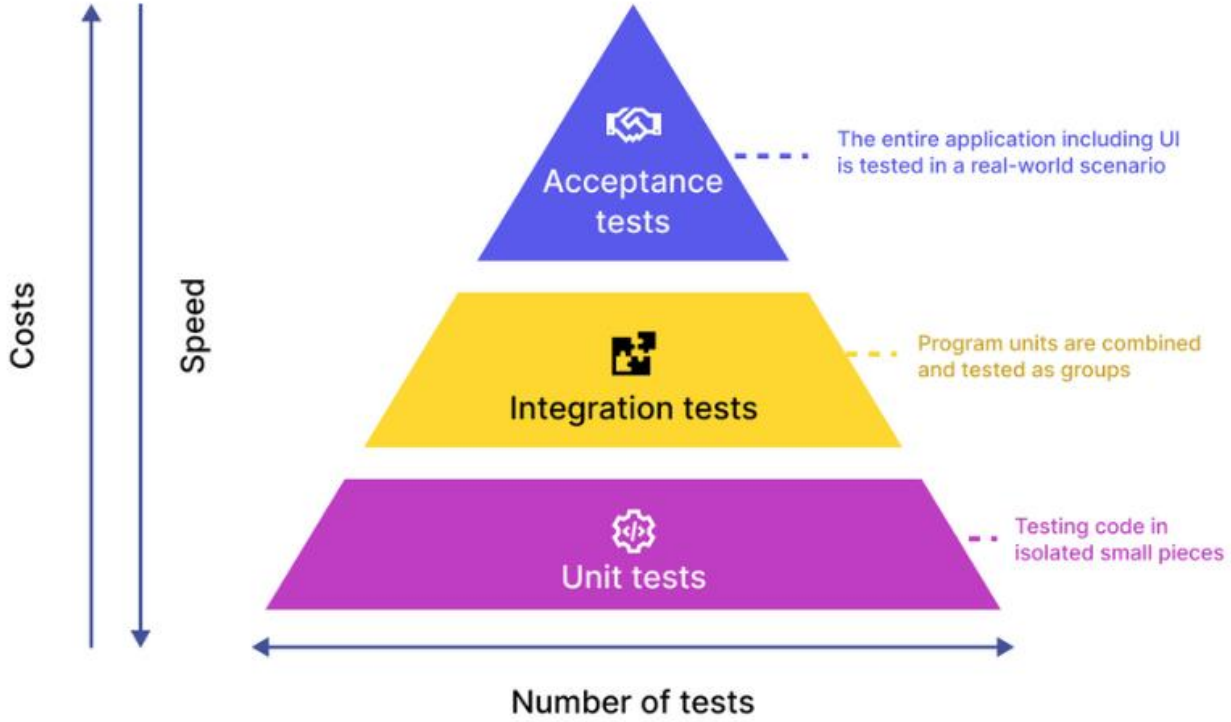
Example

Project: Online Banking App

Daily Report Shows:

- 98 tests run → 90 passed, 8 failed
- 5 flaky tests identified (same failures across runs)
- Avg execution time: 6.5 mins
- Coverage: 80% of core flows tested
- Shared with QA & Dev on Slack + CI dashboard

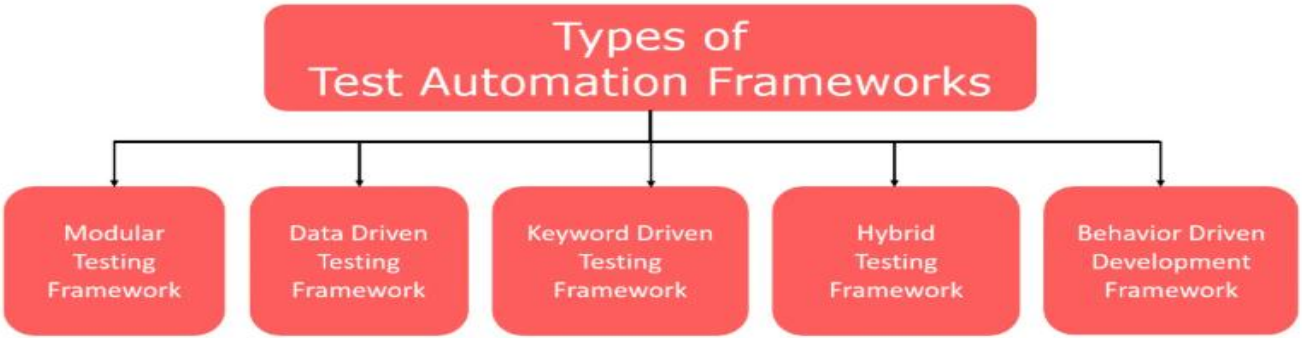
Test Automation Pyramid



Classification of Test Automation

Test Automation Frameworks

- A test automation framework is a set of **guidelines, tools,** and **practices** designed to help automate software testing in a structured and efficient way.
- It's the **foundation and toolbox** you use to write, organize, and run automated tests consistently across your project.



Key Features of Automation Frameworks

- **Reusability** – Common functions and utilities used across tests.
- **Maintainability** – Easier to update tests as the app changes.
- **Scalability** – Supports adding more tests without breaking.
- **Consistency** – Standardizes test writing, naming, and structure.
- **Reporting** – Provides logs, pass/fail summaries, screenshots, etc.
- **Integration** – Can connect with CI/CD tools, bug trackers, etc

Linear Scripting Framework

- Also known as the '**Record and Playback**' framework
- **Simple** and best for small projects or beginners
- Each test is written individually without much reuse
- **Advantages:**
  - Coding knowledge is not required
  - A quick way to generate test scripts
- **Disadvantages**
  - Hard to maintain as project grows
  - Lack of reusability
- **Example:** Selenium IDE – record browser actions and play them back.

Modular Testing Framework

- Breaks tests into independent, reusable modules.
- Each test case calls these modules as needed.
- Use page object model (POM) for maintainability
- **Advantages:**
  - Better scalability and easier to maintain
  - Can write test scripts independently
- **Disadvantages**
  - Requires more initial effort to develop scripts
  - Requires coding skills to set up the framework

Example:

```
# login_module.py
def login(username, password):
    # login steps here


# test_cart.py
from login_module import login
login('user1', 'pass123')
# continue with cart tests
```

Data-driven Framework

- Focused on separating the test scripts logic and the test data from each other.
- Test data set is kept in the external files or resources such as MS Excel Sheets, MS Access Tables, SQL Database, XML files, etc.,
- **Advantages:**
  - It supports multiple data sets
  - Modifying the test scripts won't affect the test data
- **Disadvantages**
  - Require coding skills
  - Setting up the framework and test data takes more time
- **Example:** Use an Excel or CSV file to test a form with 100 inputs.

Keyword-driven Framework

- **Advantages:**
  - No need to be an expert to write test scripts
  - It is possible to reuse the code. We can point the different scripts to the same keyword
- **Disadvantages:**
  - Take more time to design
  - The initial cost is high

Keyword-driven Framework

Example: Login to a website.

Step	Keyword	Object	Test Data
1	OpenBrowser	chrome	URL
2	EnterText	username_field	testuser
3	EnterText	password_field	password123
4	Click	login_button	
5	VerifyText	welcome_label	Welcome, User!
6	CloseBrowser		

Hybrid Framework

- Combines two or more approaches above (like data + keyword driven).
- Most real-world projects use hybrid frameworks.
- **Example:** Selenium with Python where test logic is modular, data is externalized in Excel, and actions are keyword-based.

Behavior-Driven Development (BDD) Framework

- BDD is a development approach that encourages collaboration between developers, testers, and business stakeholders.
- It focuses on writing tests in **natural language** that non-technical stakeholders can understand.
- Tests are written in **Gherkin syntax** (Given–When–Then format), which describe expected behavior of the system from a user perspective.

Behavior-Driven Development (BDD) Framework

Gherkin Syntax

Feature: *Login functionality*

Scenario: *Successful login with valid credentials*

Given *the user is on the login page*

When *the user enters valid username and password*

And *clicks the login button*

Then *the user should be redirected to the dashboard*

Automation Frameworks & Tools

Framework Type	Tools/Libraries
Linear	Selenium IDE, Katalon Recorder
Modular	Selenium + Python/Java
Data-Driven	TestNG + Excel, JUnit + CSV
Keyword-Driven	Robot Framework, Katalon Studio
Hybrid	Selenium + TestNG + Apache POI
BDD	Cucumber (Java), Behave (Python)

Integrating Test Automation with CI/CD

- What is CI/CD?
  - **Continuous Integration**: Automating the integration of code changes into the main codebase.
  - **Continuous Delivery**: Automating the deployment process to ensure new changes are automatically released.
- Automation in CI/CD:
  - Integrating automated tests into the CI/CD pipeline to run tests automatically on every code commit.
  - Benefits: Faster feedback loops, early bug detection, and seamless delivery process.

Challenges in Test Automation

High Initial Investment
Significant time and cost to choose tools, develop frameworks, and train staff

Test Maintenance Overhead
Changes in UI or application logic, requiring frequent updates.

Unstable Tests (Flaky Tests)
Sometime passes, sometime fails. Causes confusion and reduce trust

Partial Test Coverage
Not all test cases can be automated

Choosing the Right Tool
Matching with tech stack, team skill level, and project needs is difficult

Lack of Skilled Resources
Requires both testing knowledge and programming skills

What are Flaky Tests

- A flaky test is a test that sometimes passes and sometimes fails without any change to the code!
- The test result is not reliable even though the code and environment haven't changed.
- **Common Causes of Flaky Tests:**

Cause	Example
Timing issues / async waits	Test clicks a button before it's actually clickable
Element not yet loaded	Trying to read a field before it appears
Network/API delays	Test fails if an API call is slow but eventually works
Dependency on external systems	Test hits a 3rd-party API that is temporarily down

Myths in Test Automation

Myth #1: Test automation is expensive and requires a lot of resources.
Myth #2: Test automation totally eliminates the need for manual testing and replaces jobs.
Myth #3: Test automation is only suitable for large projects with ample resources.
Myth #4: It's better to use Selenium or another open-source tool to test.
Myth #5: You need to be a technical expert to utilize automation.
Myth #6: Test automation can be done absolutely by anyone.
Myth #7: Test automation is just a fad.
Myth #8: Test automation is a one-time activity.
Myth #9: Test automation is only for regression testing.
Myth #10: Test automation is inflexible and cannot adapt to changes in software requirements. It is hard to maintain.

Assignment 2

- Part A (4 marks) – Report on Software Test Automation
- Part B (4 marks) – Demonstration of Test Automation
- Part C (12 marks) – Individual Viva Presentation

Tutorial / Quiz

Thank You

Q & A

suresh.n@sliit.lk | 755841849