1. **Definition:-** Functional Testing is a type of software testing that verifies each function of the software application operates according to requirements.

   Focus: - What the system does (not how it does it).

   Example: - Calculator

   - **What it does:** Adds, subtracts, multiplies, divides numbers.
   - **How it does it:** Internal functions, variables, or loops in the program.
   - **Functional Testing cares about:** If 5+2 = 7, 10/2 =5
   - **It does NOT care about:** The code logic used to calculate the result.

   Program:

```
def add(a, b):
    return a + b
def subtract(a, b):
    return a - b
def multiply(a, b):
    return a * b
def divide(a, b):
    if b == 0:
       return "Error: Division by Zero"
    return a / b

print("Simple Calculator")
print("1. Add\n2. Subtract\n3. Multiply\n4. Divide")
choice = int(input("Enter choice (1-4): "))
x = int(input("Enter first number: "))
y = int(input("Enter second number: "))
if choice == 1:
   print("Result:", add(x, y))
elif choice == 2:
   print("Result:", subtract(x, y))
elif choice == 3:
   print("Result:", multiply(x, y))
elif choice == 4:
   print("Result:", divide(x, y))
else:
   print("Invalid Choice")
```

Output:

Add

```
Simple Calculator
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1-4):  1
Enter first number:  13
Enter second number:  16
Result: 29
```

Subtract

```
Simple Calculator
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1-4):  2
Enter first number:  5
Enter second number:  4
Result: 1
```

Multiply

```
Simple Calculator
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1-4):  3
Enter first number:  13
Enter second number:  16
Result: 208
```

Division

```
Simple Calculator
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1-4):  4
Enter first number:  15
Enter second number:  5
Result: 3.0
```

2. Test Scripts

Aim: To design and execute test scripts that can verify application behavior using multiple sets of
input data (Data-Driven Testing).

Objectives:

- To understand the concept of a test script.
- To explore the difference between manual test scripts and automated test scripts.
- To implement data-driven testing (DDT) by running the same script with multiple input

values.

Test Script:

- A test script is a sequence of instructions (manual or automated) that tests whether a software feature works as expected.
- It defines test steps, test data, and expected results.
- Test scripts ensure repeatability, accuracy, and efficiency in software testing.

Test Script vs Test Case

Feature	**Test Case (Manual)**	**Test Script (Automation)**
Definition	Step-by-Step test procedure	Program/script for automation
Execution	Manual Effort	Automated by tools/scripts
Origin	Derived from test scenarios	Derived from test cases

Reuse	Limited	Highly reusable

Advantages:

- High reusability.
- Better test coverage.
- Easy to maintain – just update data file.

Requirements:

- Any programming language (Python / Java / C#).
- Optionally, automation tools like Selenium for UI testing.
- CSV / Excel file (or in-code dataset).

Example Test Scenario:

Application: Login page

Test Data Table:

Username	**Password**	**Expected Result**
user1	**pass1**	**Login Successful**
user1	**wrong**	**Invalid Credentials**
wrong	**pass1**	**Invalid Credentials**
(empty)	**(empty)**	**Username/Password Required**

Automated Test Script (Python Demo Code)

```
test_data = [
    {"username":"user1","password":"pass1","expected":"Login Successful"},
    {"username":"user1","password":"wrong","expected":"Invalid Credentials"},
    {"username":"wrong","password":"pass1","expected":"Invalid Credentials"},
    {"username":"","password":"","expected":"Username/Password required"},
]
# Function to simulate login
def login(username, password):
    # Hardcoded valid login
    if username=="user1" and password=="pass1":
        return"Login Successful"
    elif username=="" or password=="":
        return"Username/Password required"
    else:
        return"Invalid Credentials"
# Running tests with multiple data
for data in test_data:
    result=login(data["username"],data["password"])
    if result == data["expected"]:
        print(f"PASS for {data}")
    else:
        print(f"FAIL for {data}|Got:{result}")
```

Output:

```
PASS for {'username': 'user1', 'password': 'pass1', 'expected': 'Login Successful'}
PASS for {'username': 'user1', 'password': 'wrong', 'expected': 'Invalid Credentials'}
PASS for {'username': 'wrong', 'password': 'pass1', 'expected': 'Invalid Credentials'}
PASS for {'username': '', 'password': '', 'expected': 'Username/Password required'}
```

Procedure

- Define the test scenario (e.g., login functionality).
- Prepare a data set with inputs and expected outputs.
- Write a test script (manual steps or automated code).

- Execute the test for each dataset.
- Record results as PASS/FAIL.
- Analyze any mismatches.

Result

- Test script executed successfully with multiple input sets.
- Behavior of the script remained consistent while data changed.
- Demonstrated that Data-Driven Testing improves efficiency and reusability.

3. Test Case: Instagram Login Page

Test Case ID	Description	Test Steps	Test Data	Expected Result	Actual Result	Status
GD01	**Valid Login**	**Enter correct username +password, click login**	**username: twi.nkle1316 pass: xyz1313**	**Redirect to instagram dashboard**	**(after test)**	**pass**
GD02	**Invalid Password**	**Enter correct username but wrong password**	**username: twi.nkle1316 pass: abcde1313**	**Error message "wrong password"**	**(after test)**	**pass**

GD03	**Empty username & password**	**Leave both fields empty & click login**	**(blank)**	**“Disabled login button”**	**(after test)**	**pass**

Test case 1: Correct username & password (redirect to dashboard)

Test case 2: Correct username but wrong password

Test case 3: leave both fields empty (Disabled login button)

4. Test Scenarios

Test Case ID	Description	Test Steps	Test Data	Expected Result	Actual Result	Status	Classification
GD01	Valid Login	Enter correct username +password, click login	username: twi.nkle1316 pass: xyz1313	Redirect to instagram dashboard	(after test)	pass	POSITIVE
GD02	Invalid Password	Enter correct username but wrong password	username: twi.nkle1316 pass: abcde1313	Error message “wrong password”	(after test)	pass	NEGATIVE
GD03	Empty username & password	Leave both fields empty & click login	(blank)	“Disabled login button”	(after test)	pass	NEGATIVE

Test case 1: Correct username & password (redirect to dashboard) (Classification: POSITIVE)

Test case 2: Correct username but wrong password

(Classification: NEGATIVE)

Test case 3: leave both fields empty (Disabled login button)

(Classification: NEGATIVE)

5. Test Metrics Life Cycle

Aim:

To implement Test Metric Tracking and understand the process of tracking and analyzing software testing progress through metrics**.**

Objectives: ● To understand the concept of test metrics.

● To learn how to use JIRA for tracking and managing testing activities.

● To analyze project and defect metrics using dashboards and reports.

Step 1: Prepare Test Data

Chart 1

	A	B	C	D	E	F	G	H	I
1	Test Case	Test Case Description	Priority	Type	Status	Defect Log	Severity	Execution Time (mins)	
2	TC001	Login functionality	High	Functiona	Passed	No	-	5	
3	TC002	Signup functionality	High	Functiona	Failed	Yes	Critical	8	
4	TC003	Search feature	Medium	Functiona	Passed	No	-	3	
5	TC004	Checkout process	High	Functiona	Failed	Yes	Major	10	
6	TC005	Profile update	Low	Functiona	Passed	No	-	4	
7	TC006	Password reset	Medium	Functiona	Failed	Yes	Minor	6	
8									

Step 2: Create Summary Metrics

B12 =AVERAGE(TestData!H2:H100)

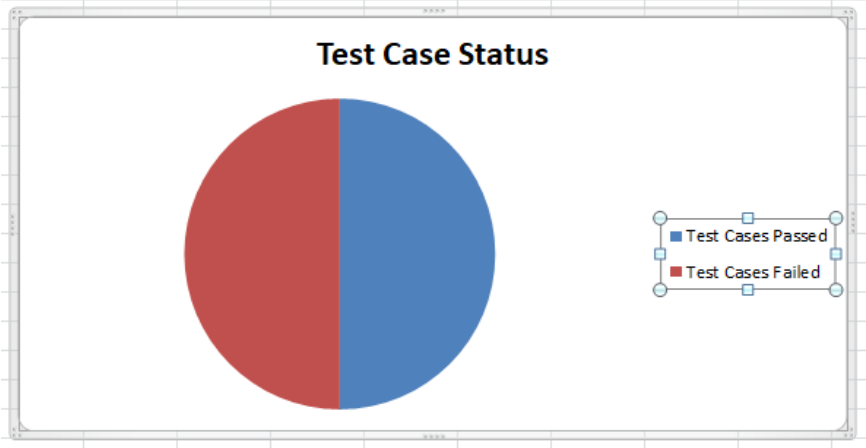
	A	B	C	D	E
1	Summary Metrics				
2					
3	Total Test Cases	6			
4	Test Cases Passed	3			
5	Test Cases Failed	3			
6	Defects Logged	3			
7					
8	Critical Defects	1			
9	Major Defects	1			
10	Minor Defects	1			
11					
12	Average Execution Time	6			
13					

Step 3: Create Charts

1. Test Case Status Pie Chart

- Data: Passed vs Failed
- Steps:

1. Select the cells containing "Test Cases Passed" and "Test Cases Failed".
2. Go to Insert → Pie Chart.

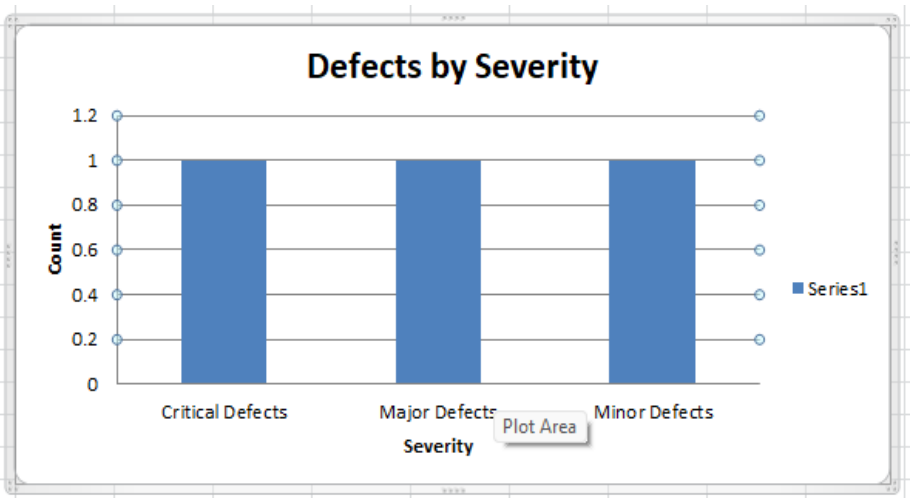


Purpose: Quickly visualize test case execution results.

2. Defects by Severity Column Chart

- Data: Critical, Major, Minor (from Step 2)
- Steps:

1. Select the severity count cells.
2. Insert → Column Chart.

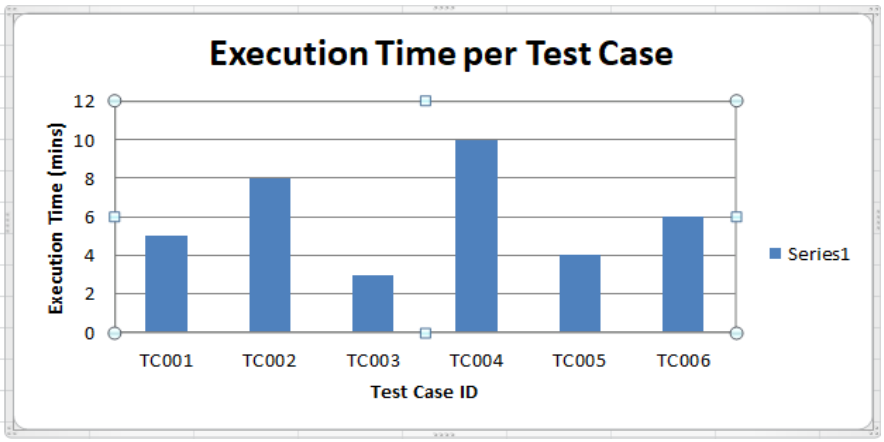


Purpose: Helps prioritize critical defects.

3. Test Execution Time Bar Chart

- Data: Test Case ID vs Execution Time
- Steps:

1. Select Test Case ID and Execution Time columns.
2. Insert → Bar Chart.
3. Optionally, sort by execution time.

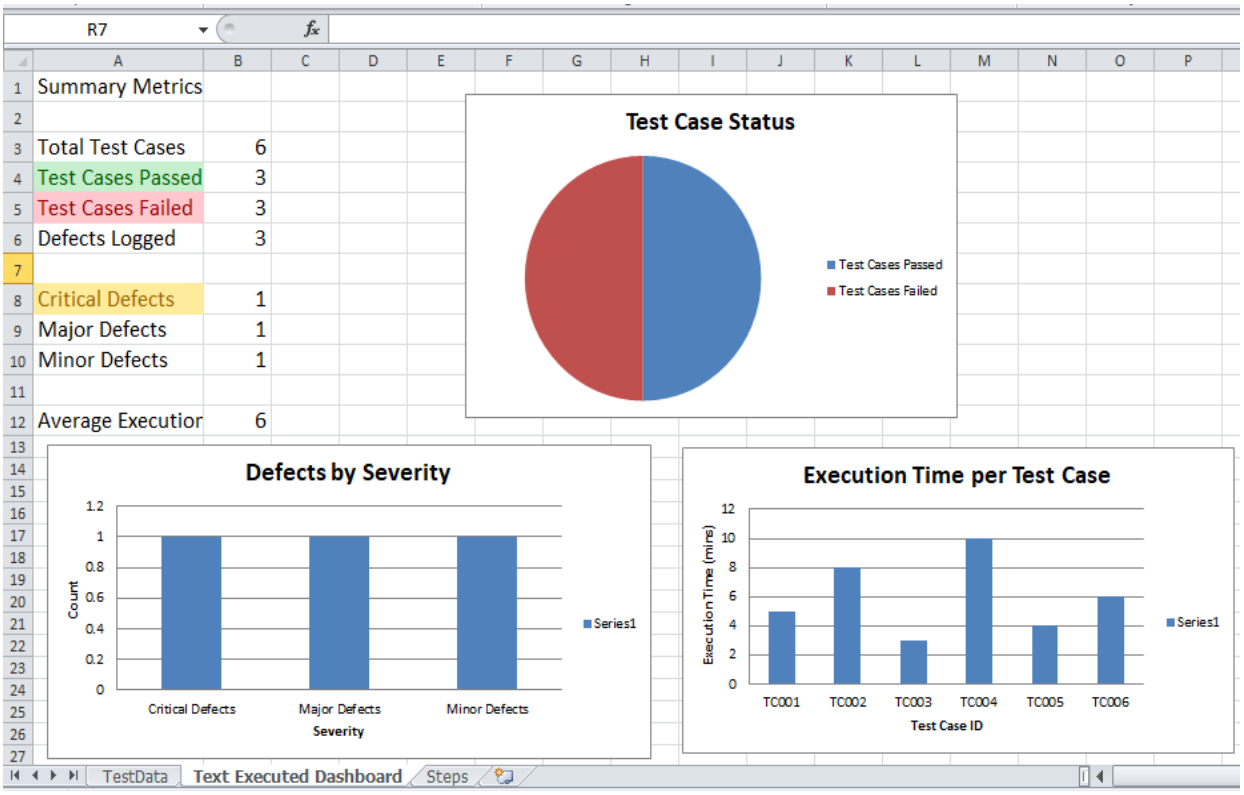


Purpose: Identify which test cases are taking longer to execute.

Step 4: Build the Dashboard

1. Arrange your charts neatly on the Dashboard sheet.

2. Add the summary metrics (Total Test Cases, Passed, Failed, Defects Logged) at the top.
3. Use Conditional Formatting for quick visualization:
o Green for Passed.
o Red for Failed.
o Yellow for High severity defects



6. Web Testing

Aim:

To test different components of a web application using functional and non-functional testing methods.

Theory Overview

A web application has multiple components such as pages, navigation bars, forms, links, and database interactions.

Testing ensures all these components work individually and together.

Test Examples

Test Case ID	Component	Test Scenario	Expected Output	Status
TC11	Responsive Layout	Open site on mobile	Adjusts layout properly	Pass
TC12	Forgot Password	Request password reset	Email with reset link sent	Pass
TC13	Broken Link Test	Check all hyperlinks	No 404 errors	Pass
TC14	Cookie Handling	Login → close browser → reopen	Session expired or remembered as per settings	Pass
TC15	Form Validation	Enter invalid email	Show "Invalid email format"	Pass
TC16	Database Test	Register new user	Data saved in database	Pass
TC17	Field Length	Enter 300 characters in username	Field should restrict extra input	Pass
TC18	Tooltip/Help Text	Hover on info icon	Show help message	Pass
TC19	File Upload	Upload profile photo	Accept image formats only	Pass
TC20	Redirection	HTTP to HTTPS redirect	Automatically redirect to HTTPS	Pass

Non-Functional Test Cases (Detailed)

Test Case ID	Parameter	Description	Tool Used	Expected Result	Actual Result	Status
NFC01	Page Load Speed	Measure page load time	GTmetrix	< 3 seconds	2.8 seconds	Pass
NFC02	Performance under Load	Simulate 50 users simultaneously	JMeter	No crash, stable response	Stable	Pass
NFC03	Browser Compatibility	Test in Chrome, Edge, Firefox	BrowserStack	Layout consistent	Consistent	Pass
NFC04	Mobile Compatibility	Check in mobile view	Chrome DevTools	Responsive layout	Working well	Pass
NFC05	Security	SQL injection attempt in login	OWASP ZAP	Error handled safely	Safe	Pass
NFC06	Accessibility	Test contrast, alt text	WAVE	No major accessibility issues	Pass	
NFC07	Usability	Evaluate ease of navigation	Manual	Easy to use	Pass	
NFC08	SEO	Analyze title, meta tags	Google Lighthouse	Optimized	Pass	
NFC09	Stress Testing	Increase server load	Apache JMeter	Server handles up to 100 users	Pass	
NFC10	Backup and Recovery	Simulate database crash	Manual check	Data recovery possible	Pass	

Result:

Functional and non-functional testing was successfully performed for multiple web components such as login, forms, search, navigation bar, and responsiveness.

The web application was found to be functionally stable, secure and compatible across browsers.