Testing

Agenda

- **What is Testing?**
- **Software Testing Concepts**
  - **White Box Testing Techniques**
  - **Black Box Testing Techniques**
  - **White Box vs Black Box Testing**
- **Unit Testing**
- **End-to-End Testing**
- **E2E vs Unit Testing**
- **Mocking**
- **Discussion and Lab time**

What is Testing?

- Process of evaluating software to detect differences between expected and actual results.
- Ensures software quality, reliability, and security.

White Box vs Black Box Testing

White Box Testing

(Code Focused):

- Testing with knowledge of the internal code.
- Focuses on internal logic and code structure

- *Ex. Unit Testing*

Black Box Testing

(User Focused):

- Focuses on input and output without looking at internal code.
- Based on specifications and user requirements.

- *Ex. End to End Testing*

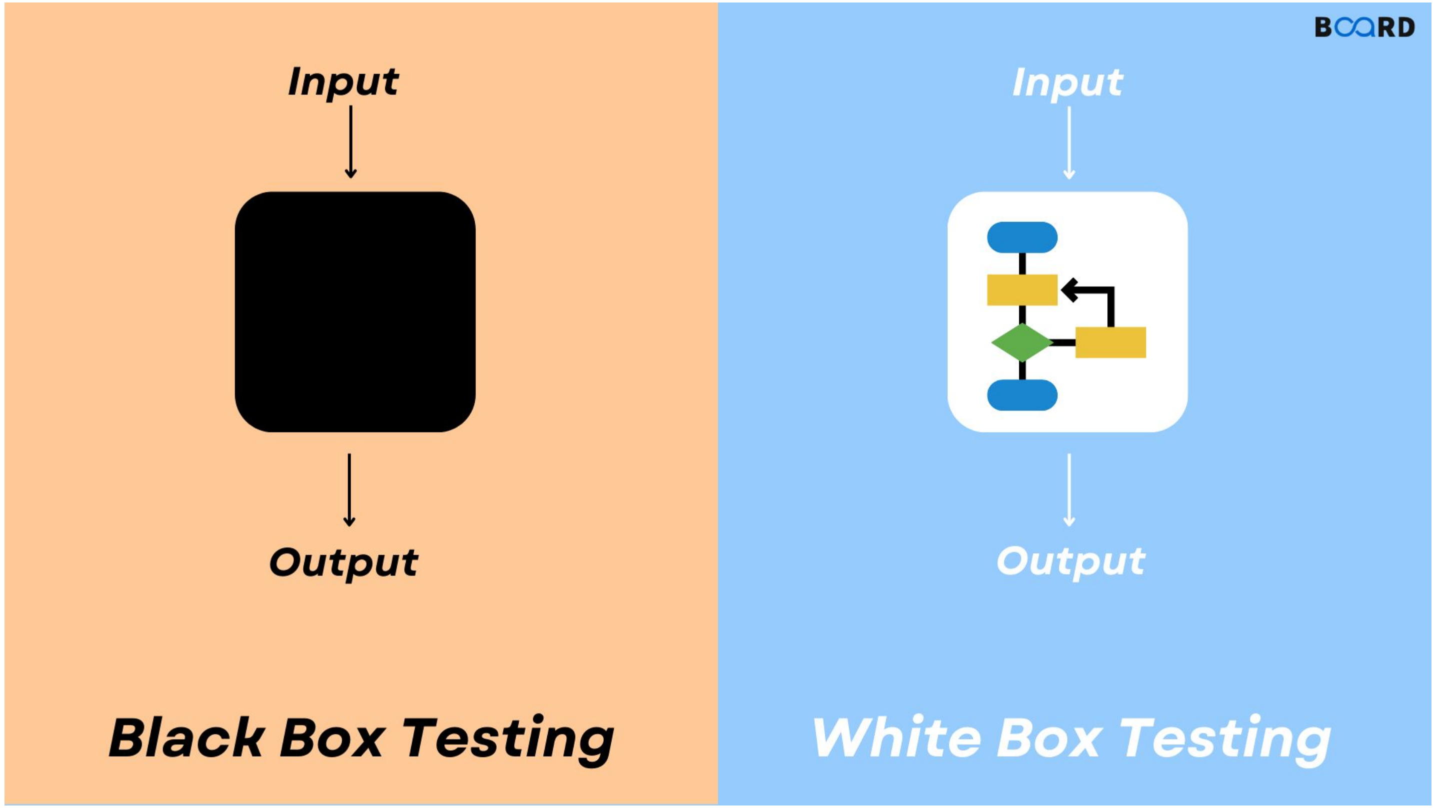


Image source:
https://www.boardinfinity.com/blog/white-box-vs-black-box/

White Box Testing Techniques

- **Statement Coverage**: Ensuring every line of code executes at least once.
- **Branch Coverage**: Ensuring every decision point is tested. (if/else)
- **Path Coverage**: Ensuring every possible execution path is tested.

Statement Coverage Example:

```
def add(a, b):
    if a < 0 or b < 0:
        return 0
    return a + b
assert add(2, 3) == 5
assert add(-1, 3) == 0
assert add(-2, -3) == 0
print("All tests passed!")
```

Black Box Testing Techniques

- **Equivalence Partitioning**: Grouping inputs into valid and invalid sets.
- **Boundary Value Analysis**: Testing at the edges of input ranges.
- **State Transition Testing**: Checking valid state changes.

Boundary value analysis example:

```
def is_valid_age(age):
    return 18 <= age <= 60


assert is_valid_age(18) == True
assert is_valid_age(60) == True
assert is_valid_age(17) == False
assert is_valid_age(61) == False
```

White Box vs Black Box Testing

```
def add(a, b):

    return a + b



assert add(2, 3) == 5
```

```
def login(username, password):

   pass

assert login("user",
"correct_password") == "Login
Successful"
```

Unit Testing

- Unit testing verifies **individual functions** or **components** in isolation to ensure they work as expected.
- Helps catch bugs early, improve code reliability, and make refactoring safer.
- Popular frameworks
    - jest in js
    - unittest python

Unit Testing

```
import unittest

def multiply(a, b):

    return a * b
```

```
class TestMathOperations(

unittest.TestCase):

  def test_multiply(self):

   assert multiply(2, 3) == 6

  assert multiply(1, 5)== -5

if __name__ == '__main__':

   unittest.main()
```

E2E Testing

- E2E testing simulates ***real user interactions*** by testing the entire application flow from start to finish.
- Ensures all components work together correctly, ***detecting integration issues*** and verifying system reliability.

End-to-End Testing [Example]

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("https://example.com/login")

driver.find_element("name", "username").send_keys("testuser")

driver.find_element("name", "password").send_keys("password123")

driver.find_element("name", "login").click()

assert "Welcome" in driver.page_source

driver.quit()
```

E2E vs Unit Testing

Unit Testing:

- Tests individual components of the software in isolation.
- Fast and reliable.
- Catch small, isolated bugs
- Example (functions test using python unit test or jest in js)

End-to-End (E2E) Testing:

- Tests the entire system flow, including integration
- Usually slow
- Ensures all parts integrate together
- Example (using Selenium for UI testing):

Mocking

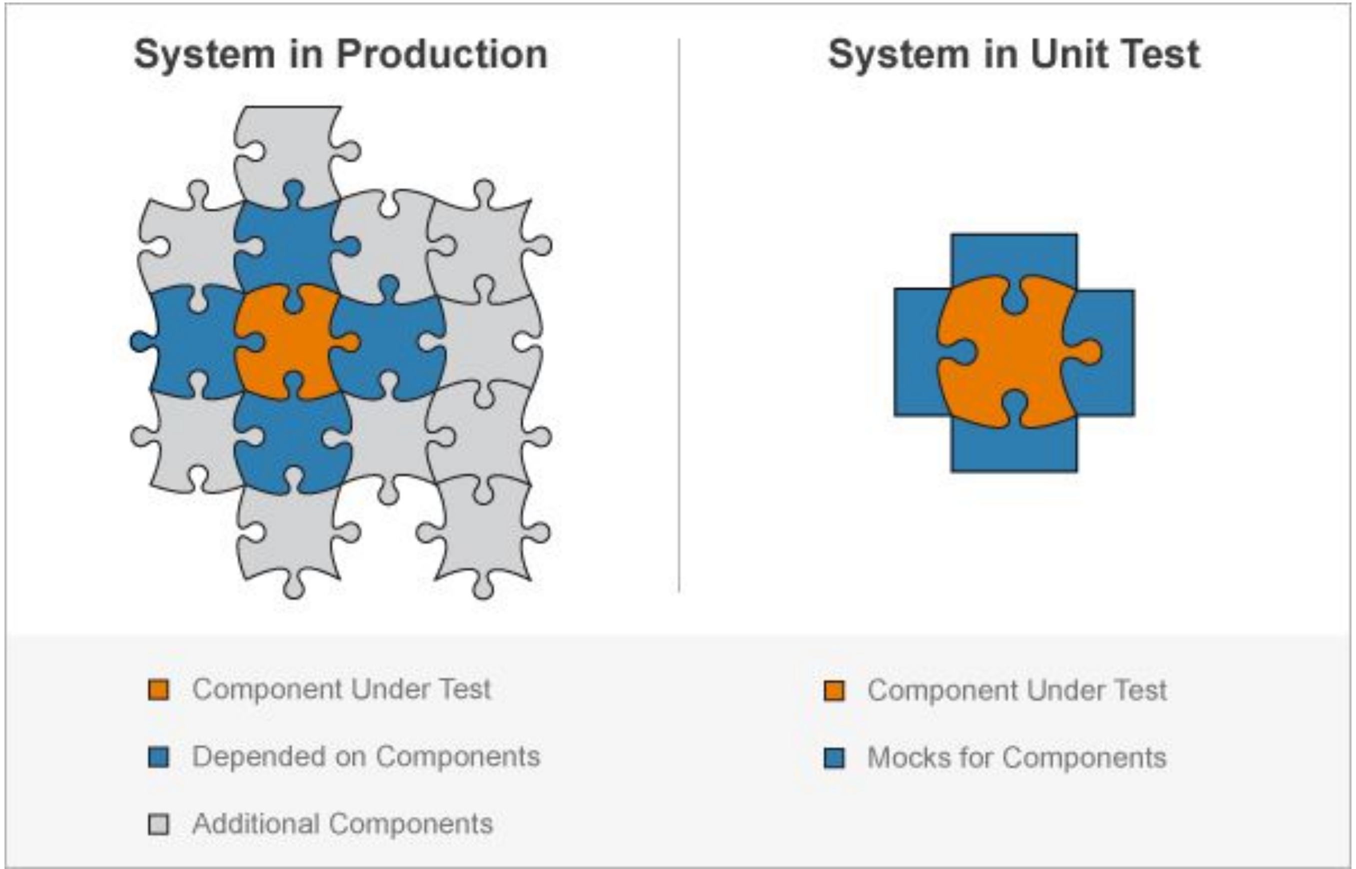
Technique that replaces real objects with fake ones during testing.

Test different simulated scenarios to control outcome.

Helps isolate unit tests from dependencies.

Dependencies could be *API calls, External Services, Databases, complex systems sensors readings, etc*.

Mocking


System in Production
System in Unit Test
🟧 Component Under Test
🟦 Depended on Components
⬜ Additional Components
🟧 Component Under Test
🟦 Mocks for Components

Mocking [Code Example]

```
from unittest.mock import Mock

class AuthService:

    def __init__(self, database):

        self.database = database

    def authenticate(self, user_id):

        user =
self.database.get_user(user_id)

        if user and
user.get("is_active"):

            return "Authenticated"

        return "Access Denied"
```

```
db_mock = Mock()

db_mock.get_user.return_value =
{"id": 1, "name": "Ahmed"}

auth_service =
AuthService(db_mock)

assert
auth_service.authenticate(1) ==
"Authenticated"
```

Mocking Payments with Stripe Test Mode

How do we test a payment system without actually charging a credit card?

Without Mocking:

App → Stripe API(Real) → Real Payment (Response)

With Mocking:

App → Stripe API (Test Mode) → No real interaction (Simulated Response)

Discussion and Lab Time