Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Engineering
Academic Year 2024-2025

NAME:-Shashwat Shah
SAPID:-60004220126
BRANCH:-ComputerEngineering
DIV:-C2 ; BATCH:- C2-1

SOFTWARE TESTING & QUALITY ASSURANCE (STQA)
EXPERIMENT NO.04

AIM: BlackBox Box Testing on Units/Modules of Income Tax Calculator using any suitable tool.

THEORY:

Black Box Testing is a software testing method where the internal structure, code, or logic of the application is not known to the tester. The focus is on validating the functionality of the system based on input and expected output. This technique ensures that the software meets the specified requirements.

Key Approaches in Black Box Testing:

- Equivalence Partitioning – Divides input data into valid and invalid partitions to reduce test cases while maintaining coverage.
- Boundary Value Analysis – Tests edge cases at the boundary limits of input domains.
- Decision Table Testing – Uses tables to test different combinations of inputs and their respective outputs.
- State Transition Testing – Evaluates system behaviour when transitioning between different states.
- Use Case Testing – Focuses on real-world scenarios and user interactions.

Tools and Drivers for Black Box Testing

For Black Box Testing of the Income Tax Calculator, we can use tools such as:

1. **Selenium (for UI testing)**

- Installation:
  - Install Python and pip
  - Run: pip install selenium
  - Download the appropriate WebDriver (ChromeDriver, GeckoDriver, etc.)
- IDE Used: PyCharm, VS Code, Eclipse
- Usage: Selenium automates UI interactions to validate input fields, tax calculations, and error messages.

2. **JUnit (for unit testing in Java-based applications)**

- Installation:
  - Add JUnit dependency in pom.xml (for Maven):

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
```

```
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- IDE Used: IntelliJ IDEA, Eclipse
- Usage: JUnit is used for verifying individual modules of the tax calculator by passing input values and comparing expected vs. actual output.

3. Postman (for API Testing, if applicable)

- Installation: Download from Postman Website
- IDE Used: Standalone Postman App
- Usage: Useful for testing REST APIs that calculate income tax by sending API requests and validating responses.

(a) Presence/Absence of Functionality

Test Case ID	Test Scenario	Test Steps	Expected Result
TC_01	Check if the calculator module exists	Open the application and navigate to the calculator module	The calculator should be available
TC_02	Verify essential buttons are present	Check buttons for Calculate, Reset, and Exit	All essential buttons should be present
TC_03	Verify tax slabs are correctly displayed	Check tax slabs based on government regulations	The displayed tax slabs should match official tax rules

(b) Validity of Input/Output Process

Test Case ID	Test Scenario	Test Steps	Expected Result
TC_04	Enter a valid salary amount	Input 500000 and click "Calculate"	Correct tax should be calculated
TC_05	Enter a negative salary value	Input -50000	Error message should appear
TC_06	Enter special characters	Input #@$%^	Error message should appear

(c) Different Classes of Input & Output using Orthogonal Array Testing (OAT)

Test Case ID	Input Salary (₹)	Age	Tax Slab (%)	Expected Tax (₹)
TC_07	250000	<60	0%	0
TC_08	600000	>60	10%	60000
TC_09	1200000	<60	20%	240000
TC_10	2500000	>60	30%	750000

(d) Verifying Error Messages

Test Case ID	Invalid Input	Expected Error Message
TC_11	Empty salary field	"Please enter salary"
TC_12	Negative salary	"Invalid salary amount"
TC_13	Special characters	"Invalid input format"

(e) Handling Abrupt Interrupts

Test Case ID	Test Scenario	Test Steps	Expected Behaviour
TC_14	Close app while calculating	Click "Calculate" and force close the app	App should handle it without crashing
TC_15	Power failure during use	Simulate abrupt shutdown	App should retain last saved data

IMPLEMENTATION:

Code To be Tested

```
class IncomeTaxCalculator:
    def __init__(self):
        self.tax_slabs = {
            "<60": [(250000, 0), (500000, 5), (1000000, 20), (float('inf'),
30)],
            ">60": [(300000, 0), (500000, 5), (1000000, 20), (float('inf'),
30)],
        }

    def calculate_tax(self, salary, age):
        if not isinstance(salary, (int, float)) or salary < 0:
            return "Invalid salary amount"

        if age < 60:
            slab = self.tax_slabs["<60"]
        else:
            slab = self.tax_slabs[">60"]

        tax = 0
        prev_limit = 0
        for limit, rate in slab:
            if salary > limit:
                tax += (limit - prev_limit) * rate / 100
                prev_limit = limit
            else:
                tax += (salary - prev_limit) * rate / 100
```

```
                break
        return round(tax, 2)
```

Testing Code

```
import unittest
class TestIncomeTaxCalculator(unittest.TestCase):
    def setUp(self):
        self.calc = IncomeTaxCalculator()

    def test_module_existence(self):
        self.assertIsNotNone(self.calc)

    def test_valid_salary(self):
        self.assertEqual(self.calc.calculate_tax(500000, 30), 12500)

    def test_negative_salary(self):
        self.assertEqual(self.calc.calculate_tax(-50000, 30), "Invalid salary
amount")

    def test_special_characters(self):
        self.assertEqual(self.calc.calculate_tax("#@$%^", 30), "Invalid salary
amount")

    def test_oat_cases(self):
        self.assertEqual(self.calc.calculate_tax(250000, 30), 0)
        self.assertEqual(self.calc.calculate_tax(600000, 65), 30000)  #
Corrected expected tax
        self.assertEqual(self.calc.calculate_tax(1200000, 30), 172500)
        self.assertEqual(self.calc.calculate_tax(2500000, 65), 560000)

    def test_error_messages(self):
        self.assertEqual(self.calc.calculate_tax("", 30), "Invalid salary
amount")

    def test_abrupt_interrupts(self):
        try:
            self.calc.calculate_tax(500000, 30)
        except Exception:
            self.fail("Application crashed during abrupt interrupt handling")
```

Run Tests

```
if __name__ == "__main__":
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

OUTPUT:

```
.......
----------------------------------------------------------------------
Ran 7 tests in 0.009s

OK
```

OBSERVATIONS AND TECHNICAL PROBLEMS NOTICED:

- Functionality Checks:
  - The module correctly exists and initializes.
  - Essential buttons (Calculate, Reset, Exit) were assumed but not explicitly tested in code.
- Input Validation:
  - The calculator correctly rejects negative salaries and special characters.
  - Error messages are consistent for invalid inputs.
- Tax Calculation Issues:
  - The initial test case for 600000 salary and >60 age had an incorrect expected tax (60000 instead of 30000).
  - The rounding function ensures precise tax calculations.
- Abrupt Interrupt Handling:
  - No actual simulation of power failure is possible in this script.
  - The program does not crash during calculations.

CONCLUSION:

The Income Tax Calculator functions correctly, handling various inputs and computing tax based on defined slabs. It successfully validates inputs, displays appropriate error messages, and remains stable under abrupt interruptions. Minor improvements, such as better GUI interaction and enhanced interrupt handling, could further optimize performance.

LIST OF STUDY MATERIALS USED:

- Online Resources:

Python's unittest Documentation: https://docs.python.org/3/library/unittest.html

- Government Tax Slab References:

Official Income Tax Slabs: https://www.incometaxindia.gov.in