Repository Link: https://github.com/Shahriar-0/Software-Testing-Course-Projects-S2025
Latest Commit Hash: 1465c747e9e7bd4151a5533ca3af069542041797

Logic, ISP, and API test

1. `SpringBootTest` vs `WebMvcTest`

`@SpringBootTest`

It is to test the full application context. Loads the entire Spring application context, including all beans, components, and configurations. It's useful for integration tests where the goal is to verify the interaction between multiple layers (e.g., Controller → Service → Repository). Slower compared to `@WebMvcTest` because it initializes the full context.

```
@SpringBootTest
class ApplicationTests {

    @Autowired
    private MyService myService;

    @Test
    void testService() {
        String result = myService.doSomething();
        assertEquals("ExpectedResult", result);
    }
}
```

`@WebMvcTest`

To test only the web layer (Spring MVC components like Controllers). Loads only the Spring MVC components, such as `@Controller`, `@RestController`, and related configurations (e.g., `@ControllerAdvice`, `HandlerMethodArgumentResolver`) and excludes non-web components like `@Service` and `@Repository`. It also mocks beans not related to the web layer unless explicitly included.

```
@WebMvcTest(MyController.class)
class MyControllerTests {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void testControllerEndpoint() throws Exception {
```

```
        mockMvc.perform(get("/api/endpoint"))
                .andExpect(status().isOk())
                .andExpect(content().string("ExpectedResponse"));
    }
}
```

Comparison Table

Feature	`@SpringBootTest`	`@WebMvcTest`
Purpose	Integration testing of the entire app	Unit testing the web layer
Application Context	Loads the entire Spring Boot context	Loads only the web layer context
Components Tested	All beans, services, repositories, etc.	Only controllers and related components
Performance	Slower due to full context initialization	Faster due to limited context
Use Case	End-to-end testing	Controller endpoint testing

2. CACC and RACC

(a) Truth Table

a	**b**	**c**	**¬a**	**¬b**	**¬c**	**¬a ∧ b**	**b ∧ c**	**¬b ∧¬c**	**p**
T	T	T	F	F	F	F	T	F	T
T	T	F	F	F	T	F	F	F	F
T	F	T	F	T	F	F	F	F	F
T	F	F	F	T	T	F	F	T	T
F	T	T	T	F	F	T	T	F	T
F	T	F	T	F	T	T	F	F	T
F	F	T	T	T	F	F	F	F	F
F	F	F	T	T	T	F	F	T	T

(b) Pairs of Rows Satisfying CACC

We can use major clauses and … but it can also get extracted from the truth table.

Clause 1: a (major clause: b = true, c = false) -> 2, 6

- **Rows:** 2, 6

Clause 2: b (major clase: {c = true, a = false, true} - {c = false, a = true}) -> 1, 3, 2, 4, 5, 7

- **Rows:** 1, 3 - 1, 7 - 5, 3 - 5, 7 - 4, 2

Clause 3: c (major clause: {b: false, a = true, false} - (b: false, true, a = true}) -> 1, 2, 3, 4, 7, 8

- **Rows:** 1, 2 - 4, 3 - 4, 7 - 8, 3 - 8, 7

(c) Pairs of Rows Satisfying RACC

Clause 1: a

- **Rows:** 2, 6

Clause 2: b

- **Rows:** 1, 3 - 2, 4 - 5, 7

Clause 3: c

- **Rows:** 1, 2 - 3, 4 - 7, 8

(d) Is the CACC Pair Set a Subset of RACC?

No cause we see pairs in CACC that are not in RACC. CACC requires other clauses to be neutral, while RACC is stricter and requires other clauses to remain fixed. Since RACC imposes a stricter condition, all RACC pairs also satisfy CACC.

(e) Does Clause Coverage Imply Predicate Coverage?

Counterexample: TTT and FFF, we have clause coverage but we don’t have predicate coverage since both of them are T, thus, CC does not imply PC.

3. ISP

Divide inputs into blocks:

- **price**:
  - A0: (<= 0) (Invalid)
  - A1: (0 < price < minPurchase) (Valid, no discount)
  - A2: (price >= minPurchase) (Valid, with discount)
- **discountRate**:
  - B0: (< 0) (Invalid)
  - B1: (0) (No discount)
  - B2: (0 < discountRate <=1) (Valid)
  - B3: (> 1) (Invalid)
- **minPurchase**:
  - C0: (<= 0) (Invalid)
  - C1: (> 0) (Valid)

Test Implementation in Python

```
import unittest

def calculate_discounted_price(price, discount_rate, min_purchase):
    if price <= 0 or discount_rate < 0 or discount_rate > 1 or
min_purchase <= 0:
        return "Invalid input"
    elif price < min_purchase:
        return f"{price:.1f}"
    else:
        discounted_price = price * (1 - discount_rate)
        return f"{discounted_price:.1f}"


class TestCalculateDiscountedPrice(unittest.TestCase):

    # A0, B0, C0
    def test_a0_b0_c0(self):
        self.assertEqual(calculate_discounted_price(-10, -0.1, -50),
"Invalid input")

    # A0, B1, C1
    def test_a0_b1_c1(self):
        self.assertEqual(calculate_discounted_price(-10, 0, 50),
```

```
"Invalid input")

    # A0, B2, C0
    def test_a0_b2_c0(self):
        self.assertEqual(calculate_discounted_price(-10, 0.2, -50),
"Invalid input")

    # A0, B3, C1
    def test_a0_b3_c1(self):
        self.assertEqual(calculate_discounted_price(-10, 1.5, 50),
"Invalid input")

    # A1, B1, C0
    def test_a1_b1_c0(self):
        self.assertEqual(calculate_discounted_price(10, 0, -50),
"Invalid input")

    # A1, B3, C0
    def test_a1_b3_c0(self):
        self.assertEqual(calculate_discounted_price(10, 1.5, -50),
"Invalid input")

    # A1, B0, C1
    def test_a1_b0_c1(self):
        self.assertEqual(calculate_discounted_price(10, -0.1, 50),
"Invalid input")

    # A1, B2, C1
    def test_a1_b2_c1(self):
        self.assertEqual(calculate_discounted_price(10, 0.2, 50),
"10.0")

    # A2, B1, C0
    def test_a2_b1_c0(self):
        self.assertEqual(calculate_discounted_price(50, 0, -50),
"Invalid input")

    # A2, B2, C0
    def test_a2_b2_c0(self):
        self.assertEqual(calculate_discounted_price(50, 0.2, -50),
"Invalid input")

    # A2, B0, C1
    def test_a2_b0_c1(self):
        self.assertEqual(calculate_discounted_price(50, -0.1, 50),
```

```
"Invalid input")

    # A2, B3, C1
    def test_a2_b3_c1(self):
        self.assertEqual(calculate_discounted_price(50, 1.5, 50),
"Invalid input")


if __name__ == "__main__":
    unittest.main()
```