Q:1:- Software Testing Techniques:
- Exploring test case design techniques, including equivalence partitioning and boundary value analysis?

Ans:- **Software Testing Techniques:**

Software testing plays a pivotal role in ensuring the reliability and functionality of applications. Among the various testing techniques available, Equivalence Partitioning and Boundary Value Analysis stand out for their ability to enhance test coverage and effectiveness. In this blog post, we delve into the intricacies of these two techniques and explore how they contribute to the robustness of software testing.

Equivalence Partitioning:

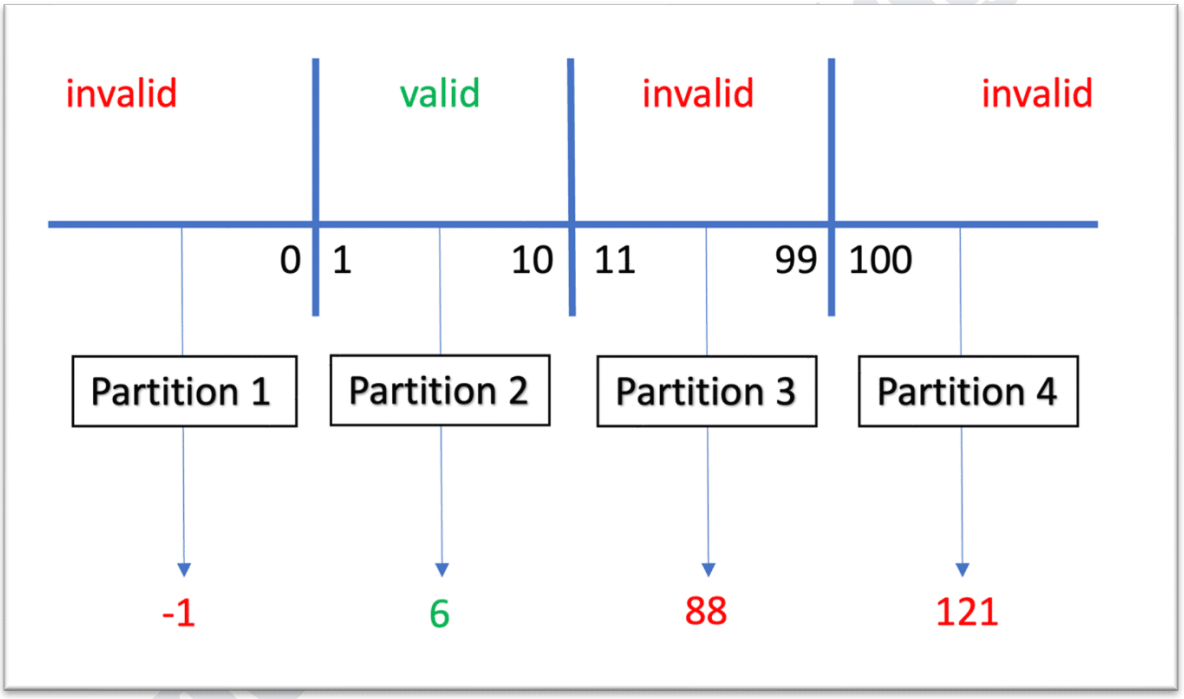
Starting with our Equivalence partitioning test technique, this test technique can be applied at all levels, such as system integration and unit testing. The most important advantage is that we can test large ranges with very few test cases. These are usually ranges of numbers. Here we break the data into parts. These are called partition or equivalence classes. These partitions can contain valid and invalid values.

The important thing here is that each partition can have only one value. So we cannot handle boundaries, for example, a value cannot belong to partition 1 and partition 2 at the same time. If necessary, it can be divided into sub-partitions, depending on the risk the tester sees. The perspective here is that if one value in a partition passes the test, so do the others, and if it fails, so do the others.

The Scenario: Online Ticket Reservation:

Let’s imagine an online platform where users can reserve tickets for events. When approving the reservation, various actions are requested depending on the number of tickets on the confirmation screen.

1. If the number of tickets is between 1 and 10, a success message should be shown.
2. If it is less than 1, an error message like “You must select at least one ticket” should be displayed.
3. If between 11 and 99, an error message like “Maximum 10 tickets can be reserved.” should be displayed.
4. If it is higher than 99, a system error should be displayed as a “General system error”.



In this case, we can consider our control values for this technique as 1, 10 and 99. If we divide them into partitions, there are 4 parts. We have invalid partitions for values less than 1, valid for values between 1 and 11, invalid for values between 11 and 100, and invalid for 100 and more.

The values we will test here are not these limits, because the values we will select with this technique should belong to a single partition.

Test Case	Number of Tickets	Expected Output
TC1	-1	"You must select at least one ticket" message
TC2	6	Success message
TC3	88	"Maximum 10 tickets can be reserved." message
TC4	121	"General system error". message

The minimum number of test cases for this example is 4. Of these 4 cases, 3 are for testing valid partitions and 1 is for testing invalid partitions. More values can be selected according to the risk seen.

Benefits of Equivalence Partitioning Testing:

Equivalence partitioning testing offers efficient test case design by grouping similar input values into classes, reducing redundancy, and optimising testing efforts. This approach ensures enhanced test coverage, early defect detection, and improved maintainability, contributing to a more organised testing process.

Challenges of Equivalence Partitioning Testing:

Despite its benefits, equivalence partitioning relies on the assumption of equal probability within classes, potentially missing defects in specific values. This method may not be suitable for all input types, and its effectiveness depends on accurate input specifications. Additionally, equivalence partitioning has limitations in handling combinatorial effects, necessitating a judicious combination with other testing techniques for comprehensive coverage.

Boundary-Value Analysis (BVA):

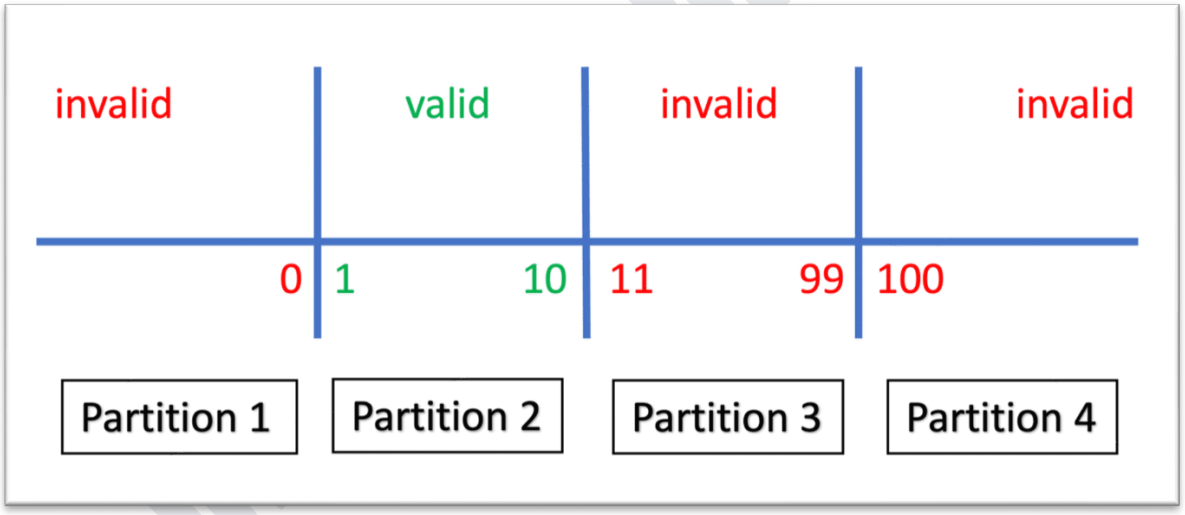
Much like Equivalence Partitioning, this testing technique is versatile and applicable across various testing levels, representing an extended version

of the Equivalence Partitioning approach. In this methodology, our attention is directed toward comprehensively defining partitions by considering not only the typical range but also the specific minimum and maximum values or the initial and final values associated with each partition.

Developers might make mistakes in handling edge cases, such as off-by-one errors, incorrect boundary checks, or issues related to the inclusivity or exclusivity of the boundaries. This technique is quite important for detecting such situations.

Number of boundaries = (Number of partitions – 1) x 2

Let's consider the above ticket reservation system example with this technique:



For Partition 1, the value "0" represents an invalid boundary. For Partition 2, "1" is the minimum, and "10" is the maximum valid boundary value. In Partition 3, "11" is the minimum, and "99" is the maximum invalid boundary value. As for Partition 4, the value "100" is an invalid boundary.

Test Case	Number of Tickets	Expected Output
TC1	0	"You must select at least one ticket" message
TC2	1	Success message
TC3	10	Success message
TC4	11	"Maximum 10 tickets can be reserved." message
TC5	99	"Maximum 10 tickets can be reserved." message
TC6	100	"General system error." message

The minimum number of test cases for all boundaries is 6. Of these 6 boundaries, 2 are for testing valid boundaries.

Benefits of Boundary Value Analysis (BVA) Testing:

Boundary Value Analysis (BVA) proves advantageous in software testing by focusing on critical points at the edges of input ranges. This precision facilitates early defect detection, as BVA targets areas where errors are more likely to occur. The efficiency in test case design, requiring fewer cases than exhaustive testing, ensures optimal resource utilisation. BVA is particularly effective for numeric inputs, addressing vulnerabilities at the boundaries of acceptable ranges.

Challenges of Boundary Value Analysis (BVA) Testing:

However, challenges include its assumption of linearity, potential oversight of defects within the input range or due to specific value combinations, and the resource-intensive nature of implementation for large and complex systems. Accurate input specifications are crucial, and BVA may not cover all possible scenarios, necessitating a complementary approach for comprehensive testing.

Combining Equivalence Partitioning and Boundary Value Analysis:

While Equivalence Partitioning and Boundary Value Analysis are powerful on their own, combining them can provide even more comprehensive test coverage. By selecting test cases at the boundaries of equivalence partitions, you ensure that both techniques complement each other, identifying defects that might be missed by using only one of the techniques.

Best Practices:

1. **Identifying Equivalence Classes:**
    a. Define equivalence classes based on similar behaviour or characteristics.
    b. Determine valid and invalid partitions for different inputs.
2. **Selecting Boundary Values:**
    a. Choose boundary values for each equivalence class.
    b. Consider the minimum, maximum, and values just inside and outside the boundaries.
3. **Designing Test Cases:**
    a. Develop test cases that cover both equivalence classes and their respective boundary values.
    b. Ensure a balanced representation of valid and invalid scenarios.
4. **Executing Tests:**
    a. Execute the test cases systematically, considering the identified equivalence classes and boundary values.

b. Record and analyse the results, identifying any discrepancies or unexpected behaviour.

Key Takeaways:

Equivalence Partitioning streamlines test case design by grouping similar input values into classes, offering enhanced coverage and early defect detection. However, its effectiveness depends on accurate input specifications.

Boundary Value Analysis focuses on critical edge points, ensuring efficient resource utilisation but faces challenges in complex systems. Combining these techniques provides a comprehensive testing approach, addressing individual limitations and maximising defect identification. Best practices involve careful class and boundary identification, balanced test case design, and systematic execution for reliable software testing.