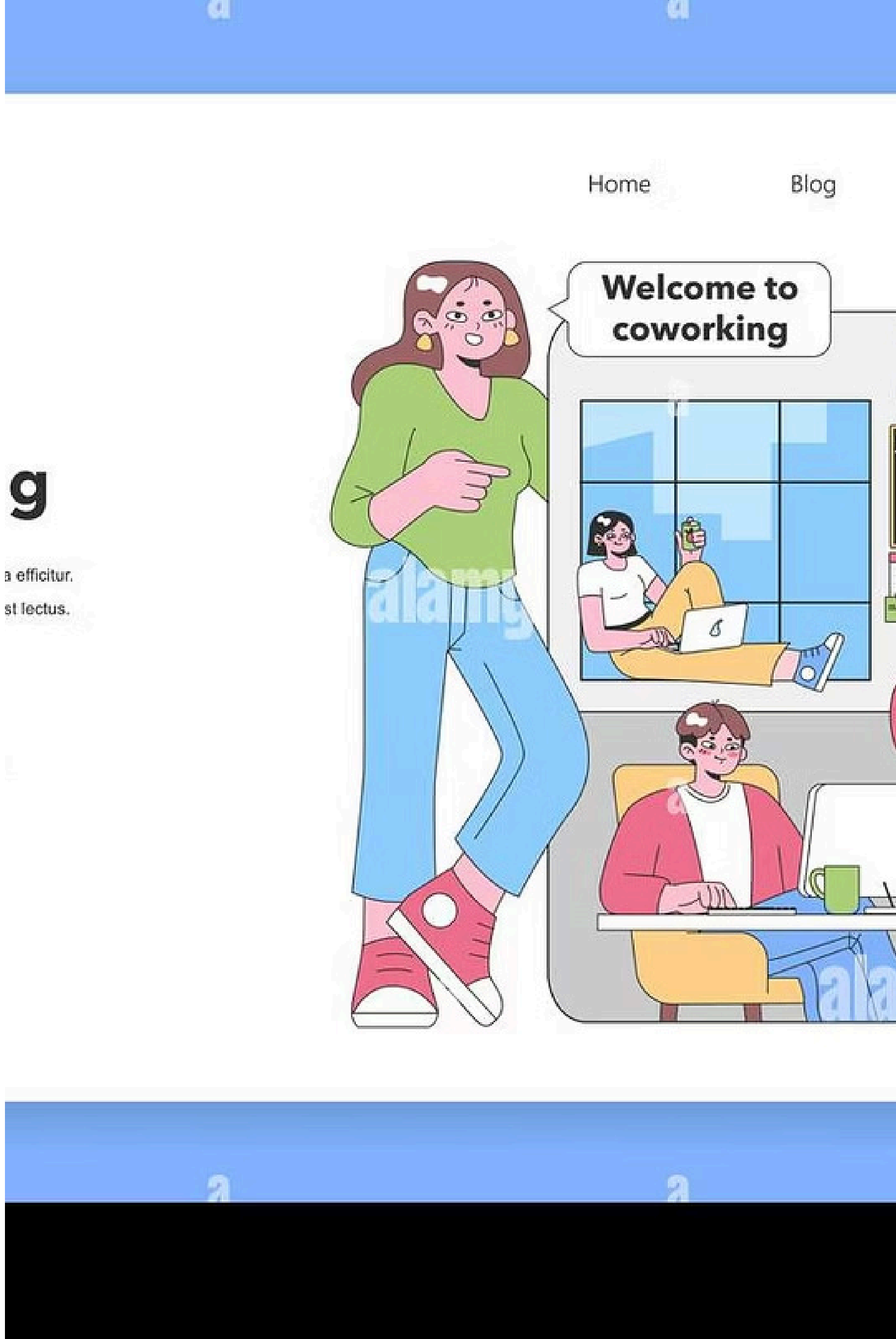
Test Design Techniques and Test Data Preparation

A Practical Guide for QA and QE Professionals



TRAINING OBJECTIVES

What You'll Learn Today

Master Core Techniques

Understand five essential test design approaches used across the software industry

Apply Real-World Examples

Learn through practical scenarios from e-commerce, banking, and enterprise applications

Prepare Test Data

Develop skills to create effective test data that ensures comprehensive coverage

Interview Readiness

Gain insights into common interview questions and mistakes to avoid

Course Overview

01

Equivalence Partitioning

Grouping similar inputs to reduce test cases whilst maintaining coverage

02

Boundary Value Analysis

Testing the edges where defects commonly hide

03

Decision Table Testing

Mapping complex business rules and conditions

04

State Transition Testing

Verifying system behaviour across different states

05

Test Data Preparation

Creating robust datasets for effective testing

06

Capstone Exercise

Applying all techniques to a comprehensive case study

SECTION 1

Equivalence Partitioning

Definition and Purpose

Equivalence Partitioning is a black-box testing technique that divides input data into logical groups (partitions) where all values are expected to behave similarly. The fundamental principle is that if one value in a partition works correctly, all values in that partition should work correctly.

This technique dramatically reduces the number of test cases needed whilst maintaining thorough coverage. Rather than testing every possible value, we select representative values from each partition.

Why Use Equivalence Partitioning?

Efficiency

Reduces redundant test cases by up to 80% whilst maintaining quality coverage. Instead of testing 100 age values, test representative values from each valid and invalid partition.

Comprehensive Coverage

Ensures all classes of input data are tested systematically. Identifies both valid partitions (accepted inputs) and invalid partitions (rejected inputs).

Early Defect Detection

Exposes logic errors in input validation and processing. Particularly effective for finding issues in data handling and business rule implementation.

Maintainability

Creates a logical test case structure that's easy to update when requirements change. New partitions can be added without redesigning the entire test suite.

Practical Example 1: E-Commerce Discount System

Scenario

An online shop applies discounts based on order value:

- Orders under £50: No discount
- Orders £50–£99.99: 5% discount
- Orders £100–£499.99: 10% discount
- Orders £500 and above: 15% discount

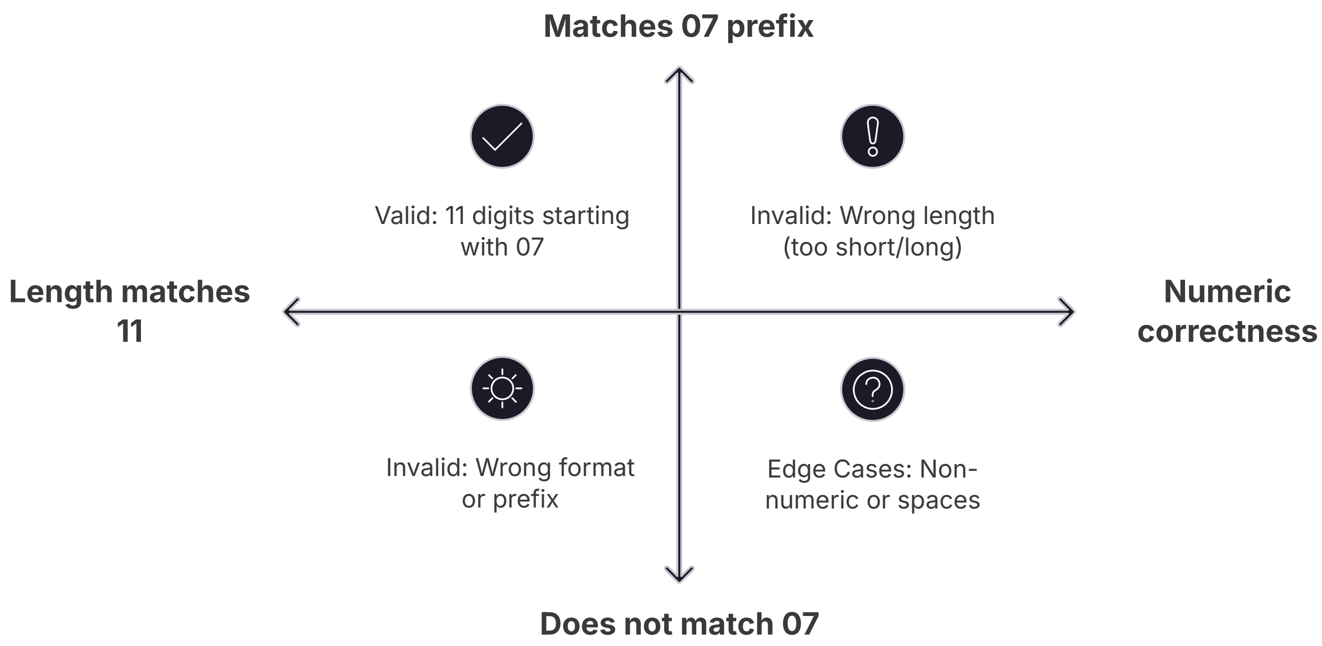
Traditional testing might use dozens of values. Equivalence Partitioning identifies four valid partitions and one invalid partition (negative values).

Partition	Range	Test Value
Invalid	Below £0	-£10
Valid 1	£0–£49.99	£25
Valid 2	£50–£99.99	£75
Valid 3	£100–£499.99	£250
Valid 4	£500+	£600

Result: 5 test cases instead of potentially hundreds, with complete logical coverage.

Practical Example 2: Mobile Number Validation

A registration form accepts UK mobile numbers starting with 07 and containing exactly 11 digits.



This approach systematically covers all input scenarios with minimal test cases.

Identified Partitions

Partition Type	Representative Test Value
Valid	07912345678
Invalid: Too short	079123456
Invalid: Too long	079123456789
Invalid: Wrong prefix	08912345678
Invalid: Non-numeric	07ABC345678
Invalid: Empty	(blank field)

Common Mistakes to Avoid

1

Overlapping Partitions

Creating partitions with ambiguous boundaries. For example, defining one partition as "1–10" and another as "10–20" creates confusion at value 10. Ensure partitions are mutually exclusive.

2

Missing Invalid Partitions

Focusing only on valid inputs whilst ignoring invalid ones. Always identify and test negative scenarios such as null values, special characters, or out-of-range data.

3

Too Many Test Cases Per Partition

Selecting multiple values from the same partition defeats the purpose. One representative value per partition is sufficient unless boundary considerations apply.

4

Ignoring Business Logic

Creating partitions based purely on data types rather than business rules. Always align partitions with actual system behaviour and requirements.

Knowledge Check: Equivalence Partitioning

Question 1

A password field accepts 8–16 characters. How many equivalence partitions exist?

A) 2 partitions
B) 3 partitions
C) 4 partitions
D) 9 partitions

Question 2

Which statement about equivalence partitioning is correct?

A) Only valid partitions need to be tested
B) Each partition requires multiple test values
C) One representative value per partition is sufficient
D) Partitions can overlap for better coverage

Question 3

What is the primary benefit of equivalence partitioning?

A) It guarantees 100% code coverage
B) It reduces test cases whilst maintaining coverage
C) It eliminates the need for boundary testing
D) It only works for numeric inputs

Answers: 1-B (Valid 8–16, Invalid <8, Invalid >16), 2-C, 3-B

Mini Case Study: Airline Baggage System

Scenario

An airline baggage system applies fees based on weight:

- 0–15 kg: Included in ticket
- 16–23 kg: £25 fee
- 24–32 kg: £50 fee
- Above 32 kg: Rejected

Discussion Questions

1. Identify all valid and invalid partitions
2. What test values would you select?
3. What edge cases might cause issues?
4. How would you test negative weights or zero?

Interview Tips

When discussing equivalence partitioning in interviews, always mention both valid and invalid partitions. Interviewers look for testers who consider negative scenarios, not just happy paths.

Be prepared to explain why you chose specific test values. Demonstrate that you understand the principle: one value represents the entire partition's expected behaviour.

Common interview question: "How does equivalence partitioning differ from boundary value analysis?" Answer: EP focuses on classes of data; BVA focuses on edges within those classes. They complement each other.

SECTION 2

Boundary Value Analysis (BVA)

Definition and Purpose

Boundary Value Analysis is a testing technique that focuses on the edges of equivalence partitions. Research shows that defects commonly occur at boundaries rather than within the middle of input ranges. BVA systematically tests minimum, maximum, and just-beyond values.

The technique is based on the observation that programmers often make mistakes in boundary conditions—using < instead of ≤, off-by-one errors, or incorrect range checks. BVA catches these errors efficiently.

When and Why to Use BVA

Numeric Ranges

Age, quantity, price, date ranges—any input with defined minimum and maximum values

String Lengths

Password fields, text inputs, character limits in forms and databases

Array Boundaries

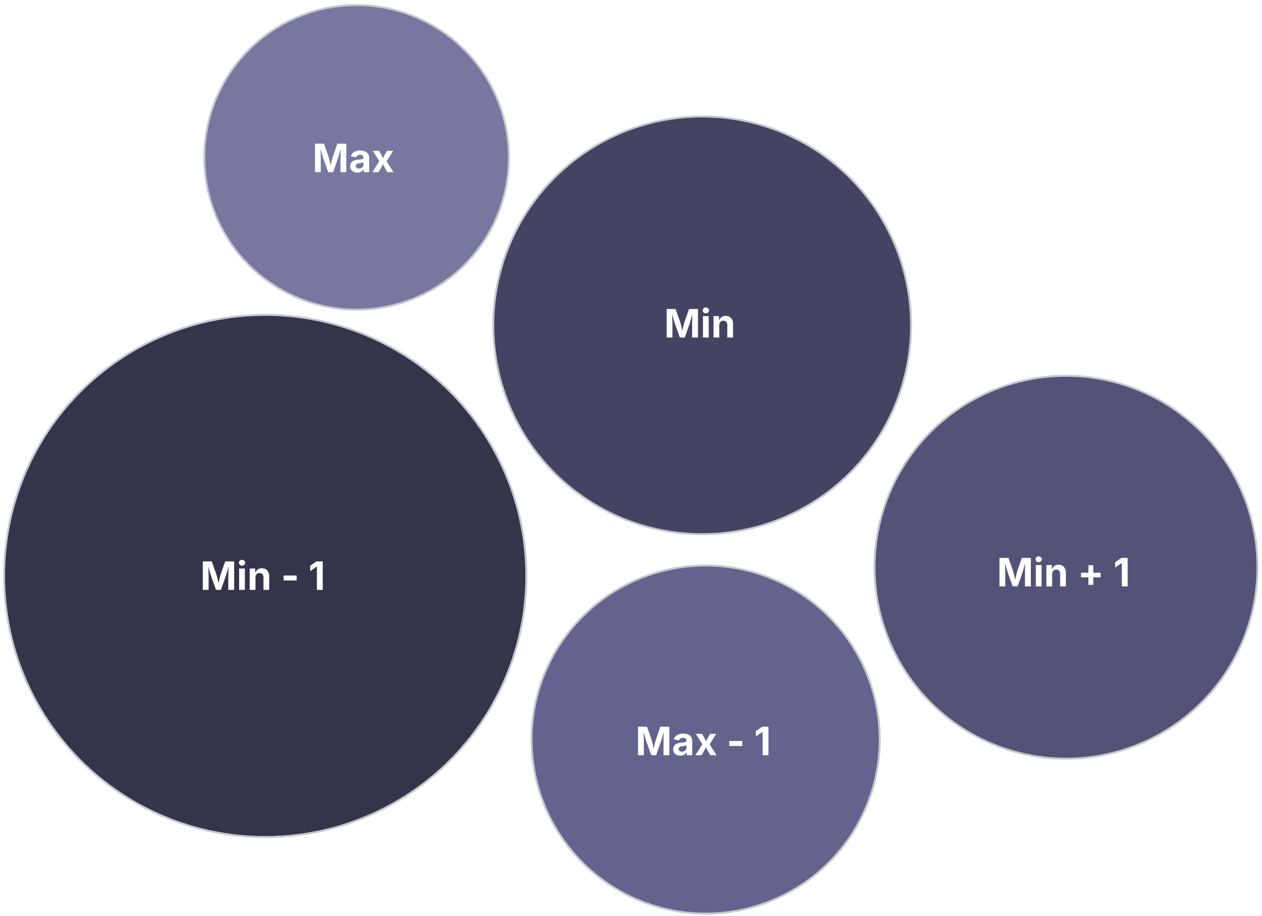
First and last elements, empty collections, maximum list sizes

Critical Values

Points where system behaviour changes, such as discount thresholds or permission levels

BVA is particularly valuable when combined with equivalence partitioning. Use EP to identify partitions, then apply BVA to test their boundaries thoroughly.

BVA Testing Strategy



This systematic approach ensures comprehensive boundary coverage whilst maintaining test efficiency.

Standard BVA Approach

For any boundary, test five critical values:

Test Point	Purpose
Minimum - 1	Just below valid range
Minimum	Lowest valid value
Minimum + 1	Just inside valid range
Maximum - 1	Just inside valid range
Maximum	Highest valid value
Maximum + 1	Just above valid range

For ranges with multiple boundaries, test each boundary independently.

Practical Example 1: Exam Scoring System

An educational platform accepts exam scores from 0 to 100. Let's apply comprehensive BVA.

Boundary Values to Test

Value	Type	Expected Result
-1	Below min	Error: Invalid score
0	Minimum	Accept: Valid score
1	Min + 1	Accept: Valid score
99	Max - 1	Accept: Valid score
100	Maximum	Accept: Valid score
101	Above max	Error: Invalid score

Additional Edge Cases

- Decimal values: 50.5, 99.99
- Null or empty input
- Non-numeric characters: "ABC"
- Very large numbers: 99999
- Negative numbers: -100

Real defect found: The system originally accepted 100.1 due to rounding logic. BVA caught this boundary error before production release.

Practical Example 2: Hotel Booking System

Business Rules

A hotel reservation system has the following constraints:

- Minimum stay: 1 night
- Maximum stay: 30 nights
- Guests per room: 1–4 people
- Booking window: 1–365 days in advance

Multiple boundaries require systematic testing of each constraint independently and in combination.

Boundary	Test Values	Scenario
Nights	0, 1, 2, 29, 30, 31	Duration limits
Guests	0, 1, 2, 3, 4, 5	Occupancy limits
Advance days	0, 1, 2, 364, 365, 366	Booking window

Total test cases: 18 boundary tests across three dimensions. This focused approach is far more efficient than testing hundreds of random combinations.

Common BVA Mistakes

Testing Only Minimums

Neglecting maximum boundaries. Both ends of the range are equally important for finding defects in validation logic.

Forgetting Invalid Boundaries

Testing only valid boundary values (min, max) without testing just-outside values (min-1, max+1). Invalid boundaries often expose critical defects.

Missing Internal Boundaries

Overlooking boundaries between equivalence partitions. If discounts change at £50, test £49, £50, and £51.

Incorrect Off-By-One

Using min+2 or max-2 instead of min+1 or max-1. The adjacent values are crucial for catching boundary implementation errors.

Knowledge Check: Boundary Value Analysis

Question 1

A field accepts values from 10 to 50. Which set includes all critical boundary values?

A) 10, 50
B) 9, 10, 11, 49, 50, 51
C) 10, 30, 50
D) 0, 10, 50, 100

Question 2

When should BVA be applied?

A) Only for numeric inputs
B) Only when equivalence partitioning is not possible
C) For any input with defined limits or ranges
D) Only during regression testing

Question 3

A password must be 8–12 characters. What is the minimum number of BVA test cases?

A) 2 test cases
B) 4 test cases
C) 6 test cases
D) 8 test cases

Answers: 1-B (Complete boundary coverage), 2-C (BVA applies to any bounded input), 3-C (7, 8, 9, 11, 12, 13 characters)

Mini Case Study: Banking Transaction Limits

Scenario

A mobile banking app has daily transaction limits:

- Minimum transfer: £1
- Maximum per transaction: £5,000
- Daily cumulative limit: £10,000
- Account balance: must remain ≥ £0

Discussion Questions

1. Identify all boundaries in this system
2. What BVA test values would you select for each boundary?
3. How would you test the daily cumulative limit?
4. What happens at exactly £0 balance?

Interview Tips

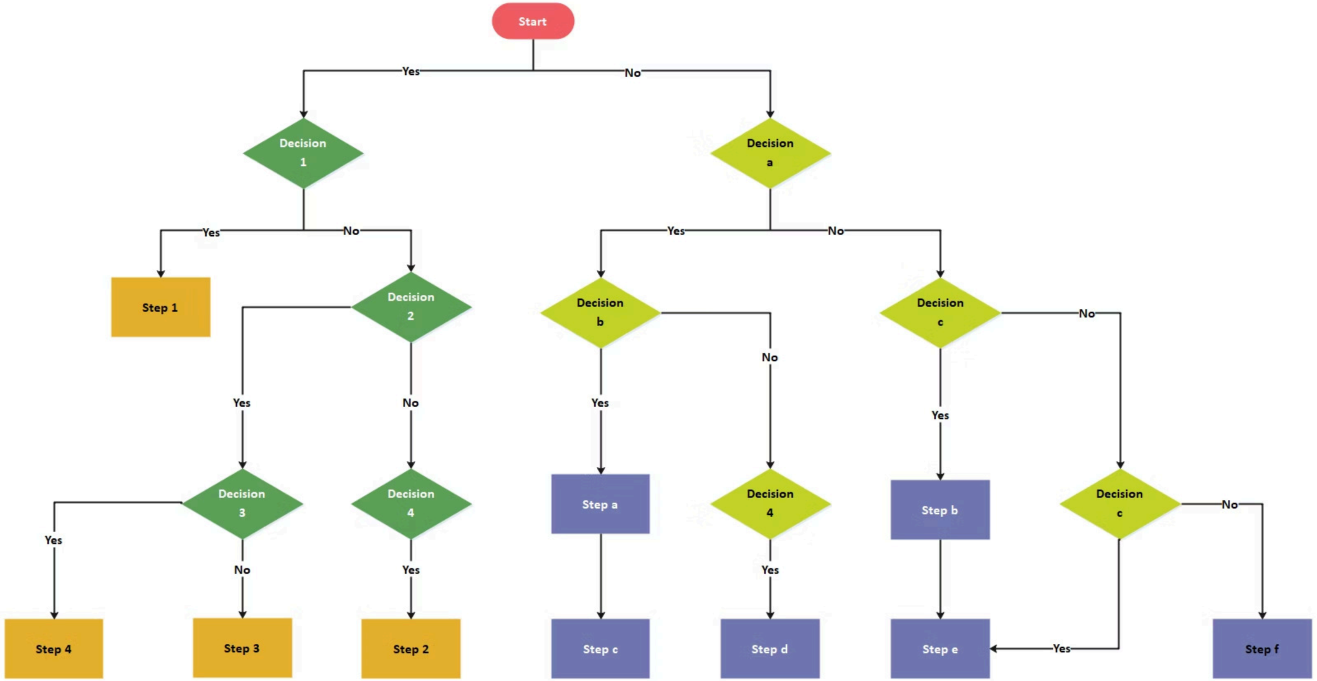
Interviewers often ask: "Why test max+1 if we know it will fail?" Answer: To verify the system handles invalid input gracefully. Testing error handling is as important as testing success scenarios.

Be ready to explain the difference between two-value and three-value BVA. Two-value tests min and max; three-value adds min-1, min+1, max-1, max+1 for comprehensive coverage.

Real-world example: Always mention that BVA helped you find a production issue, such as an off-by-one error or incorrect inequality operator.

SECTION 3

Decision Table Testing

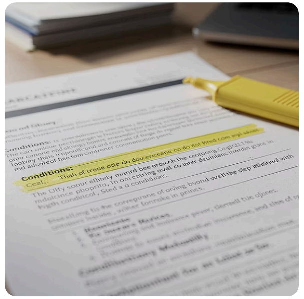


Definition and Purpose

Decision Table Testing is a systematic technique for handling complex business logic involving multiple conditions and their combinations. It creates a matrix showing all possible condition combinations and their corresponding actions or outcomes.

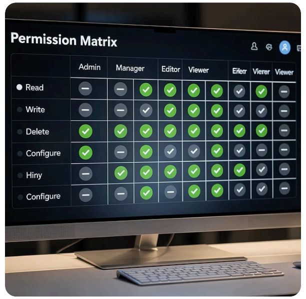
This technique excels when system behaviour depends on several interrelated factors. Decision tables make invisible logic visible, ensuring no combination is overlooked and making test coverage measurable and complete.

When to Use Decision Tables



Complex Business Rules

Insurance eligibility, loan approvals, pricing engines with multiple factors determining outcomes



Multiple Conditions

User permissions based on role, department, and seniority; access control with 3+ variables

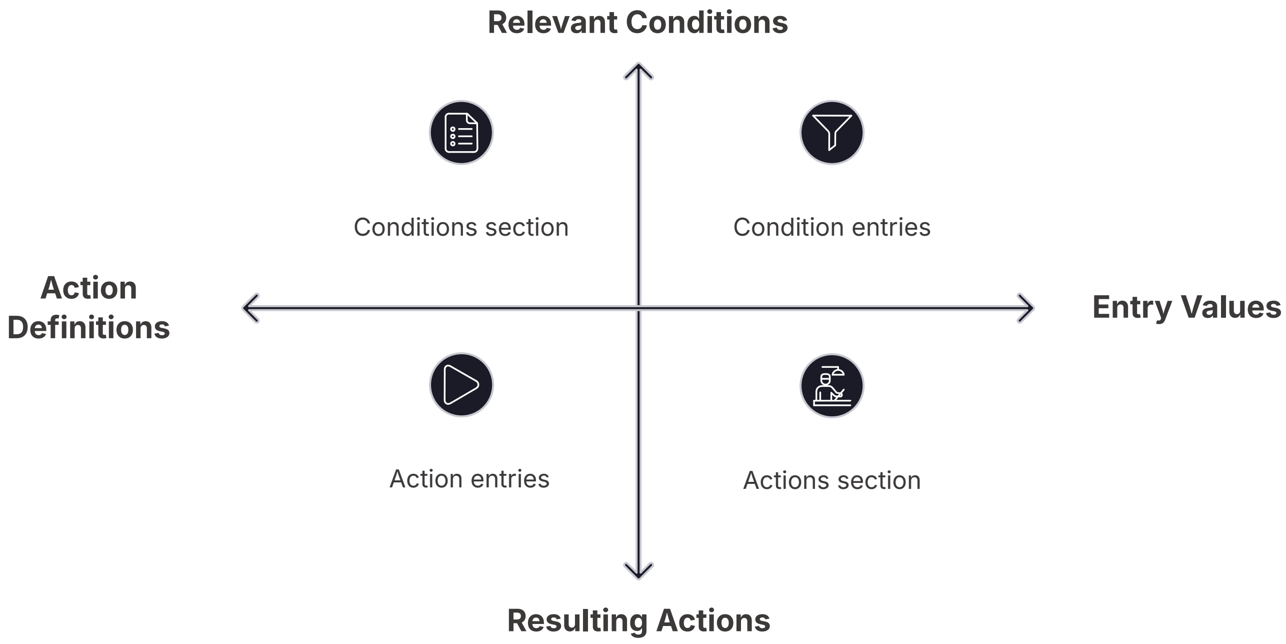
Combinatorial Logic

Promotional discounts based on customer type, order value, and membership status

Validation Rules

Form validation where field requirements change based on selections in other fields

Decision Table Structure



Each column represents one complete test case, making it easy to track coverage and identify missing scenarios.

Component Breakdown

- **Conditions:** All factors that influence the outcome (age, membership, location, etc.)
- **Condition Entries:** True/False or specific values for each condition in each scenario
- **Actions:** All possible system responses or outcomes
- **Action Entries:** Which actions execute for each combination (marked with X or ✓)

The number of columns equals 2^n where n is the number of binary conditions. For 3 conditions: $2^3 = 8$ possible combinations.

Practical Example 1: Online Shopping Discount Logic

An e-commerce site offers discounts based on customer status and order value.

Business Rules

- Premium members get 15% off orders ≥ £100
- Premium members get 10% off orders < £100
- Regular customers get 5% off orders ≥ £100
- Regular customers get 0% off orders < £100

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Premium Member?	Yes	Yes	No	No
Order ≥ £100?	Yes	No	Yes	No
Actions				
Apply 15% discount	✓			
Apply 10% discount		✓		
Apply 5% discount			✓	
No discount				✓

Result: 4 comprehensive test cases covering all combinations. Each rule becomes one test case with specific inputs and expected outcomes.

Practical Example 2: Loan Approval System

A bank's loan approval depends on credit score, employment status, and loan amount.

Conditions	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**
Credit Score ≥ 700	Y	Y	Y	Y	N	N	N	N
Employed Full-Time	Y	Y	N	N	Y	Y	N	N
Loan ≤ £50,000	Y	N	Y	N	Y	N	Y	N
Actions								
Approve Immediately	✓							
Manual Review		✓	✓		✓			
Reject				✓		✓	✓	✓

This table reveals that only one scenario (R1) leads to immediate approval, whilst five scenarios result in rejection. Such insights help stakeholders understand system behaviour and identify potential business logic issues early.

Decision Table Optimisation

Full Decision Table (8 Rules)

Condition	1	2	3	4	5	6	7	8
Age ≥ 18	Y	Y	Y	Y	N	N	N	N
Has ID	Y	Y	N	N	Y	Y	N	N
Has Ticket	Y	N	Y	N	Y	N	Y	N
Allow Entry	✓							

Optimised Table (4 Rules)

Condition	1	2	3	4
Age ≥ 18	Y	Y	N	—
Has ID	Y	N	—	—
Has Ticket	Y	—	—	—
Allow Entry	✓			

The dash (—) means "don't care"—the condition doesn't affect the outcome. If under 18, having ID or ticket is irrelevant.

Optimisation reduces 8 rules to 4 test cases by combining scenarios with identical outcomes. This improves test efficiency without sacrificing coverage.

Common Mistakes in Decision Tables

1

Incomplete Coverage

Missing condition combinations. With 3 binary conditions, you need 8 rules. Systematically enumerate all possibilities before optimising.

2

Contradictory Rules

Same conditions producing different actions. Review for logical conflicts—identical condition entries must have identical action entries.

3

Too Many Conditions

Including irrelevant factors that don't affect outcomes. Keep tables focused—5+ conditions create unwieldy tables with 32+ rules.

4

Premature Optimisation

Collapsing rules before verifying completeness. First create the full table, validate it, then optimise by merging rules with identical outcomes.

Knowledge Check: Decision Tables

Question 1

How many rules are needed for a decision table with 4 binary conditions (before optimisation)?

A) 4 rules
B) 8 rules
C) 12 rules
D) 16 rules

Question 2

What does a dash (—) represent in an optimised decision table?

A) An error in the table
B) The condition is false
C) The condition doesn't matter for this rule
D) The condition needs more testing

Question 3

When is decision table testing most appropriate?

A) For simple linear workflows
B) When multiple conditions affect the outcome
C) Only for financial applications
D) To replace all other testing techniques

Answers: 1-D ($2^4 = 16$), 2-C (Don't care condition), 3-B (Complex combinatorial logic)

Mini Case Study: Travel Insurance Eligibility

Scenario

A travel insurance company determines coverage based on:

- Traveller age (Under 65 / 65+)
- Destination risk (Low / High)
- Trip duration (≤14 days / >14 days)

Outcomes

- Standard coverage
- Premium coverage required
- Not eligible

Discussion Questions

1. Create a complete decision table (8 rules)
2. Can you optimise it? How many rules remain?
3. What test data would you use for each rule?

Interview Tips

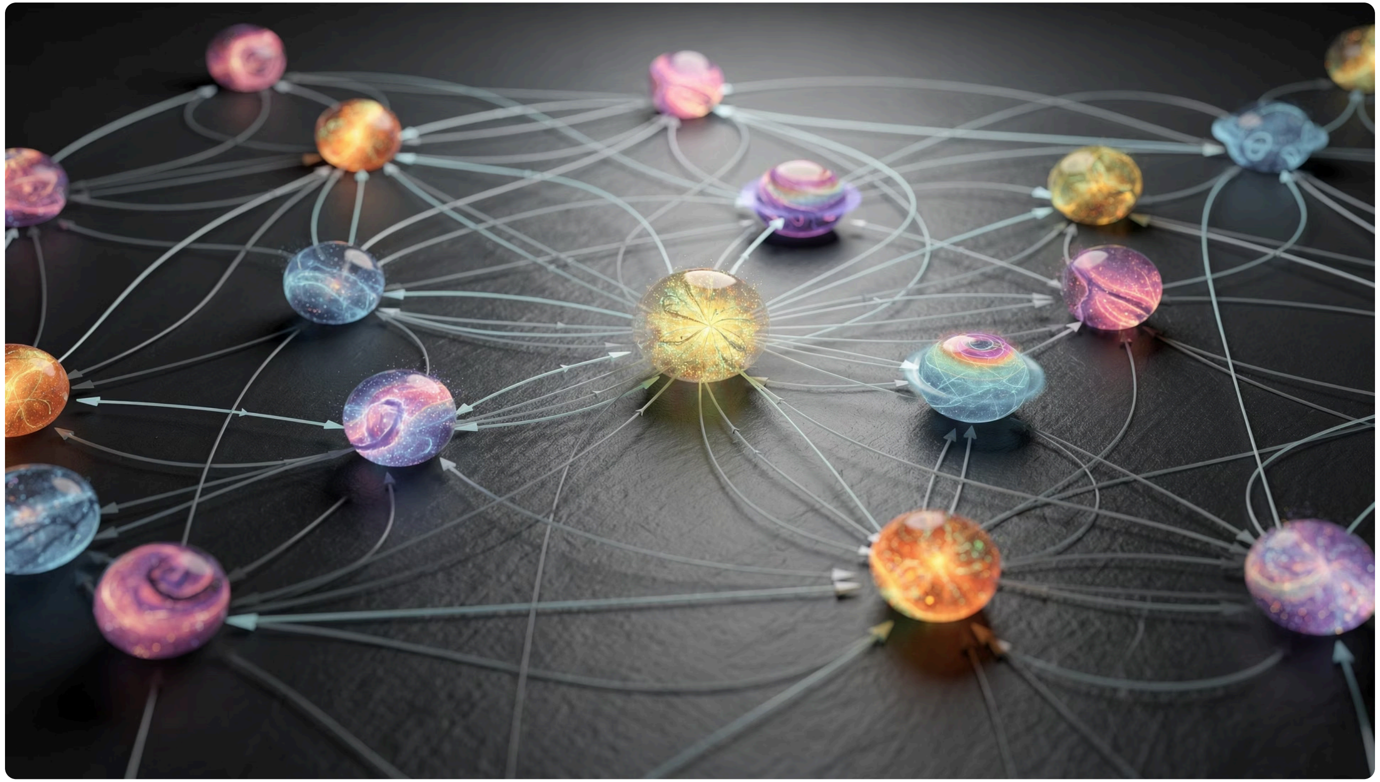
When asked about decision tables, emphasise their value for requirements validation. They often reveal ambiguities, missing scenarios, or contradictory rules before coding begins.

Mention that decision tables serve as both test design tools and documentation. They clearly communicate complex logic to developers, business analysts, and testers.

Be prepared to draw a simple decision table on a whiteboard. Interviewers may give you a scenario and ask you to create the table on the spot.

SECTION 4

State Transition Testing



Definition and Purpose

State Transition Testing verifies that a system moves correctly between different states based on events or inputs. A state represents a condition or situation during an object's lifetime; transitions are the changes between states triggered by specific events.

This technique is essential for systems where current behaviour depends on past actions—order processing, user authentication, workflow engines, and device controls. It ensures all valid transitions work correctly and invalid transitions are properly rejected.

Core Concepts of State Transition

States

Distinct conditions the system can be in. Example: New, In Progress, Completed, Cancelled

Transitions

Movements from one state to another. Example: Submit order → moves from Draft to Submitted

Events

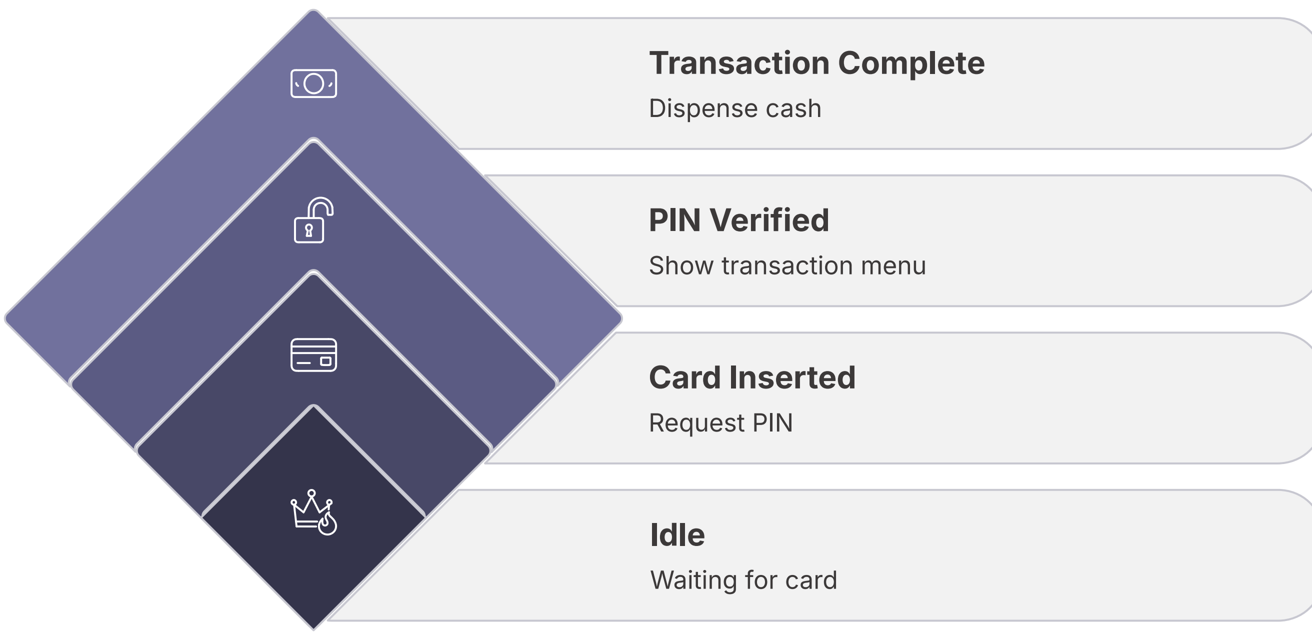
Actions or inputs that trigger transitions. Example: User clicks "Submit", payment succeeds

Guards

Conditions that must be true for a transition to occur. Example: Can only submit if form is valid

State transition testing uses diagrams and tables to visualise and verify all possible state changes, ensuring the system behaves correctly in every scenario.

Practical Example 1: ATM Cash Withdrawal



State Transition Table

Current State	Event	Next State	Action
Idle	Insert Card	Card Inserted	Request PIN
Card Inserted	Correct PIN	PIN Verified	Show menu
Card Inserted	Wrong PIN (3x)	Idle	Retain card
PIN Verified	Select Withdrawal	Processing	Check balance
Processing	Sufficient Funds	Dispensing	Dispense cash
Dispensing	Cash Taken	Idle	Eject card

Test all valid transitions and verify invalid ones are rejected (e.g., can't withdraw from Idle state).

Practical Example 2: Order Management System

An e-commerce order progresses through multiple states from creation to delivery.

States and Valid Transitions

- **Draft:** Can → Submit or Delete
- **Submitted:** Can → Confirm or Cancel
- **Confirmed:** Can → Ship or Cancel
- **Shipped:** Can → Deliver or Return
- **Delivered:** Can → Return (within 30 days)
- **Cancelled:** Terminal state
- **Returned:** Terminal state

Invalid Transitions to Test

Invalid Transition	Expected Behaviour
Draft → Shipped	Error: Must submit first
Delivered → Draft	Error: Cannot reset
Cancelled → Shipped	Error: Order cancelled
Shipped → Draft	Error: Cannot go backwards

Testing invalid transitions is crucial—they often reveal security issues, data corruption risks, or poor error handling.

State Transition Test Coverage

Coverage Levels

0-Switch Coverage: Visit every state at least once. Minimum viable coverage—confirms all states are reachable.

1-Switch Coverage: Execute every valid transition at least once. Standard approach—ensures each state change works correctly.

2-Switch Coverage: Test all pairs of transitions. Comprehensive approach—verifies behaviour after sequences of state changes (A→B→C).

Most projects use 1-switch coverage as the baseline, applying 2-switch coverage to critical workflows. Testing invalid transitions adds another dimension of coverage.

Common Mistakes in State Transition Testing

1. **Ignoring Invalid Transitions**

   Only testing valid paths. Invalid transitions often expose critical defects in validation and error handling.

2. **Missing Guard Conditions**

   Forgetting prerequisites for transitions. Example: Can only ship if payment is confirmed—test both scenarios.

3. **Incomplete State Coverage**

   Not testing all states or all ways to reach a state. Some states may have multiple entry paths requiring separate tests.

4. **Overlooking Terminal States**

   Failing to verify that terminal states (Cancelled, Completed) truly prevent further transitions.

Knowledge Check: State Transition Testing

Question 1

What is the primary focus of state transition testing?

A) Testing data boundaries
B) Verifying state changes based on events
C) Testing user interface layouts
D) Measuring code coverage

Question 2

A system has 4 states and 8 valid transitions. What is the minimum number of test cases for 1-switch coverage?

A) 4 test cases
B) 8 test cases
C) 12 test cases
D) 16 test cases

Question 3

Why should you test invalid state transitions?

A) They are not important
B) To verify the system rejects them appropriately
C) To increase the number of test cases
D) Only if time permits

Answers: 1-B (Core purpose is verifying state changes), 2-B (One test per transition), 3-B (Invalid transitions reveal critical defects)

Mini Case Study: Document Approval Workflow

Scenario

A document approval system has these states:

- Draft → Author can edit
- Submitted → Awaiting review
- Under Review → Reviewer assigned
- Approved → Final state
- Rejected → Returns to Draft

Discussion Questions

1. Draw the state transition diagram
2. List all valid transitions with their triggering events
3. Identify 5 invalid transitions to test
4. What guard conditions might exist? (e.g., can only submit if complete)

Interview Tips

State transition questions often appear in interviews. Be ready to sketch a state diagram quickly and explain your approach to achieving full transition coverage.

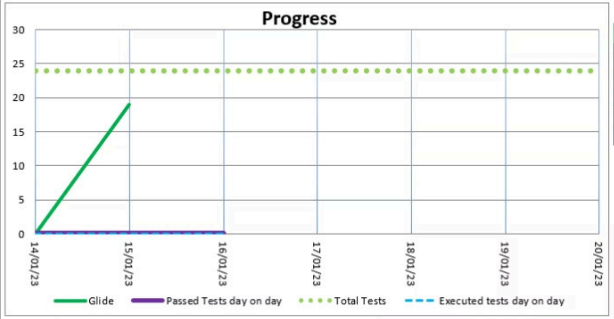
Emphasise that state transition testing finds defects other techniques miss—particularly issues with workflow logic, concurrent access, and state persistence across sessions.

Mention real examples: "In my previous project, state transition testing revealed that users could delete records in 'Processing' state, causing data corruption."

SECTION 5

Test Data Preparation

Total Tests	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday
24	14/01/23	15/01/23	16/01/23	17/01/23	18/01/23	19/01/23	20/01/23	21/01/23	22/01/23	23/01/23	24/01/23	25/01/23	26/01/23	27/01/23
Tests per FTE	0	19	15	15	15	15	0	0	15	15	15	15	15	0
Number of FTE	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Baseline tests per hour	0	19	0	0	0	0	0	0	0	0	0	0	0	0
Glide	0	19	19	19	19	19	19	19	19	19	19	19	19	19
Executed tests Main	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Executed tests Regression	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Executed tests Extra	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Executed tests	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Daily passed Main	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Daily passed Regresstion	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Daily passed Extra	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Daily passed	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Passed Tests day on day	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Executed tests day on day	0	0	0	0	0	0	0	0	0	0	0	0	0	0
% of tests completed	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%



	Totals
Phase 1	24
Pass	0
Fail	0
Not Run	24
In Progress	0
Blocked	0
% Completed	0.00%

% Of Tests Attempted	0.00%
% Of Tests Completed	0.00%

Phase 1	Total Number of Issues	Closed	New	In Progress	Issues Resolved
0.00%	0	0	0	0	

Definition and Purpose

Test Data Preparation involves creating, managing, and maintaining datasets used during testing. High-quality test data is essential for executing effective tests—poor data leads to missed defects, unreliable results, and wasted effort.

Test data must be representative of production scenarios whilst covering edge cases, valid and invalid inputs, and diverse user profiles. It should be consistent, reusable, and protect sensitive information through masking or anonymisation.

Test Data Strategies

Production Data Cloning

Copy sanitised production data to test environments. Pros: Realistic scenarios. Cons: Requires data masking for privacy, large storage needs.

Synthetic Data Generation

Create artificial data using tools or scripts. Pros: Privacy-safe, customisable. Cons: May miss real-world edge cases.

Manual Creation

Handcraft specific test scenarios. Pros: Precise control over edge cases. Cons: Time-consuming, not scalable.

Data Subsetting

Extract representative samples from production. Pros: Manageable size, realistic. Cons: Must maintain referential integrity.

Test Data Categories and Examples

Positive Test Data

Valid inputs that should be accepted:

- Email: user@example.com
- UK postcode: SW1A 1AA
- Phone: 07912345678
- Date: 15/03/2024

Negative Test Data

Invalid inputs that should be rejected:

- Email: user@, @example.com, user
- Postcode: 12345, ABCDEF
- Phone: 123, 0791234567890
- Date: 32/13/2024, 00/00/0000

Boundary Test Data

Values at limits:

- Age field (18-65): 17, 18, 19, 64, 65, 66
- Text field (max 50 chars): 49, 50, 51 characters

Special Characters

Edge cases often overlooked:

- Names with apostrophes: O'Brien
- Unicode: Müller, 北京
- SQL injection attempts: ' OR '1'='1
- Null, empty strings, whitespace

Test Data Best Practices

- [ ] **Maintain Data Independence**

  Tests should not depend on specific data being present or in a particular state. Create or set up required data as part of test setup, clean up afterwards.

- [ ] **Use Data-Driven Testing**

  Separate test logic from test data. Store data in external files (CSV, JSON, Excel) and iterate through datasets. One test script can execute dozens of scenarios.

- [ ] **Version Control Test Data**

  Track test data changes alongside code. Document what each dataset represents. Enables reproducing historical test results and understanding failures.

- [ ] **Protect Sensitive Information**

  Never use real customer data without anonymisation. Apply data masking: replace names, addresses, credit cards with realistic but fake values. Comply with GDPR and data protection regulations.

- [ ] **Automate Data Preparation**

  Write scripts to generate or restore test data. Reduces manual effort, ensures consistency, enables parallel test execution by creating isolated datasets per test run.

Technique Comparison Matrix

Technique	Best For	Complexity	Coverage Type	Effort	Defect Detection
Equivalence Partitioning	Input validation	Low	Input classes	Low	Medium
Boundary Value Analysis	Numeric ranges	Low	Edge values	Low	High
Decision Tables	Complex rules	Medium	Combinations	Medium	High
State Transition	Workflows	Medium-High	State changes	Medium-High	High
Test Data Prep	All testing	Varies	Foundational	High	N/A (Enabler)

These techniques are complementary, not mutually exclusive. Effective test design combines multiple approaches—use EP to identify partitions, BVA for their boundaries, decision tables for complex logic, and state transition for workflows. Quality test data underpins them all.