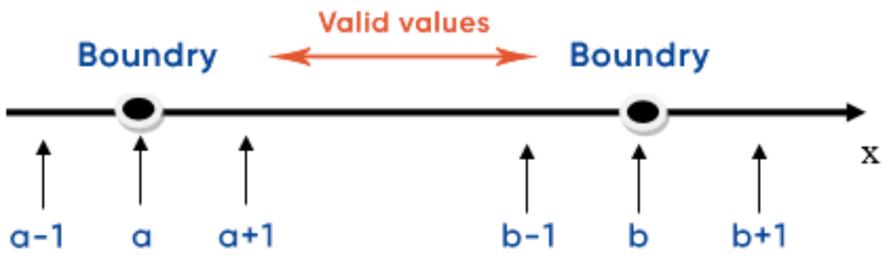
Important Test Case Design Techniques

1. Boundary Value Analysis (BVA)

Boundary value analysis is a black-box testing technique to test the boundaries between partitions instead of testing multiple values in the equivalence region. This is because we often find a large number of errors at the boundaries rather than the center of the defined input values, and we suppose that if it is true for boundary values, it is true for the whole equivalence region. Also, BVA is considered an additional test case design type for equivalence classification.

Guide to Boundary Value Analysis design: Select input variable values at their minimum/maximum, just below the minimum/maximum, just above the minimum/maximum, a nominal value (optional).



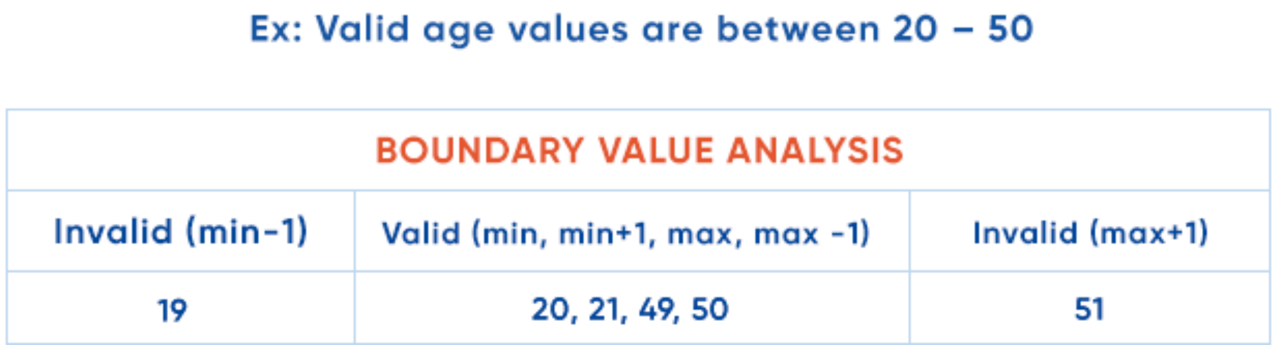
Boundary Value Analysis (BVA) test case design technique

Example: Valid age values are between 20 – 50.

- Minimum boundary value is 20
- Maximum boundary value is 50
- Take: 19, 20, 21, 49, 50, 51
- Valid inputs: 20, 21, 49, 50
- Invalid inputs: 19, 51

So, test cases will look like:

- Case 1: Enter the value 19: Invalid
- Case 2: Enter number 20: Valid
- Case 3: Enter number 50: Valid
- Case 4: Enter number 51: Invalid


Ex: Valid age values are between 20 – 50

BOUNDARY VALUE ANALYSIS		
Invalid (min-1)	Valid (min, min+1, max, max -1)	Invalid (max+1)
19	20, 21, 49, 50	51

Boundary Value Analysis test case design example

2. Equivalence Class Partitioning

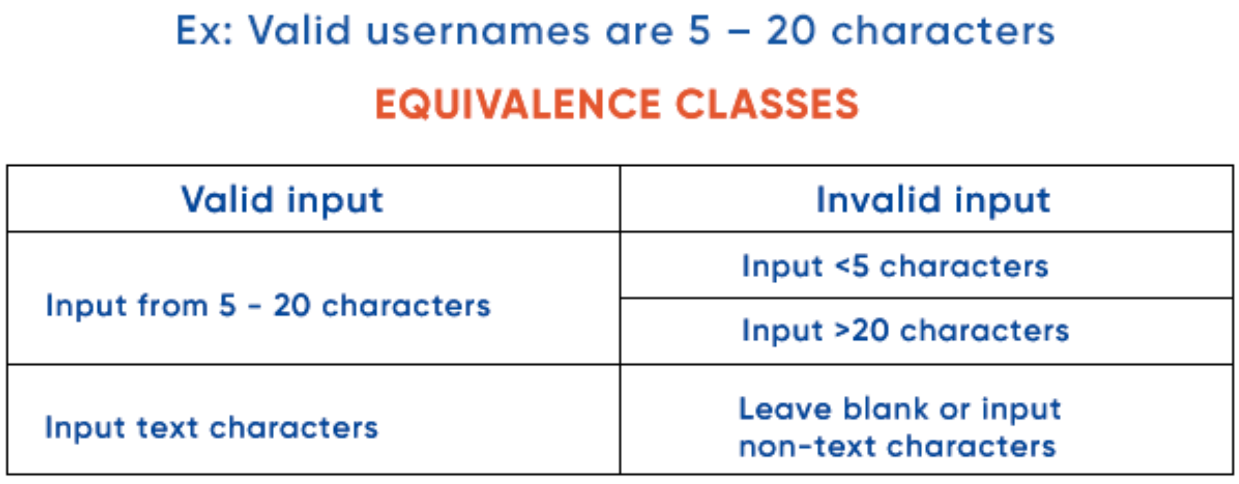
Equivalent Class Partitioning (or Equivalent Partitioning) is a test case design method that divides the input domain data into various equivalence data classes, assuming that data in each group behaves the same. From that, we will design test cases for representative values of each class that can stand for the result of the whole class.

The concept behind the Equivalent Partitioning testing technique is that the test case of a typical value is equal to the tests of the rest values in the same group. Hence, it helps reduce the number of test cases designed and executed.

Steps to design Equivalent Partitioning test case:

- Define the equivalence classes
- Define the test cases for each class

Example: Valid usernames are within 5 – 20 text-only characters.


Ex: Valid usernames are 5 – 20 characters

EQUIVALENCE CLASSES

Valid input	Invalid input
Input from 5 - 20 characters	Input <5 characters
	Input >20 characters
Input text characters	Leave blank or input non-text characters

Equivalence Class Partitioning test cases design example

So, test cases will look like:

- Case 1: Enter within 5 – 20 text characters: Pass
- Case 2: Input <3 characters: Display error message "Username must be from 5 – 20 characters"
- Case 3: Enter >20 characters: Display error message "Username must be from 5 – 20 characters"
- Case 4: Leave blank or no-text characters: Display error message "Invalid username"

3. Decision Table

Decision Table is a software testing technique based on cause-effect relationships, also called a cause-effect table, used to test system behavior in which the output depends on a large combination of input. For instance, navigate a user to the homepage if all blanks/specific blanks in the log-in section are filled in.

First and foremost, you need to identify the functionalities where the output responds to different input combinations. Then, for each function, divide the input set into possible smaller subsets that correspond to various outputs.

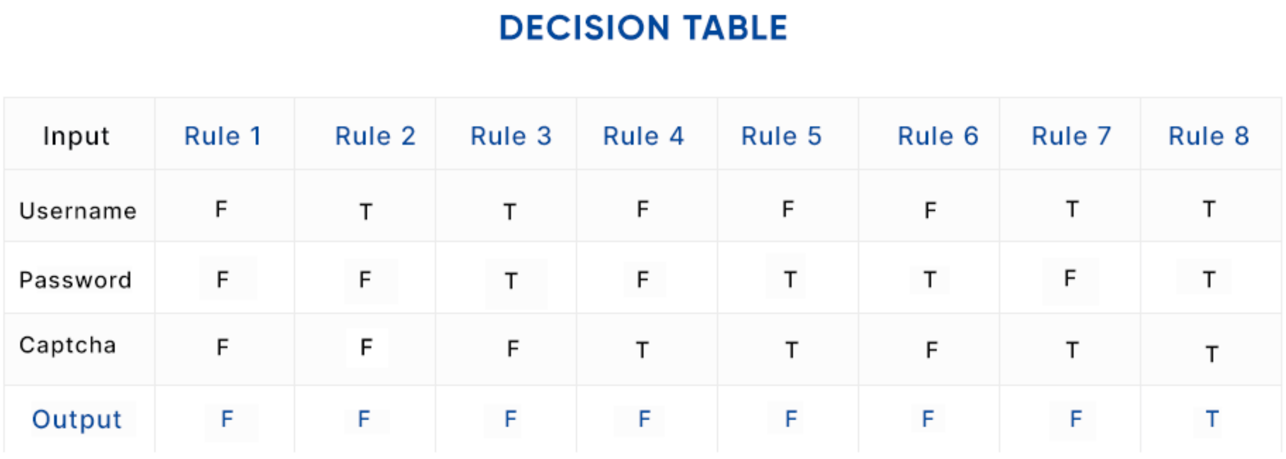
For every function we will create a decision table. A table consists of 3 main parts:

- A list of all possible input combinations
- A list of corresponding system behaviour (output)
- T (True) and F (False) stand for the correctness of input conditions.

Example:

- Function: A user will be navigated to the homepage if successfully log in.
- Conditions for success log in: correct username, password, captcha.
- In the Input section: T & F stands for the correctness of input information.
- In the Output section: T stands for the result when the homepage is displayed, F stands for the result when an error message is shown.

Look at the image below for more details.



DECISION TABLE

Input	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Username	F	T	T	F	F	F	T	T
Password	F	F	T	F	T	T	F	T
Captcha	F	F	F	T	T	F	T	T
Output	F	F	F	F	F	F	F	T

Decision table test cases design example

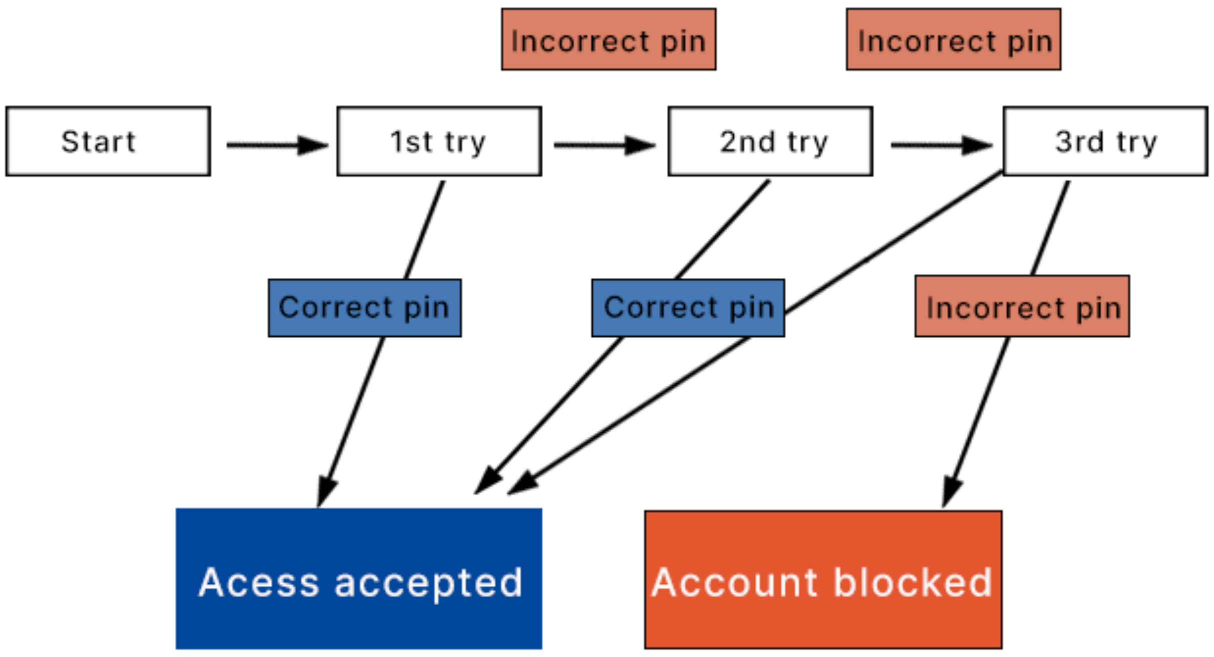
So, test cases will look like:

- Enter correct username, password, captcha: Pass
- Enter wrong username, password, captcha: Display error message.
- Enter correct username, wrong password and captcha: Display error message.
- Enter correct username, password and wrong captcha: Display error message.

4. State Transition

State Transition is another way to design test cases in black-box testing, in which changes in the input make changes to the state of the system and trigger different outputs. In this technique, testers execute valid and invalid cases belonging to a sequence of events to evaluate the system behavior.

For example: When a user logs into an e-banking app on his mobile phone, if he enters the wrong password 3 times in a row, his account will be blocked. That means he will be able to log in if he enters the correct password at the 1st, 2nd, 3rd try, and the state will be transitioned into Access Accepted. Look at the diagram in the image below.



State transition diagram example – Test cases design techniques

The State Transition technique is often used to test the functions of the Application Under Test (AUT) when the change to the input makes up changes in the state of the system and produces distinct outputs.

5. Error Guessing

In the Error Guessing testing technique, test cases are designed mostly based on experiences of the test analysts. He/she will try to guess and assume the possible errors or error-prone situations which can prevail in the code; hence the test designers must be skilled and experienced testers.

In Error Guessing, the test cases could be based on:

- Previous experience of testing related/similar software products.
- Understanding of the system to be tested.
- Knowledge of common errors in such applications.
- Prioritised functions in the requirement specification documents (to not miss them).