UNIT 2: Black Box Testing

Introduction, need of black box testing, Requirements Analysis, Testing Methods - Requirements based testing, Positive and negative testing, Boundary value analysis, Equivalence Partitioning class, Domain testing, Design of test cases, Case studies of Black- Box testing.

Software Testing

The aim of program testing is to help realiseidentify all defects in a program. However, in practice, even after satisfactory completion of the testing phase, it is not possible to guarantee that a program is error free. This is because the input data domain of most programs is very large, and it is not practical to test the program exhaustively with respect to each value that the input can assume. Consider a functiontaking a floating point number as argument. If a tester takes 1sec to type in a value, then even a million testers would not be able to exhaustively test it after trying for a million number of years. Even with this obvious limitation of the testing process, we should not underestimate the importance of testing. We must remember that careful testing can expose a large percentage of the defects existing in a program, and therefore provides a practical way of reducing defects in a system.

There are essentially two main approaches to systematically design test cases:

Black-box approach

White-box (or glass-box) approach

In the black-box approach, test cases are designed using only the functional specification of the software. That is, test cases are designed solely based on an analysis of the input/out behaviour (that is, functional behaviour) and does not require any knowledge of the internal structure of a program. For this reason, black-box testing is also known as functional testing. On the other hand, designing white-box test cases requires a thorough knowledge of the internal structure of a program, and therefore white-box testing is also called structural testing. Black- box test cases are designed solely based on the input-output behaviour of a program. In contrast, white-box test cases are based on an analysis of the code. These two approaches to test case design are complementary. That is, a program has to be tested using the test cases designed by both the approaches, and one testing using one approach does not substitute testing using the other.

BLACK-BOX TESTING

In black-box testing, test cases are designed from an examination of the input/output values only and no knowledge of design or code is required.

What is Black Box Testing?

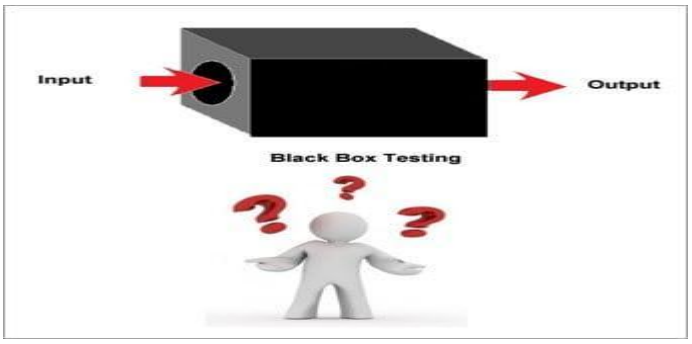
Black Box Testing is also known as behavioral, opaque-box, closed-box, specification-based or eye-to-eye testing.

It is a Software Testing method that analyzes the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value.

The main focus of Black Box Testing is on the functionality of the system as a whole. The term **'Behavioral Testing'** is also used for Black Box Testing.

Behavioral test design is slightly different from the black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged. Each testing method has its own advantages and disadvantages. There are some bugs that cannot be found using black box or white box technique alone.

A majority of the applications are tested using the Black Box method. We need to cover the majority of test cases so that most of the bugs will get discovered by the Black-Box method.
This testing occurs throughout the Software Development and Testing Life Cycle i.e in Unit, Integration, System, Acceptance, and Regression Testing stages.
This can be either Functional or Non-Functional.



Black Box Testing Techniques/Test Case Design Techniques

In order to systematically test a set of functions, it is necessary to design test cases. Testers can create test cases from the requirement specification document using the following Black Box Testing techniques:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing
- Error Guessing
- Graph-Based Testing Methods

Equivalence Class Partitioning

In the equivalence class partitioning approach, the domain of input values to the program under test is partitioned into a set of equivalence classes. The partitioning is done such that for every input data belonging to the same equivalence class, the program behaves similarly. The main idea behind defining equivalence classes of input data is that testing the code with any one value belonging to an equivalence class is as good as testing the code with any other value belonging to the same equivalence class.
Equivalence classes for a unit under test can be designed by examining the input data and output data. The following are two general guidelines for designing the equivalence classes:
If the input data values to a system can be specified by a range of values, then one valid and two invalid equivalence classes need to be defined.
For example, if the equivalence class is the set of integers in
the range 1 to 10 (i.e., [1,10]), then the invalid equivalence classes are $[-\infty,0]$, $[11,+\infty]$.
If the input data assumes values from a set of discrete members of some domain, then one equivalence class for the valid input values and another equivalence class for the invalid input values should be defined.
For example, if the valid equivalence classes are {A,B,C}, then the invalid equivalence class is -{A,B,C}universe of possible input values.

Examples of equivalence class partitioning-based test case generation .

Equivalence Partitioning

This technique is also known as Equivalence Class Partitioning (ECP). In this technique, input values to the system or application are divided into different classes or groups based on its similarity in the outcome.

Hence, instead of using each and every input value, we can now use any one value from the group/class to test the outcome. This way, we can maintain test coverage while we can reduce the amount of rework and most importantly the time spent.

For Example:

As present in the above image, the "AGE" text field accepts only numbers from 18 to 60. There will be three sets of classes or groups.

Two invalid classes will be:

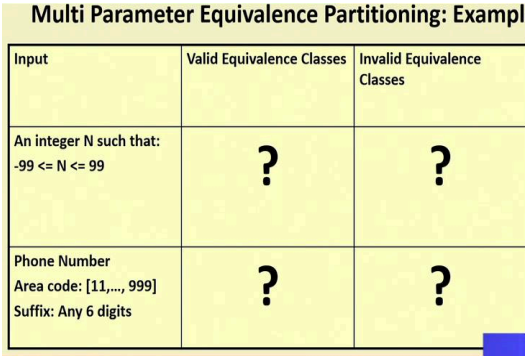
a) Less than or equal to 17.

b) Greater than or equal to 61.

A valid class will be anything between 18 and 60.

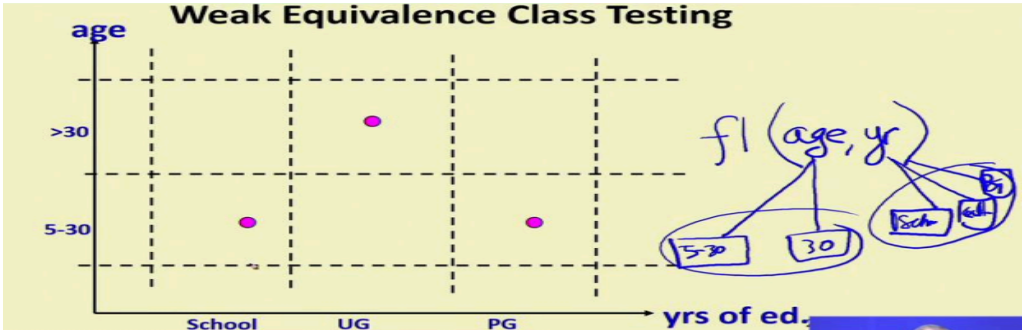
We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities. So, testing with any one value from each set of the class is sufficient to test the above scenario.

Multi Parameter Function



The input to a certain function is 2 parameters. One is an integer value such that the integer lies between -99 to +99. It also takes another parameter, which is a phone number; the phone number contains an area code of any 6-digit number. We know that we have to define valid equivalence class and invalid equivalence class. So, for each of these, let's try to first define each of these 2 parameters, what the valid and invalid set of equivalence classes are and then based on that we will try to answer that how to consider the combinations of these two different parameters

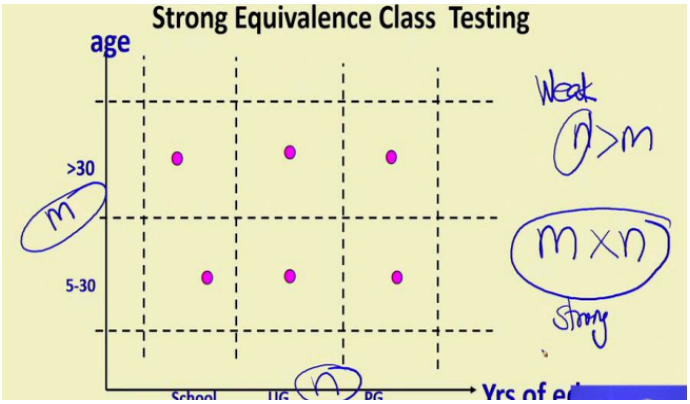
Weak Equivalance Class Testing



One way is to combine two different parameters called as weak equivalence class testing. Let's say that a certain function takes 2 parameters, one is a call that function as f1 or something and it takes 2 parameters, one is age and the other is years of education. Age is an integer and for age we have identified the equivalence classes 5 to 30, greater than 30, number of years of education based on that we have identified school, education, college, under graduate college or post graduate. So, the first parameter has 2 equivalence classes, the second has 3 equivalence classes.
We plot these 2 parameters here in the above diagram. We try to cover all of these that is our test cases should be able to test both the age equivalence classes, 1 value form the age equivalence class and also 1 value of the year of education. In that case using just 3 test cases will be able to cover both the valid age equivalence and the valid year of education. Similarly, if there are 3 or 4 parameters, we can try to cover all of them and that we call them as the weak equivalence testing.

no of test cases=max(n,m)
n=equivalence classes for the first parameter
m=equivalence classes of the second parameter

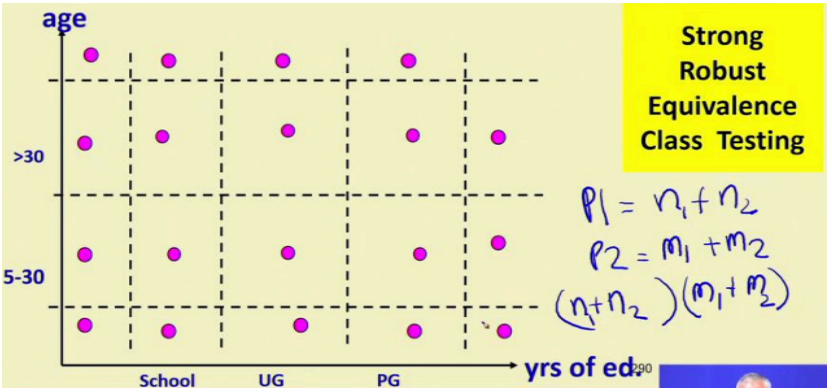
Strong Equivalence Testing



We can also define a strong equivalence class testing here obviously, the number of test cases is much more. Here, not only that we want to cover all values of the 2 parameters, but also we ensure all possible combinations of the 2 parameters. So, 5-30 is combined with all the 3 of this, 3 valid of the years of education similarly greater than 30 is combined with all the 3 or if we look at another way. For the second parameter year of education, let's say school is combined with both the valid age, UG is combined with both the valid age equivalence classes and so on.
So, the number of test cases required here, if the first parameter has m number of equivalence classes, and the second has n number of equivalence classes. Obviously, we need m*n number of test cases to achieve strong equivalence class testing. But what about weak equivalence class testing, if a 2 parameters with m and n and let's say n is greater than m, what is the number of test cases that we need ? We may just think about it, but then that is equal to n. So, this is the number of test cases required

for weak equivalence class testing and this is the number of test cases required for strong equivalence class testing

Strong Robust Testing



But, we might also like to consider the invalid values for both the parameters, then we call it as robust equivalence class testing. Here the number of test cases required is total number of valid equivalence classes for 1 parameter plus the number of invalid equivalence classes. Similarly, the number of valid and invalid equivalence class for the other parameter. Let's say the parameter 1 there are n1 valid equivalence classes and n2 invalid equivalence classes and for parameter 2, we have m1 invalid equivalence classes, and m2 valid equivalence classes. Then for robust equivalence class testing we need (n1+n2)*(m1+m2) number of test cases and obviously, the robust equivalence class testing will expose many more bugs than a weak testing or just strong testing.

Example 1 For a software that computes the square root of an input integer that can assume values in the range of 0 and 5000. Determine the equivalence classes and the black box test suite.

Answer: There are three equivalence classes—The set of negative integers, the set of integers in the range of 0 and 5000, and the set of integers larger than 5000. Therefore, the test cases must include representatives for each of the three equivalence classes. A possible test suite can be:

{–5,500,6000}.

Example 2 Design the equivalence class test cases for a program that reads two integer pairs (m1, c1) and (m2, c2) defining two straight lines of the form y=mx+c. The program computes the intersection point of the two straight lines and displays the point of intersection.

Answer: The equivalence classes are the following:

• Parallel lines (m1 = m2, c1 ≠c2)

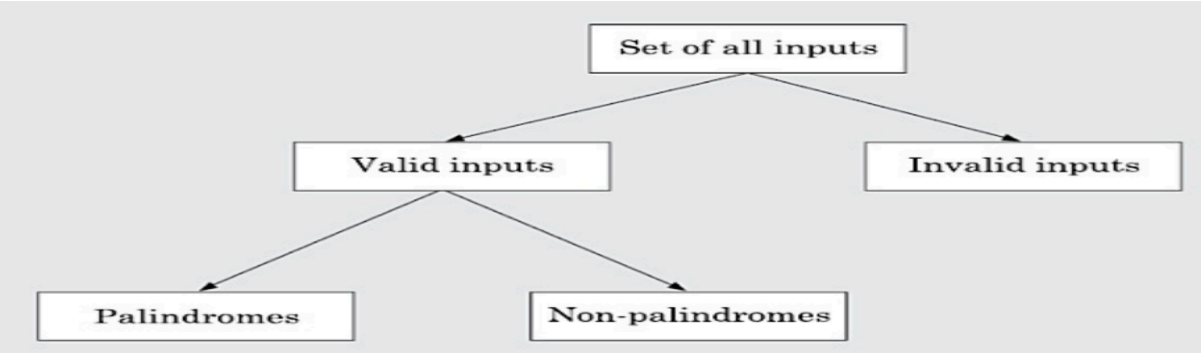
• Intersecting lines (m1 ≠m2)

• Coincident lines (m1 = m2, c1 = c2)

Now, selecting one representative value from each equivalence class, we get the required equivalence class test suite {(2,2)(2,5),(5,5)(7,7), (10,10) (10,10)}

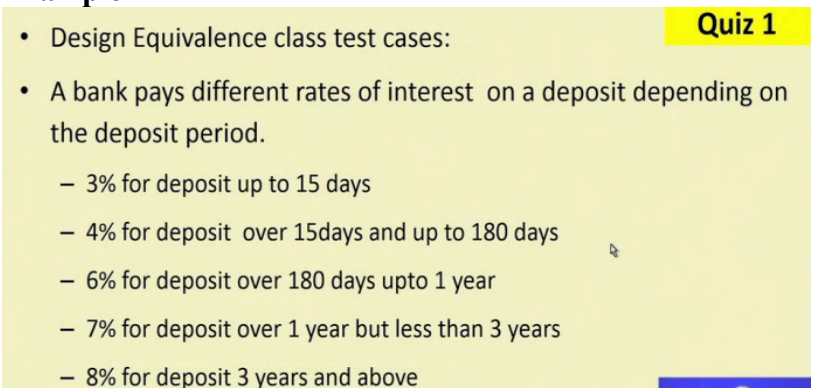
Example 3 Design equivalence class partitioning test suite for a function that reads a character string of size less than five characters and displays whether it is a palindrome.

Answer:



The equivalence classes are the leaf level classes shown in Figure . The equivalence classes are palindromes, non-palindromes, and invalid inputs. Now, selecting one representative value from each equivalence class, we have the required test suite: {abc,aba,abcdef}

Example



we have a function that computes the interest for a certain principal and there are different rates of interest on a deposit depending on the deposit period. So, wehave this function computeInterest, it takes deposit amount and a period as parameter.

We know that the interest computed depends on the period, it should be 3 percent for deposit up to 15 days, 4 percent for 15 days to 180 days, 6 percent for 180 days to 1 year, 7 percent for deposit over 1 year, but less than 3 years and 8 percent for deposit above 3 years. Now, observe here that even though there are 2 parameters, the behavior of the function does not matter much and the amount. So, amount there is only 1 equivalence class, 1 valid equivalence class and then we have invalid equivalence class. Whereas, the period there are 1, 2, 3, 4, 5 so, there are 5 equivalence classes for the period, because depending on the period, it provides different behavior that is different interest rate.

Now, the number of test cases required here is for strong equivalence testing is equal to 1, 2, 3, 4, 5. For weak equivalence class testing, it is also similar because the first one is one.

So, both weak and strong equivalence class testing will require 5 test cases, but for robust testing we have 1 valid and 1 invalid for this so, there are 2 classes for the first parameter. The second parameter has 1 invalid class. So, we have 6 here. So, the number of test cases becomes 12 for robust testing.

Boundary Value Analysis

A type of programming error that is frequently committed by programmers is missing out on the special consideration that should be given to the values at the boundaries of different equivalence classes of inputs. The reason behind programmers committing such errors might purely be due to psychological factors. Programmers often fail to properly address the special processing required by the input values that lie at the

boundary of the different equivalence classes. For example, programmers may improperly use < instead of <=, or conversely <= for <, etc
Boundary value analysis-based test suite design involves designing test cases using the values at the boundaries of different equivalence classes.
To design boundary value test cases, it is required to examine the equivalence classes to check if any of the equivalence classes contains a range of values. For those equivalence classes that are not a range of values
(i.e., consist of a discrete collection of values) no boundary value test cases can be defined. For an equivalence class that is a range of values, the boundary values need to be included in the test suite. For example, if an equivalence class contains the integers in the range 1 to 10, then the boundary value test suite is {0,1,10,11}.
Example 1. For a function that computes the square root of the integer values in the range of 0 and 5000, determine the boundary value test suite.
Answer: There are three equivalence classes—The set of negative integers,the set of integers in the range of 0 and 5000, and the set of integers larger than 5000. The boundary value-based test suite is: {0,-1,5000,5001}.

Important steps in the black-box test suite design approach:
Examine the input and output values of the program.
Identify the equivalence classes.
Design equivalence class test cases by picking one representative value from each equivalence class.
Design the boundary value test cases as follows. Examine if any equivalence class is a range of values. Include the values at the boundaries of such equivalence classes in the test suite.
The strategy for black-box testing is intuitive and simple. For black-box testing, the most important step is the identification of the equivalence classes.
Error Guessing Technique

Error Guessing is a Software Testing technique on guessing the error which can prevail in the code.
It is an experience-based testing technique where the Test Analyst uses his/her experience to guess the problematic areas of the application. This technique n
The test cases for finding issues in the software are written based on the prior testing experience with similar applications. So, the scope of the test cases would generally depend upon the kind of testing Test Analyst was involved in the past. The Error Guessing technique does not follow any specific rules.
For Example, if the Analyst guesses that the login page is error-prone, then the testers will write detailed test cases concentrating on the login page. Testers can think of a variety of combinations of data to test the login page.
To design test cases based on the Error Guessing technique, the Analyst can use past experiences to identify the conditions.
This technique can be used at any level of testing and for testing the common mistakes like:

- Divide by zero
- Entering blank spaces in the text fields
- Pressing the submit button without entering values.
- Uploading files exceeding maximum limits.
- Null pointer exception.
- Invalid parameters

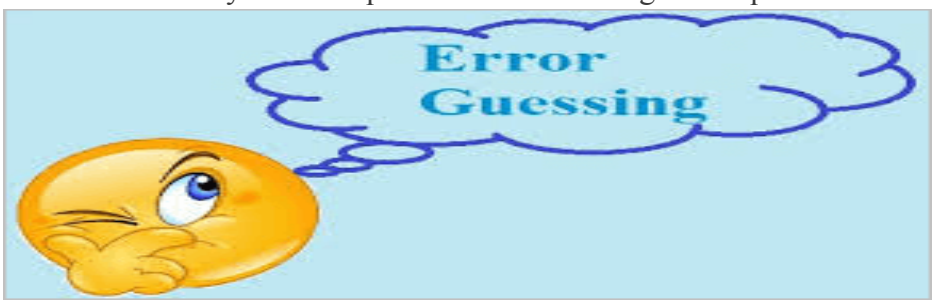
The achievement rate of this technique does mainly depends upon the ability of testers.

Purpose Of Error Guessing In Software Testing

The main purpose of this technique is to guess possible bugs in the areas where formal testing would not work.

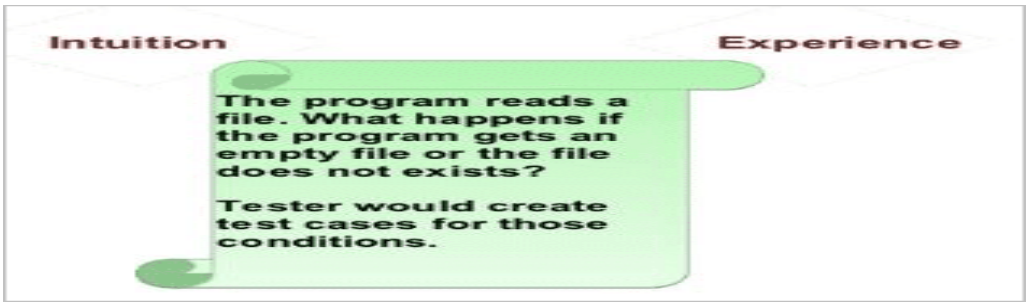
It should obtain an all-inclusive set of testing without any skipped areas, and without creating redundant tests.

This technique compensates for the characteristic incompleteness of Boundary Value Analysis and Equivalence Partitioning techniques.



Factors Used To Guess The Errors

Error Guessing technique requires skilled and experienced tester. It is mainly based on intuition and experience**.**



Following factors can be used to guess the errors:

Lessons learned from past releases
Tester's intuition
Historical learning
Previous defects
Production tickets
Review checklist
Application UI
Previous test results
Risk reports of the application
Variety of data used for testing.
General testing rules
Knowledge about AUT

When to perform Error Guessing?

It should be usually performed once most of the formal testing techniques have been applied.

Guidelines For Error Guessing

Remember previously troubled areas: During any of your testing assignments, whenever you come across an interesting bug, note it down for your future reference. There are generally some common errors that happen in a specific type of application. Refer to the list of common errors for the type of application you are working on.

Improve your technical understanding: Check how the code is written and how the concepts like a null pointer, arrays, indexes, boundaries, loops, etc are implemented in the code.

Gain knowledge of the technical environment (server, operating system, database) in which the application is hosted.

Do not just look for errors in the code but also look for errors and ambiguity in requirements, design, build, testing and usage.

Understand the system under test

Evaluate historical data and test results

Keep awareness of typical implementation errors

Error Guessing Example

Suppose there is a requirement stating that the mobile number should be numeric and not less than 10 characters. And, the software application has a mobile no. field.

Now, below are the Error Guessing technique:

What will be the result if the mobile no. is left blank?

What will be the result if any character other than a numeral is entered?

What will be the result if less than 10 numerals are entered?

Advantages of Error Guessing technique

Proves to be very effective when used in combination with other formal testing techniques.

It uncovers those defects which would otherwise be not possible to find out, through formal testing. Thus, the experience of the tester saves a lot of time and effort.

Error guessing supplements the formal test design techniques.

Very helpful to guess problematic areas of the application.

Drawbacks of Error Guessing technique

The focal shortcoming of this technique is that it is person dependent and thus the experience of the tester controls the quality of test cases.

It also cannot guarantee that the software has reached the expected quality benchmark.

Only experienced testers can perform this testing. You can't get it done by freshers.

Decision Table Testing

Decision Table Testing is an easy and confident approach to identify the test scenarios for complex Business Logic**.**

There are several test case design techniques. In this article, we will learn how to use **the Decision Table technique** effectively to write test cases for an application with complex Business Logic.

Here is an illustration:

We all know that the rules and validations of business take up a major portion of the requirements given by the customers. While observing how these requirements are represented and communicated to the entire project team by Business Analysts or customers, we come to know that most of such business rules and logic and are presented in a logical process flow diagram.

A logical process Flow diagram for a complex requirement comprises of many branches, nodes, and decision boxes. Hopefully, we testers are expected to cover all

those branches and touch every nook and corner of such a complex logical tree. I have also faced such complex business flows and tried many test case/test scenario preparation techniques for making the process easier.

Example: Writing Test cases for a login screen using the Decision Table Technique:

Let's take a **Decision Table example** of below business requirement for a login screen.

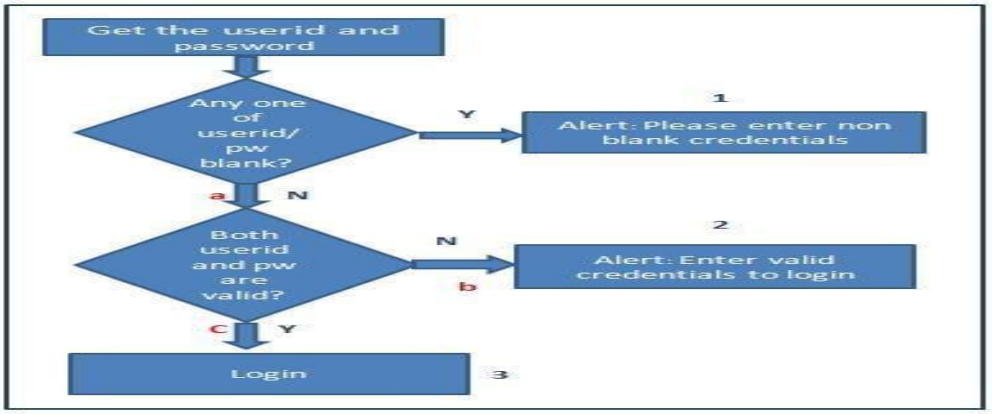


Fig: 1.0 Sample business flow diagram

The first step we do is to name all the branches and leave with numbers or alphabets as below.

1, 2, 3 are the leaves and a, b & c are the branches.

Then, we have to create a Decision table as shown below:

Sno	Branch	Leaf	Validations		Expected Results	Combinations
			Userid/pw blank	Userid and pw are valid		
1	a	1	YES	NA	Please enter non blank credentials	Both blank Either of them blank
2	b	2	NO	NO	Enter valid credentials to login	Both invalid Either of them invalid
3	c	3	NO	YES	Login	NA

Fig 1.1 Decision table for business flow fig 1.0

Advantages Of Using Decision Table Technique

1) Any complex Business flow represented as a diagram can be easily covered in this technique.

2) It provides quick confidence on the test cases. One need not have to review his own test cases multiple times to gain confidence.

3) Easy to understand. Anyone can make test cases from this Decision table template.

4) Rework on the test cases and test scenarios can be totally avoided, as it gives complete coverage at the first shot.

Limitations Of Using Decision Table Technique

1) Certain test case preparation techniques like Boundary value analysis, Equivalence partitioning cannot be directly accommodated in this template. But, one can note it down in the combinations column and use them while writing test cases.

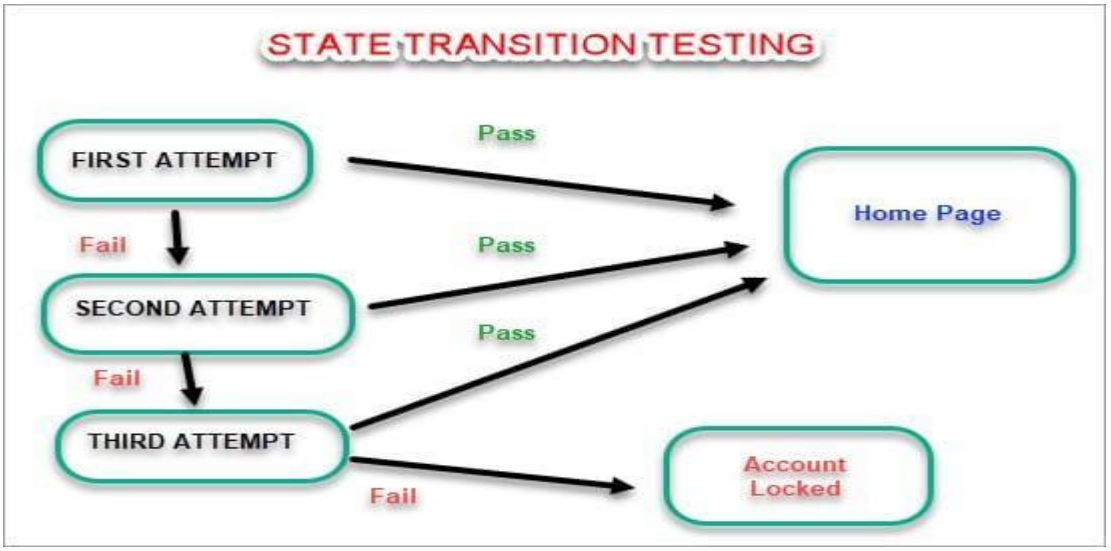
State Transition Testing

State Transition Testing is a technique that is used to test the different states of the system under test. The state of the system changes depending upon the conditions or

events. The events trigger states which become scenarios and a tester needs to test them.

A systematic state transition diagram gives a clear view of the state changes but it is effective for simpler applications. More complex projects may lead to more complex transition diagrams thereby making it less effective.

For Example:

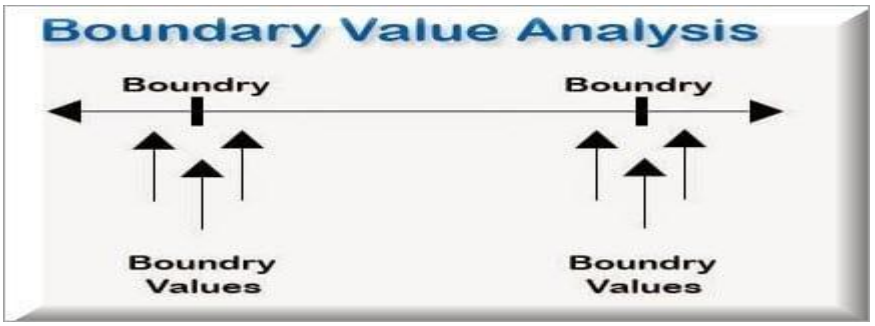


o.

Boundary Value Analysis

The name itself defines that in this technique, we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries. Boundary refers to values near the limit where the behavior of the system changes. In boundary value analysis, both valid and invalid inputs are being tested to verify the issues.

For Example:



If we want to test a field where values from 1 to 100 should be accepted, then we choose the boundary values: 1-1, 1, 1+1, 100-1, 100, and 100+1. Instead of using all the values from 1 to 100, we just use 0, 1, 2, 99, 100, and 101.

QUESTION BANK

UNIT 2: Black Box Testing

1. Demonstrate the black box testing. List black box testing techniques.What is need of testing?

2. Design the equivalence class test cases for a program that reads two integer pairs (m1, c1) and (m2, c2) defining two straight lines of the form y=mx+c. The program computes the intersection point of the two straight lines and displays the point of intersection.

3. Illustrate equivalence class partitioning technique with suitable example.CIA1

4. Describe boundry value analysis and write test suit for following scenerio:

As per HR policy if the applicant is between 0 to 12 years age, do not hire; if the applicant is between 12 to 18 years of age can only hire as an intern; and if it between 18 to 65,can hire full time; and above 65, do not hire.

5. To perform boundry value testing how many test cases required to test

Given f(x,y) with constraints $a \leq x \leq b$

$c \leq y \leq d$

Describe the test cases required .

6. Illustrate Boundary Value Analysis test case design technique with the help of suitable example?

7. For a function that computes the square root of the integer values in the range of 0 and 5000, determine the boundary value test suite.

8. Illustrate Decision Table technique for test case design with suitable example?

9. Illusterate state transition test case design technique with the help of suitable example?

10. Assume that the function try takes four parameters w,x,y and z as argument.Each of this four parameters can assume values in the range 0 to 100. How many boundry value test cases should be designed for robust testing of function try?

11. Compare Black box and white box testing.

12. Demonstarte weak ,strong and strong robust equilance class testing with suitable example?

13. Describe week equivalence class testing, strong equivalence testing, strong robust equivalence class test cases for following scenerios

For deposite of 1Lakh, rate of interest: 6% for deposite up to 1 year, 7% for deposite over 1 year but less than 3 years,8% for deposite 3 years and above

For deposite of more than 1Lakh, rate of interest:7% for deposite up to 1 year, 8% for deposite over 1 year but less than 3 years, 9% for deposite 3 years and above

13. Consider the program that reads the age of employees and compute the average age. Assume valid age is 1 to 100

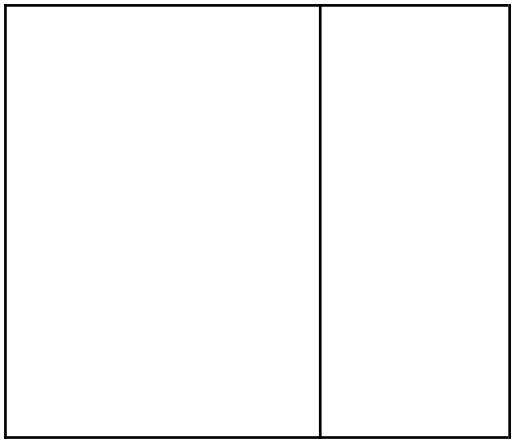
i. How would you test this?

ii. how many test cases would you generate?

iii. What would be the test data?

14. Consider the function “compute grade” written by education institute to compute grade of student from marks obtained. Assume following grading scheme is used.

During black box testing of this function ,altogether atleast how many equilence test cases are needed to be designed,assuming robust testing is targated?



15.

16. What are the limitations of boundry value analysis techniques?

17. Policy for charging for customers certain in-flight services is as follows:

If the flight is more than half- full and ticket cost is more than Rs 3000, free meals are served unless it is a domestic flight, the meals are charged on all domestic flights.

Develop a decision table for all possible combinations. Reduce the table by elimaniting redundant columns. Write final test cases.

18. Explain multiparameter equilance class partationing with suitable example?

19. What are the Combinatorial Testing techniques , List combinatorial testing techniques and explain any one?