Software Engineering

Software Testing Assignment

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. This can be either Functional or Non-Functional.

Test Case 1:

User enters password or email Id in incorrect format.

It should give the user a message to enter the email or password as per the format mentioned.

Test Case 2:

User enters incorrect email Id or password.

It should give an error message to the user which says "User Id or Password incorrect".

Equivalence Partitioning

Equivalence Class Partitioning is another name for ECP. In this strategy, the system or application's input data are split into distinct classes or groups based on based on its similarity in the outcome.

As a result, instead of testing each and every input value, we may now test any value from the group/class. We can retain test coverage while reducing rework and, most crucially, time wasted in this manner.

Test case 1:

Age limit:
Lower cap : 8
A valid class will be anything greater than or equal to 9

An invalid class will be age less than 8. (considering them to be an kid)

Test case 2:

Consider the Order Pizza Text Box's behavior.

- Pizza values ranging from 1 to 10 are categorised legitimate. The message "success" is shown.
- While values 11 to 99 are regarded invalid for ordering, an error notice "Only 10 Pizza may be ordered" will show.

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

We can't test all of the potential values because it would result in more than 100 test cases. To solve this problem, we apply the equivalence partitioning principle, which divides the potential values of tickets into groups or sets, as illustrated below, where the system behaviour is similar.
Equivalence Partitions or Equivalence Classes are the split sets. Then, for testing, we select only one value from each split. This approach is based on the idea that if one condition/value in a partition passes, all others will as well. Similarly, if one condition in a partition fails, the partition's other criteria will also fail.

Test case 3:
Consider the order rating from the user.

Here is the test condition

- Rating from 4 to 5 will be considered “Awesome”.
- Rating from 2 to 3 will be considered “Average”.
- Rating below 0 to 1 will be considered “tasteless”.

Test case 4:

Let us consider an example of software application. There is function of software application that accepts only a particular number of digits, not even greater or less than that particular number.
Consider an OTP number that contains only a 6 digit number, greater and even less than six digits will not be accepted, and the application will redirect customer or user to the error page. If the password entered by the user is less or more than six characters, that equivalence partitioning method will show an invalid OTP. If the password entered is exactly six characters, then the equivalence partitioning method will show valid OTP.

Valid	**Invalid**
Digits=6	Digits!=6

Test case 5:

Product ID for an item or dish should be valid. Here, we consider integer product ID to be valid and all other product ID as invalid. Also, the product ID should not be already existing for some other item.

Valid	**Invalid**
Type(product_ID)==integer	Type(product_ID)!=integer
Product_ID should not be already present.	Product_ID already present.

Boundary Value Analysis:

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 every equivalence partitioning, boundary value analysis can be done.

Test Case 1:

For test case 1 of our equivalence partition, we can formulate boundary value analysis.

Since we are allowing only integer values in Age, we can test the system around the threshold, which in our case is 8.

It should classify age 7 as invalid and age 9 as valid.

Test Case 2:

The allowed quantity of an item (let say Pizza) is between 1 to 10 for a single user. For applying boundary value analysis over this test case, we will test the system for 0,2,9 and 11 pizzas, where 0 and 11 should give invalid results and 2 and 9 should give valid results.

Here also, since we are using integer data type for quantity of a product, we will not check for decimal values.

Test Case 3:

Considering the test case 3 of out equivalence partitioning, in which we have categorised the restaurant on the basis of the rating given by the user, we can apply boundary value analysis for different values mentioned below:

NOTE: since we are allowing float entries for this test case, up to one decimal places, we will consider the boundaries according to the same.

- We will first check if the rating is from 0.0 to 1.9, we will classify them as tasteless.
- The ratings from 2.0 to 3.9 will be considered Average.
- The ratings from 4.0 to 5.0 will be considered Awesome. (since our upper limit is 5)

Test Case 4:

Based on our test case 4 of equivalence partitioning.

If OTP of 5 digits is entered, it should be invalid.

An OTP of 6 digits should be considered as valid.

An OTP of 7 digits should also be considered as invalid.

Integration Testing:

Test Case No.	**Test Case Objective**	**Test Case Description**	**Expected Outcome**
1	Test the interface link between login page and Home page.	Enter login info and click login.	Clicking the login should lead us to home page of the user.
2	Test the integration between sign up page and database.	Enter sign up information and click sign up	Clicking the sign up button should save the user's information in the database.
3	Test the interface link between login and sign up.	If the user is not registered, they should click the "Don't have an account yet? Sign up here".	Clicking on the link should lead the user to sign up page.
4	Test the interface between search page and restaurant home page	User searches for a restaurant and clicks on it.	It should lead the user to restaurant home page.
5	Test the interface between cart and order placed.	User adds items to cart and clicks on place order.	It should lead the user to a confirmation page that would say "Order Placed!"

AUTOMATED TESTING SCRIPTS

```
if (special_char in password and number in password and len(password)>5):
  login()
else:
  show("Your password must be 6-10 letters, must contain special character, must contain a number")
```

```
if (email in database and password in database and index(email)==index(password)):
    login()

else :
    print("incorrect credentials")
```

```
if (age<8):
    print("sorry! cannot create an account for you")
else:
    sign_up()
```

```
if (quantity>10):
    print("sorry, you cannot order more than 10 items")
else:
    place_order()
```

```
if (rating>=0 and rating<2):
    print("tasteless")
else if (rating>=2 and rating<4):
    print("Average")
else:
    print("Awesome")
```

```
if (age==7):
    validity=0
if (age==9):
    validity=1
```

```
if (quantity==0):
    validity=0
if (quantity==2):
    validity=1
if (quantity==9):
    validity=1
if (quantity==11):
    validity=0
```

```
if (rating==0.0):
    print("tasteless")
if (rating==1.9):
    print("tasteless")
if (rating==2.0):
    print("average")
if (rating==3.9):
    print("average")
if (rating==4.0):
    print("Awesome")
if (rating==5):
    print("Awesome")
```

```
if (email in database and password in database and index(email)==index(password)):
    if (user=="customer"):
        cutomer_login()
    else if (user=="restaurant"):
        restaurant_login()
```

```
if (sign_up):
    database.append([Name, Age, EmailId, Password, user])
```