MITE-414: Software Testing

Equivalence Class Testing

EC Testing is when you have a number of test items (e.g. values) that you want to test but because of cost (time/money) you do not have time to test them all. Therefore, you group the test item into class where all items in each class are supposed to behave exactly the same. The theory is that you only need to test one of each item to make sure the system works.

Example 1

Children under 2 ride the bus for free. Young people pay $10, Adults $15 and Senior Citizen pay $5. Classes:
Price:0 -> Age:0-1
Price:10 -> Age:2-14
Price:15 -> Age:15-64
Price:5 -> Age:65-infinity

Example 2 (more than one parameter)

Cellphones K80, J64 and J54 run Java 5. K90 and J99 run Java 6. But there are two possible browsers Firefox and Opera, J models run FF and K models run O. Classes:
Browser:FF, Java:5 -> Phones:J64,J54
Browser:FF, Java:6 -> Phones:J99
Browser:O, Java:5 -> Phones:K80
Browser:O, Java:6 -> Phones:K90

Dangers Of Equivalence Class Testing

There is a danger of using EC Testing that is rarely mentioned in the testing books but is very important to remember. Just because two items/values are supposed to be in the same class and behave the same, does not mean they DO behave the same.
That means that just because you test one value in the class that ALL values in the class behave the same. Real world example of mine is with cell phones that all had a certain Java Platform. They were supposed to all work the same but they didn't in reality. So, testing just one value in a class is good, but not good enough. EC Testing is a good tool, but it's not fool proof and be careful with it. If test cases are cheap and fast (like automation), test more, or why not test them all!

Boundary Value Testing

BV Testing is when you decide to test the values on the edge of each Class you have identified. The theory is that most defects is around the edges of a class. Example Classes:
Price:0 -> Age:0-1 (Boundary values 0, 1)
Price:10 -> Age:2-14 (Boundary values 2, 14)
Price:15 -> Age:15-64 (Boundary values 15, 64)
Price:5 -> Age:65-infinity (Boundary values 65)

What is State Transition Testing?

State Transition testing, a black box testing technique, in which outputs are triggered by changes to the input conditions or changes to 'state' of the system. In other words, tests are designed to execute valid and invalid state transitions.

When to use?

- When we have sequence of events that occur and associated conditions that apply to those events
- When the proper handling of a particular event depends on the events and conditions that have occurred in the past
- It is used for real time systems with various states and transitions involved

Deriving Test cases:

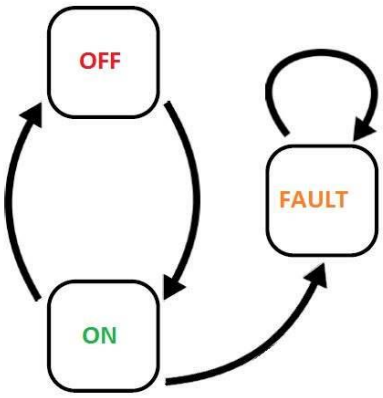
- Understand the various state and transition and mark each valid and invalid state
- Defining a sequence of an event that leads to an allowed test ending state
- Each one of those visited state and traversed transition should be noted down
- Steps 2 and 3 should be repeated until all states have been visited and all transitions traversed
- For test cases to have a good coverage, actual input values and the actual output values have to be generated

Advantages:

- Allows testers to familiarize with the software design and enables them to design tests effectively.
- It also enables testers to cover the unplanned or invalid states.

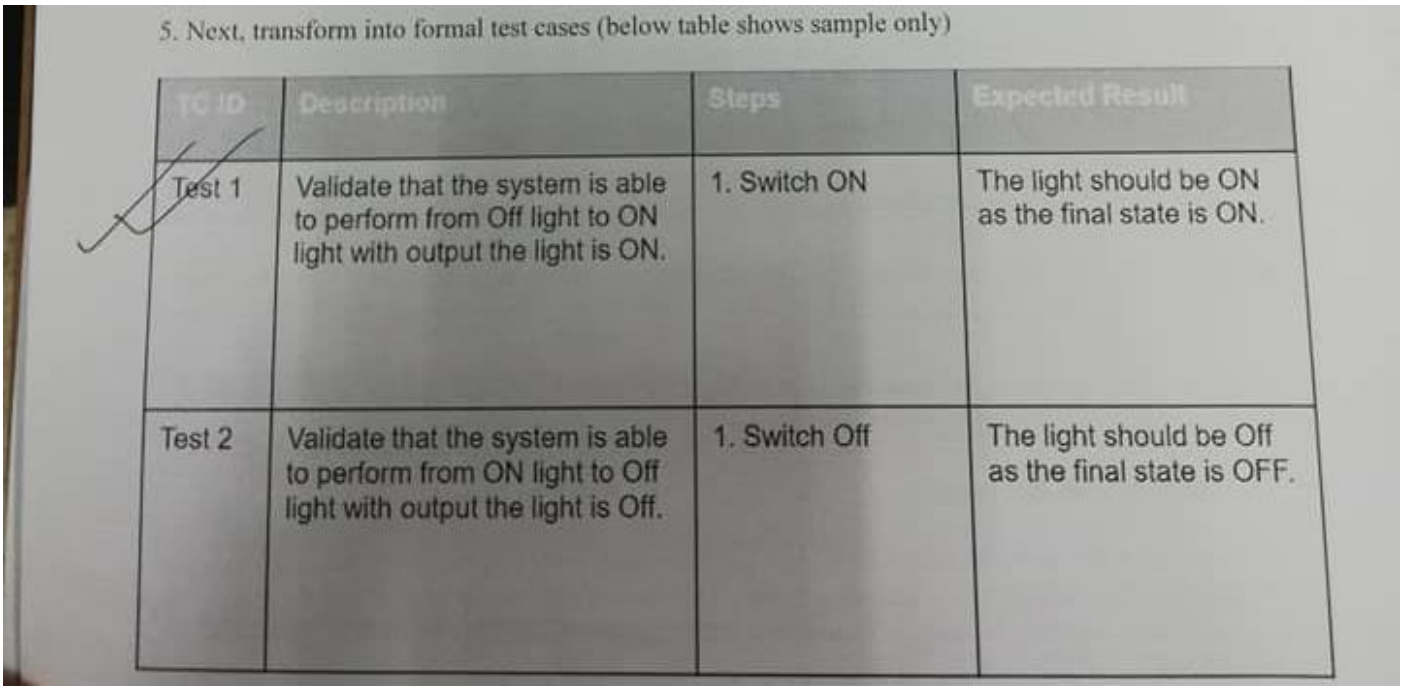
Example:

A System's transition is represented as shown in the below diagram:



The tests are derived from the above state and transition and below are the possible scenarios that need to be tested.

Tests	**Test 1**	**Test 2**	**Test 3**
Start State	Off	On	On
Input	Switch ON	Switch Off	Switch off
Output	Light ON	Light Off	Fault
Finish State	ON	OFF	On



5. Next, transform into formal test cases (below table shows sample only)

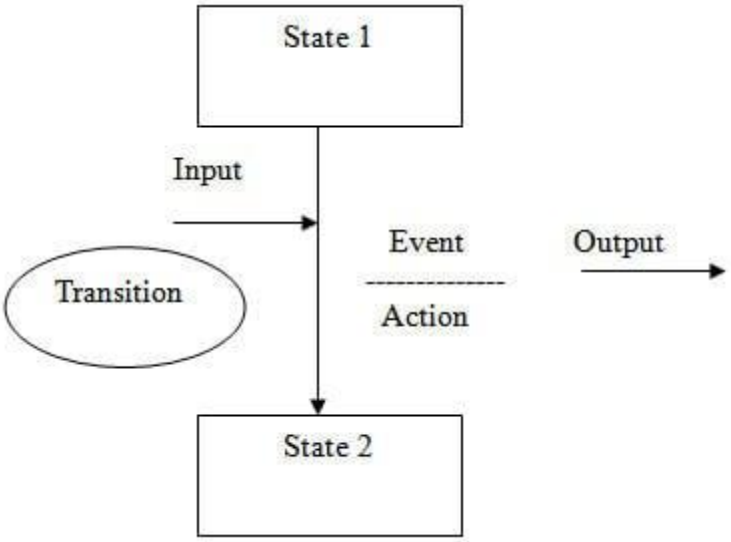
TC ID	Description	Steps	Expected Result
Test 1	Validate that the system is able to perform from Off light to ON light with output the light is ON.	1. Switch ON	The light should be ON as the final state is ON.
Test 2	Validate that the system is able to perform from ON light to Off light with output the light is Off.	1. Switch Off	The light should be Off as the final state is OFF.

State Transition Testing

State transition testing is a black-box testing technique, which can be applied to test 'finite state machines'. A 'Finite state machine (FSM)' is a system which will be in different discrete states (like "ready", "not ready", "open", "closed",...) depending on the inputs or stimuli. The discrete states that the system ends up with, depends on the rules of transition of the system. That is, if a system gives a different output for the same input, depending on its earlier state, then it is a finite state system. Further, if every transaction is tested in the system, it is called "0-switch" coverage. If testing covers 2 pairs of valid transactions, then it is "1-switch" coverage, and so on.

What is the State Transition Testing Technique?

State transition technique is a dynamic testing technique, which is used when the system is defined in terms of a finite number of states and the transitions between the states are governed by the rules of the system. Or in other words, this technique is used when features of a system are represented as states which transforms into one another. The transformations are determined by the rules of the software. The pictorial representation can be shown as:



So here we see that an entity **transitions** from State 1 to State 2 because of some **input** condition, which leads to an **event** and results in an **action** and finally gives the **output**.

To explain it with an example:

You visit an ATM and withdraw $1000. You get your cash. Now you run out of balance and make exactly the same request of withdrawing $1000. This time ATM refuses to give you the money because of insufficient balance. So here the **transition**, which caused the **change in state** is the earlier withdrawal

State Transition Testing Definition

Having understood what state transition is, we can now arrive at a more meaningful definition for state transition testing. So, it is a kind of black box testing in which the tester has to examine the behavior of AUT (application under test) against various input conditions given in a sequence.

The behavior of the system is recorded for both positive and negative test values.

When to use State Transition Testing?

State transition testing can be employed in the following situations:

- When the application under test is a real-time system with different states and transitions encompassed.
- When the application is dependent upon the event/values/conditions of the past.
- When the sequence of events needs to be tested.
- When the application needs to be tested against a finite set of input values.

When not to use State Transition Testing?

You should not rely upon State transition testing under the following situations:

- When testing is not required for sequential input combinations.
- When different functionalities of the application are required to be tested (more like exploratory testing).

State Transition Testing Example in Software Testing

In the practical scenario, testers are normally given the state transition diagrams and we are required to interpret it.

These diagrams are either given by the Business Analysts or a stakeholder and we use these diagrams to determine our test cases.

Let's consider the below situation:

Software name – Manage display_changes

Specifications – The software responds to input requests to change display mode for a time display device.

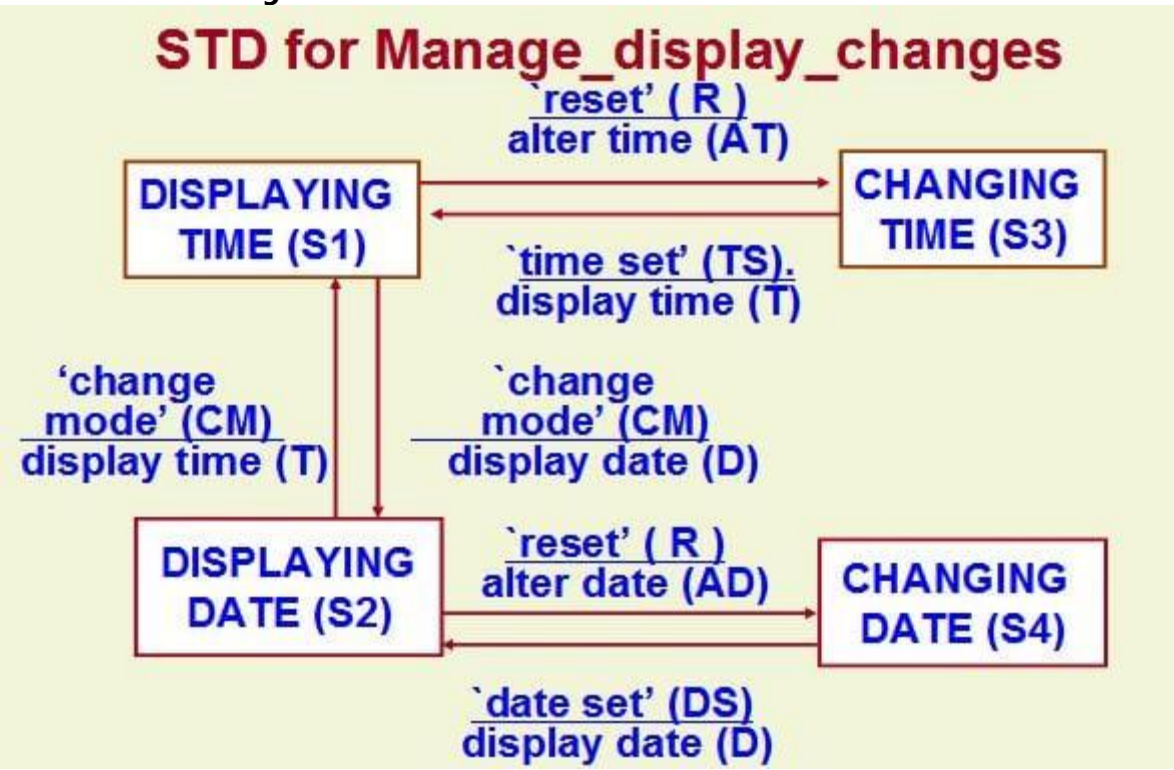
The display mode can be set to one of the four values:

- Two corresponding to displaying either the time or date.
- The other two when altering either the time or the date.

The different states are as follows:

- ✓ **Change Mode (CM)**
  Activation of this shall cause the display mode to move between "display time (T)" and "display date (D)".
- ✓ **Reset (R)**
- ✓ If the display mode is set to T or D, then a "reset" shall cause the display mode to be set to "alter time (AT)" or "alter date (AD)" modes.
- ✓ **Time Set (TS)**
  Activation of this shall cause the display mode to return to T from AT.
- ✓ **Date Set (DS)**
  Activation of this shall cause the display mode to return to D from AD.

State transition diagram



Now, let's move to interpret it:

Here:

1) Various states are:

- Display Time(S1),
- Change Time(S3),
- Display Date(S2) and
- Change Date (S4).

2) Various Inputs are:

- Change Mode(CM),
- Reset (R),
- Time Set(TS),
- Date Set(DS).

3) Various Outputs are:

- Alter Time(AT),
- Display Time(T),
- Display Date(D),
- Alter Date (AD).

4) The Changed States are:

- Display Time(S1),
- Change Time (S3),
- Display Date (S2) and
- Change Date (S4).

Step 1:

Write all of the start states. For this, take one state at a time and see how many arrows are coming out from it.

- For State S1, there are two arrows coming out of it. One arrow is going to state S3 and another arrow is going to state S2.
- For State S2 – There are 2 arrows. One is going to State S1 and other going to S4
- For State S3 – Only 1 arrow is coming out of it, going to state S1
- For State S4 – Only 1 arrow is coming out of it, going to state S2

Let's put this on our table:

Start State	S1	S1	S2	S2	S3	S4

Since for state S1 and S2, there are two arrows coming out, we have written it twice.

Step -2:

For each state, write down their final transitioned states.

- For state S1 – The final states are S2 and S3
- For State S2 – The final states are S1 and S4
- For State S3 – the Final state is S1
- For State S4 – Final State is S2

Put this on the table as output / resultant state.

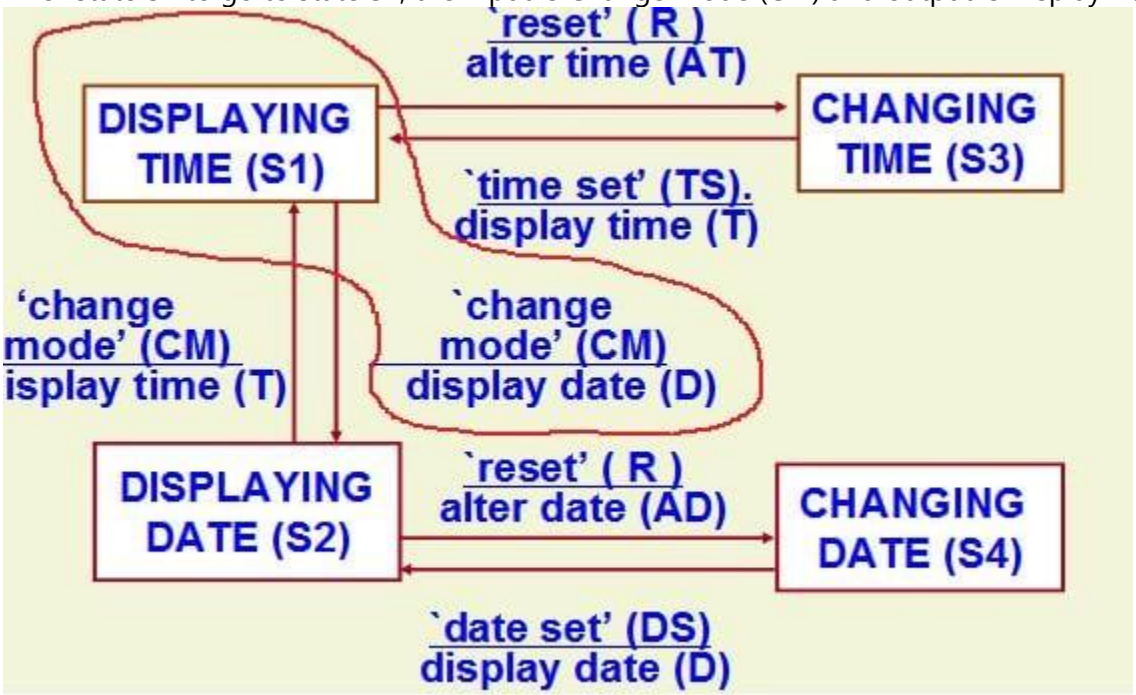
Start State	S1	S1	S2	S2	S3	S4

Finish State	S2	S3	S4	S1	S1	S2

Step 3:

For each start state and its corresponding finish state, write down the input and output conditions

– For state S1 to go to state S2, the input is Change Mode (CM) and output is Display Date(D) shown below:



In a similar way, write down the Input conditions and its output for all the states as follows:

Start State	S1	S1	S2	S2	S3	S4
Input	CM	R	R	CM	TS	DS
Output	D	AT	AD	T	T	D
Finish State	S2	S3	S4	S1	S1	S2

Step 4:

Now add the test case ID for each test shown below:

Test Case	TC1	TC2	TC3	TC4	TC5	TC6
Start State	S1	S1	S2	S2	S3	S4
Input	CM	R	R	CM	TS	DS
Output	D	AT	AD	T	T	D
Finish State	S2	S3	S4	S1	S1	S2

Now let's convert it to formal test cases:

TCID	Description	Steps	Expected Results
TC1	Validate that the system is able to do the transition from Display time to Display date with output as Display date.	1. Open the application. 2. Enter the Input state as Change mode.	Output should be Display Date and the final state is Display Date.
TC2	Validate that the system is able to do the transition from Display time state to Change time state with output as Alter time	1. Open the application. 2. Enter Input as Reset.	Output should be "Alter Time" and the state is Alter Time.

In this way, all the remaining test cases can be derived. I assume the other attributes of the test cases like preconditions, severity, priority, environment, build etc. are also included in the test case.

Summarizing the steps once again:

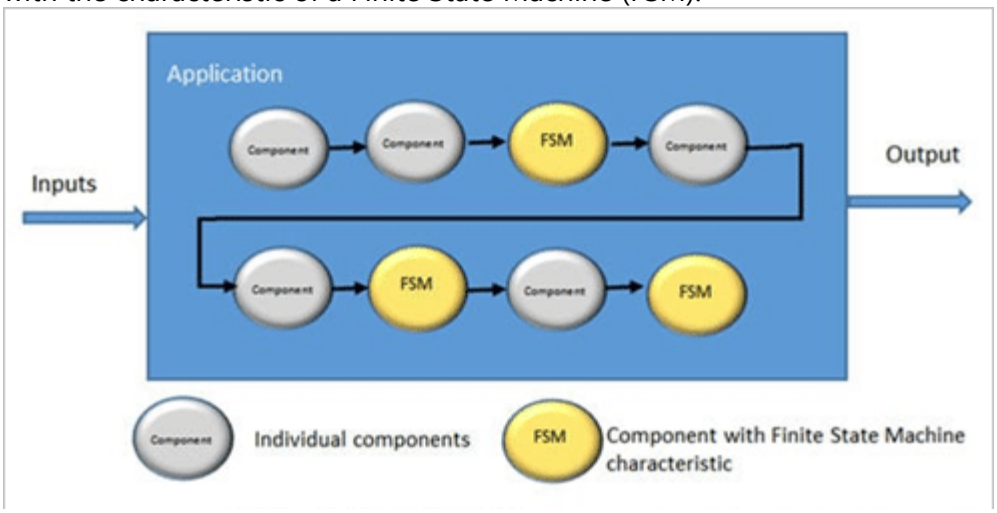
1. Identify the initial states and their final state based on the lines/arrows that are coming out of the initial state
2. For each initial state, find out the input condition and the output result
3. Mark each set as a separate test case.

More Examples of State Transition Technique

Here is one more example of State Transition Testing technique in bigger software applications.

Description:

'**Stateful Functional Testing'** approach can be used to test specific parts or components of the application, with the characteristic of a Finite State Machine (FSM).



Steps in implementation:

1. The first step in implementing the 'Stateful Functional Testing' is to identify different components/parts of the application that can be categorized as FSMs. The inputs, states, and outputs are carefully tracked for each of these FSMs.

2. The next step would be to develop test cases for these FSMs based on transition rules, inputs, outputs, and transition states.

3. The third step would be to integrate the testing of these components with other interfacing components for validating the application end to end.

This can be explained by means an example of an application named as "House Project", which tracks the construction of a house, with various application components like approval of architecture of the house, registration of the plot and house, selection of the building contractor, approval of housing loan, etc.

Example:

We shall consider testing of one FSM component of the "House Project" application: Approval of the Housing Loan.

Housing Loan Approval Application (HLA)

The HLA application will be run by an independent Loan Processing User, who processes the loan application. The different steps in the processing of the application are detailed below:

1.1.1 Step 1: Collection of documents

The first step is the collection of relevant documents for applying for the loan as mentioned in the table below. They are the 'conditions' for a successful application. The applicant collects the required documents and applies for the home loan. The Loan Processing User acknowledges the receiving of the documents and transitions the state of the Loan Application (that is the state of the HLA Application component) to "Applied" state.

Slno	Document
1	Personal Identity – like Driver's license/Voter's ID/ Ration Card
2	Marriage Registration Certificate
3	Medical Fitness Certificate
4	Two most recent Salary Payslips
5	Employment Appointment Letter
6	Most recent Payment Summary
7	If self-employed, the last two year's full tax returns
8	Proof of any existing rental income
9	Recent Savings Account Bank Statement
10	Other income certificates like share certificates, Mutual Funds, etc.
11	Most recent statement for all credit cards, personal loans, leases etc.
12	Document of Contract of sale for property being purchased

Table 1: List of Documents

1.1.2 Step 2: Loan Assessment

At this stage, the lender assesses the Loan Application to determine whether it meets his credit requirements. The supporting documents are verified at this time.

Slno	Document	Critical (Y/N)
1	Personal Identity – like Driver's license/Voter's ID/ Ration Card	Y
2	Marriage Registration Certificate	Y
3	Medical Fitness Certificate	Y
4	Two most recent Salary Payslips	Y
5	Employment Appointment Letter	Y
6	Most recent Payment Summary	N
7	If self-employed, the last two year's full tax returns	Y
8	Proof of any existing rental income	N
9	Recent Savings Account Bank Statement	Y
10	Other income certificates like share certificates, Mutual Funds, etc.	N
11	Most recent statement for all credit cards, personal loans, leases etc.	N
12	Document of Contract of sale for property being purchased	Y

Table 2: Criticality of Documents

The documents required for the assessment, that is the "conditions" which need to be validated at this stage, are validated. Each condition has a criticality attached to it (mentioned as 'Y' in the table above). Once all the required critical conditions are satisfied, the application moves to the state "Confirmed" – that is the HLA application component is in the "Confirmed" state.

Point to note:

#1: This principle brings in a structure and objectivity to the test conditions and "state" definitions of the system.

Also, not all the "conditions" for validating the system are critical for it to reach this "Confirmed" state. In the table above, 4 conditions are marked as "Not Critical" for the application to reach the "Confirmed" state.

#2: Number of validations can be optimally reduced, depending on the risk or criticality of the rules required for each state. This will significantly reduce the time required for test execution, and at the same time not compromising on Quality of testing.

#3: This is not only useful for testing the individual components, but also for testing the system end to end.

#4: Also, very useful while creating regression test suites.

So, at this stage, it is a 0-switch type of testing. But later stages of approval can be 1-switch or 2-switch types of validations for that stage. For example, "Marriage Certificate" may not be too relevant at this stage, but in the later stages of approval when the risk of the applicant to pay the EMI is being considered, the marriage certificate may become relevant – that is, if the spouse is employed too, it reduces the risk, and if not employed, it increases the risk.

#5: The above principle can be used for expanding the test conditions depending on the requirement of the component at that stage.

1.1.3 Step 3: Conditional Approval

The current state of the application is "Confirmed". The lender would give 'Conditional approval" for the loan process to move forward. Further validations are required for moving the HLA application to the state "Approved".

1.1.4 Step 4: Approval

Critical validations are conducted at this stage:

1. Assessment by the Lenders Mortgage Insurance (LMI): this would involve 2-switch or more validations for the property's genuineness.
2. The Lender may demand information which was not given during the "Confirmation" stage.

Once the above conditions are satisfied, the application moves to "Approved" state. The Final authority of the approval process may cross-check the credibility of the Loan applicant by asking for more details or may not ask if the Applicant's other documents are conclusive. That is, more inputs from different components of the main application would be required to prove the validity.

#6: In other words, more validations may be required (or reduced) for the transition to a different state depending on the input conditions to the component from other components of the application.

The diagram below depicts the approval process.

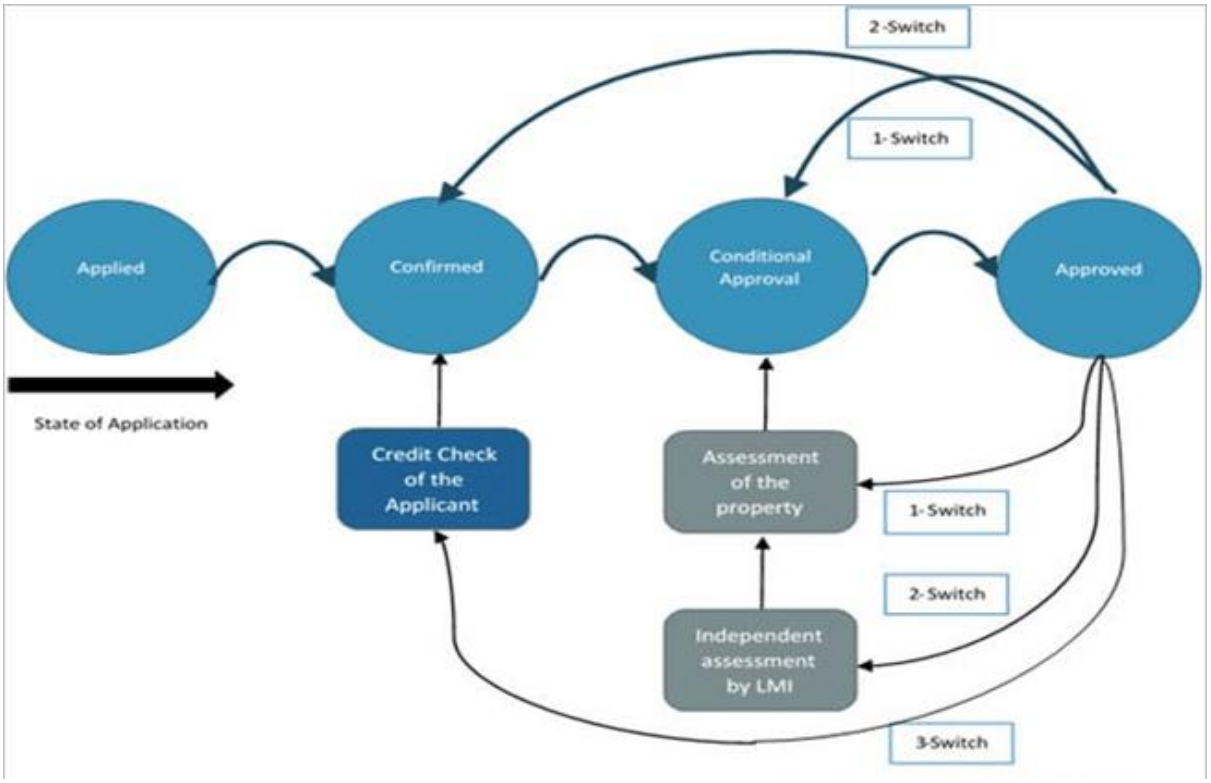


Figure 1: Loan Approval Process

Risks and Challenges

1. For large applications, deep application knowledge is essential to break the application into different logical components to enable categorization as FSMs and regular components. This might require costly time from SMEs.
2. Not all applications would have the feasibility of this kind of FSM categorization.
3. Since FSM components interact with regular components in the application, inputs to FSMs from different components require careful planning and execution.

Advantages of state transition testing

- Under this technique, by using a pictorial or tabular representation of system behavior, the tester becomes familiar with the application design and feels easy to cover & design the tests effectively and efficiently.
- The unplanned or invalid states of the system get also covered by using this technique.
- Using the state transition diagram, its easy to verify if all the conditions are covered.

Disadvantages of state transition testing

- This technique can't be used for nonfinite state systems.
- Defining all possible states for large and complex systems is a quite cumbersome task.

Conclusion

What is State transition testing in software testing?

State transition testing is a helpful approach when different system transitions are required to be tested for finite state systems.

State transition testing is used where some aspect of the system can be described in what is called a 'finite state machine'. This simply means that the system can be in a (finite) number of different states, and the transitions from one state to another are determined by the rules of the 'machine'. This is the model on which the system and the tests are based.

- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
- A finite state system is often shown as a **state diagram** (see Figure 4.2).
- One of the advantages of the state transition technique is that the model can be as detailed or as abstract as you need it to be. Where a part of the system is more important (that is, requires more testing) a greater depth of detail can be modeled. Where the system is less important (requires less

testing), the model can use a single state to signify what would otherwise be a series of different states.

- A **state transition model has four basic parts:**
    - The states that the software may occupy (open/closed or funded/insufficient funds);
    - The transitions from one state to another (not all transitions are allowed);
    - The events that cause a transition (closing a file or withdrawing money);
    - The actions that result from a transition (an error message or being given your cash).

Hence, we can see that in any given state, one event can cause only one action, but that the same event – from a different state – may cause a different action and a different end state.

For example, if you request to withdraw $100 from a bank ATM, you may be given cash. Later you may make exactly the same request but it may refuse to give you the money because of your insufficient balance.

This later refusal is because the state of your bank account has changed from having sufficient funds to cover the withdrawal to having insufficient funds. The transaction that caused your account to change its state was probably the earlier withdrawal.

A state diagram can represent a model from the point of view of the system, the account or the customer.

Let us consider another example of a word processor. If a document is open, you are able to close it. If no document is open, then 'Close' is not available. After you choose 'Close' once, you cannot choose it again for the same document unless you open that document. A document thus has two states: open and closed.

We will look first at test cases that execute valid state transitions.

Figure 4.2 below, shows an example of entering a Personal Identity Number (PIN) to a bank account. The states are shown as circles, the transitions as lines with arrows and the events as the text near the transitions. (We have not shown the actions explicitly on this diagram, but they would be a message to the customer saying things such as 'Please enter your PIN'.)

The state diagram shows seven states but only four possible events (Card inserted, Enter PIN, PIN OK and PIN not OK). We have not specified all of the possible transitions here – there would also be a time-out from 'wait for PIN' and from the three tries which would go back to the start state after the time had elapsed and would probably eject the card.

There would also be a transition from the 'eat card' state back to the start state. We have not specified all the possible events either – there would be a 'cancel' option from 'wait for PIN' and from the three tries, which would also go back to the start state and eject the card.

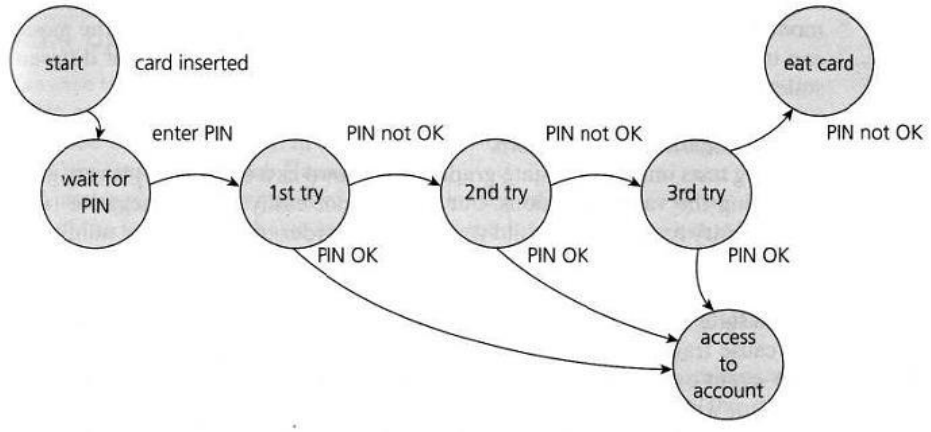


FIGURE 4.2 State diagram for PIN entry

In deriving test cases, we may start with a typical scenario.

- First test case here would be the normal situation, where the correct PIN is entered the first time.
- A second test (to visit every state) would be to enter an incorrect PIN each time, so that the system eats the card.
- A third test we can do where the PIN was incorrect the first time but OK the second time, and another test where the PIN was correct on the third try. These tests are probably less important than the first two.

- Note that a transition does not need to change to a different state (although all of the transitions shown above do go to a different state). So there could be a transition from 'access account' which just goes back to 'access account' for an action such as 'request balance'.

Test conditions can be derived from the state graph in various ways. Each state can be noted as a test condition, as can each transition. However this state diagram, even though it is incomplete, still gives us information on which to design some useful tests and to explain the state transition technique.
We need to be able to identify the coverage of a set of tests in terms of transitions. We can also consider transition pairs and triples and so on.
Coverage of all individual transitions is also known as 0-switch coverage, coverage of transition pairs is l-switch coverage, coverage of transition triples is 2-switch coverage, etc. Deriving test cases from the state transition model is a black-box approach.
Measuring how much we have tested (covered) will discuss in a white-box perspective. However, state transition testing is regarded as a black-box technique.

Example scenario: "**A marketing company wishes to construct a decision table to decide how to treat clients according to three characteristics: Gender, City Dweller, and age group: A (under 30), B (between 30 and 60), C (over 60). The company has four products (W, X, Y and Z) to test market. Product W will appeal to female city dwellers. Product X will appeal to young females. Product Y will appeal to Male middle aged shoppers who do not live in cities. Product Z will appeal to all but older females.**"

Decision tables are used to model complicated programming logic. They can make it easy to see that all possible combinations of conditions have been considered; when conditions are missed, it is easy to see this. The tables are composed of 4 parts: conditions, actions, condition alternatives (each column is a rule), and actions for the rules.

The process used to create a decision table is the following:

1. Identify conditions and their alternative values.

- ✓ There are 3 conditions: gender, city dweller, and age group. Put these into table as 3 rows in upper left side.
- ✓ Gender's alternative values are: F and M.
- ✓ City dweller's alternative values are: Y and N
- ✓ Age group's alternative values are: A, B, and C

2. Compute max. number of rules.

- ✓ Determine the product of number of alternative values for each condition.
- ✓ 2 x 2 x 3 = 12.
- ✓ Fill table on upper right side with one column for each unique combination of these alternative values. Label each column using increasing numbers 1-12 corresponding to the 12 rules. For example, the first column (rule 1) corresponds to F, Y, and A. Rule 2 corresponds to M, Y, and A. Rule 3 corresponds to F, N, and A. Rule 4 corresponds to M, N, and A. Rule 5 corresponds to F, Y, and B. Rule 6 corresponds to M, Y, and B and so on.

3. Identify possible actions

- Market product W, X, Y, or Z. Put these into table as 4 rows in lower left side.

4. Define each of the actions to take given each rule.

- For example, for rule 1 where it is F, Y, and A; we see from the above example scenario that products W, X, and Z will appeal. Therefore, we put an 'X' into the table's intersection of column 1 and the rows that correspond to the actions: market product W, market product X, and market product Z.

	1	2	3	4	5	6	7	8	9	10	11	12
Gender	F	M	F	M	F	M	F	M	F	M	F	M
City	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
Age	A	A	A	A	B	B	B	B	C	C	C	C

MarketW	X				X				X			
MarketX	X		X									
MarketY								X				
MarketZ	X	X	X	X	X	X	X	X		X		X

5. Verify that the actions given to each rule are correct.
6. Simplify the table.

- ✓ Determine if there are rules (columns) that represent impossible situations. If so, remove those columns. There are no impossible situations in this example.
- ✓ Determine if there are rules (columns) that have the same actions. If so, determine if these are rules that are identical except for one condition and for that one condition, all possible values of this condition are present in the rules in these columns. In the example scenario, columns 2, 4, 6, 7, 10, and 12 have the same action. Of these columns: 2, 6, and 10 are identical except for one condition: age group. The gender is M and they are city dwellers. The age group is A for rule 2, B for rule 6, and C for rule 10. Therefore, all possible values of condition 'age group' are present. For rules 2, 6, and 10; the age group is a "don't care". These 3 columns can be collapsed into one column and a hyphen is put into the age group location to signify that we don't care what the value of the age group is, we will treat all male city dwellers the same: market product Z.

	1	2	3	4	5	6	7	8	9	10
Gender	F	M	F	M	F	M	F	M	F	M
City	Y	Y	N	N	Y	N	N	Y	N	N
Age	A		A	A	B	B	B	C	C	C
MarketW	X				X			X		
MarketX	X		X							
MarketY							X			
MarketZ	X	X	X	X	X	X	X			X

Decision Table Methodology

1. Identify Conditions & Values	Find the data attribute each condition tests and all of the attribute's values.
2. Compute Max Number of Rules	Multiply the number of values for each condition data attribute by each other.
3. Identify Possible Actions	Determine each independent action to be taken for the decision or policy.
4. Enter All Possible Rules	Fill in the values of the condition data attributes in each numbered rule column.
5. Define Actions for each Rule	For each rule, mark the appropriate actions with an X in the decision table.
6. Verify the Policy	Review completed decision table with end-users.
7. Simplify the Table	Eliminate and/or consolidate rules to reduce the number of columns.

Decision Table - Example

Save Supermarket has a policy for cashing customers checks. If the check is a personal check for$75.00 or less the check can be cashed. If the check is the customers pay check, it can be cashed for over $75.00 providing it is a company accredited by the supermarket.

Step 1 - Name the conditions and the values for each condition -

- ✓ Type of check - personal (PE), payroll (PA)
- ✓ Amount of check - equal to or less than $75.00 (=<$75), greater than $75.00 (>$75)
- ✓ Accredited company - Yes (Y), No (N)

Step 2 - name all possible actions

- ✓ Cash the check
- ✓ Don't cash the check

Step 3 - list all possible rules

2 x 2 x 2 = 8 rules

	Rules							
Conditions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Type of check	PE	PE	PE	PE	PA	PA	PA	PA
Amount of check	=<$75	=<$75	>$75	>$75	=<$75	=<$75	>$75	>$75
Accredited Company	Y	N	Y	N	Y	N	Y	N

Step 4 - Define the Actions for each rule

	Rules							
Conditions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Type of check	PE	PE	PE	PE	PA	PA	PA	PA
Amount of check	=<$75	=<$75	>$75	>$75	=<$75	=<$75	>$75	>$75
Accredited Company	Y	N	Y	N	Y	N	Y	N
Actions								
Cash the check	X	X			X		X	
Don't cash the check			X	X		X		X

Step 5 - Simplify the Decision Table

	Rules							
Conditions	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Type of check	PE	PE	PE	PE	PA	PA	PA	PA
Amount of check	=<$75	=<$75	>$75	>$75	-	-	-	-
Accredited Company	-	-	-	-	Y	N	Y	N
Actions								
Cash the check	X	X			X		X	
Don't cash the check			X	X		X		X

Remove indifferent conditions. Combine rules 1 and 2, 3 and 4, 5 and 7, and 6 and 8

Simplified Decision Table

	Rules			
Conditions	**1**	**2**	**3**	**4**
Type of check	PE	PE	PA	PA
Amount of check	=<$75	>$75	-	-
Accredited Company	-	-	Y	N
Actions				
Cash the check	X		X	
Don't cash the check		X		X

Cause and Effect

Cause and effect graph is a dynamic test case writing technique. Here causes are the input conditions and effects are the results of those input conditions.

Cause-Effect Graphing is a technique which starts with a set of requirements and determines the minimum possible test cases for maximum test coverage which reduces test execution time and ultimately cost.

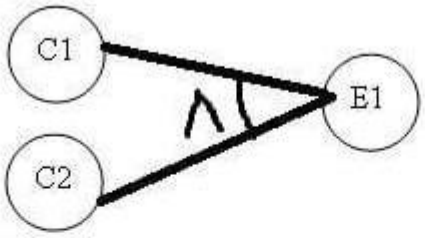
The goal is to reduce the total number of test cases still achieving the desired application quality by covering the necessary test cases for maximum coverage.

But at the same time obviously, there are some downsides of using this test case writing technique. It takes time to model all your requirements into this cause-effect graph before writing test cases.

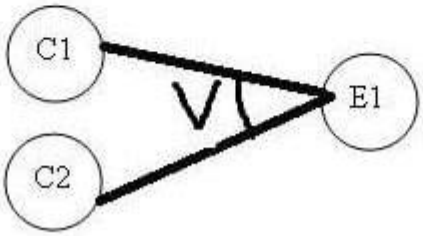
The Cause-Effect graph technique restates the requirements specification in terms of the logical relationship between the input and output conditions. Since it is logical, it is obvious to use Boolean operators like AND, OR and NOT.

Notations we are going to use:

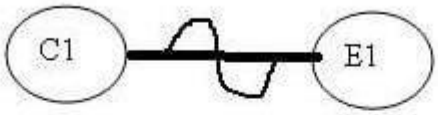
AND – For effect E1 to be true, both the causes C1 and C2 should be true



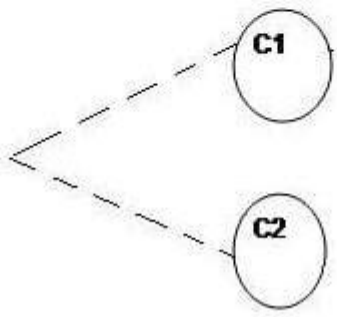
OR – For effect E1 to be true, either of causes C1 OR C2 should be true



NOT – For Effect E1 to be True, Cause C1 should be false



MUTUALLY EXCLUSIVE – When only one of the causes will hold true.



Now let's try to implement this technique with some example.

1. Draw a cause and effect graph based on a requirement/situation
2. Cause and Effect graph is given, draw a decision table based on it to draw the test case.

Let's see both of them one by one. Let's draw a cause and effect graph based on a situation

Situation:

The "Print message" is software that read two characters and, depending on their values, messages must be printed.

- The first character must be an "A" or a "B".
- The second character must be a digit.
- If the first character is an "A" or "B" and the second character is a digit, the file must be updated.
- If the first character is incorrect (not an "A" or "B"), the message X must be printed.
- If the second character is incorrect (not a digit), the message Y must be printed.

Solution:

The causes for this situation are:

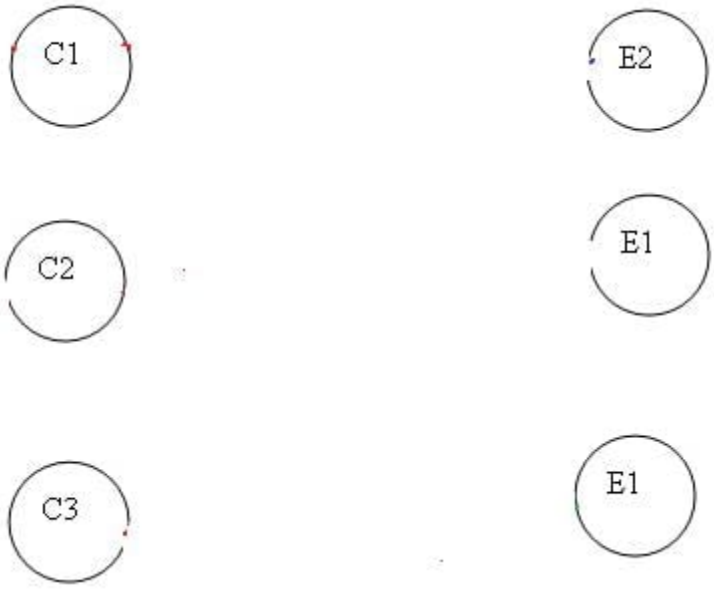
C1 – First character is A
C2 – First character is B
C3 – the Second character is a digit

The effects (results) for this situation are

E1 – Update the file
E2 – Print message "X"
E3 – Print message "Y"

LET'S START!!

First, draw the causes and effects as shown below:



Key – Always go from effect to cause (left to right). That means, to get effect "E", what causes should be true**. In this example, let's start with Effect E1.**

Effect E1 is to update the file. The file is updated when

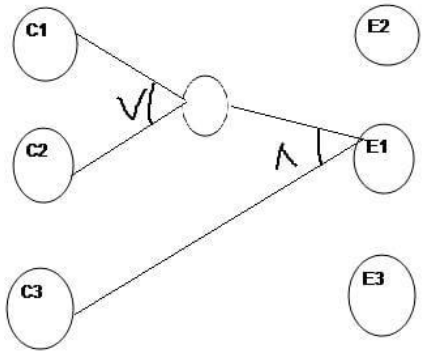
– The first character is "A" and the second character is a digit
– The first character is "B" and the second character is a digit
– The first character can either be "A" or "B" and cannot be both.

Now let's put these 3 points in symbolic form:

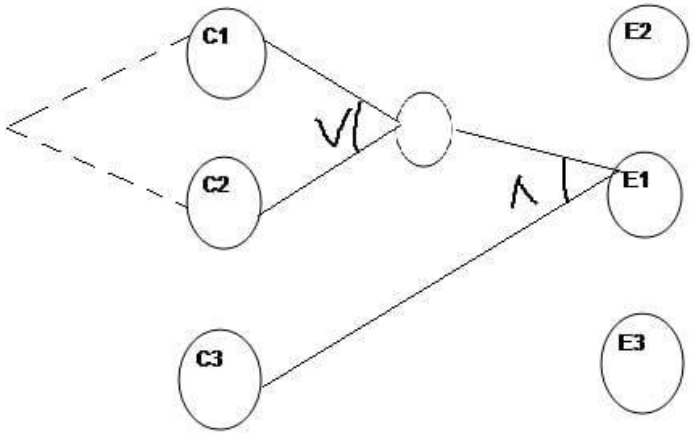
For E1 to be true – following are the causes:

– C1 and C3 should be true
– C2 and C3 should be true
– C1 and C2 cannot be true together. This means C1 and C2 are mutually exclusive.

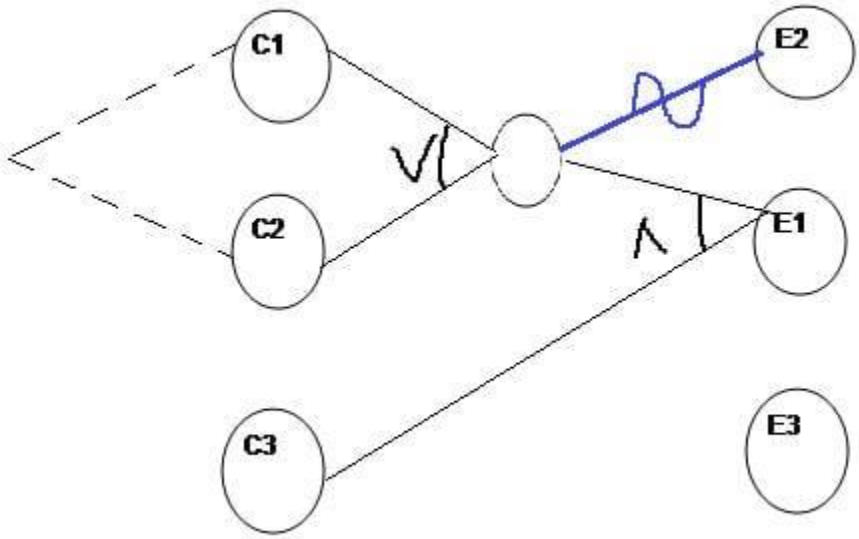
Now let's draw this:



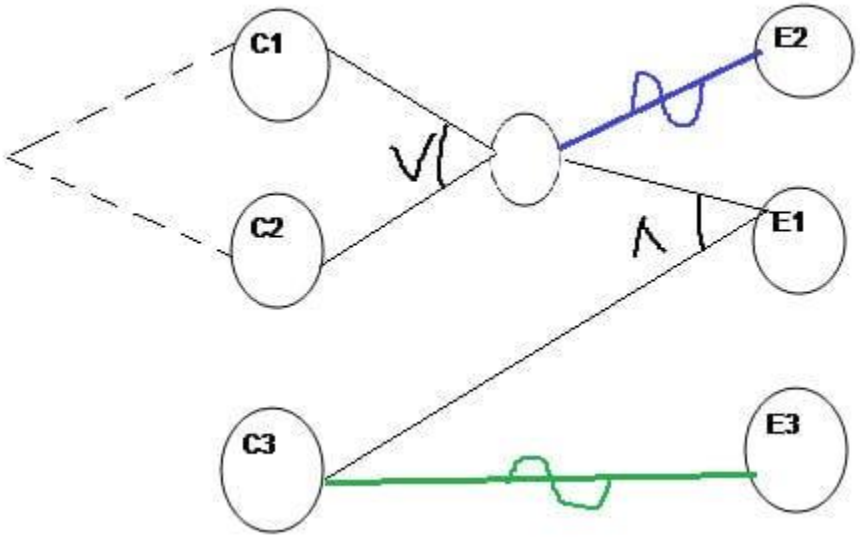
So as per the above diagram, for E1 to be true the condition is (C1 ∨ C2) ∧ C3
The circle in the middle is just an interpretation of the middle point to make the graph less messy.
There is a third condition where C1 and C2 are mutually exclusive. So the final graph for effect E1 to be true is shown below:



Let's move to Effect E2:
E2 states to print message "X". Message X will be printed when the First character is neither A nor B.
Which means Effect E2 will hold true when either C1 OR C2 is invalid. So the graph for Effect E2 is shown as (In blue line)



For Effect E3.
E3 states to print message "Y". Message Y will be printed when Second character is incorrect. Which means Effect E3 will hold true when C3 is invalid. So the graph for Effect E3 is shown as (In Green line)



This completes the Cause and Effect graph for the above situation. Now let's move to draw the **Decision table based on the above graph**. Writing Decision table based on Cause and Effect graph
First, write down the Causes and Effects in a single column shown below

Actions
C1
C2
C3
E1
E2
E3

Key is the same. Go from bottom to top which means traverse from effect to cause. Start with Effect E1. For E1 to be true, the condition is (C1 ∨ C2) ∧ C3. Here we are representing True as **1** and False as **0**
First, put Effect E1 as True in the next column as

Actions	
C1	
C2	
C3	
E1	1
E2	
E3	

Now for E1 to be "1" (true), we have the below two conditions – C1 AND C3 will be true C2 AND C3 will be true

Actions		
C1	1	
C2		1
C3	1	1
E1	1	1
E2		
E3		

For E2 to be True, either C1 or C2 has to be false shown as

Actions				
C1	1		0	
C2		1		0
C3	1	1	0	1
E1	1	1		
E2			1	1
E3				

For E3 to be true, C3 should be false.

Actions						
C1	1		0		1	
C2		1		0		1
C3	1	1	0	1		
E1	1	1				
E2			1	1		
E3					1	1

So, it's done. Let's complete the graph by adding 0 in the blank column and including the test case identifier.

Actions	TC1	TC2	TC3	TC4	TC5	TC6
C1	1	0	0	0	1	0
C2	0	1	0	0	0	1
C3	1	1	0	1	0	0
E1	1	1	0	0	0	0
E2	0	0	1	1	0	0
E3	0	0	0	0	1	1

Writing Test cases from the decision table

I am writing a sample test case for test case 1 (TC1) and Test Case 2 (TC2).

TC ID	TC Name	Description	Steps	Expected result
TC1	TC1_FileUpdate Scenario1	Validate that system updates the file when first character is A and second character is a digit.	1. Open the application. 2. Enter first character as "A" 3. Enter second character as a digit	File is updated.
TC2	TC2_FileUpdate Scenario2	Validate that system updates the file when first character is B and second character is a digit.	1. Open the application. 2. Enter first character as "B" 3. Enter second character as a digit	File is updated.