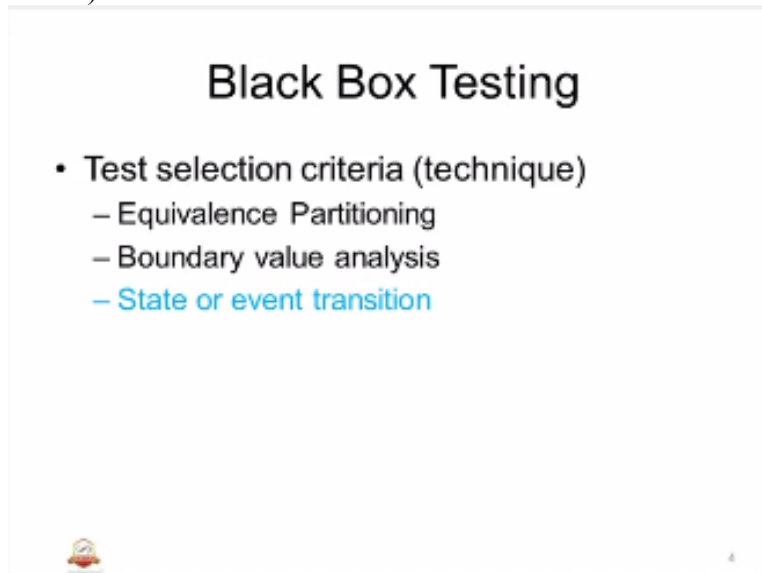


Welcome you to the next embedded software testing session.
(Refer Slide Time: 00:07)



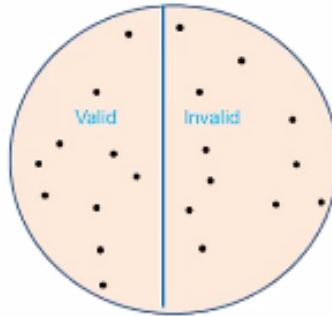
Lecture 15 of unit 2 on us discussed both those testing methods in test selections criteria in previous session,
(Refer Slide Time: 00:23)



We studied about equivalence partitioning and in the last class we studied about boundary value analysis and equivalence partitioning both and today we will discuss and study about state or event transition. We know state how it is going to behave in embedded system. Before that I guess we will just do a test selection, criteria for equivalence function and criteria analysis what is the difficulty we had? We know that equivalence classes.
(Refer Slide Time: 01:05)

Equivalence Partitioning forms

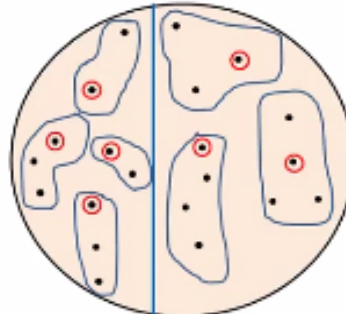
- First-level partitioning: Valid vs. Invalid test cases



We are going to have in terms of valid and invalid where the normal behavior of the system or normal expectation of the requirements are under valid equivalence classes and other side of the valid equivalence classes is the invalid equivalence classes.
(Refer Slide Time: 01:22)

Boundary Value analysis

- Create test cases to test boundaries of equivalence classes



- For each identified boundary in input and output, create two test cases. One test case on each side of the boundary but both as close as possible to the actual boundary line

And also we had studied about after we choose a good test case out of this equivalence classes using that group we start the boundary for each of that equivalence class valid or invalid in the test above that equivalence class and above the groups below the equivalence class in terms of the values and basically we had to reduce the number of test cases in terms of requirements.
(Refer Slide Time: 02:0-0)

Boundary Value Analysis - examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?



Prof. students.cs.bvu.edu

Wherein the behavior is same could be valid or invalid and also we said an example of boundary value analysis from -99 to +99. We had seen number of -110, -99, -98, -10, -9, -1, 0 -1 then we had 9, 10, 98, 99, 100 means the boundary value is the test above the boundary and below the boundary of a lower like this define it. So 17, we explained in the next slides in terms of minimum just above minimum nominal average or average discount.

(Refer Slide Time: 02:49)

Applying Boundary Value Analysis

- In general, application of Boundary Value Analysis can be done in a uniform manner.
- The basic form of implementation is to maintain all but one of the variables at their nominal (normal or average) values and allowing the remaining variable to take on its extreme values. The values used to test the extremities are:
 - Min ----- - Minimal
 - Min+ ----- - Just above Minimal
 - Nom ----- - Average
 - Max- ----- - Just below Maximum
 - Max ----- - Maximum



The middle one also it can be called as intermediate mode and we have a maximum and just above the maximum, so this all will do covered in terms of equivalence classes.

(Refer Slide Time: 03:02)

BVA important aspects with ex.

- **The Next Date problem:**

- The NextDate problem is a function of three variables: day, month and year. Upon the input of a certain date it returns the date of the day after that of the input. The input variables have the obvious conditions:
 - $1 \leq \text{Day} \leq 31$.
 - $1 \leq \text{month} \leq 12$.
- Min ----- - Minimal
- Min+ ----- - Just above Minimal
- Nom ----- - Average
- Max- ----- - Just below Maximum
- Max ----- - Maximum



11

While doing the equivalence classes we have to careful identifying the tests as especially for the problems like next date problems where we see we cannot afford to give empirical with a month of 31 which is not realistic and we should have knowledge of the requirement aspective. (Refer Slide Time: 03:27)

BVA important aspects with ex. contd.

- **The Triangle problem:**

- In fact the first introduction of the Triangle problem is in 1973, Gruenburger. There have been many more references to this problem since making this one of the most popular example to be used in conjunction with testing literature.
- The triangle problem accepts three integers (a, b and c) as its input, each of which are taken to be sides of a triangle. The values of these inputs are used to determine the type of the triangle (Equilateral, Isosceles, Scalene or not a triangle).
- For the inputs to be declared as being a triangle they must satisfy the six conditions:
 - C1. $1 \leq a \leq 200$.
 - C2. $1 \leq b \leq 200$.
 - C3. $1 \leq c \leq 200$.
 - C4. $a < b + c$.
 - C5. $b < a + c$.
 - C6. $c < a + b$.

Otherwise this is declared not to be a triangle.

The type of the triangle, provided the conditions are met, is determined as follows:

1. If all three sides are equal, the output is Equilateral.
2. If exactly one pair of sides is equal, the output is Isosceles.
3. If no pair of sides is equal, the output is Scalene.

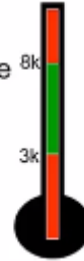


12

Similarly that triangle column also we have discussed there in able to define whether it is triangle or non triangle. Non triangle inputs of no use so we need to define what type of an input is on the definition of the requirement such as triangle or excel triangles. So that was what about the boundary value analysis and equivalence partitions. (Refer Slide Time: 03:55)

Exercise question

- Develop BVA for the Example: "A refrigerator has a red and a green indicator. The optimal temperature in the refrigerator is between +3 and +8 degrees. If the temperature is within this interval, the green indicator is lit, otherwise the red indicator is lit."
- The temperature range can be divided into three intervals (equivalence classes).
 1. From $-\infty$ (-273?) to but not including +3 resulting in a red light
 2. From +3 to +8 resulting in green light
 3. From but not including +8 to $+\infty$



We had an X types pro active infatuate that we are clearer on this, so what we had exercise for boundary value analysis at a temperature meter example the refrigerator as two signals, so two indicators are identical. The optimal temperature in the refrigerator pre indicator is lit that means between 3 and 8 which shows that green.

Whereas if it is below 3 and above 8 it will indicate as red the temperature in this can be divided in to three intervals we know that from infinity to infinity means here any number on the lower side, three degrees but not including 3 degrees it is here let give them in the, we have include and from 3 including 3 till 8 including 8. Similarly not from 8 but just above 8 it is a part one or whatever it is till another infinity values.

Such as 100, 200, 300 that are the capability of that particular temperature sensor.

(Refer Slide Time: 05:24)

- When using boundary value analysis, there should be one test case for each boundary in every equivalence class:
- Test case 1a:
 - Negative infinity, even -273 is a little hard to create, and furthermore not very likely to occur. So a good (?) estimation could be -20,000.
- Test case 1b:
 - Here we have the problem of being close enough to the boundary since being on the boundary is outside this interval. Is five valid digits a good estimate?
- Test cases 2a and 2b:
 - Both boundaries are inside the interval so these values are the ones to choose.
- Test case 3a:
 - Same discussion as in 1b.
- Test case 3b:
 - Same discussion as in 1a.



So for this example when using your boundary value analysis there should be one test case for each boundary in every equivalence class, so there are about 5 test cases identified for equivalence classes for this example the first one being test case of A negative infinity even -273

is a little hard to create and furthermore not very likely to occur. That means the temperature needs to be managed 273 such as or very less.

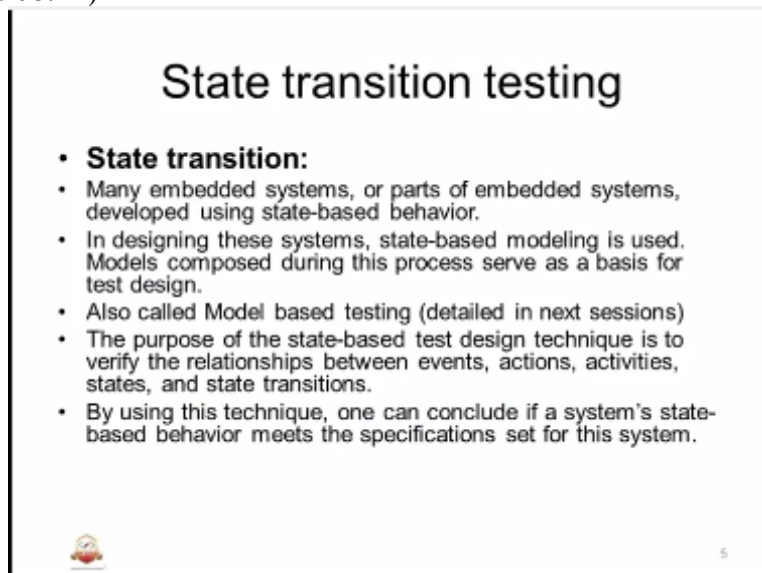
So we could give something like this 70 max so next one you hat test case one B here we have the problem of being close enough to the boundary since being on the boundary is outside this interval. Is five valid digits a good estimate? That it is can we have 3 digits or 4 digit of embedded software specification we should have an answer specificities how much it can go? Or it will don't have.

We should have a valid embedded system where the N service capable of so and so ranges. Those requirements will take care of only 3 and the we should have a knowledge of the underneath sensor or the hardware which is capable of taking a max of both the size so that is the test case 1a and 1b equal, then we have test cases 2a 2b with second set of test cases both modules are inside the interval.

So these values are the ones to choose that means there is a interval above 8 dependency so one of that similarly we have the other one also below 3 k it could be anything 3- 10, 30 depending on that capability of that sensor. Then we have test case 3a same discussions as 1b on the other side and test case 3b same as 1a on the other side that is on the left side so that is how we can have a boundary value and square this example.


Now the last test selection criteria technique that we will discuss today is about state or event transition. So what is a state?

(Refer Slide Time: 08:12)



State transition testing

- **State transition:**
- Many embedded systems, or parts of embedded systems, developed using state-based behavior.
- In designing these systems, state-based modeling is used. Models composed during this process serve as a basis for test design.
- Also called Model based testing (detailed in next sessions)
- The purpose of the state-based test design technique is to verify the relationships between events, actions, activities, states, and state transitions.
- By using this technique, one can conclude if a system's state-based behavior meets the specifications set for this system.

 5

Let us briefly understand about a state. Many embedded system as are parts of embedded systems or basically with the help of states state based behavior. It is something a different modes which you can also call it as where the embedded system will move from one to another one that means it has a pre defined set of actions or executions for such mode or such event, so based on the criteria that in terms are that,

Based on the criteria that it takes is from that particular state it has the number of behavior that it has to behave so basically any of the embedded systems are partly the pieces of the embedded systems can be intermitted with the help of state based behavior so state based embedded

systems are state mission is also called as if they implemented in melanin embedded systems especially complex systems in the industry.

Such that telecom or mobile any GPS or GSM system it take the control motive for an engine control systems where engine control has he many features or functionalities given based on the different states, similarly we have aerospace products the A matrix sub systems having state based or state driven applications. If you see in these systems the state based modeling is used. So basically in state machines,

Or state based embedded systems are a developed with a help of module because modeling is a good concept in developing such states so we have a state diagram from that maybe we can have a small session on what is state based modeling for the development purpose? Similarly we can understand the state model based testing that we have studied model based testing in one of the class in the previous sessions.

So models composed during this process are basics for the test designs for this all these modules are the basis of designing these systems having state based behavior so the testing is called module based testing. The purpose of state based technique is to verify any relationships between events, actions, activities, states and state transitions. Transition is something like the entry and exit of different states one state to other state the moment is called transition or the entry exit from one state to entry to another state is called the transition.

So upon occurrence of certain conditions those conditions are called events so based on those conditions or events that condition will occur. The events could be some actions, sorry as a result of event the matches will take it, as a result of action some activities are going to be happening or executed, so that is how I state transition will happen, by using this technique one can conclude this system state based behavior meets the specifications set for this system.

What you mean this with the help of this technique we know that how the system is being and existing on state based behavior and testing is we know that how we are going to reconstruct specification such for the system testing different conditions different events of state transition we will help us know whether it is transiting or not whether it is doing a entry in a state and whether it is exiting from the previous state etc...

All this can be known it as technique called state transition testing and the state based behavior are the projected using tables or activity charts or state charts, so just name that I will put this so that we get to aware this the embedded software tester, so basically state this systems are represented with the help of tables state charts, we will study little all this how they are tables also will be there? Like there are few elements which are part of these tables.

This table is also called as group of table using function pointer in C this is basically implemented with a very critical aspect in the embedded software systems where we have complex mechanism in terms of different states and numerous events triggered based on a numerous signals artery that means we have try to push suppose on state this under state and we have a state change wi6thin these two.

And of course we need to define a state entry also how this state can be entered? And how this state can be exited? So let us say state one this is state two 2 states are there these two plugs you can have a multiple states in the embedded system suppose state 1 needs to be enter so there are

certain events that needs to be e1 or e2 or e3. Systems under satisfaction of one of the events or all the events could result in enter ate in the state 1.

Suppose let us define the state 1 as initialization so upon power up there are certain conditions power up is one condition the assistant will enter into any state once the init state is complete that means there are different activities the action is two activities such as initialization or software etc so this state will be complete so once the state is complete that means the condition is that have to be or the actions that are to be concluded or completed within the state.

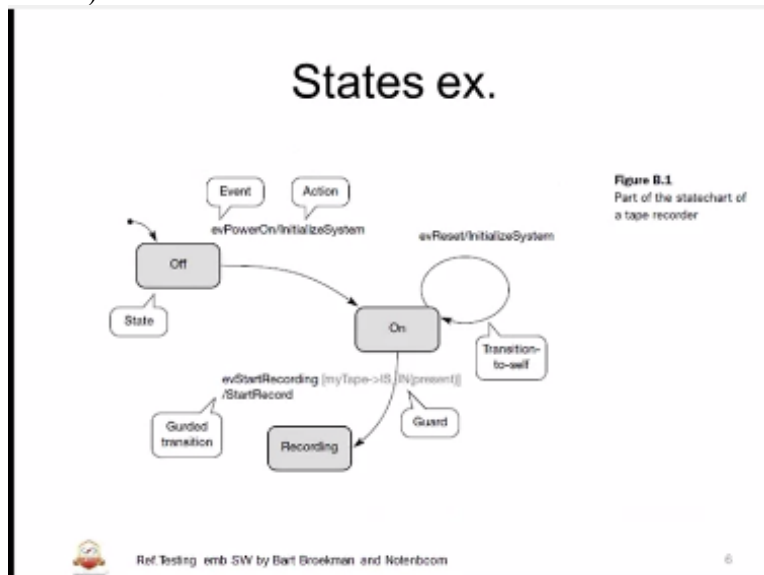
So that means we are good for going for next state use exit criteria is satisfied, that means we have an entry criteria and state to in satisfied. Here this entry here this exit so we know that state 2 is entered this could be a main application similarly we can have next state, next state like this with the help of a state diagrams and this state can go to first state also based on the pre set or some events that will result in switching between the states.

So that is what definition of state, this state diagram is basically state diagram this state diagrams are basically represented with the help of text I think having expenditure example how tables will apply, that table will basically define the state and events state charts also will define complete state machine entry exit all the values and everything and we have a lookup table identifying the functions.

Which are functions which are responsible for writing into different stages these are the executors and aspects that are part6 of the embedded software systems. So showing format of those states of condition mechanism in order to test it so how we are going to test it? Mostly with the help of model based testing will because state machines are state transitions which terms basically there is no model.

We study about model based testing in the next class or next sessions. So that is about state transition you can see an example of states.

(Refer Slide Time: 18:41)



This is again from the book embedded soft ware testing by Balt brook man and Nottingham so here he is depicted a state machine a state chart basically will it of a state recorder, let us try to understand from those understanding the states and the state chart. So there are pre states you can keep the grade one off on recording so take recorder can be in of state if take recorder can be in on state, take recording can enter in to recording state.

How half state will enter? There is a small dark emergency that is from the switch of we can say so we have switching icon here which will help in terms of going to the state. So that is why it is called as test state and terrific rated cam go into on state with the help of some operation or some action that is remote of event, event power on initialized system that means once you power on with this power on switch and you can connect to the power supply.

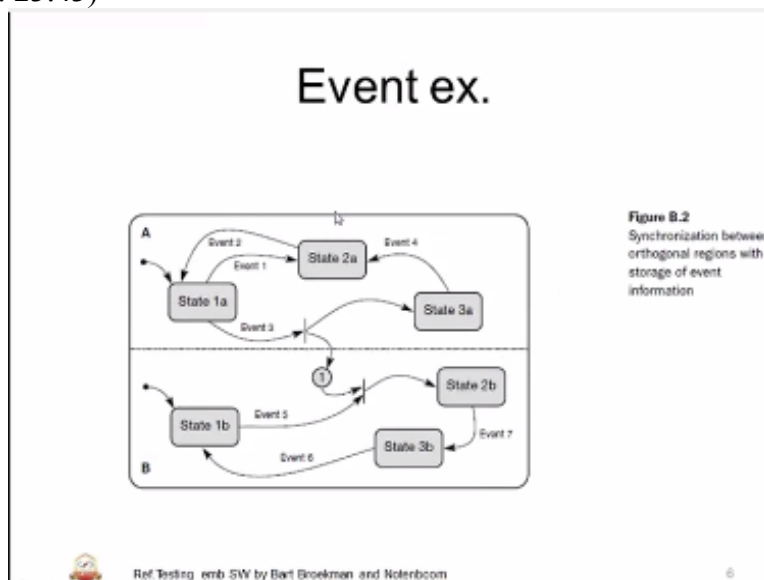
And make switch from off to on it will enter so the event is event power on so action is the initialized system so these things are important state event and action with this, this state is defined or shifting o the nest state is defined. Next is on what is going to happen within on? Or what are the events or actions that are going to be taking place this state that is on state you can see a circular arrow transition to self this is an important turn.

When on it can be there in on only because user has not done any operation here just followed on see we waiting for the user to operate further could be play or as you can see in next state recording or anything. So till then it is going to be these unite. So that event could be called as event result or initialize system that means not on it is just waiting for the result event to happen or it is waiting for the initialization any further initialization for the next.

The next part is recording so upon user tresses the recording that on what will happen is? It will enter into recording state, element is called start recording on the action is start recording, so there will be guard there will be turnings called guard so what you digest from on to recording guard will be triggered this is called guarded transition and of course some recording we can again go back to on state by stopping it so you can have another event called stop recording.

And the guard could be the next one similarly recording to half we can go but there are take the guard us with the we should first do a stop and we should go for off. So likewise it defines state path entered and exited so there times are very important concept state, event, action. So that is about an example about states how the state transition is going to happen here we can see three these are state, so what we have seen the different events.

(Refer Slide Time: 23:43)



Those are expiring in the next table next bit so let us do a small example on this o there are two levels of aspiring this or the flow between A and B you can see, laporfiding A those side as B the simpler turn we will try to understand, so what is told about the synchronization between ortho original groups stored on each information that means two orthonal events are going to occur with different states and how they taken care is what is been explained.

So there is a state 1a, state 2a state 3a in the first orthonal section and it will enter into the next orthonal with the help of state1b, state2b, state3b state 1b and for system within this stage

executing this stage there are different events that is to occur so state 1 A is been entered in type of forearm and event1 will go to state A and from state A event 2 it can happen it can come back to state 1 similarly state 1 into state 3 directly you can go the event is 3.

Similarly from 3A we can fall back state 2A with the help of event 4 then also there is a second orthogonal region the inventory can also give up to the next one which will result in state 2B state 3B state 1B etc state 1B can be directly enter with the help of this so there is a synchronization mechanism that is depicted here in the one around so that is not go in detail about this but we have to understand from this is that define states are responsible,

For taking into the inputs from different events and going to the next state in the fall switching and three times are state in action basically enter remember for embedded software system.

(Refer Slide Time: 26:20)

State transition testing

- The behavior of a system can be classified into the following three types.
- 1. *Simple behavior*. The system always responds in the exact same way to a certain input, independent of the system's history.
- 2. *Continuous behavior*. The current state of the system depends on its history in such a way that it is not possible to identify a separate state.
- 3. *State-based behavior*. The current state of the system is dependent on the system's history and can clearly be distinguished from other system states.



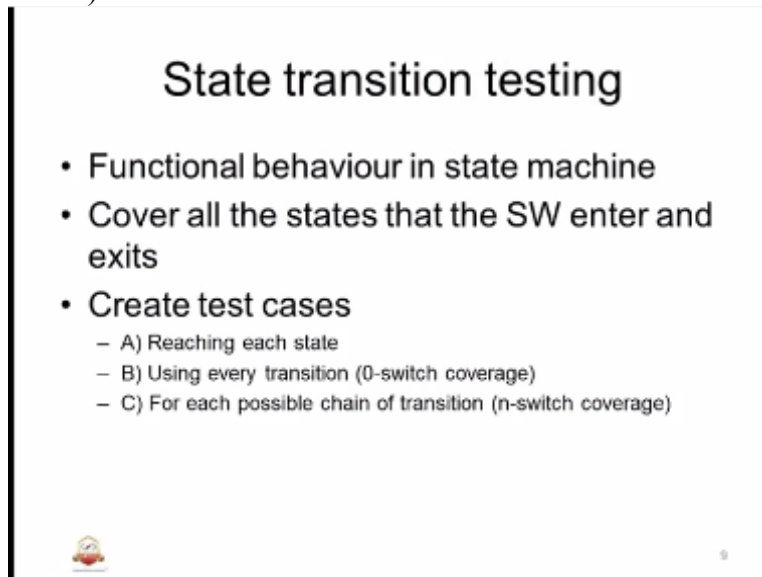
Now the behavior of the system can be classified into the following as three types. The state transition behavior is divided into three types simple behavior the system always response in the exact same way to a certain input independent of the system's history. That means it won't care whatever happen in the past but it will behave the same way all the time for certain input. Second type of behavior is continuous behavior.

The continuous behavior the current state of the system depends on its history such a way that it is not possible to identify a separate state. That means based on the history it will decide whether it will switch it or it will notch it so basically it is not possible to rectify a separate state this is continuous behavior. The third one is state based behavior where the current state of the system dependant on the system's history and can clearly be distinguished from other system states.

That means we know where the system it lie in this sort of a behavior that means the current state of the system is very well prepared on the system's history. But it is possible to identify a distinguishes state on the other state where it is, so this is a state based behavior second one is the continuous behavior where we don't know or we are not able to identify the specific states for the particular input it is again depending on the history.

Simpler behavior we have clear input and it is very much independent so it will behave of the same all the time same that is what it means. So now let us move on to next,

(Refer Slide Time: 28:26)



State transition testing

- Functional behaviour in state machine
- Cover all the states that the SW enter and exits
- Create test cases
 - A) Reaching each state
 - B) Using every transition (0-switch coverage)
 - C) For each possible chain of transition (n-switch coverage)

Functional behavior in state machine so basically we need to understand what are the functionalities we are going to execute, so what we understood from this? There are different actions based on different events all this all part of the functions one on the other day that the particular piece of software is going to behave those functions are something like procedure or different sound systems.

Which are driving the embedded system into switch between the different states so all this functional behavior e are going to test it and cover all the states with the software enter and exit that means the coverage is very important for a particular state as per the specification. So we need to create test cases as per three steps which each state how to reach state? Using every transition that from the default to switching.

That means every transition we are going to cover with the base to the next state with different transitions, similarly the next one is for each possible chain of transition which means N to switch coverage's which means maximum of possible transition in the complete part of that we are going to have it so we are going to create test cases accordingly. So state machine based testing is called the quite useful model based black box testing.

See any type of function can be represented as a finest transition can be tested using this technique so finest state machine in all beyond this you have but you can study that in embedded system basis where we have different state machines define and its specifications are get it. So it is transition based testing will help and identify the issues with those implementations of state transitions, so somewhat it is the first step what we do using the state machine testing is?

Respect the model exam that means we are going to construct the model how the mode is going to be behave here so we are going to with the help of test we are going to trigger some of the inputs so that we could model is constructed, some ties state machines are used by a imprisonment constructors as implementation tools. The models are used or developed so in those case the state machines and of course been used directly we can directly use the state machines otherwise this is a state machine model asked to construct based on the requirements based transitions.

That's it model base testing where we developed the model first then defined the states or based on the requirements depends on these simplicity of the complexity of the requirement of the underneath transition we are going to develop the state machines. So what will happen is during the construction of this statement is false can be there so the key properties with the state machine is that all input files can occur regardless on the state of the machine.

Implemented through model based design or it could be implemented directly on the requirements the requirements should be so good and clean fact it is implemented directly of the requirements, so saying that you can test it if the models are level we are going to attach the models to test it is the requirements of that statements we will brunch on the functionalities that is underneath the requirements.

Need or different states that are responsible for state machines, so there are other detail like the model based testing and brunch probably we will study that it will of the model based testing class. So basically we need to understand the strategies to elaborate or fouls on the transitions so very important in to understand the concept of transitioning and functioning which happen with the help of that and event will result action.

So all this part of the state N I will say N, so it is very important from different state from 0 to state N from state N to A whatever it is all are based on the transition. So we need to have a strategy test on the transitioning between the tests so 0 test is coverage requires 1 test case because we know that 1 terms and for each possible transition we have a one digit or one possible transition is there.

So let likewise we have for N to 1 conscecute transitions where an N to switch coverage requirement is there. So that is how state transition testing is taken care where transition has to be understood very well. Now having understood the different behavior and its states, events actions you need to categories the faults so what are the fault category that are for incorrect state behavior?

(Refer Slide Time: 35:21)

State Transition fault categories

- Incorrect state-based behavior can have three causes.
 - 1.The first is that the state chart(s) do(es) not represent a correct translation of the system's functional specifications. The state-based test design technique is not capable of revealing these types of faults because the statecharts themselves are used as the basis of the tests.
 - 2.A second cause is that the statecharts are syntactically incorrect or inconsistent. These faults can be revealed by static testing (for example, by using a checklist or tools). If the faults found this way are corrected, then the statechart constitutes the basis for dynamic testing using the state-based test design technique.
 - 3.The third cause is the translation from statecharts to code. It is becoming increasingly common that this translation is performed automatically. The code generated this way is (hopefully) an exact representation of the statecharts' behavior. The use of the statecharts as the basis for test design in this case is, therefore, not useful and the application of the state-based test design technique is superfluous. However, if coding based on statecharts is applied without the use of a generator, then the state-based test design technique should be used.



We know that states can be mistreating two different errors or faults and how we can categorize those faults in the state based embedded system behavior? The first is that the state chart do or

does not represent a correct translation or the systems functional specifications that means it is actually present in the functional transition a state behavior test result technique is not capable of revealing these types of faults because,

The state charts themselves are used as the basis of tests. So we not be able to identify the faults it will be because our tests are based on these state based functional specification it is a trolled specifications are switching and all that we will not be able to find the problem the second is called is state charts or syntactically it correct or inconsistent that means this faults can be revealed by static testing.

So these are some of the hues that we can come across while doing the embedded software testing using state analysis so this type of faults it could reveal the doing the testing itself or let us strategy what we want or alternates we need to find out after finding these faults. So second will happen because syntax errors are there or syntactically it is all implemented or it is not consistently behaving the implementation or it is wrong.

So these faults can be revealed by static testing so static testing anyway we will study it, so we have analysis or inspection methods with the help of that there are additional adding tools just or any other tools with the help of that we can reveal this kind of a inconsistent insist of input output in the events those kind of a problems are found out with the help of that as static testing if the faults found this way corrected.

Then the state has constitutes the basis or dynamic testing using state base testing analysis. So after the static testing this will be used for doing the state based testing the third cause is the transition sorry the translation from the state charts to code that means how the code is implemented from the state chart whether it is implemented exactly what is transition that is where the fault category lies.

It will becoming increasing the common that translation is performed automatically that means we have a generator holds generator from the models such as antics under doll net model based design they do they have a auto core feature where they generate the code from the model itself with the help of that they integrate and execute the code but it is not 100% that the code that is developed from the model isn't auto code.

Still within to heighten it to make it exactly what it is there, the code generated this way and exact representation of state charts within that the use of the state charts as which is used for the design it therefore not useful and the application of the state based design in super close, however the coding based on state charts is applied without the use of generator when a state based is tested and techniques will be used.

Auto code generator we have later we are not able to use then we should be stick to the testing with the state based testing and the technique what we have seen with the help of transitioning state based testing. The next we will study in detail about,

(Refer Slide Time: 40:05)

State Transition fault categories contd.

- The following faults can occur in statecharts and software:
- **1. States**
 - 1.1 States without incoming transitions (specification and/or implementation fault).
 - 1.2 Missing initial states – all paths in a statechart must be defined. In the case of a transition to a superstate in which the resulting substate is not indicated, the superstate must contain an initial state. If the initial state is not indicated, the state in which the transition terminates can not be predicted (specification and/or implementation fault).
 - 1.3 An additional state – the system turns out to have more states than is represented in the statechart (implementation fault).
 - 1.4 A missing state – a state that is shown on the statechart but is not present in the system (implementation fault).
 - 1.5 A corrupt state – a transition takes place to a non-valid state, leading to a system crash (implementation fault).



11

The faults different faults can occur for doing the state charts and a software having implemented the state techniques state analysis step, states can have issue states without incoming transitions that means without inputs specification and our implementation faults that means states are happening without any transitions. Second type of faults can occur missing initial states that means the inputs are not there all passed in the state chart must be prepared.

In the case of the transition the super state in which the result is substitute E is not indicated the super state must contain an initial state that means where we don't have a resulting sub style there should be a initial state at least to switch on if the easy state is not indicated the state at which that the ambition cannot be addicted we don't know where it is going to be terminated the states specification are implementation fault.

Specification could be wrong or the implementation could be wrong so this is one of the faults missing in each state third one is an additional state that means an extra state something like the system turns out to have a worst stage than it is represented in the statement which something like a dead line or a dead state, so it is again a problem with a implementation. So it implements it and it more states sop that those part cannot be covered.

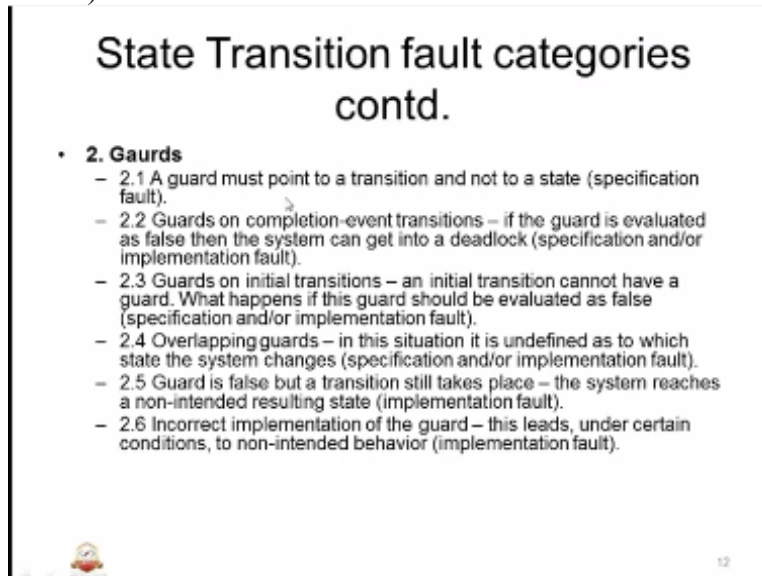
Or that is a fault so fourth one is the missing state that is one state is missing suppose to be it support sot have four states which for that each states are the one file with the help of the code then that is a fault. The state that is shown on the state chart but it is not presented in the system, the other one is states are there when particular state is complete the transition takes place to non valid states or the state is going to be not existing.

But the code is done which is going to go somewhere but it may crash or it may be predictable behavior will difficult to test sometimes so you say issue basically so we have specification or implementation issue where we will not going to have inconvenience state or some of the initial states are missed and implementation wise we have states function state we have missing states where there is no state at all and the states are there.

But it is corrupted so that system will not behave properly or it may crash so this some of the faults that can occur for th4e state charts. The second categories is on guards so guards be issue point to a vision and not to a state so guards are something like what is said in relating the way

event where continuously it is happening in terms of one example we can say by take is switching to easy that means present, so it is in guarded transition. So what are the faults that we can have in guarded or in guards?

(Refer Slide Time: 44:01)



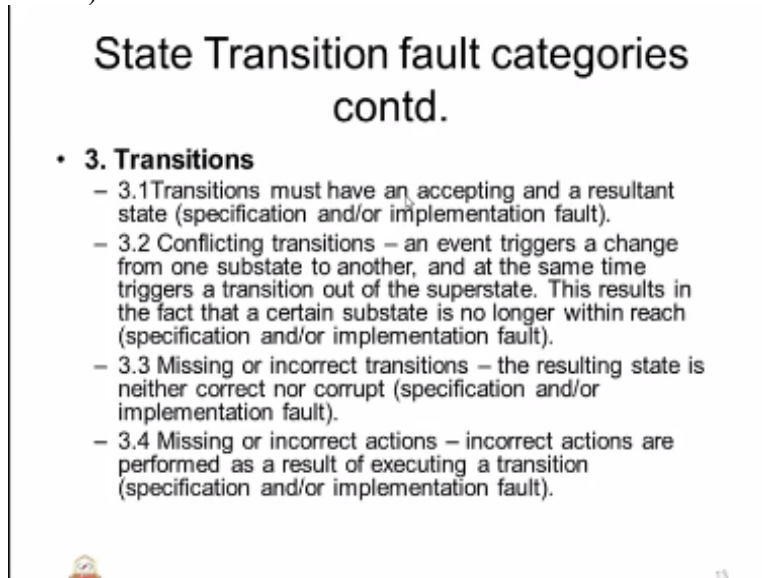
State Transition fault categories contd.

- **2. Guards**
 - 2.1 A guard must point to a transition and not to a state (specification fault).
 - 2.2 Guards on completion-event transitions – if the guard is evaluated as false then the system can get into a deadlock (specification and/or implementation fault).
 - 2.3 Guards on initial transitions – an initial transition cannot have a guard. What happens if this guard should be evaluated as false (specification and/or implementation fault).
 - 2.4 Overlapping guards – in this situation it is undefined as to which state the system changes (specification and/or implementation fault).
 - 2.5 Guard is false but a transition still takes place – the system reaches a non-intended resulting state (implementation fault).
 - 2.6 Incorrect implementation of the guard – this leads, under certain conditions, to non-intended behavior (implementation fault).

12

So called must point to a transition that not in a state always guard should be identified in transition what transitions are used? Transitions,

(Refer Slide Time: 44:22)



State Transition fault categories contd.

- **3. Transitions**
 - 3.1 Transitions must have an accepting and a resultant state (specification and/or implementation fault).
 - 3.2 Conflicting transitions – an event triggers a change from one substate to another, and at the same time triggers a transition out of the superstate. This results in the fact that a certain substate is no longer within reach (specification and/or implementation fault).
 - 3.3 Missing or incorrect transitions – the resulting state is neither correct nor corrupt (specification and/or implementation fault).
 - 3.4 Missing or incorrect actions – incorrect actions are performed as a result of executing a transition (specification and/or implementation fault).

13

So transition must have accepting and a resultant state conflicting the transitions have to be there and even triggers a change from one sub state to another sub state and at the same time triggers the transition out of the super state that means the main state itself is not able to achieve there is a conflict between different transitions it may happen to that state or this state while happening that we pick you out of the main state itself.

So this results in the fact that a certain sub state that no longer within reach that means it will never go to that sub state because in a complex system we ask in many states and many sub

states so without coming to the main states it is not possible to go for a sub states so I will take with raw as whichever is possible hear so that is a main straight to many states are been there and there are small sub states but into that next state suppose then we have a transition between these we know transition between these two we know and transitions from this we know.

This we know etc this is measure main state to this is called super state these are all sub states and we should have mechanism to this sub state through this main state similarly the other way also through so it cannot be conflicting for state transition between the sub states to main states to sub states instead all this will be predictable such a way that the state transition is executed. This is what the meaning of the reachable sub states should be reachable.

So confliction should not be there for the state transition. Next one is missing or incorrect transitions that means the answer I told you in the previous state itself the states are not there but transition is there sorry whereas the states are there but transitions are not proper all this is the result in state is a neither correct or nor correct in terms of switching there transition, so other one is the missing or incorrect actions.

So we know that different actions are there due to which the transition occur so that has to be action that could be missing or the actions are improper so incorrect actions are performed as a result of an executing a transition, specification issue and implementation issues all this could be of specification also or it is a implementation process, so transition and states these two are very much important in terms of testing the embedded systems where states are implemented.

(Refer Slide Time: 47:56)

State Transition fault categories contd.

• 4. Events

- 4.1 Missing event – an event is ignored (specification and/or implementation fault).
- 4.2 Hidden paths – a reaction to a defined event takes place, although this reaction is not defined in the statechart (also known as "sneak path") (implementation fault).
- 4.3 A reaction takes place to an undefined event (also known as a "trap door") (implementation fault).

So fourth one the last one is and of course we have a miscellaneous and we will go through that also. Events so we know that the events due to which the state will target we have a missing event a fault could be there in the missing event and event is ignored that means the events are not happening specification and or implementation fault that means events are happening but we don't know that why the vent is happening,

The reaction to a defined event explains how the reaction is not defined in this example that means the safe path it is also called as the safe path say implementation fault so some events are happening but there is no reason for that actions or something like that events are happening that

is but hidden oath event. The third one is a reaction takes place to a UN identified event that means a reaction is happening there is no event.

This is also known as the trap door that means it is a implementation called so reaction is the due to something but there is no event, this is the fourth type of faults in the category of states of embedded system.

(Refer Slide Time: 49:34)

State Transition fault categories contd.

- **5. Miscellaneous**

- The use of synchronization in an orthogonal region – a synchronization substate is situated in one orthogonal region without a connection to an event in another orthogonal region. A wrong implementation of the synchronization pseudostate can lead to rarely or not at all reachable states, no synchronization at all, or to a barely detectable deviation from the required system behavior.



15

The last one is miscellaneous this we will talk about synchronization in an orthogonal region a synchronization sub state is situated in one orthogonal region without a connection to an event in another orthogonal region that means we have an two spheres having different states within but there is no proper sequensation with the two regions. A wrong implementation of the synchronization at all can lead to rarely or not at all regions of states.

No synchronization at all or to a barely detectable deviation from the required system behavior, so this is basically talking about the synchronization between the say sub systems where both the sub systems lets us take a two sub systems and both sub systems have implemented the states technician so in this case, so the events that are happening within the sub systems and in between these sub systems for interaction between these two sub systems.

Or based on the synchronization, this has to be in that in order to work it properly so this is what is the fault could occur then in this region of the synchronization.

(Refer Slide Time: 51:19)

State transition Fault categories coverage

Table 11.3
Coverage of the fault categories

Fault category	Checklist	State transition technique
1.1		✓
1.2	✓	✓
1.3		✓
1.4		✓
1.5		✓
2.1	✓	
2.2	✓	✓
2.3	✓	✓
2.4 ¹	✓	✓
2.5		✓
2.6 ²		
3.1	✓	✓
3.2		✓
3.3		✓
3.4		✓
4.1		✓
4.2		✓
4.3 ³		
5.1	✓	

¹ This fault category can be tested by both a checklist and the test design technique. The checklist suffices as long as code is generated from the model. However, if programming is based on the model then this fault category should also be included using the test design technique.
² Category 4.3 is not covered by the test design technique. Error guessing is, in this case, the only available option. Category 2.6 is very difficult to detect with a test design technique – statistical usage testing is probably the only technique which has a chance of detecting this type of fault.



And we will try to have a small table state transition fault category coverage how it is defined this is the last slide so coverage of the fault categories we will define the different faults in 1.1, 1.2, 1.3 so this is check list straight on transition technique we can apply or not. So based on this fault categories are applied this different fault categories are all what we have about discussed here 4.1, 4.2, 4.3 then 3.1 to 3.4, 2.1 guards.

States have 1 to 5 likewise we have different faults. So we have a check list, check list in the sense whether it is a right or it is wrong base on that we are going to have it while defining the state transition techniques so very important so that the coverage's so that is what the meaning of this you can have this, this, this have not covered embedded software testing, so we know that this check list with the help of this we have the state transition techniques.

So this point category can be tested by both the check list and the test design technique that means we are going to have a coverage for all that is 2.4 can be covered with the help of the test design technique and with the help of check lit also, category 4.3 is not covered by the test design technique arrow based technique is probably touch wise so at on this something like we identify an error or a fault this is just see what could be the problem.

So this will happen during the dynamic exhibition only because the tester will have an test driving or the system of how to executing only when he is doing some other tests he will guess some errors while doing some other tests so 4.3 is what type of sink it is an reaction takes place to an undefined event so we know that as per specification we will have tested but what will happen that test is that some reaction is happening.

But that is no event for that so trap door it is also called so that is what the category is about this can be done with the help of errors based, so basically it will help in terms of coverage of state transition this is the elaboration of this so you will continue this transition with some exercise and other testing mechanisms and methods I the next class so that is for this session.