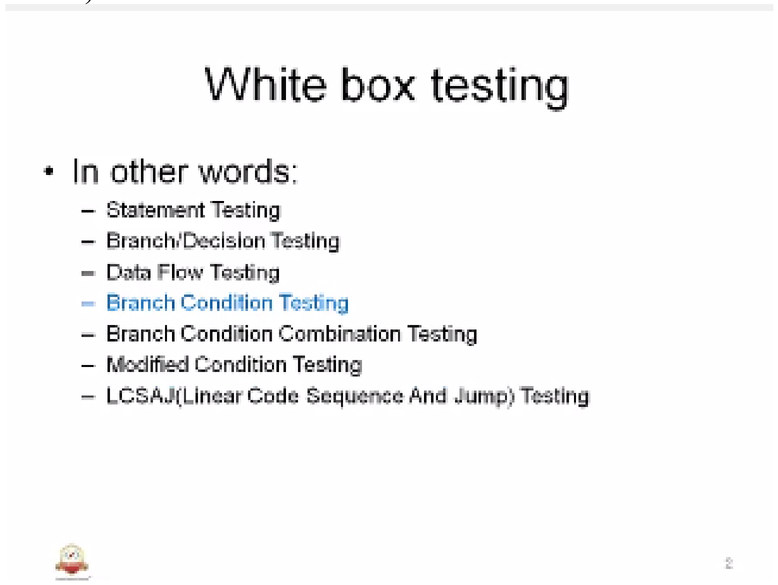
Before joining the next session of embedded software testing,
(Refer Slide Time: 00:08)

Embedded Software Testing
Unit 2: Testing Methods

Lecture 20
Seer Akademi – NPTEL MOU

1

Unit 2 series we got to move on to this second unit with couple of sessions we will take out practical and we are more focusing on the embedded software testing white box testing aspect and technique, we will study about the white box testing techniques in more details and in continuation of the earlier sessions.
(Refer Slide Time: 00:38)

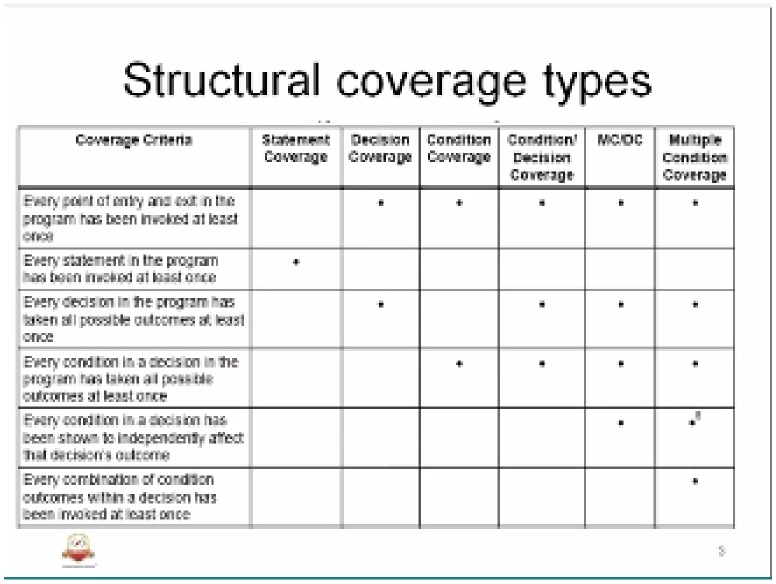


White box testing

- In other words:
  - Statement Testing
  - Branch/Decision Testing
  - Data Flow Testing
  - Branch Condition Testing
  - Branch Condition Combination Testing
  - Modified Condition Testing
  - LCSAJ(Linear Code Sequence And Jump) Testing

2

Today we will study about the branch condition testing, branch condition combination testing, modified condition testing and LCSAJ testing. So before that we just touch base on to the previous session here we had gone through a table of structural coverage analysis or structural coverage types,
(Refer Slide Time: 01:02)



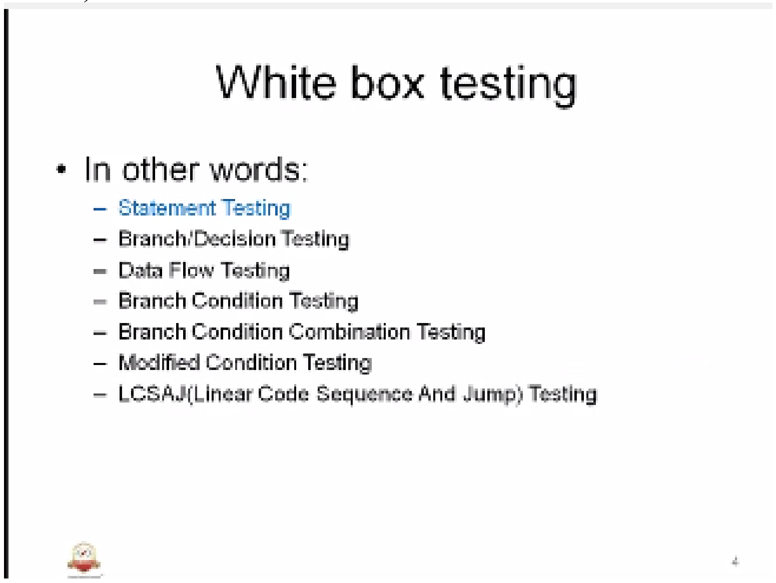
Structural coverage types

Coverage Criteria	Statement Coverage	Decision Coverage	Condition Coverage	Condition/ Decision Coverage	MC/DC	Multiple Condition Coverage
Every point of entry and exit in the program has been invoked at least once		•	•	•	•	•
Every statement in the program has been invoked at least once	•					
Every decision in the program has taken all possible outcomes at least once		•		•	•	•
Every condition in a decision in the program has taken all possible outcomes at least once			•	•	•	•
Every condition in a decision has been shown to independently affect that decision's outcome					•	•[8]
Every combination of condition outcomes within a decision has been invoked at least once						•

3

Basically which is used for structural coverage analysis, structural coverage analysis are extremely very important analysis method we are using for industries like aero space where the analysis meet the provide with the structural like, data, space industries manufactures, such as buying air bus etc..

(Refer Slide Time: 01:35)



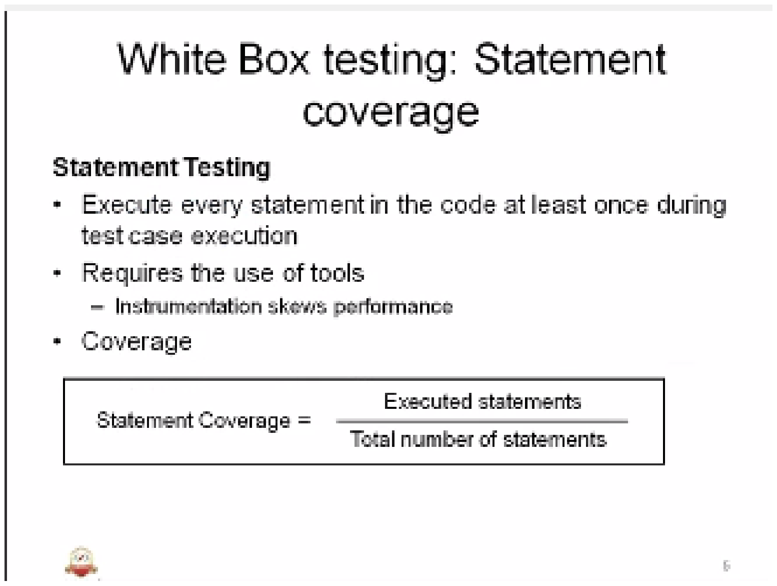
White box testing

- In other words:
  - Statement Testing
  - Branch/Decision Testing
  - Data Flow Testing
  - Branch Condition Testing
  - Branch Condition Combination Testing
  - Modified Condition Testing
  - LCSAJ(Linear Code Sequence And Jump) Testing

4

We started the white box testing technique in the statement testing the coverage to cover the,

(Refer Slide Time: 01:41)



White Box testing: Statement coverage

Statement Testing

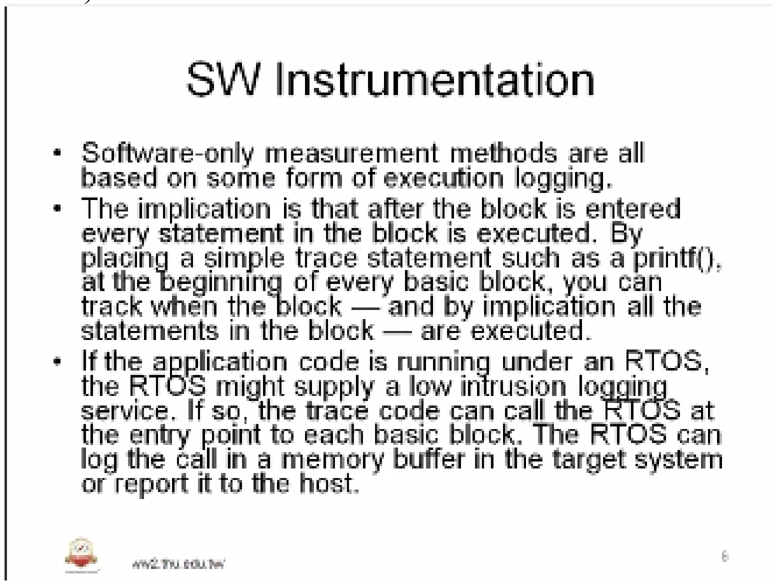
- Execute every statement in the code at least once during test case execution
- Requires the use of tools
  - Instrumentation skews performance
- Coverage

$$\boxed{\text{Statement Coverage} = \frac{\text{Executed statements}}{\text{Total number of statements}}}$$

6

Statements that are in the code and we also learned about software instrumentation,

(Refer Slide Time: 01:48)



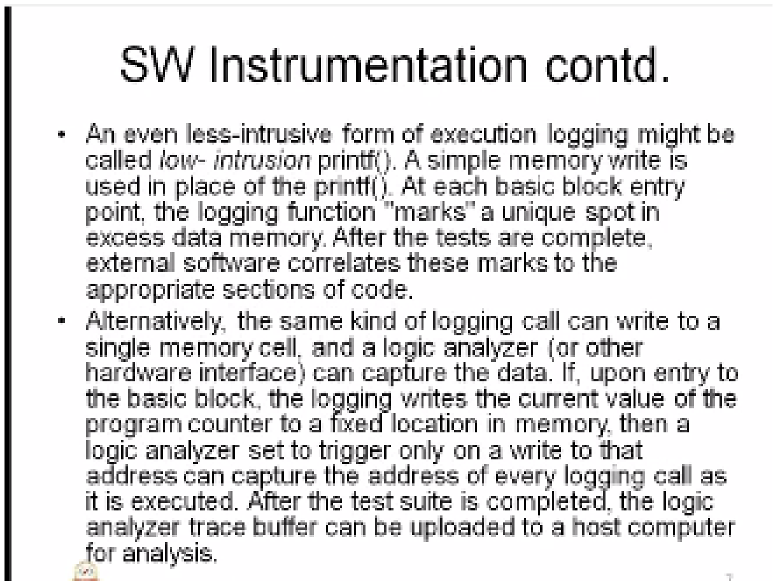
SW Instrumentation

- Software-only measurement methods are all based on some form of execution logging.
- The implication is that after the block is entered every statement in the block is executed. By placing a simple trace statement such as a printf(), at the beginning of every basic block, you can track when the block — and by implication all the statements in the block — are executed.
- If the application code is running under an RTOS, the RTOS might supply a low intrusion logging service. If so, the trace code can call the RTOS at the entry point to each basic block. The RTOS can log the call in a memory buffer in the target system or report it to the host.

ww2.thu.edu.tw 6

Which does the,

(Refer Slide Time: 01:49)

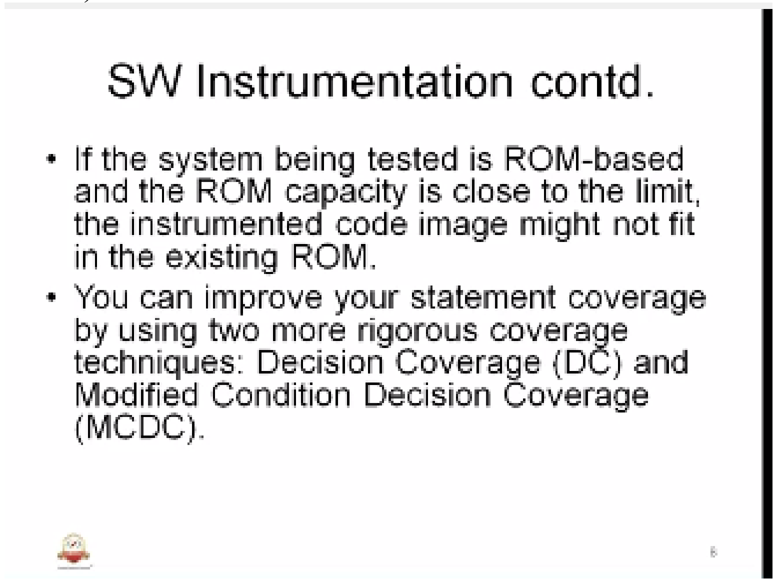


SW Instrumentation contd.

- An even less-intrusive form of execution logging might be called *low- intrusion* printf(). A simple memory write is used in place of the printf(). At each basic block entry point, the logging function "marks" a unique spot in excess data memory. After the tests are complete, external software correlates these marks to the appropriate sections of code.
- Alternatively, the same kind of logging call can write to a single memory cell, and a logic analyzer (or other hardware interface) can capture the data. If, upon entry to the basic block, the logging writes the current value of the program counter to a fixed location in memory, then a logic analyzer set to trigger only on a write to that address can capture the address of every logging call as it is executed. After the test suite is completed, the logic analyzer trace buffer can be uploaded to a host computer for analysis.

Inclusive hooks in terms of entire for the marks which are required within the code so that the cordage will be traced, it is a tracing mechanism it is called software instrumentation.

(Refer Slide Time: 02:03)

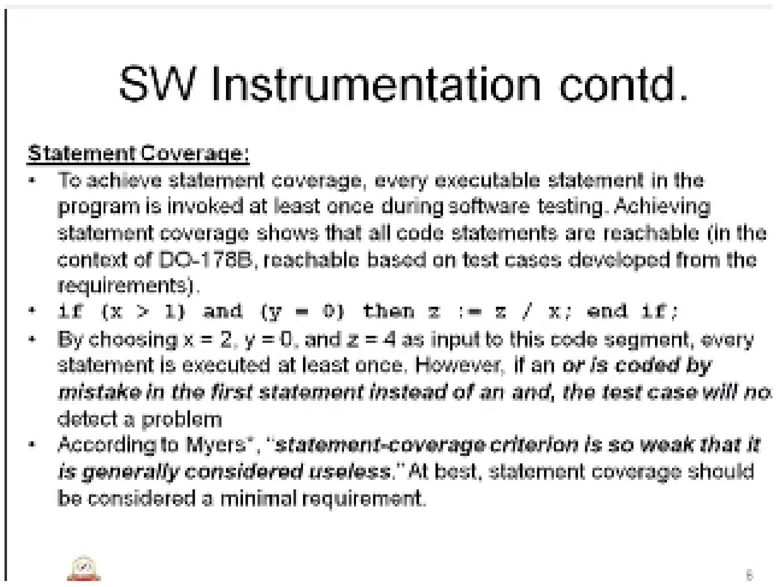


SW Instrumentation contd.

- If the system being tested is ROM-based and the ROM capacity is close to the limit, the instrumented code image might not fit in the existing ROM.
- You can improve your statement coverage by using two more rigorous coverage techniques: Decision Coverage (DC) and Modified Condition Decision Coverage (MCDC).

Since the code basis is huge, so we cannot afford to have it manually. So there are number of tools that are used for software instrumentation like, vector caste LDRA RTRT, etc., with the help of instrumentation. We run the unit testing statement coverage and coverage due to in terms of percentage,

(Refer Slide Time: 02:26)



SW Instrumentation contd.

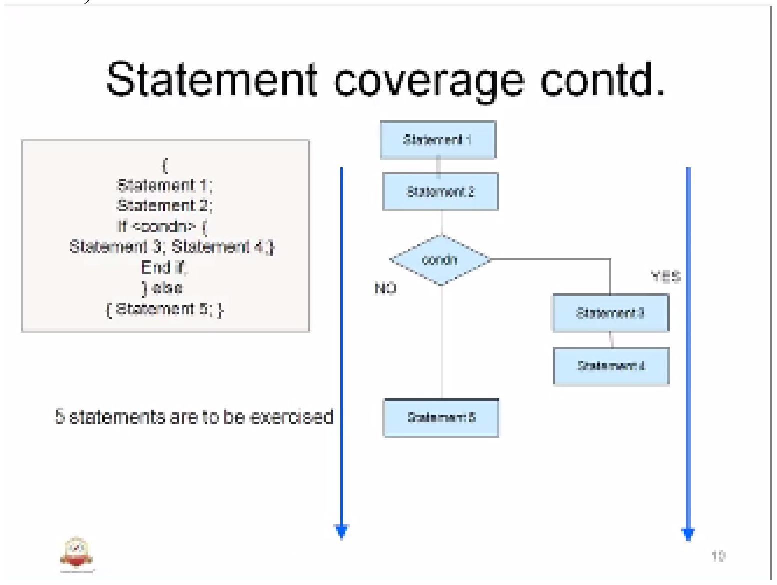
Statement Coverage:

- To achieve statement coverage, every executable statement in the program is invoked at least once during software testing. Achieving statement coverage shows that all code statements are reachable (in the context of DO-178B, reachable based on test cases developed from the requirements).
- `if (x > 1) and (y = 0) then z := z / x; end if;`
- By choosing x = 2, y = 0, and z = 4 as input to this code segment, every statement is executed at least once. However, if an ***or is coded by mistake in the first statement instead of an and, the test case will no*** detect a problem
- According to Myers*, "***statement-coverage criterion is so weak that it is generally considered useless.***" At best, statement coverage should be considered a minimal requirement.

9

If the percentage is not 100% we will need a manual analysis of gaps and the gaps could be in terms of decisions also.

(Refer Slide Time: 02:35)



Statement coverage contd.

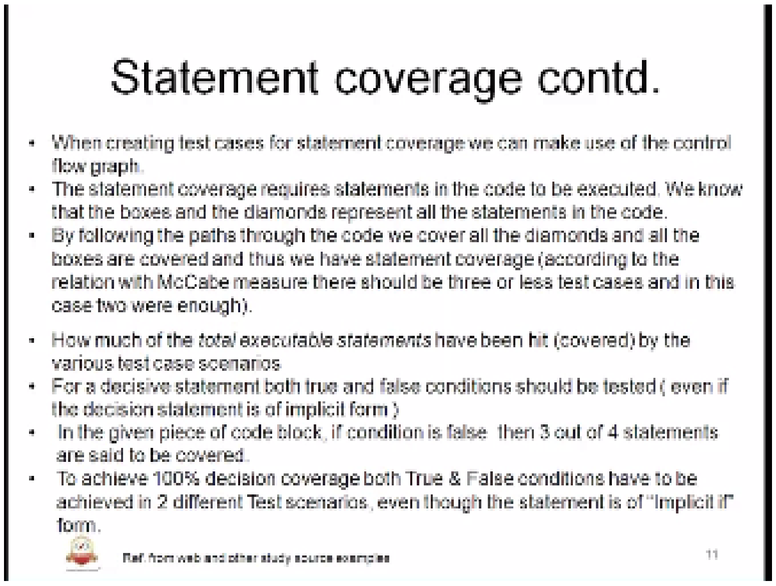
```
{
Statement 1;
Statement 2;
If <condn> {
Statement 3; Statement 4;}
End if,
} else
{ Statement 5; }
```

5 statements are to be exercised

10

So in that case what we do? Branch decision coverage we will take care where the coverage is done, the type of total number of executions divided by total number of decisions multiplied by 2, because 2 times it has to be the decisions box need to be analyzed. Why because there are,

(Refer Slide Time: 02:55)



Statement coverage contd.

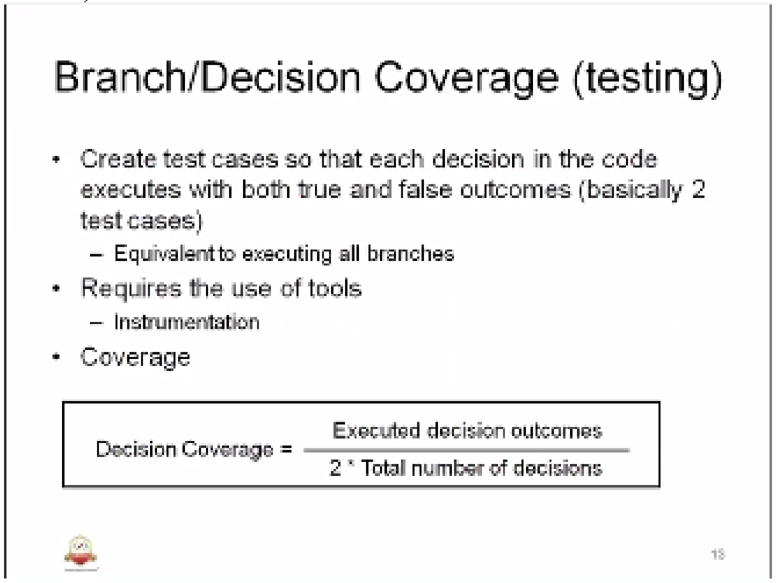
- When creating test cases for statement coverage we can make use of the control flow graph.
- The statement coverage requires statements in the code to be executed. We know that the boxes and the diamonds represent all the statements in the code.
- By following the paths through the code we cover all the diamonds and all the boxes are covered and thus we have statement coverage (according to the relation with McCabe measure there should be three or less test cases and in this case two were enough).
- How much of the *total executable statements* have been hit (covered) by the various test case scenarios
- For a decisive statement both true and false conditions should be tested (even if the decision statement is of implicit form)
- In the given piece of code block, if condition is false then 3 out of 4 statements are said to be covered.
- To achieve 100% decision coverage both True & False conditions have to be achieved in 2 different Test scenarios, even though the statement is of "Implicit if" form.

Ref. from web and other study source examples

11

Fourth comes of the decision box that is diamond box.

(Refer Slide Time: 02:59)



Branch/Decision Coverage (testing)

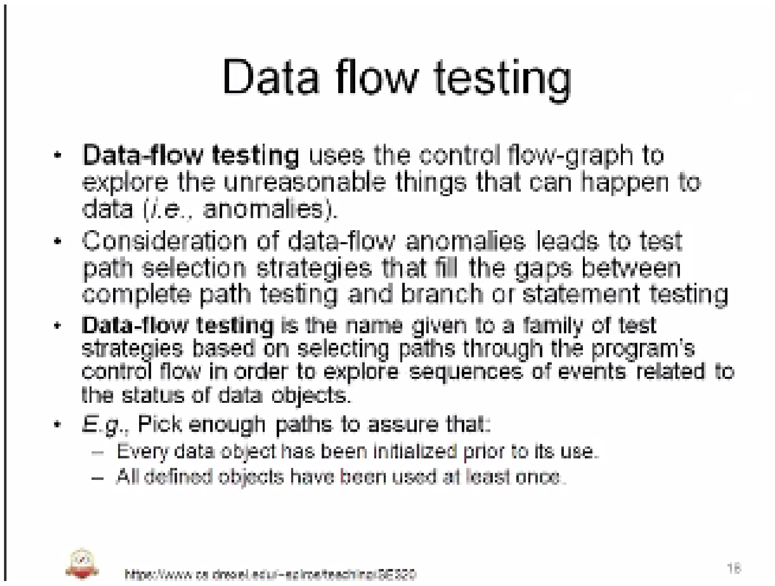
- Create test cases so that each decision in the code executes with both true and false outcomes (basically 2 test cases)
  - Equivalent to executing all branches
- Requires the use of tools
  - Instrumentation
- Coverage

$$\text{Decision Coverage} = \frac{\text{Executed decision outcomes}}{2 * \text{Total number of decisions}}$$

18

It could be true or it could be false, and in continuation of the white box testing, we have gone through the data flow testing where the way of the data optics or the analyzed in terms of the flow and

(Refer Slide Time: 03:20)



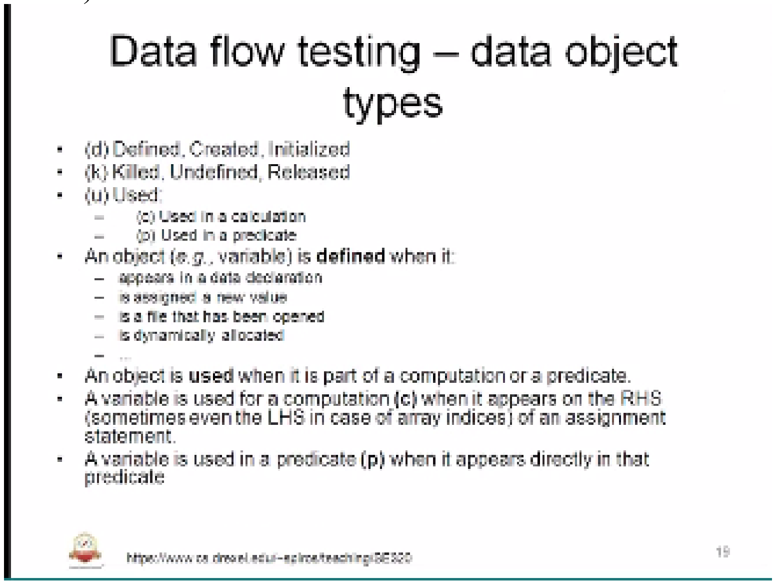
Data flow testing

- **Data-flow testing** uses the control flow-graph to explore the unreasonable things that can happen to data (*i.e.*, anomalies).
- Consideration of data-flow anomalies leads to test path selection strategies that fill the gaps between complete path testing and branch or statement testing
- **Data-flow testing** is the name given to a family of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects.
- *E.g.*, Pick enough paths to assure that:
  - Every data object has been initialized prior to its use.
  - All defined objects have been used at least once.

https://www.cs.drexel.edu/~spiros/teaching/SE320 18

Every data that are used or the objects that are used,

(Refer Slide Time: 03:25)



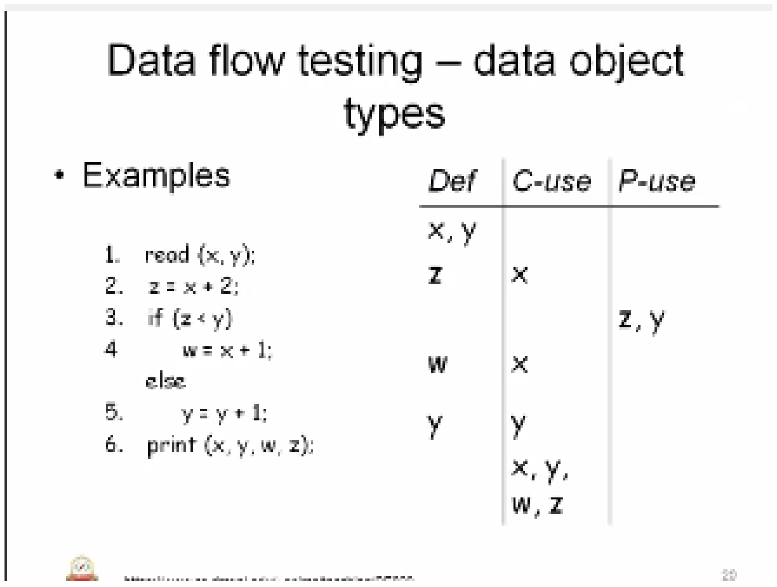
Data flow testing – data object types

- (d) Defined, Created, Initialized
- (k) Killed, Undefined, Released
- (u) Used:
  - (c) Used in a calculation
  - (p) Used in a predicate
- An object (*e.g.*, variable) is **defined** when it:
  - appears in a data declaration
  - is assigned a new value
  - is a file that has been opened
  - is dynamically allocated
  - ...
- An object is **used** when it is part of a computation or a predicate.
- A variable is used for a computation **(c)** when it appears on the RHS (sometimes even the LHS in case of array indices) of an assignment statement.
- A variable is used in a predicate **(p)** when it appears directly in that predicate

https://www.cs.drexel.edu/~spiros/teaching/SE320 19

Will need to be analyzed and tested here. The data object types could be DKU used could be it is used in the calculations or used in a predicate, so in the objectives defined when it is in the data declaration of the new value or just been opened or the3 file has been opened that is also called as an object. And is a memory using in variable is allocated is been done with it or using constants or the allocations we can use so this data object types are.

(Refer Slide Time: 04:05)



Data flow testing – data object types

- Examples

```
1.  read (x, y);
2.   z = x + 2;
3.  if (z < y)
4       w = x + 1;
    else
5.      y = y + 1;
6.  print (x, y, w, z);
```

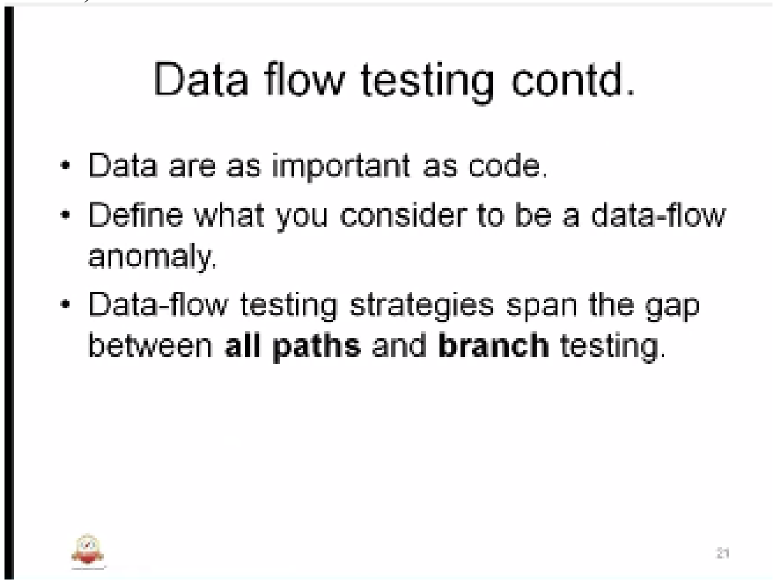
Def	C-use	P-use
x, y		
z	x	
		z, y
w	x	
y	y	
	x, y, w, z	

20

Used for data flow testing where segregate different types of data in columns such as definition C use, P use is a calculation purpose you see X, Y, W, C in this example used as C use or confession use that as P use or the predicate use, is Z and Y you can see if bracket within that Z and Y is closed so you can see in the predicate value directly in the bracket so that is why it is called as C use.

All these data will be observed and analyzed in term of data flow testing is should be tested against the indented functionality or the specification of the system alert test.

(Refer Slide Time: 05:02)



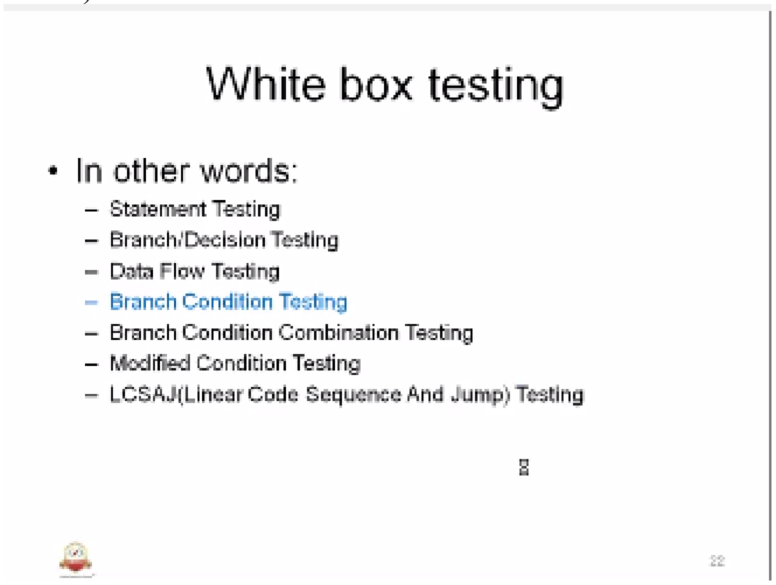
Data flow testing contd.

- Data are as important as code.
- Define what you consider to be a data-flow anomaly.
- Data-flow testing strategies span the gap between **all paths** and **branch** testing.

21

So there gaps between paths and the statements and the branch or the decision or the data flow will obment it in terms of strategies that can happen, so each of the client of code and the data objects will analyzed and using different values for example x in to y as per it can take value between 0 to 10 we know that how we can provide inputs 0 to 10 with the help of equals class and boundary value on the things also will depends the if condition below so that it is satisfied it could take the path within these or it could take the path within the else, so all these gaps will be analyzed with the help of data flow testing.

(Refer Slide Time: 05:53)



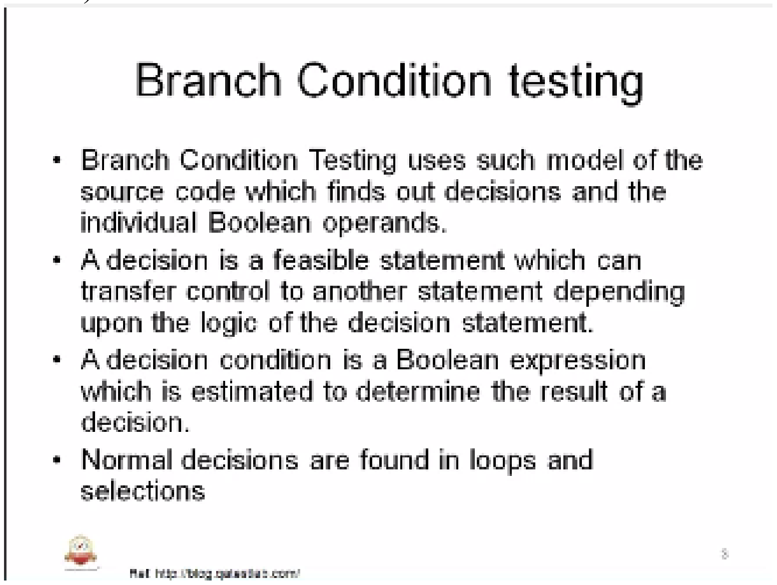
White box testing

- In other words:
  - Statement Testing
  - Branch/Decision Testing
  - Data Flow Testing
  - Branch Condition Testing
  - Branch Condition Combination Testing
  - Modified Condition Testing
  - LCSAJ(Linear Code Sequence And Jump) Testing

22

Our next is the branch condition testing which we will study today, so what is the branch condition testing?

(Refer Slide Time: 06:02)



Branch Condition testing

- Branch Condition Testing uses such model of the source code which finds out decisions and the individual Boolean operands.
- A decision is a feasible statement which can transfer control to another statement depending upon the logic of the decision statement.
- A decision condition is a Boolean expression which is estimated to determine the result of a decision.
- Normal decisions are found in loops and selections

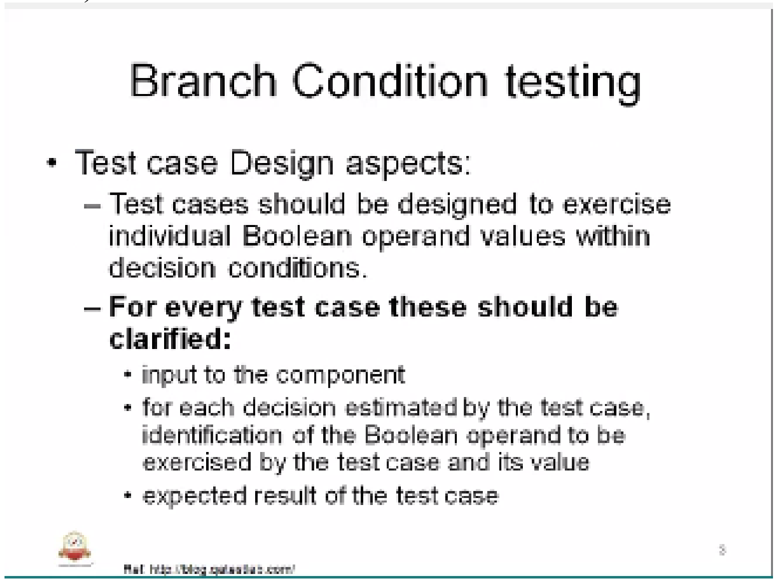
Ref: http://blog.qatestlab.com/

3

Branch condition testing uses such model of the source code which finds out decisions on the individual Boolean operands. We know Boolean operands such as less than, greater than, less than equal to, greater than equal to etc. so that decisions coming out of that are tested with the help of branch condition testing. A decision is a feasible statement which can transfer control to another statement depending upon the logic of the decision statement.

This decision having a logic, having the Boolean operands and as a result of that the control can be transferred to different statements or single statement depending on the logic that is been applied within the decision statement and a decision condition is a Boolean expression which is

estimated to determine the result of a decision, so decision basically the condition so it will have the Boolean expression in terms of lesser than, greater then, less than equal to, greater than equal to or equal to etc.
So that we will decide basically the result of the decision whether it is one control or one sort of statements or it should take another sort of statements, so basically these decisions are used in loops and selections. So loops you know it could be a why loop or for loops or do why likewise we have based on the land like constructs we have different groups similarly collection sin terms of cases we know that.
So this decision is implied within this type of statements, you will see in of this how branch condition testing is taken care, so in terms of branch condition testing.
(Refer Slide Time: 08:16)



Branch Condition testing

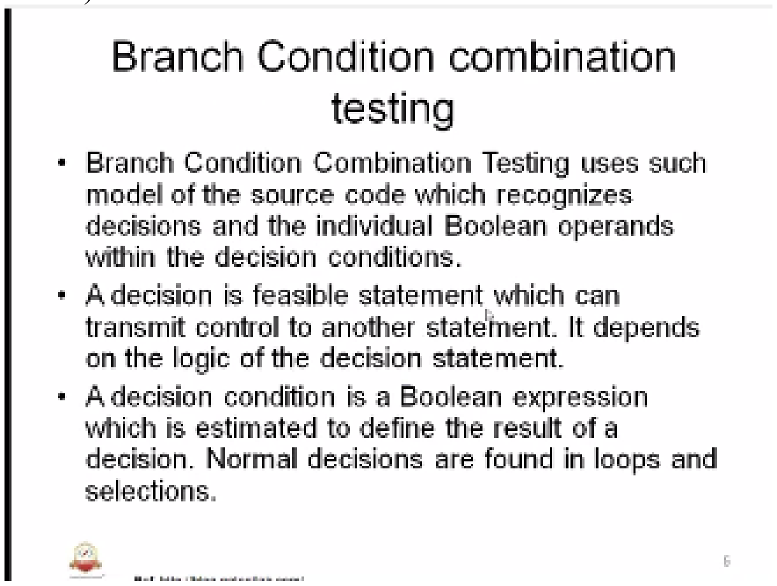
- Test case Design aspects:
  - Test cases should be designed to exercise individual Boolean operand values within decision conditions.
  - **For every test case these should be clarified:**
    - input to the component
    - for each decision estimated by the test case, identification of the Boolean operand to be exercised by the test case and its value
    - expected result of the test case

Ref: http://blog.qatestlab.com/

8

Test case design aspects how we do have a test case design aspect? So before that, so 100% condition coverage means that test data clears the coverage's also saying that how you are going to do but 1005 coverage here means that the data will ensure that every condition that is executed at least once during the testing coverage that will include also the branch condition don't get confused with the branch or decision control here condition coverage called the in terms of the Boolean operands and the extensions part of the decisions will be tested.
And in terms of designing the branch condition testing test cases should be designed to exercise individual Boolean operand values within decision conditions. For every test case these should be clarified before we do the design we should have an input to the component that is part of the decision condition, for each decision estimated by the test case, identification of the Boolean operand to be exercised by the test case and its values.
Expected result of the test case, so these three should be clear for the tester before re designs so all the Boolean operands within the decision condition will have to be exercised operand values such as you can see an expression, let's start with a for loop for I is 02 I less than 50 I ++ and A less than B and in to this things etc. It could be multiple statements as well as a complied lock also, will be called depending on the nature of the performance.

So here we have a for loop this lock will be executed in terms of 02 less than 50 means 50 times it will be executed, here the index is I so obviously the index will have a incremental or the incremental already there is based on the index certain conditions are used and you can see on the right hand side along with this it is very important to have this right hind side expression also to be satisfied are revaluated before the lock is getting executed.
So we know that statements embedded statements and we know that there is a block A less than B we are going to do it and true it will done if false this will not be executed and now since we have A less than B as a condition branch condition we need to evaluate all the aspects of A as well as B in order to test the Boolean operands, so the Boolean operands could be logical expressions such as or and etc.
So basically the test or needs to respond what are the inputs to this lock that needs to be taken care for each decision estimated in the test case identification of the Boolean operand to be exercised with the test case this are the action that lock is going to do for each of these Boolean operands expression? In terms of inputs will be exercised and also it carries out in terms of execution.
In codes what is expected as for each condition while it is getting executed that is also very important to understand. So that is about branch condition testing and next one we have is slide modification branch condition, branch condition combination testing here what we do?
(Refer Slide Time: 14:08)

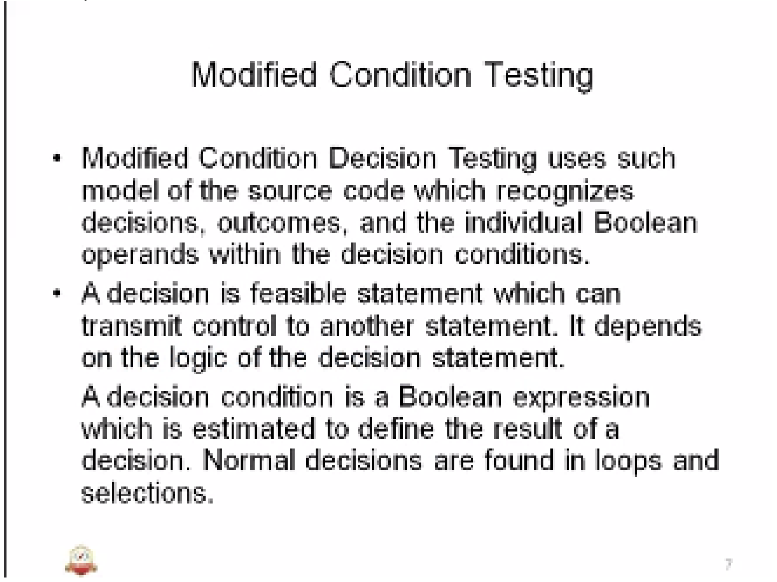


Branch Condition combination testing

- Branch Condition Combination Testing uses such model of the source code which recognizes decisions and the individual Boolean operands within the decision conditions.
- A decision is feasible statement which can transmit control to another statement. It depends on the logic of the decision statement.
- A decision condition is a Boolean expression which is estimated to define the result of a decision. Normal decisions are found in loops and selections.

Branch condition combination testing uses such as model of the source code which recognizes decisions and the individuals Boolean operands within the decision conditions, that mean there could be decision within the operands itself during operands so that also will be exercised. So whereas in case of branch condition alone what we will do, we take care of the decisions and the individual Boolean operands.
Here we take care of decisions and the individual Boolean operands in the decision conditioning so we have decisions also we have lot of Boolean operands or expressions or operands or path of the decisions that also will be exercised in sever they get as I mean so that is why it is called as a

branch condition combination testing. A decision is feasible statement which can transmit control to another statement we know that, it depends on the logic of the decision statement.
That means how the logic is going to be derived based on that decision is feasible decision is called a feasible statement. A decision condition is a Boolean expression which is estimated to define the result of a descion that means it is a condition basically Boolean expression condition which is testing at a define the result of a decision that means the Boolean expression will have drive the values in such a way as the decision outcome will be earned.
Normal decisions are found in loops and selections same way as the we have combination of that Boolean operands within the decision condition are tested with the help of this so each of this Boolean operand is need to be tested or the variables is taking care. So test cases design how we are going to do? For branch condition combination test cases should be designed left side combination of the Boolean operand values within decision condition.
For every test case these should be clarified the inputs to the components, for every decision estimated by the test case identification of the combination of Boolean operands and their values the expected results of the test case. So similar to the branch condition testing in addition we have a combination of the Boolean operands within the decisions the operands all that need to be exercised so that is the purpose of branch condition combination. So next one is about modified condition testing,
(Refer Slide Time: 16:57)



Modified Condition Testing

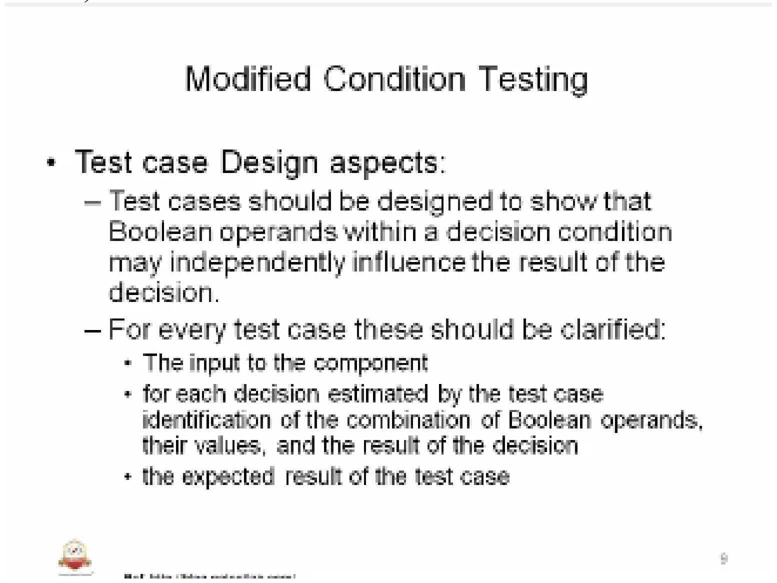
- Modified Condition Decision Testing uses such model of the source code which recognizes decisions, outcomes, and the individual Boolean operands within the decision conditions.
- A decision is feasible statement which can transmit control to another statement. It depends on the logic of the decision statement.

  A decision condition is a Boolean expression which is estimated to define the result of a decision. Normal decisions are found in loops and selections.

7

So what modified condition testing does? Modified condition decision testing uses such model of the source code which recognizes decision, outcomes, and the individual Boolean operands within the decision conditions here additionally we have outcomes also important of this output. What sort of output the speck or this function expected? So based on the need of that we are going to test it how we are going to test it?
It is with the help of all the decisions that means decisions, individual Boolean operands, decision conditions and the outcomes of the result that is also a deciding factor for the modified condition testing. So we need to have all this possible combination in terms of modified

condition testing. A decision is a feasible statement which can transmit control to another statement. It depends on the logic of the decision statement.
Again the logic is very important here, based on how and what are the values that we are going to type on this it is going to decide the decision output as well as the Boolean operand within the decision conditions. So a decision condition is a Boolean expression which is estimated to define the result of a decision. Normal decisions are found in loops and selections. So a grand this whole we expect some result and the result we want as per the specification.
And that result we want to achieve and to be done with the help of such values which can be driven within the Boolean expression part of the decision conditions, that is how modified condition testing is executed may be we will touch this,
(Refer Slide Time: 18:56)



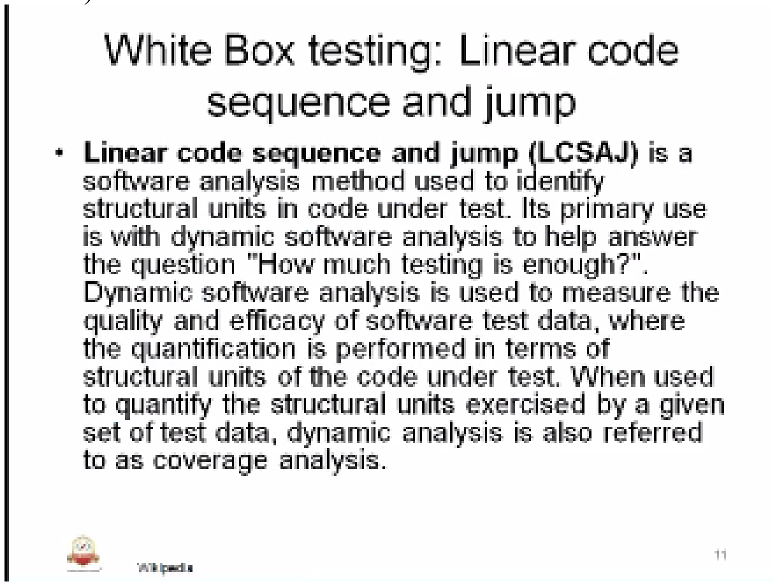
Modified Condition Testing

- Test case Design aspects:
  - Test cases should be designed to show that Boolean operands within a decision condition may independently influence the result of the decision.
  - For every test case these should be clarified:
    - The input to the component
    - for each decision estimated by the test case identification of the combination of Boolean operands, their values, and the result of the decision
    - the expected result of the test case

9

Modified condition with decision coverage in detail with respect to new cinerary process which is followed in the aspects in this, Test case design aspects, test cases should be designed to show that Boolean operands within a decision condition may independently influence the result of the decision. That means to understand here suppose variable A and B are used to derive C, C is supposing horizontal.
So this modified condition is such that independency need to be verified that means because of A I am going to give the result in C and because B I am keeping A as constant so you will have two paths on this, so like this we will have multiple condition and Boolean expressions are in number of various groups and logic that in all the cases need to measure the independency it is very important.
Independency need to be verified so that is what the modified condition testing because A is having some value such as 0 and sorry and B equalizes 0 so what will happen A and B will become 0 which could result in something processing some result 1 to say so nearly we have proven that when A has become 0 C the result of the operation will have an impact on the output, similarly when A is 0 B any value 1, 2, 3 whatever it is.
But when the impact is due to A because A added with any of the value within thin the B will result in unknown value it could be a result wrong depending on whatever the expression is so A

and B will drive independent, 1 will keep it as constant like B as we will keep 1 next to the sorry we will keep A as constant next time we will verify B in terms of 1, 2, 3 etc so the independency for the B is been verified.
So that is how we are going to have so you have the independency see that the independency is achieved, because of 1 path or 1 input the decision is condition is going to go so that is what it means test cases should be designed to show that Boolean operands within a decision condition may independently increase about the decision whatever input here implement the Boolean expression as an so we are going to verify the operands for its independency. For every test case B should be valid we need to have a input what are the inputs that are going to drive for each decision estimated by the test case identification of the combination of Boolean operands, their values and the result of the decision the expected result of the test case that also need to be understood before the modified condition testing.
So that is about modified condition testing, the next one is linear code sequence and jump,
(Refer Slide Time: 23:23)



White Box testing: Linear code sequence and jump

- **Linear code sequence and jump (LCSAJ) is a** software analysis method used to identify structural units in code under test. Its primary use is with dynamic software analysis to help answer the question "How much testing is enough?". Dynamic software analysis is used to measure the quality and efficacy of software test data, where the quantification is performed in terms of structural units of the code under test. When used to quantify the structural units exercised by a given set of test data, dynamic analysis is also referred to as coverage analysis.

Wikipedia 11

This generally we follow not with this movement indirectly through modified condition testing will add it certain specific industries or applications that report to have this, so this is a software analysis method basically used to identify structural units in code under test. Its primary use is with dynamic software analysis top help answer the question how much testing is enough? That means basically it will have an analysis of how much is tested?
In terms of dynamic software behavior or analysis, dynamic software analysis is used to measure the quality and efficacy of software test data, where the quantification is performed in terms of structural units of the code under test that is based on the structural unit and its behavior or the functionality the dynamic software analysis is used so it used to measure the quality identification of the software test data.
When used to quantify the structural units exercised the given set of test data, dynamic analysis is also referred to as coverage analysis. So when we are done with the dynamic analysis we will also come up with the coverage analysis, so that is how this is done basically to drive how much

of the testing based on the test data is needed is what is been derived for the linear code sequence and jump testing.

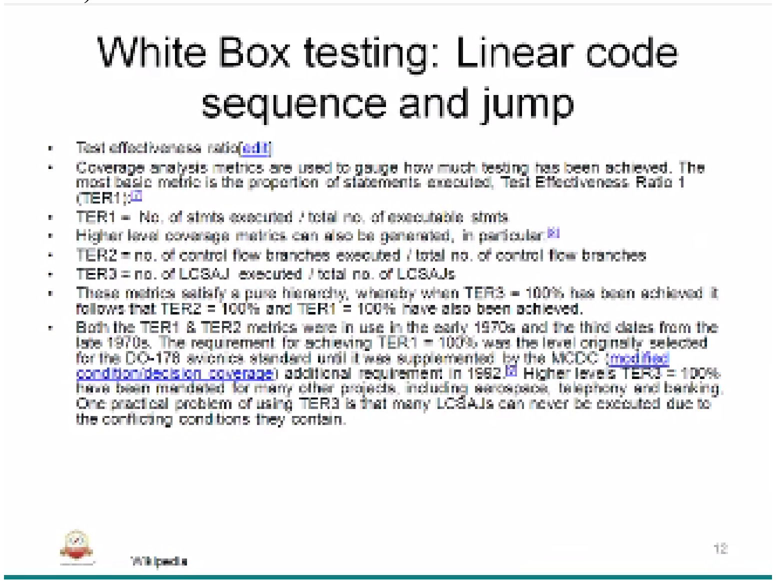
Is basically an software code path fragment consisting of a sequence of code that means a linear code sequence followed by a control code jump in consist of the free items basically start of the linear sequence of executable segments at end of the sequence and target line in which the control flow is transferred at this end of this, suppose function 1, 2 I am just mentioning in numbers like which one in number of function laws are executed by the fragments are there in the function.

so what do we do with the LCSAJ basically how much are you test for this is what is going to tell that is its basically code path, code path fragment I may test1, I may test 2, I may test some by randomly N 20 may be N. so that means consisting of sequence of code it is called linear code sequence. And jumping to the different linear it is called jump control flow jump basically so what do we do? Do you have a code path fragment?

Consistent of sequence of code linear code sequence followed by a control flow jump so we start with this and then we will have a control flow jump and basically the jump could be achieved with a following three types that is start, then end you know that we need to have end for the function in law and the target line. That means with the help of this target line how we have jumped to the end basically.

That mean the target line each control flow is transferred at the end of the sequence, so with the help of this will basically used where we have numerous functionality having expected output and similar sort of inputs you have some of the functions so that is how you use the LCSAJ method. So in the continuation of this basically LCSAJ method will have the TER which is called the test effective ratio.

(Refer Slide Time: 28:46)



White Box testing: Linear code sequence and jump

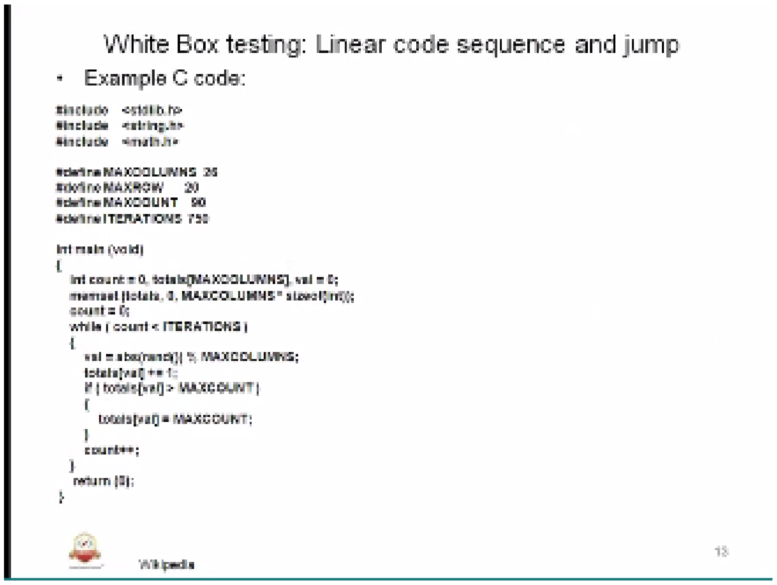
- Test effectiveness ratio[edit]
- Coverage analysis metrics are used to gauge how much testing has been achieved. The most basic metric is the proportion of statements executed, Test Effectiveness Ratio 1 (TER1):[7]
- TER1 = No. of stmts executed / total no. of executable stmts
- Higher level coverage metrics can also be generated, in particular:[8]
- TER2 = no. of control flow branches executed / total no. of control flow branches
- TER3 = no. of LCSAJ executed / total no. of LCSAJs
- These metrics satisfy a pure hierarchy, whereby when TER3 = 100% has been achieved it follows that TER2 = 100% and TER1 = 100% have also been achieved.
- Both the TER1 & TER2 metrics were in use in the early 1970s and the third dates from the late 1970s. The requirement for achieving TER1 = 100% was the level originally selected for the DO-178 avionics standard until it was supplemented by the MCDC (modified condition/decision coverage) additional requirement in 1992.[9] Higher levels TER3 = 100% have been mandated for many other projects, including aerospace, telephony and banking. One practical problem of using TER3 is that many LCSAJs can never be executed due to the conflicting conditions they contain.

Wikipedia

12

So what is test effective ratio? It is number of statement executed by number of effective statements, number of control flow branches executed by total number of control flow branches they have number of LCSAJ executed by total number of LCSAJ. Altogether you can have a

TER that is effectiveness ratio how much of the code we have sequenced and jump in terms of testing.
So accordingly we will have this sort of testing and coverage also will take care of this, coverage or metrics are used to botch and hours testing has been achieved the most terrific metrics is proportion of statements test effective ratio1 TER1, is called as number of statements executed by total number of executive statements. Higher level coverage metrics can also be generated in particular thing.
The next level of TER control flow branches is in to whatever admission are branches that we have seen before you are right total number of control flow branches how much is there? So whether we are able to test it or we are able to free value the metrics, so this TER will give basically in the sequences whatever we have done, so TER 3 number of number of executive statements divided by total number of LCSAJ.
So this metrics that is for the hierarchy whereby when the TER1 100% has been achieved it follow that TER2 should be 100% then TER 1 should be 100%, so I used to have TER1 as to be 100% so that the number of LCSAJ are executed by the total number of series become 100%. So this is been traditionally very old method so both the TER1 and 2 get results for 70 and the third results for the later 70.
And the requirements actually for the TER1 100% was the level and used to be called TER and then they define TER1, TER2, TER3 like this levels of testing based mention based on the safety and embedded software they will have a definitions of the embedded software levels. Later they have a MCDC modified condition decision coverage that we have studied in the numerous slides.
So it was added 1992 higher levels of TER 3 100 has been mandatory for many other projects including aero space telephone and banking. So where there is a mandated TER3 should be 100% liners force sequence and jump is 100% mandatory when practical problem of using TER3 is that many LCSAJ can never be executed due to the conferencing condition they may contain. We know that we have different jumps sequence that is start of the linear sequence of executive statement.
And the target line to which control flow is transferred at the end of the executive statement, so it is basically it is the units in the map that executed all of them to subjective but it is mandatory.
(Refer Slide Time: 32:21)



White Box testing: Linear code sequence and jump

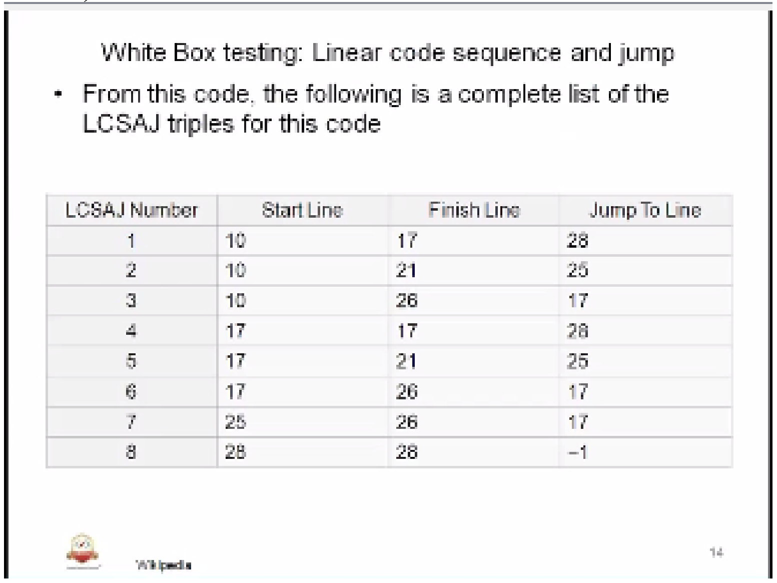
- Example C code:

```
#include   <stdlib.h>
#include   <string.h>
#include   <math.h>

#define MAXCOLUMNS  26
#define MAXROW      20
#define MAXCOUNT    90
#define ITERATIONS  750

int main (void)
{
   int count = 0, totals[MAXCOLUMNS], val = 0;
   memset (totals, 0, MAXCOLUMNS * sizeof(int));
   count = 0;
   while ( count < ITERATIONS )
   {
      val = abs(rand()) % MAXCOLUMNS;
      totals[val] += 1;
      if ( totals[val] > MAXCOUNT )
      {
         totals[val] = MAXCOUNT;
      }
      count++;
   }
   return (0);
}
```

Wikipedia

13

So here you see an example code.
(Refer Slide Time: 32:32)



White Box testing: Linear code sequence and jump

- From this code, the following is a complete list of the LCSAJ triples for this code

LCSAJ Number	Start Line	Finish Line	Jump To Line
1	10	17	28
2	10	21	25
3	10	26	17
4	17	17	28
5	17	21	25
6	17	26	17
7	25	26	17
8	28	28	−1

Wikipedia

14

And for white box testing of LCSAJ recent example using example from Wikipedia and just used here you have a C code having a library included here you have function of tattoos which is consume columns, consume rows, iterations and 750 in the mean are the main and we are initializing a called equal to 0 and array totals of fix and value and local variable equal to 0 and then we shut the memory with totals.
It has oversight of that integer it counts 0 in what it does is while the count is less than iterations that means 750 times this block will be executed this value will be execute of some number and modules consume columns and total array will be assigned with 1 implemented if the total of the particular value is greater than maximum code it will assign the ledger count, the count will get implemented so that up to 750 times it will be executed. Hence it will not be assigned.
So with help of this example so there is a LCSAJ triples it is called as I said start fine finish line jump line, so there are eight LCSAJ numbers have been identified and start line is 10 or the first

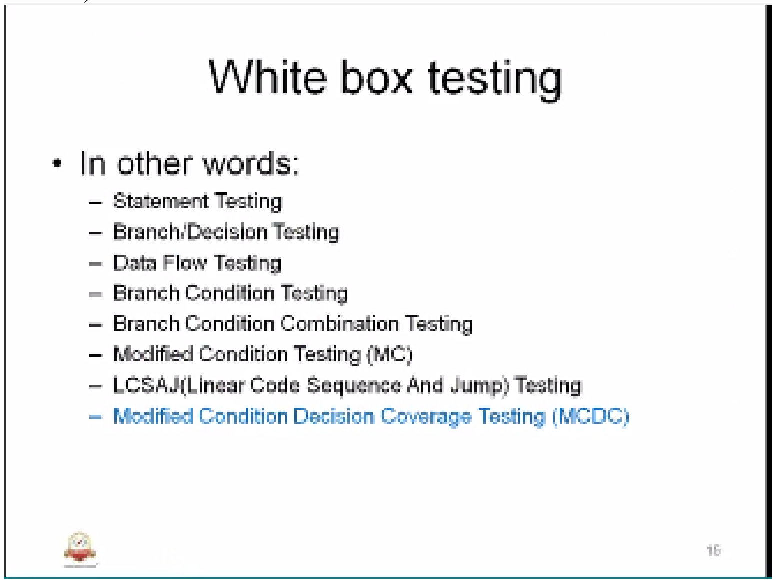
three or then we have the start line of 17 for the next three and then 7th one is 25 and the 8th one is 28. So based on this follows which there are LCSAJ numbers for that and total number of LCSAJ are 28.

The first one in LCSAJ then start line 10 is used and finish line is 17that means 10 is wiped after these values and 17 is the 28 positive last power backing. So within this what are going to happen like assumed as one LCSAJ similarly we have 10 then 21 and finish line is 17 jump line is 28 that means it is 17 so this is a 10 set and we have 17 11, 12, 13, 14, 15, 16, 17 and the jumped line 28

So likewise we are going to have start line 17 and finish line 21 they draw a table so that is about LCSAJ understanding where the use the TER test effective ratio with help of the LCSAJ is the sequences where the sequence of executable when it starts and it ends and the target line with the control flow is getting targeted, so that is how they are going to test it. And you can see some of the table tasks about started line are the same.

Which is same here 28- 28 is same that is the last one written 0 is the same and the jump line that last program, similarly we have 17-17 the last line is 28. So that is how the key aspects will be tested in the LCSAJ type of white box testing.

(Refer Slide Time: 37:42)



White box testing

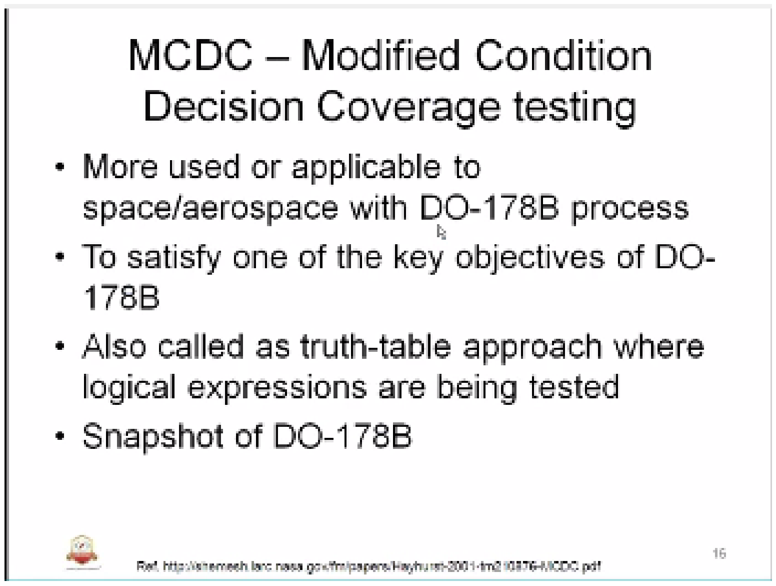
- In other words:
  - Statement Testing
  - Branch/Decision Testing
  - Data Flow Testing
  - Branch Condition Testing
  - Branch Condition Combination Testing
  - Modified Condition Testing (MC)
  - LCSAJ(Linear Code Sequence And Jump) Testing
  - Modified Condition Decision Coverage Testing (MCDC)

15

So the last one being I have added purposefully because they use this type of testing in aero space so very important to understand this, we said different standards where the different levels are safety is being defined based on the second level of the particular embedded software and the system, so for that type of system that device is going to be mandatory. Which is called modified condition decision coverage testing, so we will quickly see what this is?

So introducing modified condition testing followed to,

(Refer Slide Time: 38:49)



MCDC – Modified Condition Decision Coverage testing

- More used or applicable to space/aerospace with DO-178B process
- To satisfy one of the key objectives of DO-178B
- Also called as truth-table approach where logical expressions are being tested
- Snapshot of DO-178B

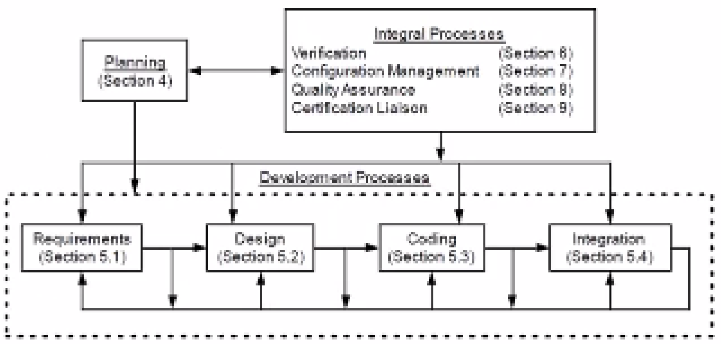
Ref. http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf

16

It is more used to or applicable to space/ aerospace with DO-178B process, to satisfy one of the key objectives of DO -178B. Also called as truth table approach where logical expressions are being tested so where we have an exam question. We have a truth table so what is a truth table? We will study that in the next class, and example snapshot of DO-178B.

(Refer Slide Time: 39:26)

MCDC – Modified Condition Decision Coverage testing


Planning (Section 4)
Integral Processes
Verification (Section 6)
Configuration Management (Section 7)
Quality Assurance (Section 8)
Certification Liaison (Section 9)
Development Processes
Requirements (Section 5.1)
Design (Section 5.2)
Coding (Section 5.3)
Integration (Section 5.4)

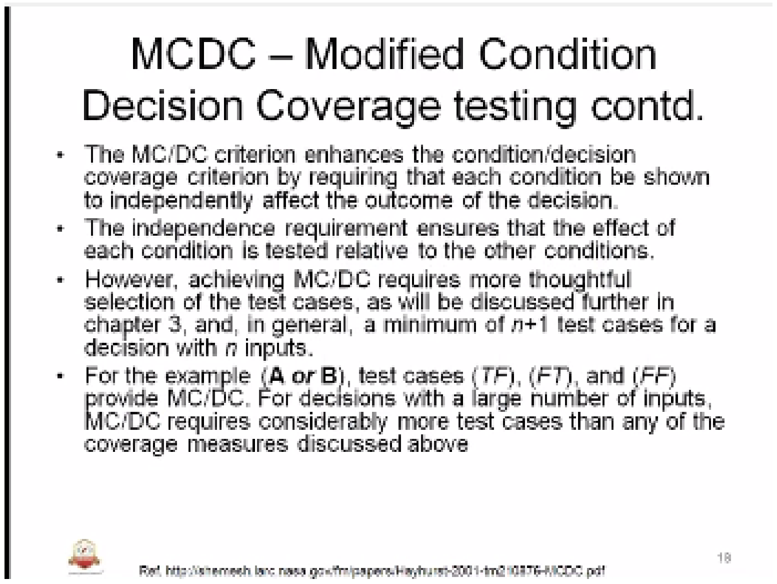
Ref. http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf

16

I will just have a look in to this, so basically the MCDC is modified condition decision coverage testing, we know we have seen condition coverage and decision coverage and condition combination decision coverage it will enhance basically by requiring that each condition we show independently is outcome of the decision process. The modified condition testing the combinations it is similar to that but it is string line process may follow it.

Independence requirement ensure that they effect of each condition is tested related to feather conditions. Whatever achieving requires more spot full selection of the test cases we need to have a truth table sort of a approach but you have a three variables we will enable 15 type of

combination, we don't need to develop them. So that is why it is very important to select the good approach and select such inputs.
And suppose in general if N inputs are there minimum of N+1 test cases for a decision with those inputs need to be there, for example A or B we will have test cases. For example we have independency of this A or B suppose this, is there then we will have some stagments executed so A can have true it can have false B also can have true and false, so what are the test that we can do it to show the independence?
So the combination could be TF, FT, FF so the independencies can mingle that it is not required because it doesn't touch the independence an you have only tested when the A is concerned as input through and A are B will result in true only, so to achieve the independency we need to have a test combinations we have FT and FF this will provide the coincidence it can have a proof also it is subjective again depending on that.
Here you can see decision coverage MCDC testing what they use, so basically we need the example snapshot DO-178B the different processors and sections that has it draws a life cycle DO-178B software life is called and this diagram is known as NASA document you can see the references and we have decrement loss and have a planning it is focal instruction four this device will be process at the different sections against the each section that has a checklist.
And each section will have interface different aspects of the MCDC coverage and the testing process so we have a planning instruction four we have integral process which is evaluation verification section 6, configuration management section 7, quality assurance section 8, and certification liaison section 9. So this will be part f the planning that is to be addressing similarly the planning also will take care of this blocks which will be plotted here.
The planning will specify about development aspects so that is part of the development we have requirements section 5.1, decision sorry design section 5.2 coding is all part of section 5.3 1, 2, 3 sections are about definition in section 4 followed the DO-178B has to be strictly followed for the embedded software development if it is DO-178B or example like space and aerospace industries so there are AVR sub systems BOS sub systems need to implement,
Based on the software level that DO-178B will be fandoms, so requirement region coding integration, so this perceptions are part of the process that needs to be pointed out or supported with the help of planning which is instruction code similarly the very important part is the integral process which has perfections, which will identify the verification process, configure management process, quality assurance, certification liaison.
Once we have this addressed that aspects of the embedded system will be called as DO-178B, further on the MCDC,
(Refer Slide Time: 45:15)



MCDC – Modified Condition Decision Coverage testing contd.

- The MC/DC criterion enhances the condition/decision coverage criterion by requiring that each condition be shown to independently affect the outcome of the decision.
- The independence requirement ensures that the effect of each condition is tested relative to the other conditions.
- However, achieving MC/DC requires more thoughtful selection of the test cases, as will be discussed further in chapter 3, and, in general, a minimum of *n*+1 test cases for a decision with *n* inputs.
- For the example (**A *or* B**), test cases (*TF*), (*FT*), and (*FF*) provide MC/DC. For decisions with a large number of inputs, MC/DC requires considerably more test cases than any of the coverage measures discussed above

Ref. http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf

18

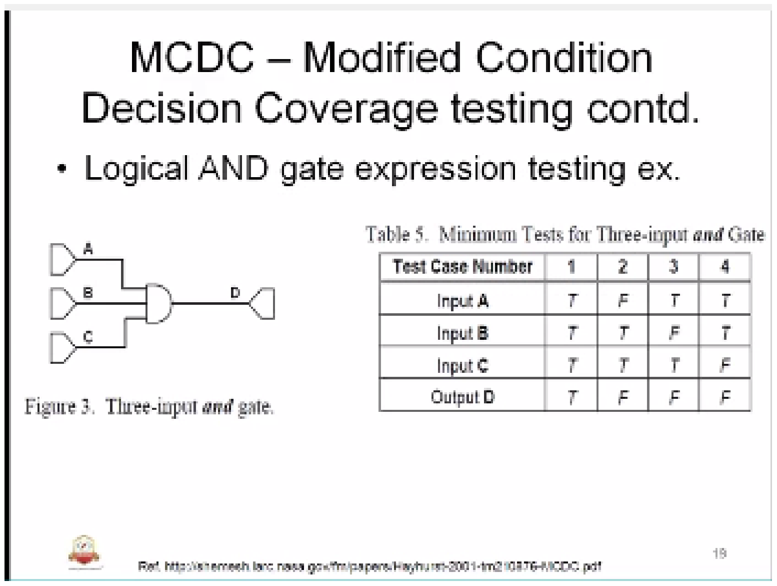
The MCDC criterion enhances the condition/decision coverage criterion requiring that each condition be shown to independently affect the outcome of the decision, same as what we have studied in modified condition combination testing here with most demonstrate aspects will be addressed. The independence requirement insures that the effect of each condition is tested relative to the other conditions that mean we have a certain combinations.

So those combinations because once you have this core is executed and a piece of code which is responsible for some of term will definitely need to be tested and traced so the rest of other piece of the core will have o be constant, so then independency of this specific block will need to be ensured, how we are going to do is? With the help of independence testing the condition and the decision coverage that we have,

However achieving MCDC requires more thoughtful selection of the test cases, as will be discussed further in the chapter 3, that is there in the you can have a look with the following having the which the blocks in detail about MCDC this is not the scope of this of the embedded software system because we are talking about in general all these aspects need further if interested maybe you can verify about MCDC for aerospace how they have addressed.

So in general the minimum of n+1 test cases for a decision with N inputs, for example you have A or B they have true false, false true, and false- false this will provide the modified condition decision coverage, for decisions with a large number of inputs where we have A, B, C, D, E if, or, and all that introducing request considers the more and any of the coverage request about, we will take some examples.

(Refer Slide Time: 47:43)



MCDC – Modified Condition Decision Coverage testing contd.

- Logical AND gate expression testing ex.

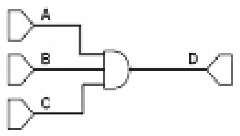


Figure 3. Three-input *and* gate.

Table 5. Minimum Tests for Three-input *and* Gate

Test Case Number	1	2	3	4
Input A	T	F	T	T
Input B	T	T	F	T
Input C	T	T	T	F
Output D	T	F	F	F

Ref. http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf

19

Here you can see a logical AND gate, where A, B, C inputs are being handed in this block and the result of this is D. so it is called the three input AND gate that means there are three inputs A, B, C there is a anklet here you can see A, B, C is a anklet B is the result, so this also represent different gates so that gate representation now all part of the numerate curriculum where R and all the stuff represented this way these are all typically used in the requirements and against those requirements test cases will be done.

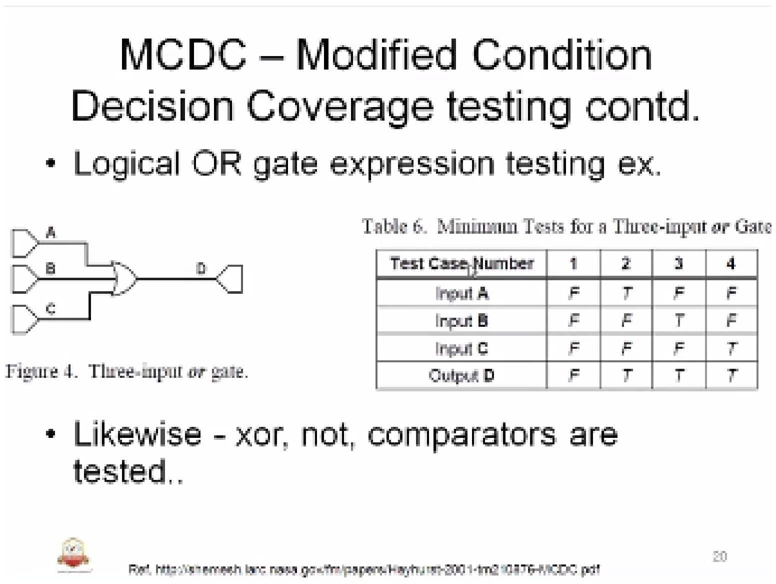
So in this type of what are number of minimum tests for this three input and gates are listed, what they listed? Let us see, test case numbers there are about1, 2, 3, 4 test cases are there sorry three test cases numbers are being percepted, each of them consider in one at a time but this A as an input B as an input, C as an input also we need to have a D as an output for this condition, so those kind of test cases need to be exercised.

What are test that can have for this MCDC? First one A, ,B, C are provided as T that is true the output will be true, similarly A, B, C we provided input as false and B has true- true the output is same or the output is false similarly third one we modify B as false and A as true that could be as false., and in fourth case the input C is provide with the input as that and the output is that, so basically type output is depending on the false conditions that can be driven either this inputs.

So because of C only the process form how we are going to excess by providing the inputs D, suppose if the D is providing false in cleaning case all this for A and B sort of that case C is in test case prove and in the next test sequence we are in to A as true and B as false likewise we are going to have a table this table is called truth table, that is why in C approach in boost for such expression where we use AND.

So these kind of tests we are going to have one test will take 1, 2, 3, 4 values sorry the columns first one keeps all the considerations as 2 second one all the considerations as different values the output as same and that 2, 3, 4 where are to equal to D there are due to independency in either A, B, C etc so that is the MCDC table is drawn and test cases are drawn, let us take another example

(Refer Slide Time: 51:47)



MCDC – Modified Condition Decision Coverage testing contd.

- Logical OR gate expression testing ex.

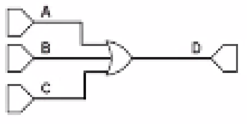


Figure 4. Three-input *or* gate.

Table 6. Minimum Tests for a Three-input *or* Gate

Test Case Number	1	2	3	4
Input A	*F*	*T*	*F*	*F*
Input B	*F*	*F*	*T*	*F*
Input C	*F*	*F*	*F*	*T*
Output D	*F*	*T*	*T*	*T*

- Likewise - xor, not, comparators are tested..

Ref. http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf

20

Here we will go through the three input OR gate where A, B, C are fed in to an object and result of output is D in the previous example we have B is an output there are from the animate so what are the test cases that you can draw so we need to preserve in the input A input B in this, in order to achieve output B out of the condition we need to exercise, so this is a table which starts about minimum tests that we need to drive.

In the first test case we are going to have input A, B, C all are as false the output will be false, in rest of the test cases you can see 234 the output is always true why because one of the input is true. So why it is called as a minimum we need to understand we may have one more suppose what are the composition we can have do we have true, true, false? Will we have false, true, true? Yes.

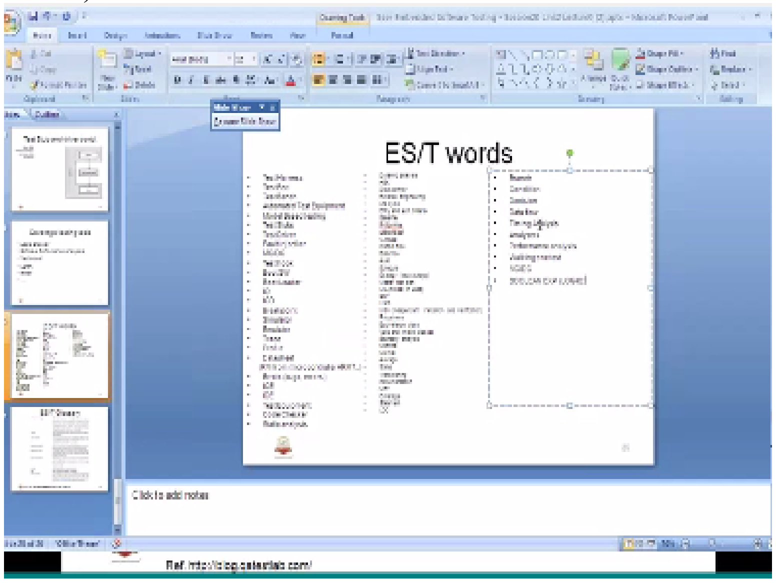
So basically which is the deciding factor here not the false, the output is derived purely based on the, the output true is purely based o the fact that any of the input is true. So we need to drive any of the input in the first case any can become A in the second case B such as a C, so other cases like both cases A and B true doesn't matter because already we have seen the output has true, so this is the minimum of tests.

Of course defining on the expression condition we can have more but it is like a PDS relevant selection but as part of the MCDC called as an independency literature. How we are going to show independencies, reach of the input that is responsible or effective in terms of the output not to be driven. Similarly we have seen many of the inputs in term of false will result in the false output so it doesn't matter true or false all three are false.

Independency visual is set for each of the input so likewise we can have for xor, not, comparators are inputs that is used in a logical expressions maybe you can take an exercise or xor and not so not is very simple you know that not of A equals B. similarly A xor B you know that truth table how to draw because logic expressions you know, so you can write what are the minimum tests that are needed test.

With expression first one and the second one, so this is an exercise or total session will tell. So that is about MCDC, you know that important method that is used in the DO-178B life cycle process. So that is about MCDC we will continue the upper aspects of the next session, we will

go through some of the volts that we have added mainly what we can add is MCDC, DO-178B extreme coverage part of this system.
Anything we have missed MCDC then test cases inputs and outputs you know then branch condition testing, branch condition combination testing, Boolean operands so what we do is that how much wants today we studied,
(Refer Slide Time: 57:29)



ES/T words

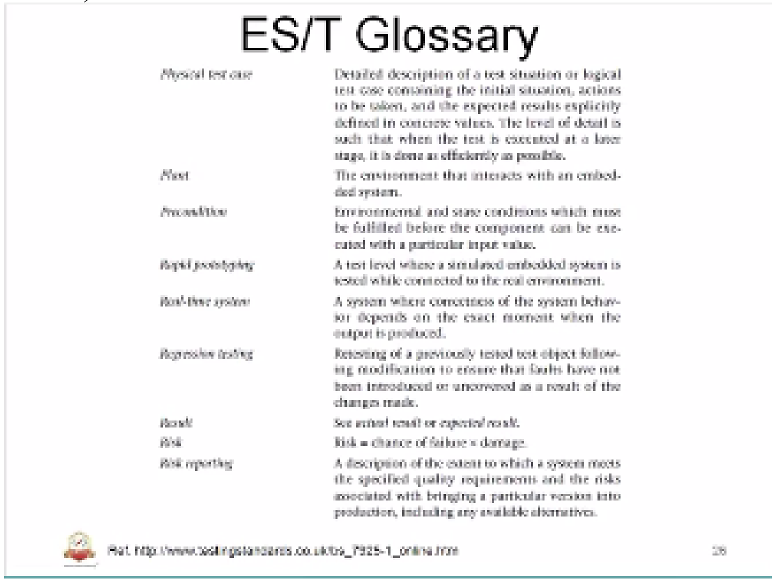
[illegible]

- Branch
- Condition
- Decision
- Data flow
- Timing analysis
- analysis
- Performance analysis
- Walking tester
- MCDC
- BOOLEAN EXP LOGIC

Click to add notes

Ref: http://blog.catalab.com/

So rest of the words like branch condition decision, timing analysis, analyses I think timing analysis analyses performances walking in test studied in these classes session we have seen, Boolean expression logic we had gone through and from the recent session we had statements, orders and will understanding about embedded software testing we will need to be aware of the knowledgeable of this type of testing.
So let us study about, few of the glossaries
(Refer Slide Time: 58:24)



ES/T Glossary

Physical test case	Detailed description of a test situation or logical test case containing the initial situation, actions to be taken, and the expected results explicitly defined in concrete values. The level of detail is such that when the test is executed at a later stage, it is done as efficiently as possible.
Plant	The environment that interacts with an embedded system.
Precondition	Environmental and state conditions which must be fulfilled before the component can be executed with a particular input value.
Rapid prototyping	A test level where a simulated embedded system is tested while connected to the real environment.
Real-time system	A system where correctness of the system behavior depends on the exact moment when the output is produced.
Regression testing	Retesting of a previously tested test object following modification to ensure that faults have not been introduced or uncovered as a result of the changes made.
Result	See *actual result* or *expected result*.
Risk	Risk = chance of failure × damage.
Risk reporting	A description of the extent to which a system meets the specified quality requirements and the risks associated with bringing a particular version into production, including any available alternatives.

Ref. http://www.testingstandards.co.uk/bs_7925-1_online.htm

28

Is refer from the test cases standards, physical test case, you can get designed description of a test situation or logical test case consuming the identical salvation, activities to be submitted, and the respected result specifically designed in concrete values, level of detailing is such that when test is executed in the latest is done this will have the concrete steps in terms of values and that sop physically we made about in a descriptive manner sort of physical tests. Plant, environment and normal cannot is the production industry is test plan etc. the environment attracts with the embedded system, we condition is an important term,

So say environmental and store conditions which are not being fulfilled before the components can be executed with a particular input value. And the sort of pre conditions attest following that pre condition that are to be executed and that pre condition cannot be executed, pre condition have to be there input. Rapid prototyping it is a developing test level where a simulated system is tested while connected to real environment,

That means rapid prototype testing is a test driver that embedded system is simulated is tested which is connected to the real environment. Real time system is a system where corrections of the system behavior depend on that exact moment when the output is produced. That means where the time is very hard this also called as hard real time system, real time system basically deals with the behavior.

Where the different units that are going o happen in the embedded system should be happening at the right time or the exact moment, for example we will take the embedded systems they have access where users process one button so it should respond sort of time. So that sort of systems we have in term embedded system. Regression testing sis very important hat as the retesting of a previously tested test object following modification,

To ensure that faults have not been introduced or uncovered as a result of the changes made, that means we have some in first or primary testing and that is been fixed like a primary department design or the same test without modified that suit test environment or the test cases that sort of the fixed code and results in achieve in test been passed. Result you know that actual result or expected result.

Actual result is what we get it from the system when it is subjected to subsequent test, the expected result is what? Been expected which is next part of the test case results. Risk is a chance of failure in to damage that means while doing the testing's testing of the embedded systems which could based on the inputs or the robustness or the lower that we have on the system it could impact efficiency or behavior we have looked.

And it could result in a dimension so this gives as a chance of failure multiplied by the damage, this report in description of the extent to which a system meets the specified quality requirements and the risks associated with bringing a particular version into production including any available alternative, that means we have multiplication how we can do this aspects all will be part of the this risk reporting, so with that we will end this session and we will take up the white box some of the pending in the next session.