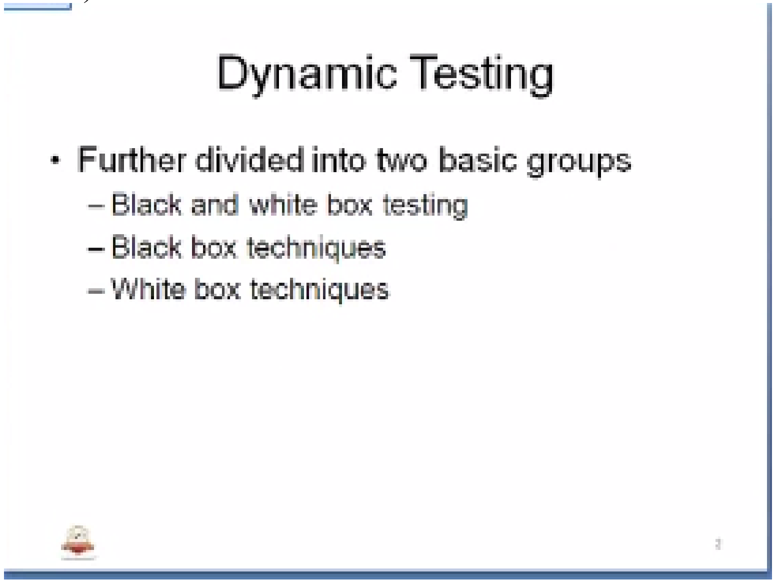
Embedded Software Testing
Unit 2: Testing Methods
Lecture 19
Seer Akademi-NPTEL MOU

The next session of embedded software testing, this is the 19 lecture of embedded software testing. This part 2 or unit 2 we are going to study about white box testing and the techniques, in the previous session we understood about the basic of,

(Refer Slide Time: 00:35)

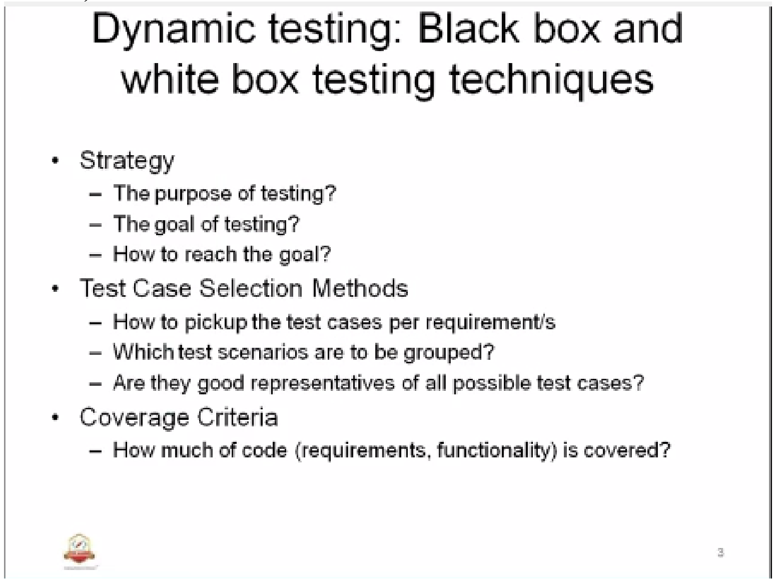


Dynamic Testing

- Further divided into two basic groups
  - Black and white box testing
  - Black box techniques
  - White box techniques

In the terms of white box and black box, we need to have strategy test case selection and coverage solution in the term of this aspect.

(Refer Slide Time: 00:46)

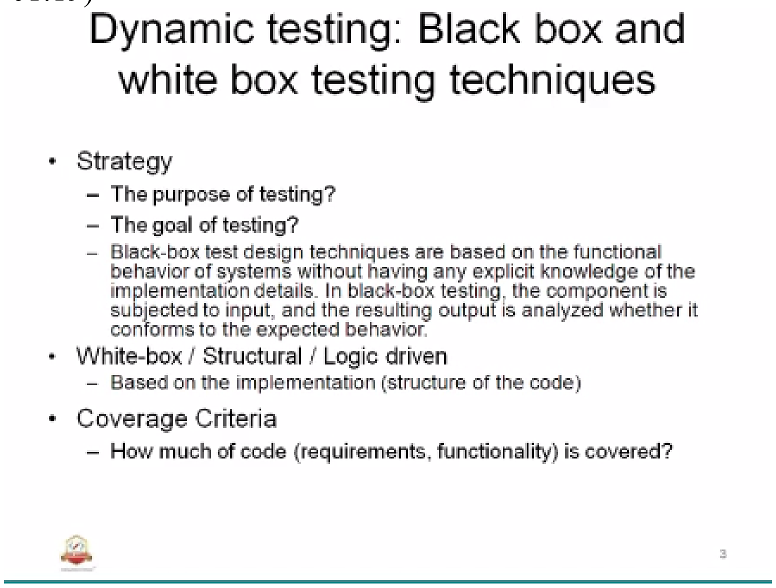


Dynamic testing: Black box and white box testing techniques

- Strategy
  - The purpose of testing?
  - The goal of testing?
  - How to reach the goal?
- Test Case Selection Methods
  - How to pickup the test cases per requirement/s
  - Which test scenarios are to be grouped?
  - Are they good representatives of all possible test cases?
- Coverage Criteria
  - How much of code (requirements, functionality) is covered?

In the black box and the white box testing any answer it should be about the difference between white box and the black box testing, basically black box the functional of data, it is expirely based on the requirements and the given technique from the function behavior of the system without any knowledge of the internal or implementation system and the test.

(Refer Slide Time: 01:19)

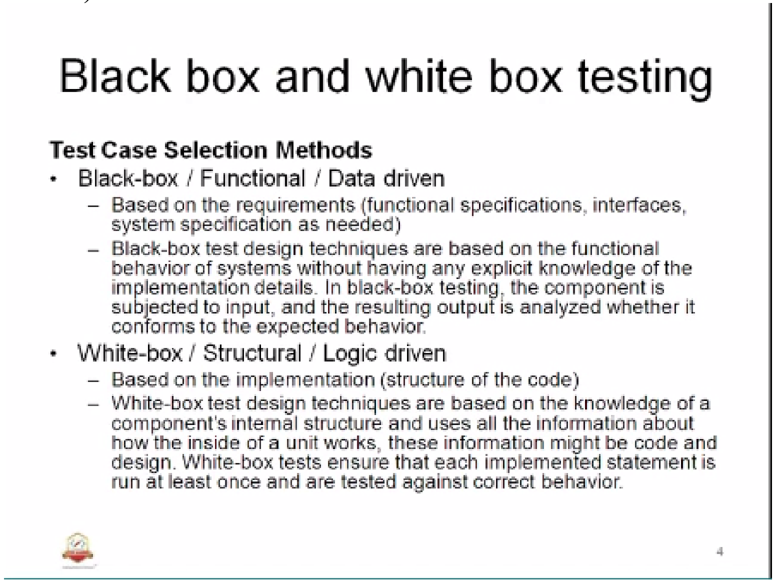


Dynamic testing: Black box and white box testing techniques

- Strategy
  - The purpose of testing?
  - The goal of testing?
  - Black-box test design techniques are based on the functional behavior of systems without having any explicit knowledge of the implementation details. In black-box testing, the component is subjected to input, and the resulting output is analyzed whether it conforms to the expected behavior.
- White-box / Structural / Logic driven
  - Based on the implementation (structure of the code)
- Coverage Criteria
  - How much of code (requirements, functionality) is covered?

3

(Refer Slide Time: 01:20)



Black box and white box testing

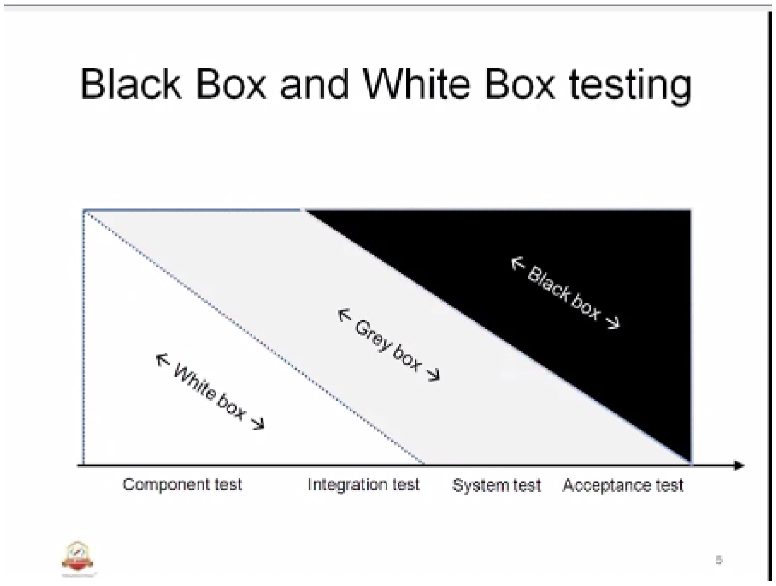
Test Case Selection Methods

- Black-box / Functional / Data driven
  - Based on the requirements (functional specifications, interfaces, system specification as needed)
  - Black-box test design techniques are based on the functional behavior of systems without having any explicit knowledge of the implementation details. In black-box testing, the component is subjected to input, and the resulting output is analyzed whether it conforms to the expected behavior.
- White-box / Structural / Logic driven
  - Based on the implementation (structure of the code)
  - White-box test design techniques are based on the knowledge of a component's internal structure and uses all the information about how the inside of a unit works, these information might be code and design. White-box tests ensure that each implemented statement is run at least once and are tested against correct behavior.

4

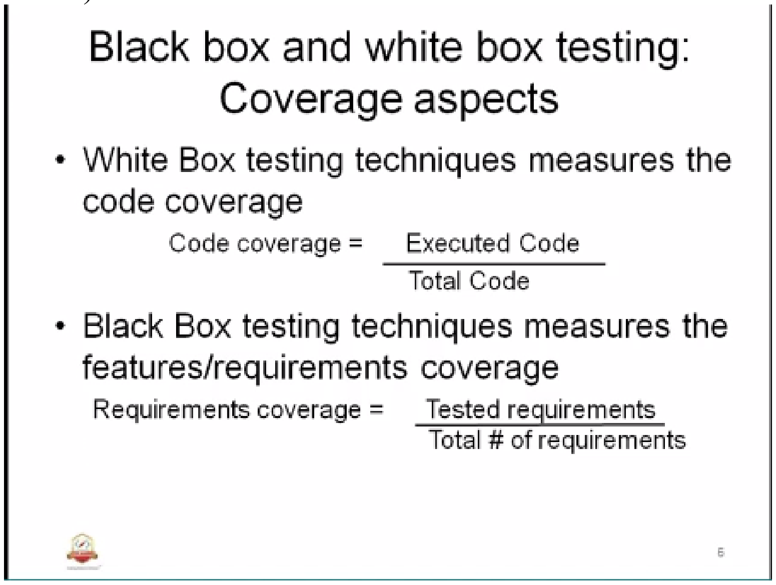
Similarly the white box testing implementation of the code and the study and partially the system will have knowledge in the entire system, but these purposes will be based on white box design techniques in the terms of the structure of the code. Also it brings the intermediately called tool box,

(Refer Slide Time: 01:45)


Black Box and White Box testing

It can have a mixture of both white box and black box, where the coverage is not possible in the white box and black box, so we can have a credit of both of them that is called grey box. And the code coverage we know,

(Refer Slide Time: 02:02)


Black box and white box testing: Coverage aspects

- White Box testing techniques measures the code coverage

$$\text{Code coverage} = \frac{\text{Executed Code}}{\text{Total Code}}$$

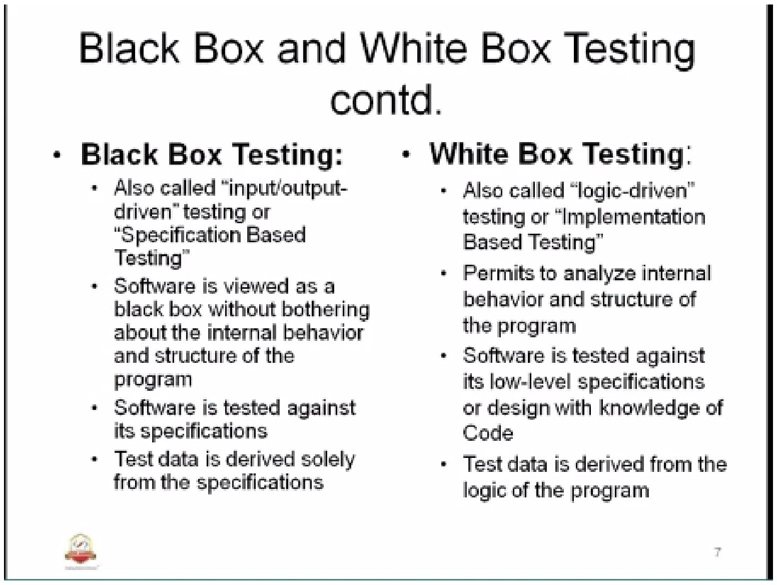
- Black Box testing techniques measures the features/requirements coverage

$$\text{Requirements coverage} = \frac{\text{Tested requirements}}{\text{Total \# of requirements}}$$

6

The number of executed code that is tested divided by the total code. That is there in the system and this is called as core inbox tool, similarly requirements are defined and this talk about the total number of tested requirements vs. total number of requirements in percentage. Also we studied about,

(Refer Slide Time: 02:26)

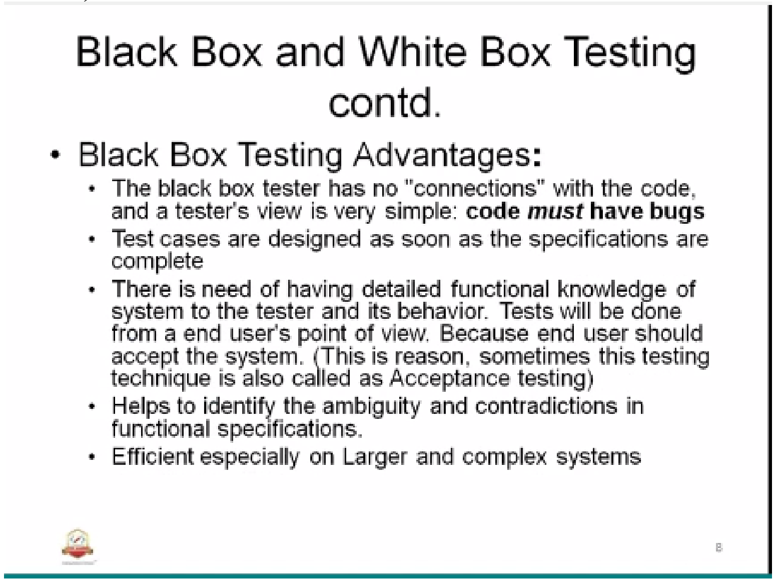


Black Box and White Box Testing contd.

- **Black Box Testing:**
  - Also called "input/output-driven" testing or "Specification Based Testing"
  - Software is viewed as a black box without bothering about the internal behavior and structure of the program
  - Software is tested against its specifications
  - Test data is derived solely from the specifications
- **White Box Testing**:
  - Also called "logic-driven" testing or "Implementation Based Testing"
  - Permits to analyze internal behavior and structure of the program
  - Software is tested against its low-level specifications or design with knowledge of Code
  - Test data is derived from the logic of the program

7

Black box testing and white box testing differences, and their advantages,
(Refer Slide Time: 02:33)

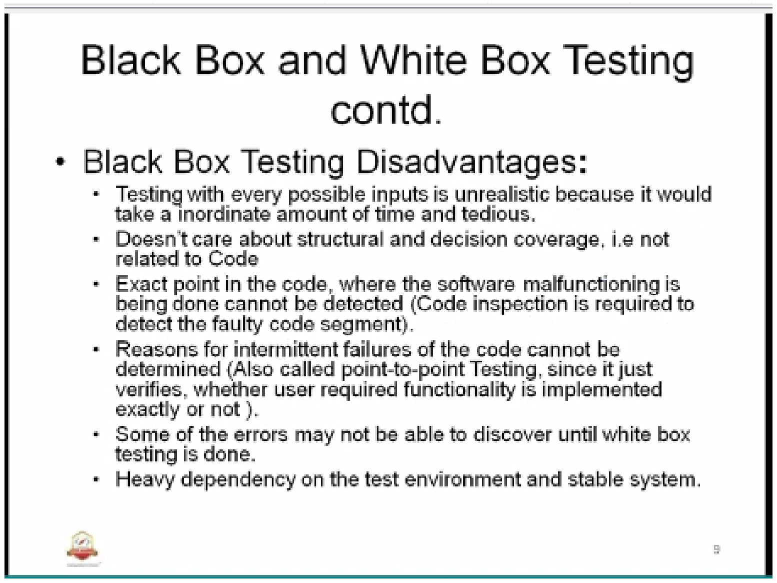


Black Box and White Box Testing contd.

- Black Box Testing Advantages**:**
  - The black box tester has no "connections" with the code, and a tester's view is very simple: **code *must* have bugs**
  - Test cases are designed as soon as the specifications are complete
  - There is need of having detailed functional knowledge of system to the tester and its behavior. Tests will be done from a end user's point of view. Because end user should accept the system. (This is reason, sometimes this testing technique is also called as Acceptance testing)
  - Helps to identify the ambiguity and contradictions in functional specifications.
  - Efficient especially on Larger and complex systems

8

Are again as well,
(Refer Slide Time: 03:37)

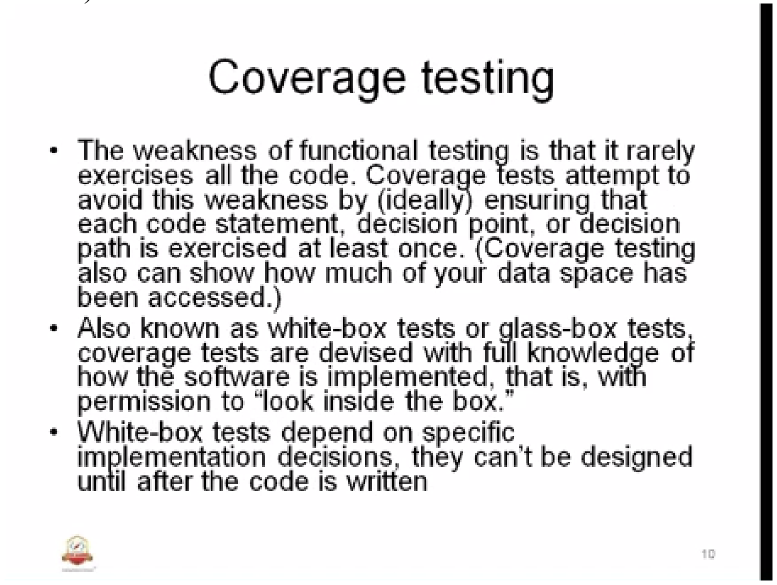


Black Box and White Box Testing contd.

- Black Box Testing Disadvantages:
  - Testing with every possible inputs is unrealistic because it would take a inordinate amount of time and tedious.
  - Doesn't care about structural and decision coverage, i.e not related to Code
  - Exact point in the code, where the software malfunctioning is being done cannot be detected (Code inspection is required to detect the faulty code segment).
  - Reasons for intermittent failures of the code cannot be determined (Also called point-to-point Testing, since it just verifies, whether user required functionality is implemented exactly or not).
  - Some of the errors may not be able to discover until white box testing is done.
  - Heavy dependency on the test environment and stable system.

9

In the continuous of that, today we study about,
(Refer Slide Time: 02:43)

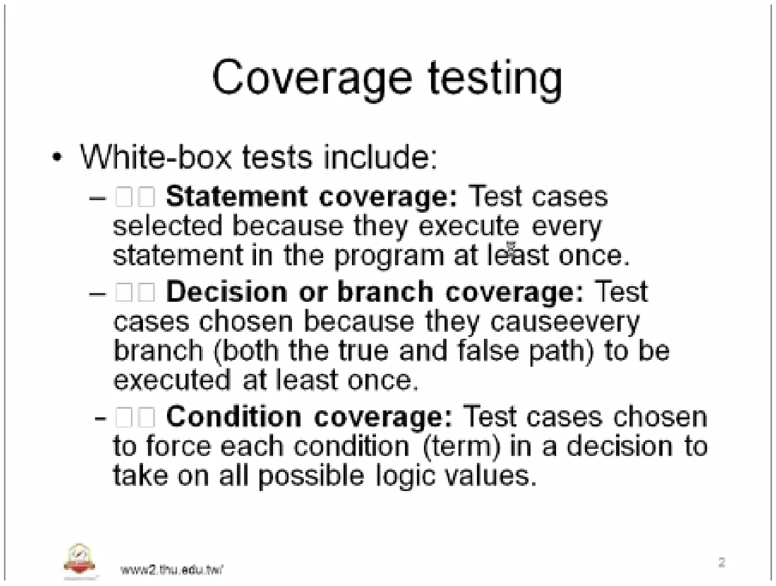


Coverage testing

- The weakness of functional testing is that it rarely exercises all the code. Coverage tests attempt to avoid this weakness by (ideally) ensuring that each code statement, decision point, or decision path is exercised at least once. (Coverage testing also can show how much of your data space has been accessed.)
- Also known as white-box tests or glass-box tests, coverage tests are devised with full knowledge of how the software is implemented, that is, with permission to "look inside the box."
- White-box tests depend on specific implementation decisions, they can't be designed until after the code is written

10

The coverage of white box testing, what the types of box are testing?
(Refer Slide Time: 02:52)



Coverage testing

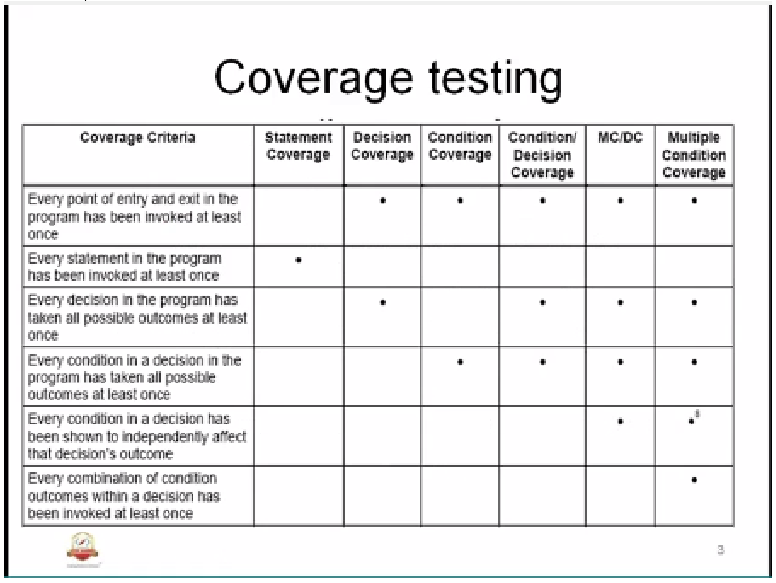
- White-box tests include:
  - – □□ **Statement coverage:** Test cases selected because they execute every statement in the program at least once.
  - – □□ **Decision or branch coverage:** Test cases chosen because they causeevery branch (both the true and false path) to be executed at least once.
  - - □□ **Condition coverage:** Test cases chosen to force each condition (term) in a decision to take on all possible logic values.

www2.thu.edu.tw/ 2

The broad categories are, statement coverage where on the executable statements in the program we need to test it, because there will be executed so we need to see that its driven and we are tested. So we need to have the test cases selected as per the same. The next one is the decision or the branch coverage, here test cases chosen because ever branch it is called branch of the decision, so the decision can be read into true or false path, so both have to be executed at least once, so that's why it is called decision coverage or branch coverage.

Similarly we have condition coverage in this type of broad level coverage; we call test cases during some force for some condition in a decision to take on all possible logic values. Suppose we have a multiple condition available and we need to test, so we need to exercise all the possible paths, that the condition take and it could be a logical expression, so we will study about that condition in detail and the coming class. Okay, so all this together is called as statement coverage.

(Refer Slide Time: 04:20)



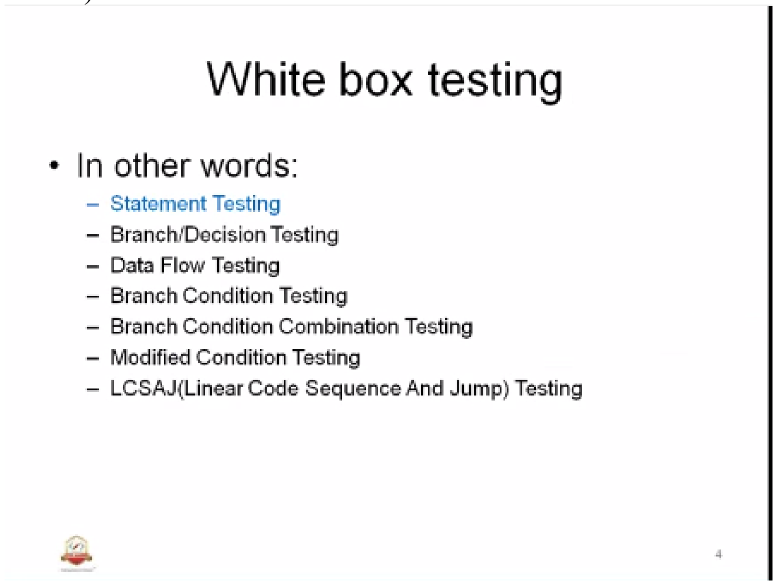
Coverage testing

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

Fffffffffff7So these are the white box testing technique on total is called structural coverage which have the statements on and the different types that are underneath the technique, in the end of the structural coverage we do a technique called structural coverage, so which is methods which credit and how much coverage is done in our correspond, so end goal is to reach 100 percent, so that is what it require the requirement, all together it is called as structural coverage, and you can see the table this typically used in industries as part of the coverage's concerned.
So you can see a different coverage's on the performance file statement coverage, mission coverage, condition coverage, condition decision coverage, MC/DC coverage and multiple condition coverage. So what are these and how these are coordinated in the different types of coverage criteria, so ever point of it entry and exit in the program has been invoked at least once, that means we have a several program units or the function units that function will get entered and then the function gets executed after the function is executed.
So whatever the possible ways of entering that functional and what are the possible ways of exiting out of the function as to be invoked by the program that sort of testing is done in the addition coverage, conditional coverage, condition decision coverage, MCDC and multiple condition coverage, this are the responsible because there are function could have conditions, decisions, monifide conditions etc.
Make sure that all the entry and exist criteria have been taken care, okay, next one is a program will have a statements all the statements is need to be touched once, all the statements must be involved with that testing method, so this testing method is called as stepping coverage, the next one is the decision in the program in all possible outcomes happens once at least, and you have a diamond box, decision box, so the decision box and certain connections, so decision can take a true or false, this part can take yes or no, so likewise all the decisions in terms of covering this part and this part, so this decision have to be invoked, this decision has to be covered. So those types of testing are covered in the decision coverage, condition decision coverage, MCDC, multiple conditions.
Next one is ever condition has a decision, the program has taken all the possible outcomes which it depends, that means a conditions that is come out in the decision should have taken whatever the possible outcomes that in this regions, so that will be taken in the condition coverage explicitly, along with the condition decision because any other decision is involved. So the condition decision in the programming is taken all possible at least once, the next one is every condition in the decision have been gone independently effect that decision of the condition, whatever the execution path be have should be independently, that means the pre condition that are required should be independent.
So that will done with the help of MCDC, this MCDC otherwise it is called as multiplied conditions, decision coverage, decision condition technique, these are mostly used in other space and space industries with the help of standards industries, I will be probably explain that aspect in detail in the test slides and this is called do ones standards basically, and this one along with the multiple condition part of it.
We have this condition coverage criteria is meet, last one is every combination of the condition comes with the decision has been invoked, and these are the multiple condition coverage where

the difference several conditions it will program the functional could have similar or different sort of diamond boxes in this statements, all that paths have to be aggregated with the several combination of them, so that is the multiple condition, so this several table talks about coverage criteria, and what are the types of coverage we have , we will study each of them in detail.
(Refer Slide Time: 10:52)



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

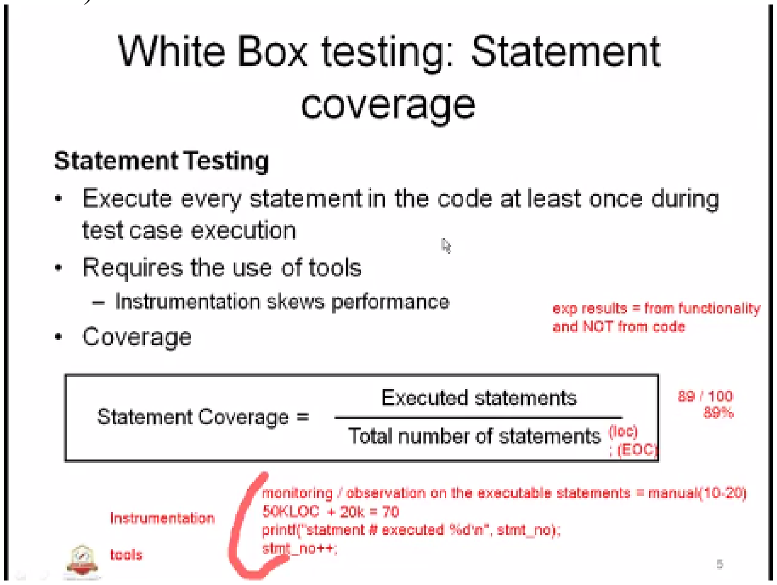
4

Okay, so white box testing have a below types of testing methods, the first one is being. Statement testing and the next one is branch decision testing, and the other one is, data flow testing, and branch condition testing we have branch combination testing, modified condition testing, and the last one is linear code sequence and jump testing.
Statement testing the idea with this statement coverage is to do enough test case for that and every statement source code as been executed at least once, and in branch decision testing idea with the decision coverage, the idea with the decision coverage's is to execute, and every single decision in the code at least twice, you know that the decision can be executed in the adrenal path true or false, that is the possible outcome of the decision should be executed in order to reach the decision coverage, that means the decision coverage is been taken to the branch or decision in testing.
Next one is the data flow testing, here what we do is, test case are designed based on variables that are used in the code. Basically a purely this is based on the data, so the data how it is represented? With the help of variables so all those variables have to be consider in the terms of every stage in the flow, and what are the values it takes, so that is taken care with the help of data flow testing.
The next one is branch condition testing, these are the test case designed in which test cases are designed to execute branch condition outcomes, next one is test case design technique, in which test cases are designed to execute combination of branch condition outcomes, that means we have different combination of the branches, that is done with the help of branch condition combination testing.
Next one is a modified condition testing, here in this design technique test cases are designed to a execute branch conditions outcomes that independently effect the decision, that means the

independency that decision will be taken consider in this modified condition testing. The last one is, linear code sequence and jump technique, in this select test case based on jump free sequence, that means some codes are like interrupts in this case and in independent, so this consist of about three types of things, executable statements, linear sequence end of that it will be there and the target line to which control flow is transferred at the end of the linear sequential, that means we have start of the sequence and end of the sequence and the target line in which the control flow is transferred end of the linear code.

So we will study in detail about this the type of this, and last one is one more technique also we will study and basically followed in a aerospace, with the help of this standard, that is NCDC, so that also we can consider basically that is having this multiplied condition, decision coverage. This is nothing but branch condition combination about it is called.

(Refer Slide Time: 15:04)



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

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

exp results = from functionality and NOT from code
89 / 100
89%
(loc)
; (EOC)
Instrumentation
tools
monitoring / observation on the executable statements = manual(10-20)
50KLOC + 20k = 70
printf("statment # executed %d\n", stmt_no);
stmt_no++;
5

Okay, statement coverage. So execute every statement in the code at least once in the test case application. That means it is basically a fundamental technique that every testing it does not matter it is embedded or non embedded equation in java whatever it is called, all these type of technique, they use this statement called coverage, in this statement coverage test cases will be created so that ever statement in this source code have been executed, so that is the aim of the stat4ement coverage.

So basically they use tools basically such as, white box statement testing tools, LDRA, RTRT, lot of tools are there, the specific tools need of that particular embedded software they use this. And the coverage is as I said earlier slide, earlier session, the coverage's taken in to the credit with the help of executed number of statements process, total number of statements. That means, suppose with the help of this statement testing that the coverage how we will arrive it, suppose we have executed totally number of lines 89, and the total number of statements that are there, there statement is nothing but, the lines of executable lines that, this exclude all commands and everything and see if you consider every form is there and the execute are called as LOC, so basically this total number of statement need to be executed in white box statement coverage, but it may not be possible to do 100 percent. So there are other methods that we will have to do it like whatever that methods are similar position coverage, so ever path needs to be executed by its own technique and their coverage done with the help of percentage, here it is written in percentage and it is written in the percentages statement coverage's so the 11 percentage will be drawn and 11 percent we need to take care.

So the work flow when we are using statement for all existing black box test cases that has been created while monitoring the execution, that means if we have done the black box requirements testing, we can do that if possible with the help of white box and monitor one of the statements that have been executed, then this monitoring is an all but simplest test cases that performed in the tools support basically.

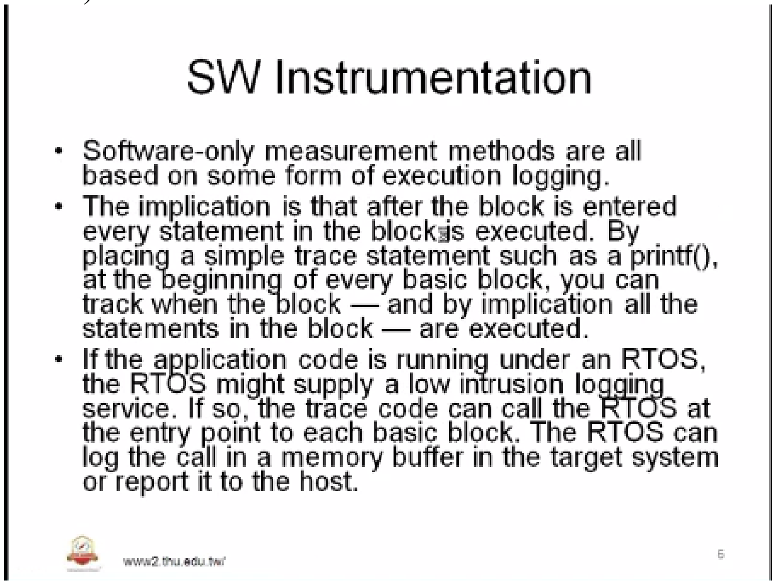
So when all black box test cases have been executed tool can report which parts of the code remain untested, that means we know that 11 percent is UN trusted, so now we need to construct the new test cases that we cover the remaining statements as much possible. So we will start with the path of the code that should be reached walk backward code to determine the values of input variables, which is required to reach the result of the report, then accordingly we will avoid the inputs, whatever the values specific values that are required for lines of code that are not called, so how we can feed the inputs? With the help of specification or whatever it is used for functionality of the variables if it is possible, and the excepted results also can be drawn for that specification.

So this new test cases which are executed are monitored, something we should take care is, it is not excepted results from the call itself, it should go through is specification and the functionality, off course code should be compliment and NOT from code and it is very important aspects. We cannot have the test code can have the local variable or un calculated values or you may get bioassays or it may be half testing and it should try to go with the functionality or that unit piece of program what we excepted accordingly we need to complete the expected results and try to see whether passes or fail and this is about statement coverage.

And as I said all the statements I have to be monitored and executed in the track, so what happens is, it is multi technique or what it is called? Observation we want to have on the executable statements, it is not possible with the help of manual, if it is, say some 10 -20, okay, we can do it, there are big embedded programs having 50k loc of code, what will you do?

This is impossible to have a manual instruction of loc. So what we need to have is, there are test hooks that can be used something like printer we can add, and see that every statement is executed, the print f will print, suppose print f statement something like this statement number and every lines of code we can have this statement number implemented, accordingly we know that, what is that? Print f and there is a missing thing, we know that how many statements are missed, but the problem is here we cannot have print f done by our selves, so that's why we have a lines on cover to this we have a tool call instrumentation, this technique is called instrumentation, instrumentation what we do is, we will provide the monitoring ways of observing how many statements have been executed or what are the statements that are mixed, so that report will be done with the instrumentation, where they existing lines of code the program will be instrumented, that will see whether the tested part will be reported.

So in this instrumentation there are lot of things, we will study about the tools so what it does is basically we will feed all the source files and it will add the instrumentation code in to that and execute on the target of the post depending on the nature of the embedded tool, there are other challenges that I will ask you about that, where we add instrumentation and it will add other lines of code and the total lines of code will increased and the instrumentation falls on the 20k it will make 70k and which cannot fit in the target system. So what we can do is, during that cases we can do partial or instrumentation and take the credit by overlapping all of them that is how the instrumentation of statement is done.
(Refer Slide Time: 23:48)

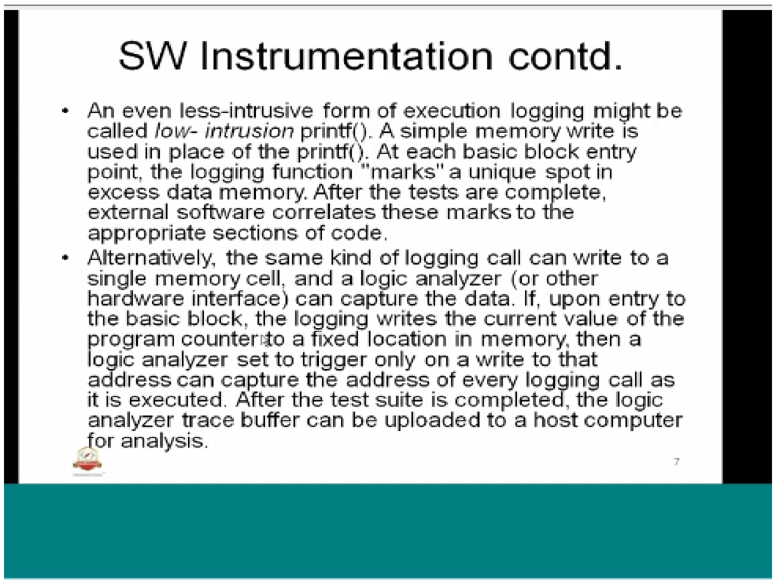


SW Instrumentation

- 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.

www2.thu.edu.tw/ 6

Okay, software instrumentation, software only the measurement methods are all based on some form of execution logging that means logging of the execution or the monitoring of the execution required to do the statement coverage. The implication is that after the block is entered every statement of the block is executed. By placing a simple trace statement such as a print at the beginning of every basic block, you can track when the block and by implication all the statements in the block are executed.
This is what I explained. If it is RTOS, the RTOS itself provide a logging service with the help of we can do it but not all embedded systems will have an RTOS for facility with RTOS we trace it in terms of the logging the calls or memory in the target system. So, we use the intrusion logging in with the help of tools, extended tools. So, they seem to see
(Refer Slide Time: 24:55)



SW Instrumentation contd.

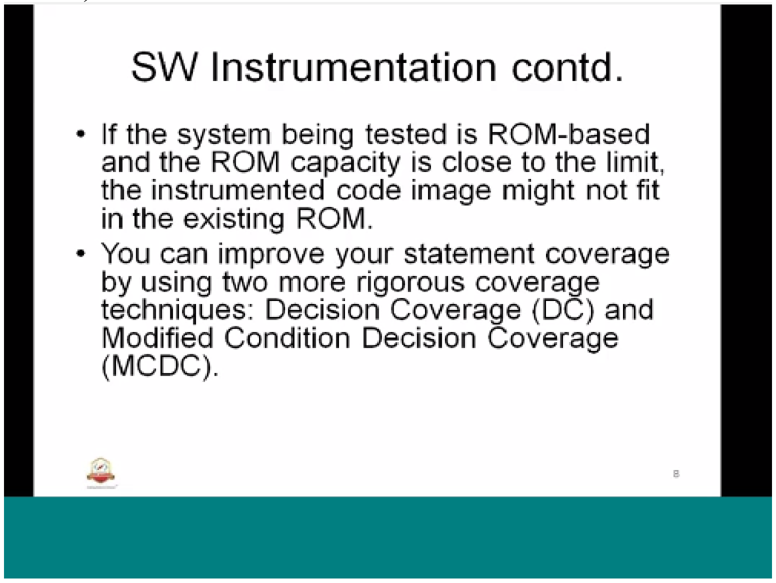
- An even less-intrusive form of execution logging might be called *low- intrusion* printf(). A simple memory write is used in place of the printf(). At each basic block entry point, the logging function "marks" a unique spot in excess data memory. After the tests are complete, external software correlates 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.

7

Form of execution logging might be called low-intrusion print f because it does not intruse much into the existing problem. A simple memory write is used in a place of the printf. Ata each basic block entry point, the logging function marks a unique spot in the excess data memory. After the tests are complete external software correlates these marks to the appropriate sections of the code, so, that the code whether that is used or not used will come to know based on the memory aspects.

Alternatively the same kind of logging call can write to a single memory cell and logic analyzer is another tool as I said for the intrusion and statement coverage they use it. It could be hardware interface such as logic analyzer which we capture data from the memory and analyzed. So, that is how the software intrusion has been done.

(Refer Slide Time: 25:55)



SW Instrumentation contd.

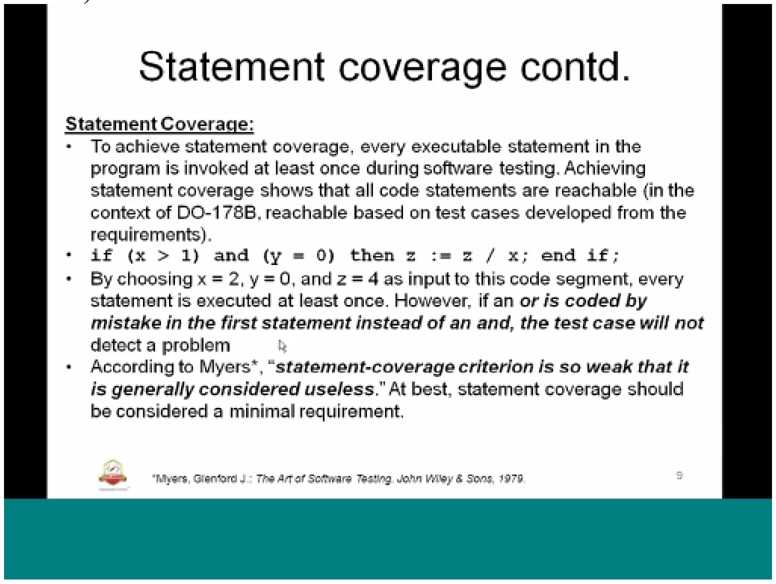
- If the system being tested is ROM-based and the ROM capacity is close to the limit, the instrumented code image might not fit in the existing ROM.
- You can improve your statement coverage by using two more rigorous coverage techniques: Decision Coverage (DC) and Modified Condition Decision Coverage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

8

And the challenging part of the intrusion part is that if the system being tested is ROM-based and the ROM capacity is close to the limit, the instrumented code image might not fit as I said to give the effective problem lines of code is there. And the instrumentation after we are done with that it could come to something like 70, 75k and where the embedded systems are have a memory of 75k definitely fix in a border or it may create more than that.

So, what you can do is you can improve the statement coverage by using two or more rigorous coverage techniques. Sorry we can improve that by the help of over-lapping technique and statements we can reduce it partially. And further testing is the statement coverage is used with the help of decision coverage and intrusion modified condition decision coverage. So, that is how the software instrumentation is done. There are various tools we will touch-case some of the tools with an example at the end of this session or in the next session. Okay,

(Refer Slide Time: 27:04)



Statement coverage 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 not*** 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.

*Myers, Glenford J.: *The Art of Software Testing*, John Wiley & Sons, 1979.

9

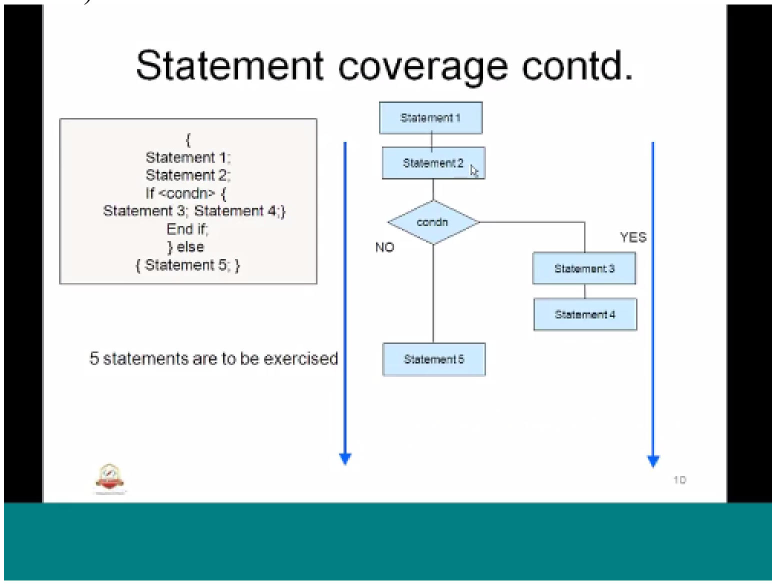
We will go in the detail of statement coverage. You can see there is examples provided to achieve statement coverage very executable statement in the program is invoked at least once during the software testing you know that. Achieving statement coverage shows that all code statement is reachable in the context of DO-178B. Reachable based on test cases developed from the requirements. Here I use either a design or the requirements to have that reach ability aspects that is what we spoken in the DO-178B process, standard. So, for example if x>1 and y=0 then Z will become Z/X end if.

So, this is one statement. So, by choosing x=2, y=0 and z =4 as input to this code segment, every statement is executed at least once. You know that there are two statements these two statements how we are going to execute are by adding such values. So, we enter into that then portion only when if x>1. Here we are choosing x as 2 definitely it will go inside this but there is one more condition called AND condition.

So this logic expression into satisfied by putting y as 0. So, when y becomes 0 and x >1, so z will become z/x and z is called as 4. So, that will become 4/- or if an OR is coded by mistake in the first statement instead of an AND, the test case will not detect a problem. That means we will not be able to detect the issue that is here because the implementation is wrong. So, this some of sort of some of segment of it goes purely by test case selection and the, for understanding.

If the code is wrong then it may difficult. As per embedded software Myers, statement coverage criterion is so weak that it is general is considered as useless. So, at best statement coverage

should be considered a minimal requirement. That means a sort of statement coverage we need to do, to have a complex that may observe statements have been reached or submit by the LIMO. In that way basically we can use this statement coverage criteria and rest of them will use the other coverage.
(Refer Slide Time: 30:18)

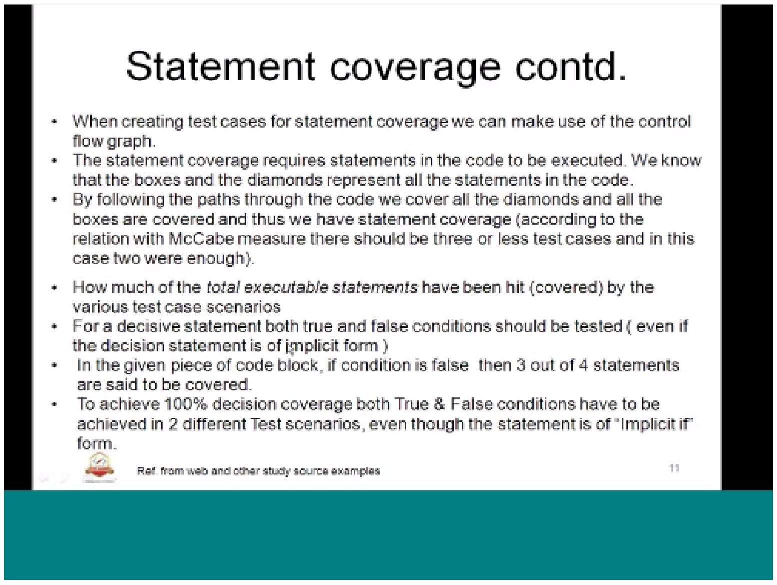


Statement coverage contd.

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

5 statements are to be exercised

Further elaboration of the statement coverage you can see there are about five statements in the program. Here is the program block within the flower bracket. So, we have multiple blocks in terms of small flower brackets. First two statements are executed first and if the condition, if the condition is fine then the statement 3 and 4 are executed. If the finishing is not good in statement 5 is executed.
There are total statements that are being exercised in this program. How are we going to do it? So, because statement coverage we need to make sure those 5 statements are involved at least once. So, we have to make sure that the condition we are going to provide such a way that, the condition can take both NO as well as YES. In those values so by statement coverage going plainly we definitely we are going to cover first two statements and the second part it will based on the result of the two statements, statement 1 or 2 it could be, it will go either as YES path or NO path. So, either case we will execute only three statements or four statements. One of this will be exercised.
So, with this statement coverage approach at best we can achieve either 3 with the NO path or 4 with YES statements are exercised. So, statement coverage can do this much because the conditions it would not get. So, that is how this is been done with the help of 5 statements with the conditions that are underneath based. Okay, in creating test cases for statement coverage
(Refer Slide Time: 32:45)



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

We can make use of control flow graph. So, control flow graph is something like how the flow is going to happen? 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. So, that is also the part of this statement.

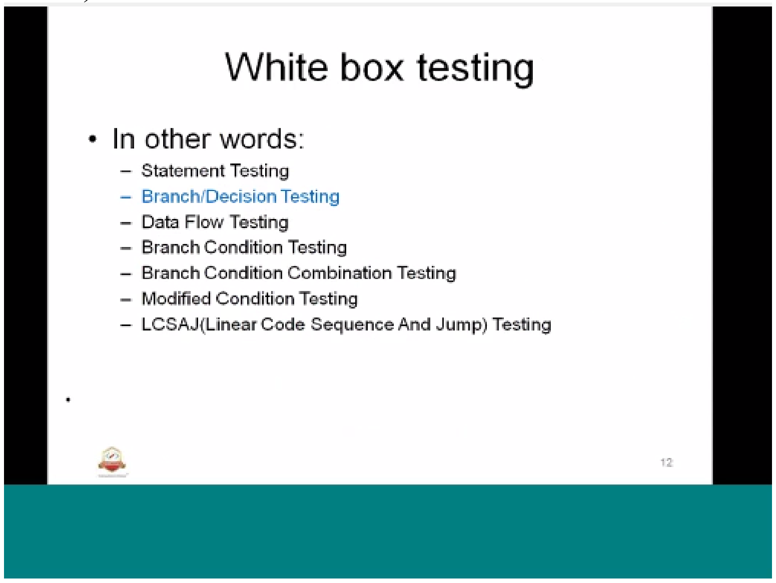
So, that also will be reached with the help of statement coverage. By following the paths through the code we cover all the diamonds and all the boxes are covered and thus we have statement coverage according to the relation with McCabe, McCabe is a person name but he is the founded or he discovered this standard measure the complexity of the program based on all these diamonds and processing statements.

So, according to the relation we have connected it measure McCabe measure it is called it should be 3 or less test cases and in this case two were enough. So, to have two paths what I said is we require two test cases. The first test case will take care of statement1, 2 and 5. And second test case will take care of statement 1, 2 and 3 and 4. So, two test cases are enough to this measure. How much of the total executable statements have been hit covered by the various test case scenarios is what is important.

For a decisive statement both true and false conditions should be tested, even if the decision statement is of implicit form. Though it is internal that means it depends on the some of the internal way of submitted files, not the external one that means to the exercised still. In the given piece of code block IF condition is false then 3 out of 4 statements are said to be covered. So, we know that the statement 1, statement 2, statement 3 sorry statement 5 have been exercised if there is a false condition exists or no condition exists.

To achieve 100% decision coverage both true and false conditions have to be achieved in two different test scenarios, even though the statements is of implicit if form. So, second form is we need to have another statement coverage with path taking as YES or through, with the help of that statement 1, 2 and 3, 4 been exercised. So, in a first form we have exercised the first one. And in the second one we have exercised the true path. That is false or no whatever you want to

call. That is how the statement coverage in terms of total number of exercised statement of each will be done. Okay, so that is what is about
(Refer Slide Time: 35:51)

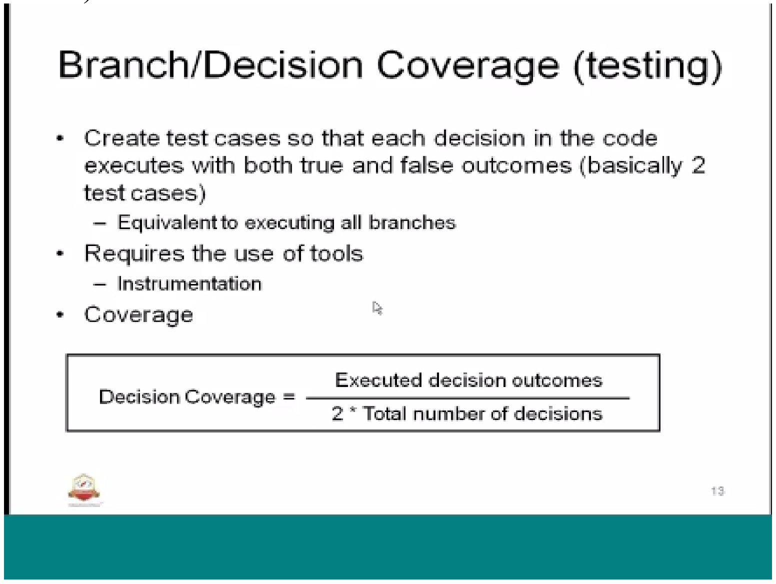


White box testing

- 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

12

Statement coverage, all the executable statements have to be reached. In next type of white box testing is on the branch testing or the decision coverage or the decision testing it is called. So, how are we going to do? Let us see. Okay, so here we create test cases
(Refer Slide Time: 36:40)



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

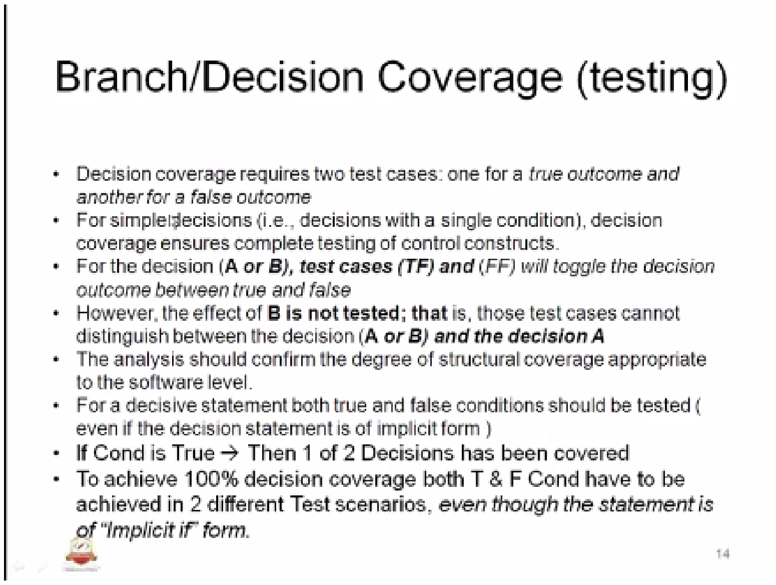
$$\text{Decision Coverage} = \frac{\text{Executed decision outcomes}}{2 * \text{Total number of decisions}}$$

13

So that each decision in the code executes both true and false outcomes, So, basically two test cases we take that simple example in the so we have exercise a statement coverage in two techniques, similarly for decision or the branch coverage testing. What we do is both true false conditions we will have it.

So, we use tools as well for these called instrument machine tools were decisions have been taken here. And here also the coverage aspects are done with the help of a percentage whereas total numbers of execution decision outcomes have been considered that means total number of

decisions how many are there against total number of executed decision. That means suppose we have a code suppose two diamonds and each diamond will have two outcomes true false true false when there are four, because 2 * total number of decision because each decision will have a two outcomes either true or false.
So, one diamond if u has excelled or two diamonds with one outcome we have exercised it means the path is to without coverage is 50%. So, this also can be done with the help of the tool. So, both are called same basically branch coverage or decision coverage. So, if the techniques similar to statement coverage so the idea with the decision coverage is to execute every decisions single decision in code at least twice it is not once, that we have to remember. Every decision in the code has to be exercised twice.
In statement coverage only once is enough here twice because the decision will take two paths. That is why we need to have this criterion very important. So both possible outcomes of the decisions is true and false could be identified in order to reach the decision coverage. Very first glance statement and decision coverage seem to exactly the same perspective at executing every decision with both true and false outcomes resulting all statements executed.
So we have to execute all statements all outcomes of every decision need to be executed that is false, so but there is a one case in which statement coverage can be reach without having fully mission coverage and that is within statement without in else class, so in that case we will not have false conditions but that is not a good programming practice to have a if symptom in some condition statement and remove else.
So in this case only one way it is possible but it is not a good practice to have one if without x, that is have to be else in some other test, some other execution, so definitely we are going to have a multiple paths true as well as false, so obliviously we still need the second test case with the false outcome to reach the decision cover. So coverage is measured by dividing the total number of execution decision outcomes and the total number of decision times multiplied by 2, so and the tools that are used in this also is the same tools for monitoring the coverage where addition is always taken care as soon as the statement later it has been meet.
So that is what basically and the decision coverage request the two test cases, one for true and false outcome, the simple decisions,
(Refer Slide Time: 42:03)



Branch/Decision Coverage (testing)

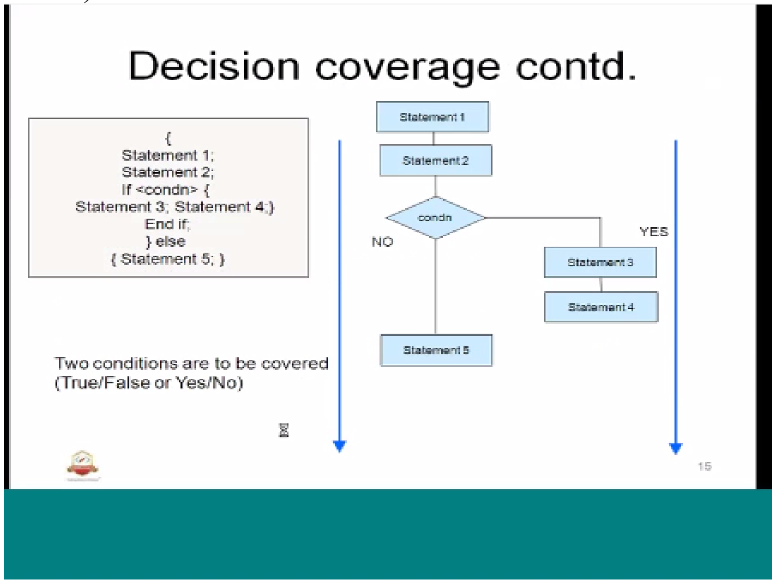
- Decision coverage requires two test cases: one for a *true outcome and another for a false outcome*
- For simple decisions (i.e., decisions with a single condition), decision coverage ensures complete testing of control constructs.
- For the decision (**A *or B*), *test cases (TF) and*** *(FF) will toggle the decision outcome between true and false*
- However, the effect of **B is not tested; that** is, those test cases cannot distinguish between the decision (**A *or B) and the decision A***
- The analysis should confirm the degree of structural coverage appropriate to the software level.
- For a decisive statement both true and false conditions should be tested (even if the decision statement is of implicit form)
- If Cond is True → Then 1 of 2 Decisions has been covered
- To achieve 100% decision coverage both T & F Cond have to be achieved in 2 different Test scenarios, *even though the statement is of "Implicit if" form.*

14

That is decisions with the single condition, decision coverage ensures complete testing and control constructs. For the decision (A or B) test cases true false and false will toggle the decision outcomes between true and false. So, we have reached that. However the effect of B is not tested. Right, that means those test cases cannot be distinguished between the decisions A and Band the decision A.

So, that analyzes should confirm the degree of structural coverage appropriate to the software level. What are the level of the software that is defining on the whether it should confirm basically whether A as an effect or not? For a decisive statement both true and false conditions should be tested, even if the decision statement is of implicit form. If condition is true then one of two decisions has been covered to achieve 100% decision coverage both true and false conditions have to be achieved in two different test scenarios even though the statement is of implicit form. Okay, so here

(Refer Slide Time: 43:07)



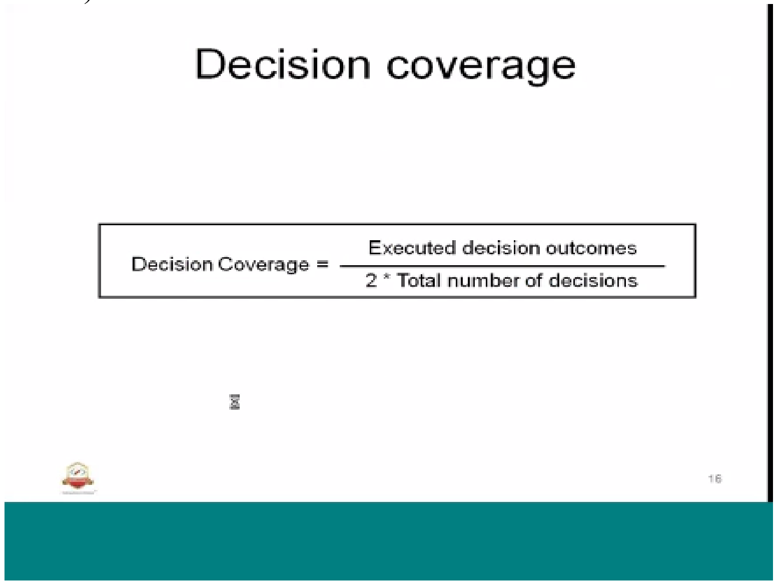
Decision coverage contd.

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

Two conditions are to be covered (True/False or Yes/No)

15

The same example you see here statement 1, 2 condition yes will take in to statement 3 and 4 and the condition no will take into the statement 5. So, two conditions are to be covered, true and false or yes or no.
(Refer Slide Time: 43:27)

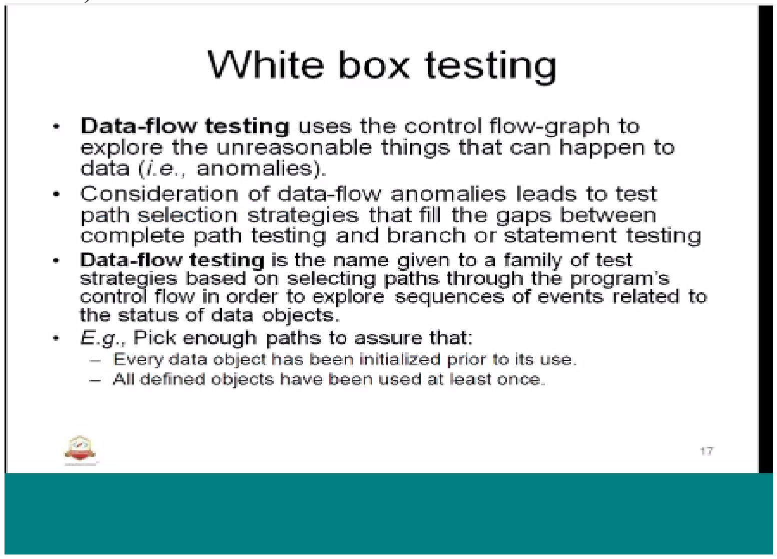


Decision coverage

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

16

Decision coverage is with the help of a percentage executed decision outcomes to the only one decision outcome is there/ total number of decisions. That is 1* 2. So, there are 2 denominator and 2 in the numerator that will be 100% decision coverage. So, that is about the decision coverage. In the next one we will see the other technique called data flow testing.
(Refer Slide Time: 44:09)



White box 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.

17

Here also the data flow testing uses the control flow-graph. Control flow-graph is you know that it is a basic control of the program how it is going to execute with the structure of the program. So, data flow testing uses the control flow-graph to explore the unreasonable things that can happen to data. That means basically we use the data in terms of controlling the flow.

Consideration of data flow anomalies leads to test oath selection strategies that fill the gaps between complete path testing and branch or statement testing.

That means we test using the control flow all these statement decision and everything sometimes what will happen is purely the flow is based on the data for certain cases. So, there are gaps by doing that selection criterion with the help of the path testing and branch or statement testing. So, in that case data flow testing is useful. Data-flow testing is the name given to a family of test strategies based on selecting paths through the programs control flow in order to explore sequences of events related to the status of data objects.

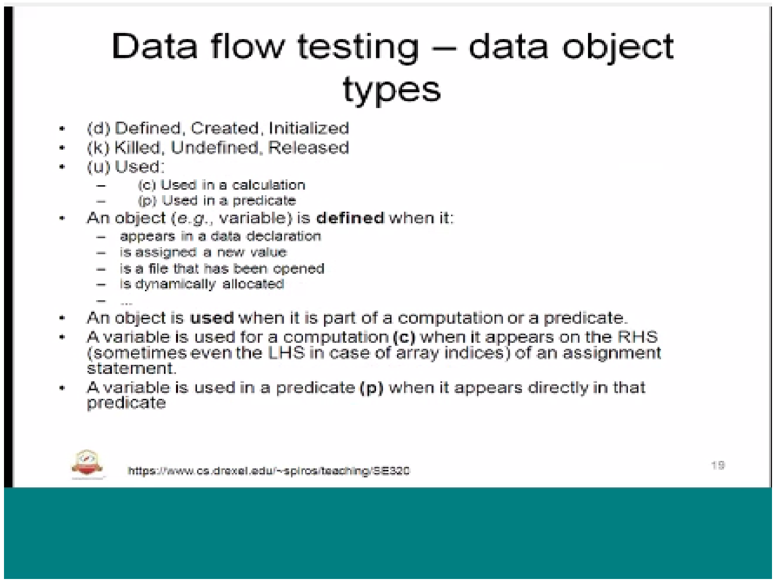
We know that data flow testing is one or the other data flow of objects where data is here data could be a variable, example pick enough paths to assure that every data object has been initialized prior to its use. All defined objects have been used at least once. So, how data-flow testing goes through? Example is that variables of the data that are used should be initialized because there is a requirement that initialization function will take care of the global variables. System gets initialized upon power up.

So, what will happen is the system gets initialized so all the variables from the data that are part of the initialization function, init function will get initialized. So, how do we do is with the help of the data-flow whether that initialization is done for each of the intended data. It could be a variable or init. So, what will happen is that init function whatever the var1, var2 we have initialized 0. So, initialized does not mean that it could be 0 it could be any value that is used as the init value throughout the entire program. So, this is very important because the data-flow is instead of 1 so I put as 0 could face because var3 could be initialized and that initial will be in static increment in the other program may use it. So, that the increment will happen if it is initialized with 1 as 2, 3 ,4 it will initialize the 0 the increment will go to wrong. Similarly, if the variable 1 and 2 are initialized with some other value than 0 the program can collapse.

That is why it is very important to have data-flow testing along with other type of testing techniques. The next one is this is also very important. Whatever the data or the objects that are defined should be used, so, very important because we cannot have a data that is dead. It is called a data object which is very not used, also called as dead objects.

So, this is very important we cannot have because unnecessarily it will forget by the memories which are new. And sometimes it will so happen there are other functions that could use the same name as 1440 in other program which is unused will result in some sort of anomalies in the program. So, we need to avoid this type of things. And this will be done with the help of data-flow testing. And further the data type objects,

(Refer Slide Time: 48:57)



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

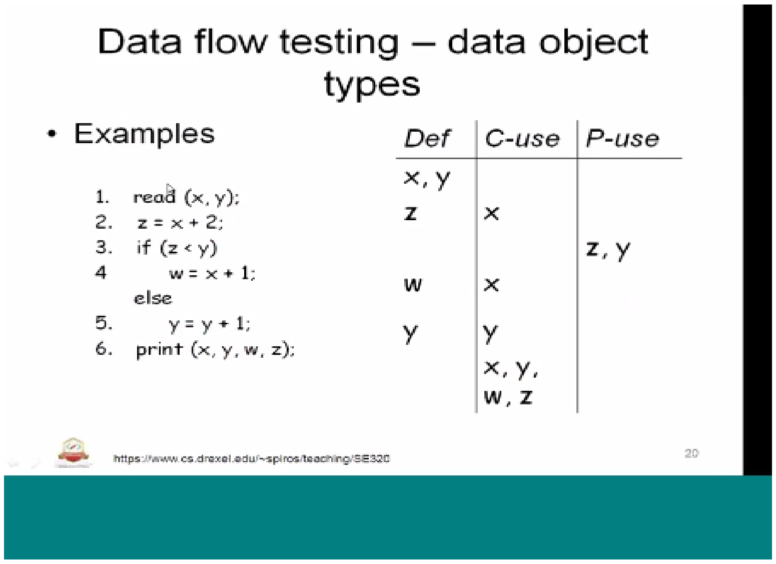
https://www.cs.drexel.edu/~spiros/teaching/SE320 19

Are defined in the data-flow testing, So, type D type of data or the objects or the types having defined, created and initialized types. K type is killed, undefined and or released ones. Mostly this is used in application embedded application or c++ applications. U type of data object is used by so used in the calculation or the predicate. An object, that is variable defined when it appears in the data declaration and assigned a new value file that has been opened is dynamically allocated etc. so, all those are called as an object.

So, there is a definitely a requirement for this and based that objective of that particular variable or the object as per the definition will need to test. So, an object is used it is called as used when it is part of a computation or a predicate. So we can see C or P type, C is for calculation P is for predicate. A variable is used for computation when it appears on the right hand side that means the variable is used when it is used for comparison or assigned etc. some sort of a computation that is called as the C type used object, data object. Sometimes even the left hand side in case of array indices also will be used of an assignment statement. Understanding RHS is right hand side, LHS is left hand side.

So, according the variables is computed. So, variable is use in a predicate when it appears directly in that predicate. That means variable can be used as a predicate by itself. That means the variable can be compared directly if varying is something then take some conditions or decisions and that var is getting updated with some sort of a ignorance of whatever it is. So, those are called predicate variables where is no assignment or RHS, LHS sort of a usage read done with the help of that variable. So, the next one is

(Refer Slide Time: 51:42)



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	

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

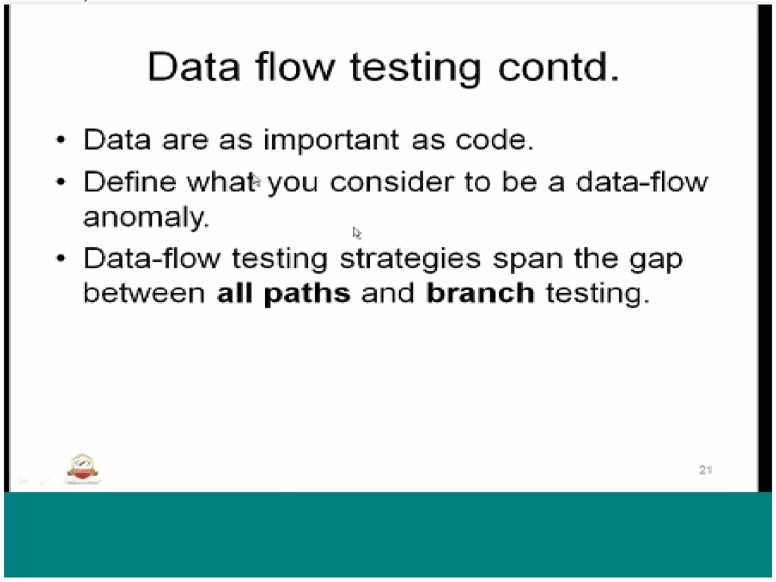
20

Examples of data flow testing-data object types we can see definitions, C-use, P-use. As I said definition is defined, created, initialized. The other one is C-use and P-use that abbreviations we use C-type of objects for predicate we use P-type of objects. We use small example program. Read(x, y), the z goes x+2, if there is condition so.

So, there is a print f statement and how are we going to segregate the different types of data objects here? Xy and x and y are defined with the help of read statement. Read function rather z is also defined one, w is other one, y is other one. C-use the computation because x is used for computations, x on right hand side also we have z, also we have y, we have w z etc because print is used we have supplied as a four parameters that is why C-use. For P-use we see we use a predicate quality where z<y is0 predicate for doing some if condition.

That is why it is called as P-use variables data object types, so, all this need to be verified in dataflow testing. So, data are as important,

(Refer Slide Time: 53:24)



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

As code because having an implementation along with the data is as important for the program, and define what you consider to be a data-flow anomaly. I mean you understand the program flow and the data that is used within the program and the technician we need to consider in terms of any anomaly how it is been used? What against what is being spoke on under the program? The data flow testing strategies span the gap between all the paths and branch testing. So, it did not just enough to have a branch testing, supplying z and y values and make sure that path4 is done 1, 2, 3, 4, 6 one path is here.

In this decision testing what we do is paths 1, 2, 3, 4 and 6 are done and other type of decision testing with the else path executed what we do is 1, 2, 3, 5 and 6 this path is executed. Still we would have missed if the variable is been assigned wrongly or z suppose have an x+2 and specification says that should be x-2 and we have missed it.

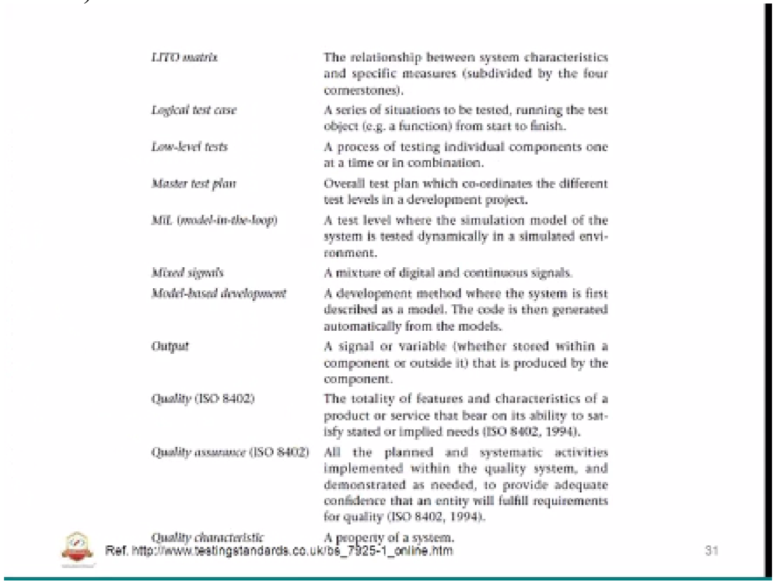
So, we believe that what are the statements is after all before the decision testing is being tested properly. So, it ios very important to have a data flow testing as well. So, how do we do? By having all the paths exercised with the focus on the data. So, there is a gap between the path and the branch testing of data flow testing takes care of.

Okay, this is how we have studied about statement testing where all the statements have to be executed and invoked man in branch or decision testing. We have seen every diamonds are the process invokes with the conditions or the decisions sorry it is not conditions, branches will be tested in data flow testing we do the data or the objects of different types will be test and in the next session we will study about the branch condition testing, branch condition combination testing, modified condition testing and the last type is LCSAJ testing and we will test base on the also. Okay, so sort of the embedded system testing was, we will go through

(Refer Slide Time: 56:42)

What we have added today's branch, condition, decision, data flows. So, along with the other words we listed these sessions how many embedded system it has done to the word. Of course there is a glossary divided some more,

(Refer Slide Time: 56:59)



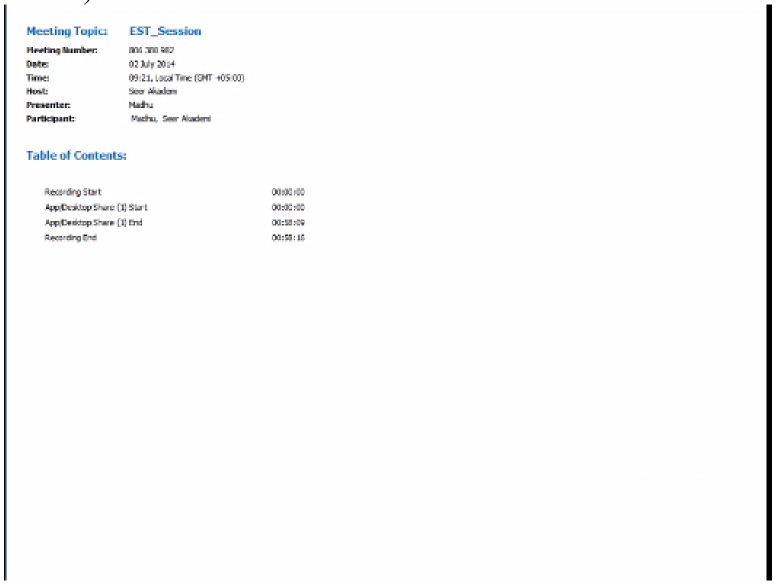
LITO matrix	The relationship between system characteristics and specific measures (subdivided by the four cornerstones).
Logical test case	A series of situations to be tested, running the test object (e.g. a function) from start to finish.
Low-level tests	A process of testing individual components one at a time or in combination.
Master test plan	Overall test plan which co-ordinates the different test levels in a development project.
MiL (model-in-the-loop)	A test level where the simulation model of the system is tested dynamically in a simulated environment.
Mixed signals	A mixture of digital and continuous signals.
Model-based development	A development method where the system is first described as a model. The code is then generated automatically from the models.
Output	A signal or variable (whether stored within a component or outside it) that is produced by the component.
Quality (ISO 8402)	The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs (ISO 8402, 1994).
Quality assurance (ISO 8402)	All the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfill requirements for quality (ISO 8402, 1994).
Quality characteristic	A property of a system.

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

31

Glossary together, such as syloto metric, logical test scope, low level test, master test plan, MIL (model in the loop), mixed signals, model based development, output, quality example (ISO8402), quality assurance, quality characteristics. So these will add likewise each sessions some of the glossary problematic, understand in embedded system testing. So with that we will end this today's session and then next session we will continue on the white box cell testing or other techniques.

(Refer Slide Time: 58:09)



Meeting Topic: **EST_Session**

Meeting Number:	806 380 982
Date:	02 July 2014
Time:	09:21, Local Time (GMT +05:00)
Host:	Seer Akademi
Presenter:	Madhu
Participant:	Madhu, Seer Akademi

Table of Contents:

Recording Start	00:00:00
App/Desktop Share (1) Start	00:00:00
App/Desktop Share (1) End	00:58:09
Recording End	00:58:16