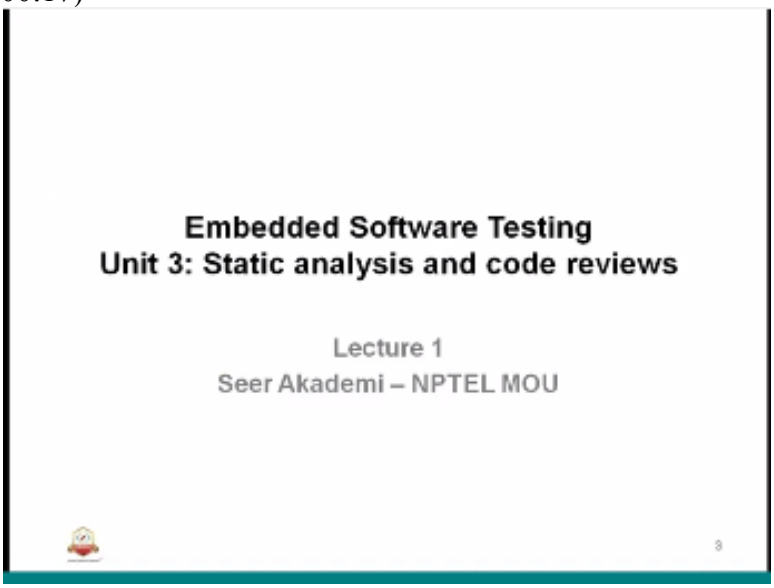
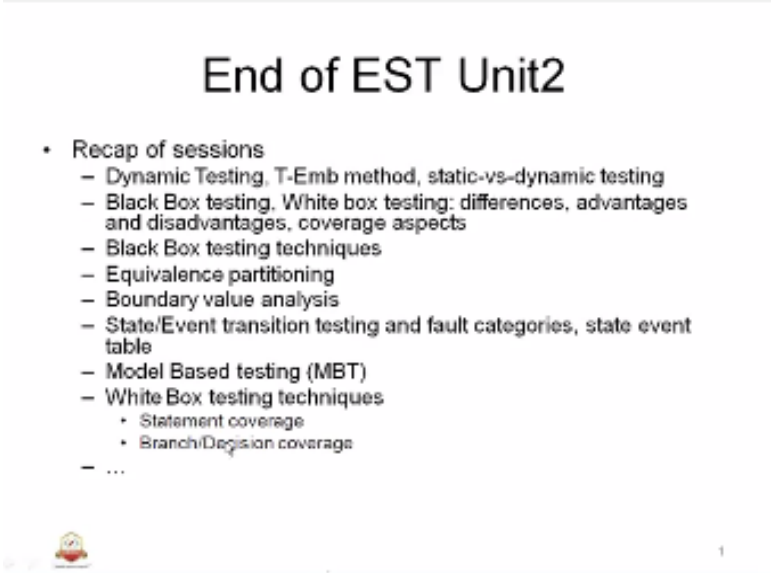
(Refer Slide Time 00:17)



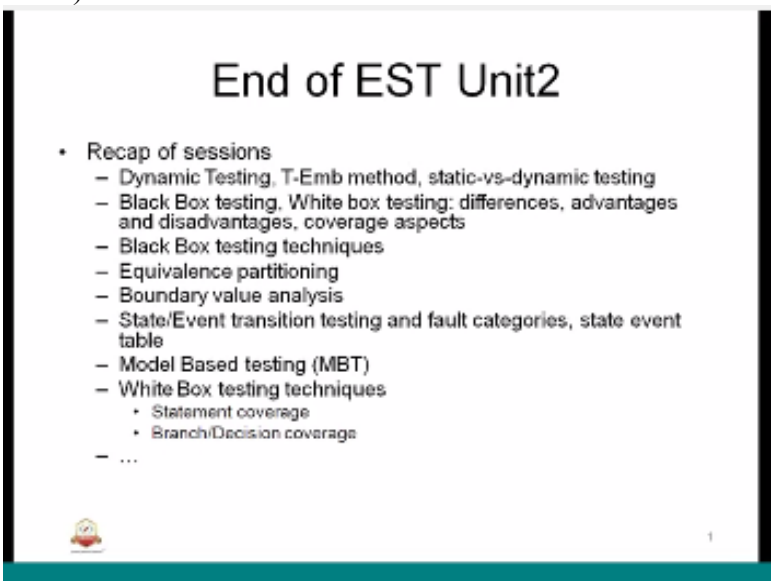
Hi all welcome you to the new section embedded software testing, it the unit 3, static analysis and code reviews. So there are few sessions in unit, studying about static analysis and code reviews inspections and matrix. So these three things we will study in unit three, so there are few session around the unit, and we know that testing can be done with the help of dynamic testing, whether, context use static testing. So in dynamic testing we know that we can test when the program is executing.

(Refer Slide Time 01:04)

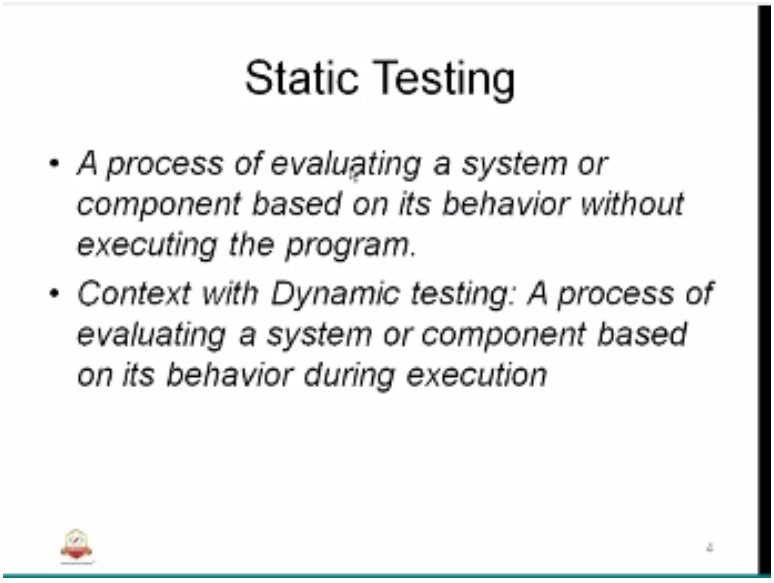


So let's do a recap of what we have done studied in unit two, unit one we will not recap because unit one we already gone through. While doing the unit two so unit one, was about the interaction, basic templates, testing aspect, etc, in unit two so, we have studied about dynamic testing, T-Emb, method static-vs-dynamic testing it differences then next part of the dynamic testing we studied about black box testing, white box testing, advantages and disadvantages also we had gone through the coverage aspects. Then we discussed about the black box testing techniques and testing techniques are equivalence partitioning, boundary value analysis, the state

event transition and fault categories, then we also have an example and go through state event table and model based design and it is testing it is called (MBT) model based testing, then white box testing and techniques we went through. So we have gone through different techniques under white box testing there are statement coverage, branch decision coverage.
(Refer Slide Time 02:32)



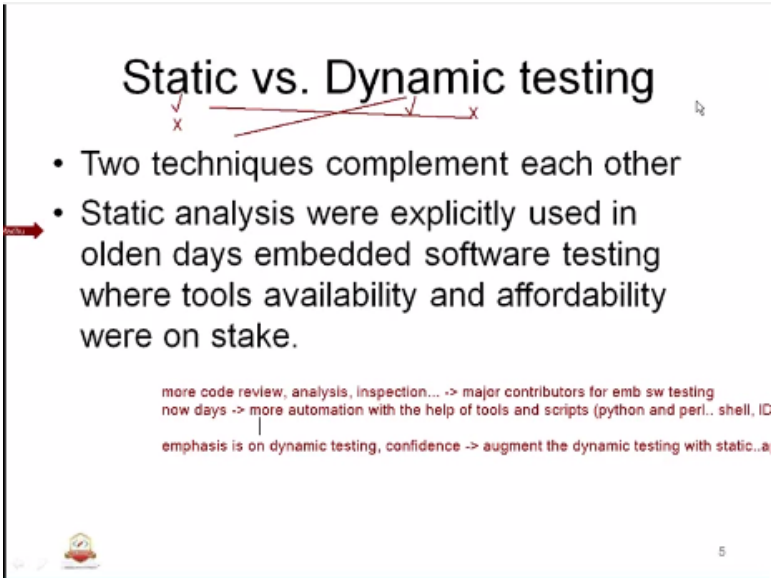
Then data flow testing, branch condition testing, branch condition combination testing, which is similar to MCDC, which is MCDC separately we studied on the (DO178b perspective) then we have modified condition testing, LCSAJ testing, codes sectors jump, then gray box testing, which is mix of white box as well as black box, then we understood about test drive and test stubs which is very important on integration or unit testing, then we go through some of the coverage tools and discussed about different tools that are used for the dynamic testing and analyzer performance, logic, memory, etc, then testing tools life cycles based on the life cycle chart on the standard book what we referred, then test automation and techniques automation branch processing we have studied here. Then the last one is risk based testing and importance of risk based testing on different strategy, how it could be done. So those are some of the sessions in unit2 in brief okay.
(Refer Slide Time 04:21)



So in next unit that is static analysis and code coverage we will try to understand what is static testing? Static testing is a process of evaluating a system or component based on its behavior without executing the program. So we know that unit testing has to done when the program is executing, we do lot of black box and white box testing techniques followed in the dynamic testing, Whereas, static testing is process of evaluating system, or the units of the components, depending on its behavior. And the behavior will be checked or verified analyzed or evaluated without the need of execution of the program. Because already we have executed the program and tested during the black box for one aspect of the testing, this is the different aspects this is also equally important to do this static testing here we do not execute the program. But the different thing are there without executing different aspect of the inputs form the test at affects, that will be used for static testing.

So let us study about that in the next slides context into context with dynamic testing, a process of evaluating a system or component based on its behavior during the execution that is in context to dynamic testing. During the execution we will do the test. That is what the basic difference of dynamic testing, and the static testing.

(Refer Slide Time 06:09)



Next one is static vs dynamic testing, so basically to do a testing as an embedded system completely both have to be done. Some of things may not be probable with dynamic testing, so other thing possible with static testing, so other things may not be possible with dynamic testing some features are definitely not possible with static testing. So likewise we have a complement of each other so, that the testing is complete in all aspects. So that is how these two techniques are mutually done.

And while doing the act we may take credit of each other, so that the coverage aspects are taken care. So, static analyses were explicitly used in olden days embedded software testing where tools availability and affordability were on stake. See just imagine that there are no tools to test in the program is executing how we can test it. So it is very difficult, so what is to be done in the earlier days, we use to run stop, run stop, where if they on the field and when the program is on alter they will extract some of the memory statically.

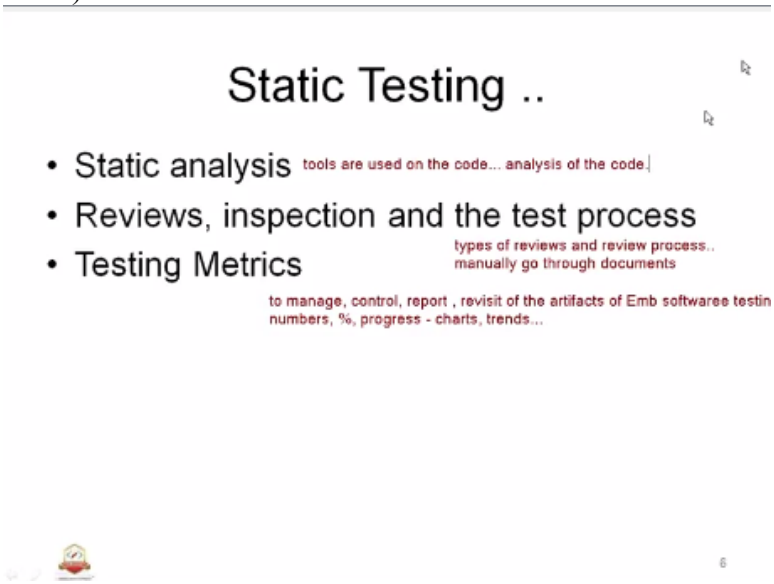
So there are fewer tracks of tools embedded tools in terms of testing under debuggers and analyzers these are very less in earlier days. That is why we used to have sort of a code review more of a code review, analysis it could be inspection. We will study about this how they are done, but I am not sure these are the major contributories for over all testing, embedded software testing. What that as changed now a days, so more automation with the help of tools and scripts I would say, there are lot of languages like python and Perl, shell. Shell could be from amen torn or the IDE short form.

So they are all evolved so, due to that what is happening is, the unit of software program that can be run independently and this tools can be applied on those run and can get the result. So what will happen is more emphasis is on dynamic testing, but to gain the confidence we will augment the dynamic testing with static testing. Static approach or testing etc, so that is how static testing complimenting to dynamic testing because so of the methods are impossible or very serious are, it is like very difficult or there is no point in doing sort of a test are there, some of the requirement it could be, some of the unit level verification, or validation all these will be done as part of the static testing. So in software development static analysis and dynamic testing are two different ways of detecting defects unfortunately there are too often sort of as competition for one another but it is not a competition hopefully. And developers are sometimes encouraged to favor one to the exclusion of the other. Because they are theirs, because they have implemented code to have different approaches and all that, so they use to be always for one of the approach.

So this inaccurate in potentially or equal integration may be consequence of confusion over the role and power of the next generation analysis tools basically, static analysis tools. So that is why to clean up all this misunderstanding, this approach are static analysis separately it is evolved and come up.

So definitely nay embedded system that they are going to have, will have static analysis and coverage. Which is done without the program, this is a important aspect in embedded software testing.

(Refer Slide Time 11:33)



The next one static testing so what are the different types are there in static testing. Which are used for doing the static testing, so there are three things that are done in the static testing, static embedded software testing, static analysis, and next one is reviews, inception and the test process, then testing metrics all these together are used under static testing to make sure that the reports and the completion, and the possible criteria are arrived.

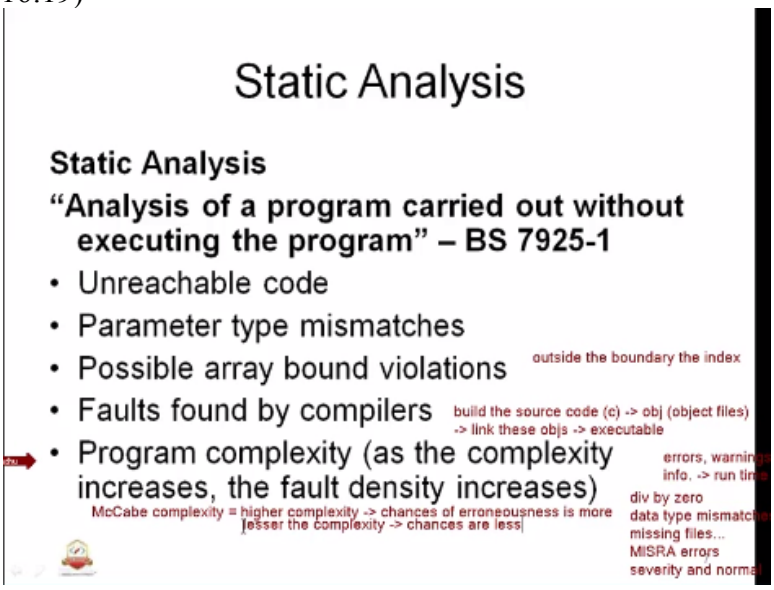
So these are the three different types, so you know that we can do a review of the requirement and the define code against what is implemented and what is working. So that could be reviewed it could be informal way through a review inception etc, under reviews. So we will study about different types of reviews and review process, is very important because every review as its own processor. What are the types of reviews? What it will bring?

Basically to go through the existing artifact and understand manually, manually go through documents. Basically what is document encoder and what is it working that should be consistence and matching, so that is how they use the reviews and inception in the test process adopted there.

So these are those specific to in terms of validating, verifying internal behavior of the system statically. And static analysis, we do with the help of tools and cheque leaves and different categories of staticals are there we will study in the next slide. And last one is testing metrics, so we need definitely a metrics to mange, control, report, revisit of the artifacts of embedded software testing. So this is very important because without metrics nothing is complete you are asked how much of the progress you have made in testing, it should be definitely telling either numbers or percentage or progress in terms of charts, trends etc, so all this will be part of the metrics. So with the help of metrics the program is well managed and controlled, singularly the metrics are used in terms of analyzing various artifacts as part of the software testing.

You could say that it is a over view, but as part of static testing this is also important so that the completion is taken as a credit in terms of overall embedded software testing. So static testing orders, static analysis, and next one is the reviews inspection on the corns poses the last one is testing metrics. So static analysis are basically done with the help of support by different tools, so the tools are applied on the code basically or the implemented part, so basically we do a analysis of the code.

(Refer Slide Time 16:19)



So, that is what we do in static analysis, so let us see each one what are they, what are the details about that, what are they do static analysis okay. Definition of static analysis, analysis of a program carried out without executing the program you know that, this is the standard definition from the standard BS 7925-1, the various aspects of analysis we do on the program without that is being executing or as for below unreachable code.

So the code of what we do is analysis the code by analysising what we will discover is there are certain pieces of code, certain software units of the code. Which are not reachable in the entire execution of the program or which are dead sort of a thing. Then parameter type mismatches, that means the parameter that is intended to be used in different function or categories are not being forward, this is a mismatch between what parameter is suppose to be used and parameters that are implemented.

The next one is possible array bound violations, so what is the meaning of array bound violation you know that we are trying to access an element or element index between the array. But where do that is been given is outside it. That is outside the boundary value of the array and to access it, the index of the supplied parameters or whatever it could be, is the used actually. So that is the issue with that particular function, the next one is the fault found by compilers, that means the compliers basically used for building the program, we know that built the source code, suppose that say that is equal. So what is does it changes the OBJ effects or the object files, then what do we do we link it text office, link these object to generate the executable. In the lecture I am just briefing this, so while doing this there could be errors, there could be warnings, there could be info, and all this have to be cleared.

Because somebody may argue or developer may have the file clear of the error, I do not care about warning, the warning are sometimes very critical such that some of the above things like parameter incorrectness is there. So of the object are properly not used and improperly assigned

then there is a index mismatch or wrong use of the index, and that is leaving to a warning and if you ignore the warning you could lead into a run time issue. So all this will lead into a run time issue, what is run time issue once you built the execute level, what we will do is, we will execute that on the target and higher that is getting because of these errors warning info.

The fault that is going to come from the program is nothing but the run time issues. So with the help of static analysis all this split faults will be brought out and of course it is depending on the type of language also. The kind of warnings the errors are getting used for example missing files could be there and data mismatch, data type mismatch. So type cast is wrongly used, possibly divide bit 0 is there, misuse of variables so these are some of the typical issues that are there, so divide by 0, because complier might not throw, but warning can be there or if you set the errors in such a level that, these are reported then it will throw a error.

So that possibility is that the faults are found by the compliers of such nature, it could be data type mismatches. There could be missing files simply it as ignored and so how it compiled, if it is not a correct file. So all these will be part of the compilers of course MISRA errors if have enabled the MISRA check while doing the completion also it will throw. You can do severity and normal sort of a rule, so better to clear up all this sort of error file. Severity errors also definitely you have to clear it up, and normal errors also sometimes leading into the issue while running the program.

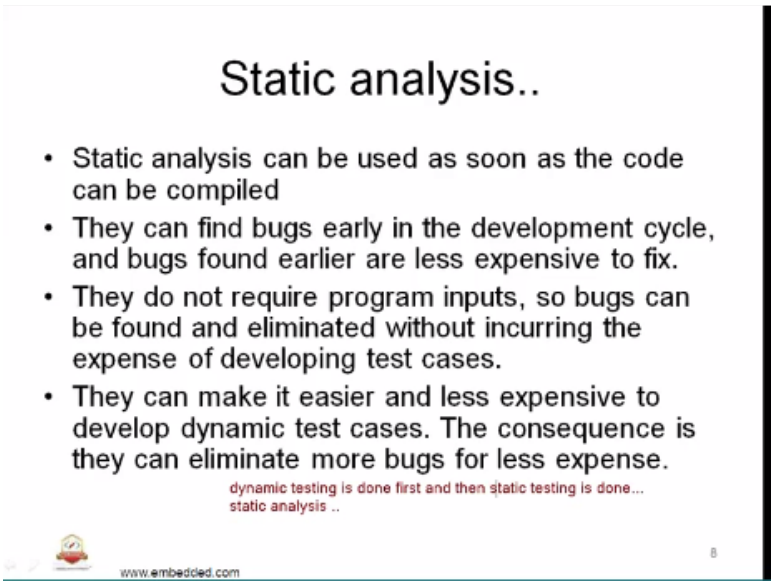
So all this have to be clearly analyzed and reported, so that is what the objective of static analysis with the help of the complier reported faults. The next one being programmed complexity, so the tools basically which can measure the complexity of a program. So what is a complexity of a program? Again it is a measure of the percentage of or the measurement of loops if statements, cases, switches and all that.

How complex the program is written, how complex the program is going to have it, all this will be part of that. So that is measured with the help of tools basically, because manually it is difficult, I will explain the complexity measurement such as make away it is one of the most popular complexity measurement tool or analysis. That they do, higher the complex or higher the complexity chance of run is more.

So lesser the complexity chance are less, that means we have high complexity for higher, lot of conditions, lot of (IF) statements, wide roots, cases, and etc, so there is a possibility that a program is become a complex and number of lines of course that also will be simple thing. We may have one single function and I would consider that as a two huge code to maintain or to test it, better to split it into stuff functions and try to use as much as well as possible. In terms of splitting the overall function.

So all this matters in terms of the program complexity is very important to avoid such things. So with the help of static analysis we will discover the program complexity and we will report the number. Definite report in number for the complexity of the program to study about what thermal complexity is using a mcCabe complexity measurement. Formula for that you will study that, so these are some of the static analysis method that I have used unreachable code, parameter type mismatches, better time mismatches divide by 0, possible array bound violations, faults found by the complier in terms of errors, warnings, info type of a report all these are consider as faults under static analysis. Those have to be analyzed and fixed, and then the last one is the program complexity that also will be analyzed with the help of static analysis.

(Refer Slide Time 25:16)



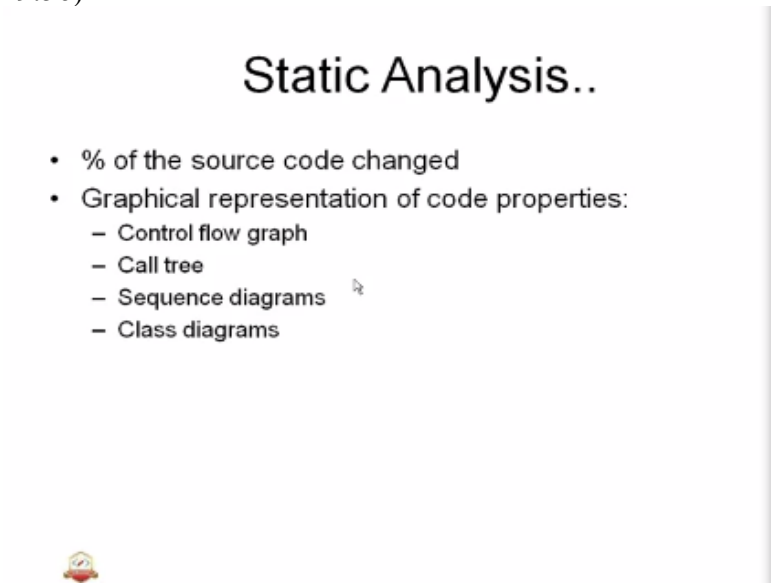
So continuation static analysis, static analysis can be used as soon as the code can be complied that means code as to be compiled it should be read. It is in a good complied state for the static analysis to be done otherwise static analysis tools may not work properly. So they can find bugs early in the development cycle and bugs found earlier are less expensive to fix. We know that as we progress in the embedded software development towards the end it is very difficult to fix and retest and deliver the product and to be more expensive to do that, then if the bugs are found in the earlier stages of the development cycle.

And static analysis they do not require program inputs so bugs can be found and eliminated without incurring the expense of developing test cases. So sometimes what will happen is we may not need a test case ideal it is better to have a test case. Over all like in terms of identifying different aspects of the analysis but more or less how it is done is manually it is done with the process what is being laid out. But there may not be a need of test case development separately, but the program itself. So we do not need to worry about the inputs with the program and all, so we can eliminate the bugs. Without incurring the expense of developing test cases, so that is what the meaning of this.

So static analysis they can make it easier and less expensive to develop dynamic test cases. The consequence is they can eliminate more bugs for less expense. So in embedded system in industry basically what I have seen is dynamic analysis, dynamic testing is done first. And then static testing is done, I think this will may not be a, what I called that a right approach to attack an embedded system. Basically those systems having complex nature why because some of the issues are the problems can be found during the static testing and where we do the analysis and issue is found in the earlier stages.

So that before doing the dynamic testing, these are all can be found out, because the expense of static analysis is always less then dynamic one. So that is why it is very important to do this static analysis, of course it may not be 100 percent, because, it is just complied so the issues in order to be fixed in functionality, still the functionality are not matured so the code is bound to change the analysis. Static analysis may get impacted, but by we are first cut of what the program is to understand what is going on better to do a static analysis. As a first hand information that can be used to fix so of the easier bugs for the dynamic testing can be done. So that is what static analysis is going to be useful, so it may not have to integrate completely with the rest of the program.

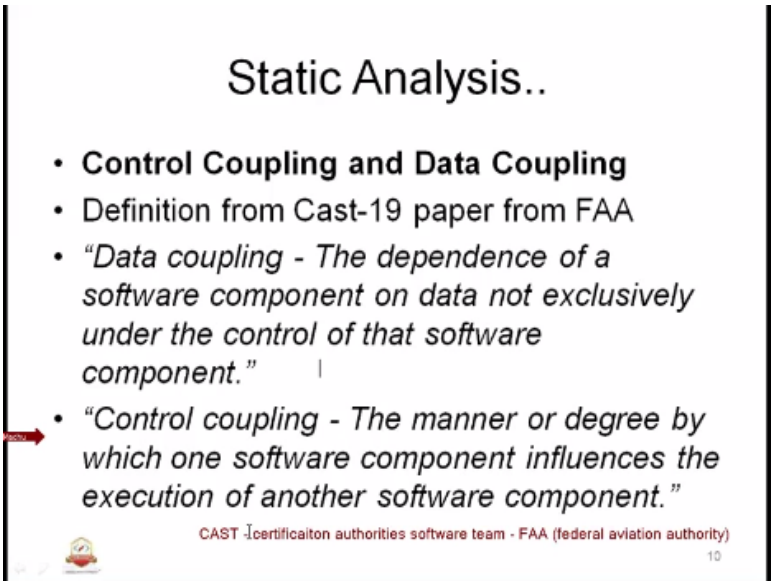
To find the bugs on techniques but still we can do a static analysis, you can also go through a single file also a data base also how it is getting the structure of the program or entering in between this. So all this can be consider for static analysis so the results are better when the entire program. But analysis of small purse can be used, thus developer can get very quick feedback on their code and the quality of the code so that is the idea of static analysis. And doing static analysis that are the things that can we do easier
(Refer Slide Time 29:56)



The percentage of the code is how much have been analyzed what is the change that is going to happen, and the various analysis aspects of the source code where the potential issue are there all this can be analyzed with the help of tools. The tools can tell, basically the percentage of source code where and all the issue, so before we start developing a test cases etc, with the help of the tools. So then the graphical representation of the code properties we can use it, basically where representation could have for the following four types of reports.

Such as control flow graph, so we will study this later, control flow graph is something like how the control of the program is going to be done and executed. And next one is the call tree that means what are the total numbers of functionalities both are the procedures. How they are getting invoked and who are all invoking both is important here. We will probably go through a tree, call tree with an example tool called understand for (C++) from sky tools, so call tree is very important invoker as well as invoked function or procedures.

So by this we will know how the flow of the program the control flow is going to be executed when the program is run. Then we have a different event and the sequences that are part of the embedded execution. So sequence diagram will be there that is useful for static analysis, so all this part of the code properties are different architectural inputs such as class diagram also used. So probably I will try to put few inputs on the class diagrams, sequence diagrams which are useful in terms of representing the code and their property. So these are all the unraised statically either it could be with the help of tool or it could be done manually so that is how static analysis is done with different aspects of the code. Basically it is done on the code,
(Refer Slide Time 32:58)



And the next important thing is control coupling and data coupling, it is a very important item they used especially they use in aerospace industry. So what is control coupling and data coupling? Definition from cast paper, cast is the certification authority software thing, so this is a dedicated team basically. Certification authorities' software team that is what is cast so that is the group basically they will define how the static analysis should be done and all that.

So as per that paper this is done by FAA, it is nothing but US autonomous organization. Federal deviation authority, for every airlines to fly or to certify FAA will help or FAA will have to mandate that this product is good to fly, so like it is not only for the entire airlines are the entire aircraft but also for the different components between the aircraft a complete avionic bus and software, hardware and communication.

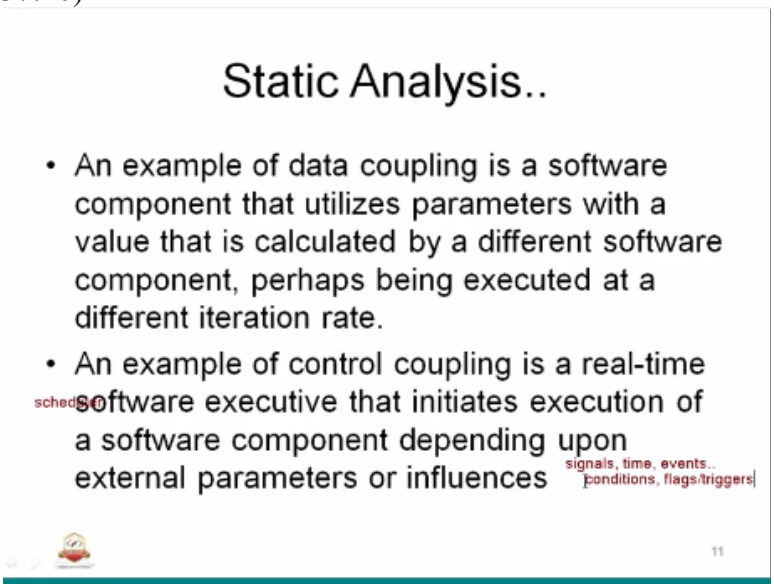
Whatever we are going to use there is a standard, so as for the standards they will verify and certify without the certification the air lines will not delivery the aircrafts. For certifying that the use certain analysis documents so they are known as cast papers let us call. So what is the definition of control coupling and data coupling on this paper data coupling, data coupling is the dependence of a software component on data not exclusively under the control of that software component.

That means the data that is going to be used in the entire software component, but it is not entirely getting controlled with that component. But it is having a dependence on various other components that is what the definition of data couplings. The other aspect is control coupling the manner or degree by which one software component influences the execution of another software component.

We know that there are functionalities those functionalities are divide into various sub functionality or sub procedure or coupling. So each one will have a dependence so how they are binded so what is binding between them is what is getting defined in control coupling. So these two are very important these two have to be analyzed in this static analysis affect with the help of various analysis tools or reference or standards, such as tasked from the FAA team, so with the help of that the data coupling analysis and control coupling analysis are done.

So this will bring the dependence of the data that are exclusively used outside and inside the software component. Also the various components influencing the different components will have a some sort of a controlling mechanism that flow and the binding of the various

components also will be analyzed. So with this analysis control coupling and data coupling will be reported. So this will help in terms of static analysis that is been reported.
(Refer Slide Time 37:40)

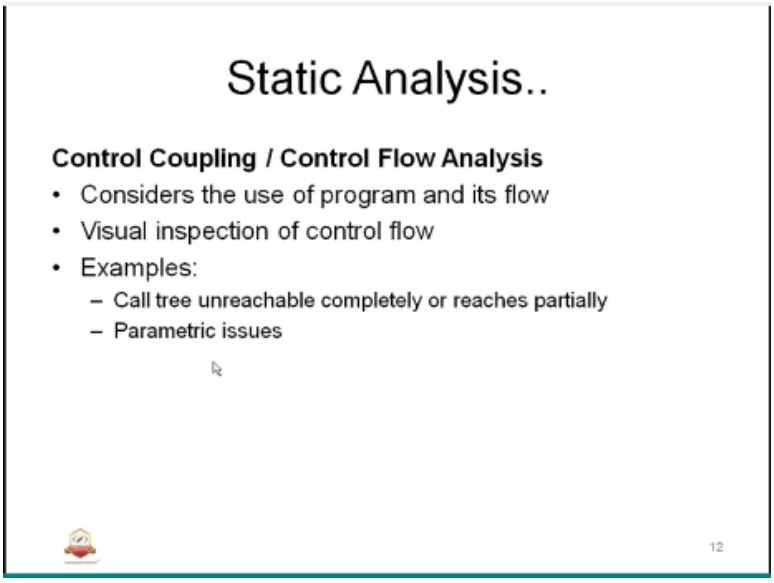


So next one is continuation of data coupling example, example of data coupling is a software component that utilize parameters with a value that is calculated by a different software component, perhaps being executed at a different iteration rate. So we have a software functionality that utilizes the parameters that are coming as an input with the value. But the values are computed or calculated by the different software component, so there is a coupling between that component as well as the utilized component. So and of course with the continuous MN component also it can be updated at the different rate.

How are you going to iterate that so that is what this data coupling will bring out? The other one is example of control coupling is a real time software executive that initiates execution of a software component depending upon external parameters or influences. That mean in the real time embedded systems we know that external parameters could be a signal or time information or an events etc, all these will influence in terms of the program flow so how this program flow is getting identified or executing is what we are talking about control software.

So control coupling example is real time software executive, so basically this scheduler, so that initiate execution of different component depending on the time it could be depending on the signal, or depending on the conditions or some events or some external flags or triggers are done. So that will have the dependences so all this will be exercised under the control coupling. So that is what about static analysis of control coupling and data coupling.

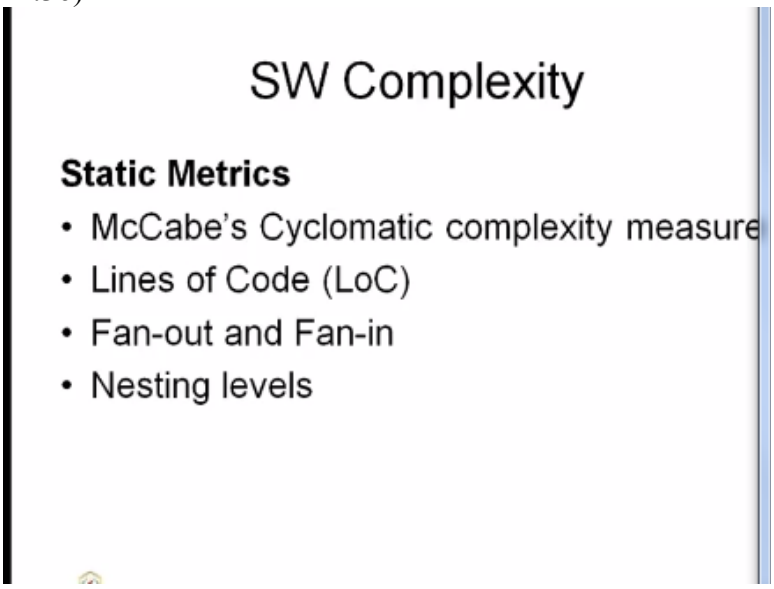
(Refer Slide Time 39:55)



So I think we will study little more on control coupling, control flow analysis is also called as control flow analysis it considers the use of program and its flow. Visual inspection of control flow is also very important so that we know where the flow is going, who is calling or who is getting called? Example call tree unreachable completely or reaches partially, parametric issues. The parameter is supplied by with the help of parameter, the call tree is not reached completely or the call trees partially reached and the program is not executing properly.

So these have to proper in control coupling and control flow analysis. So it is very important to understand control coupling and data coupling aspects. And you should not get confused with data flow testing especially data flow coupling and any other testing aspects like we did on white box, black box and all, this is the different aspects part of the static analysis.

(Refer Slide Time 41:36)



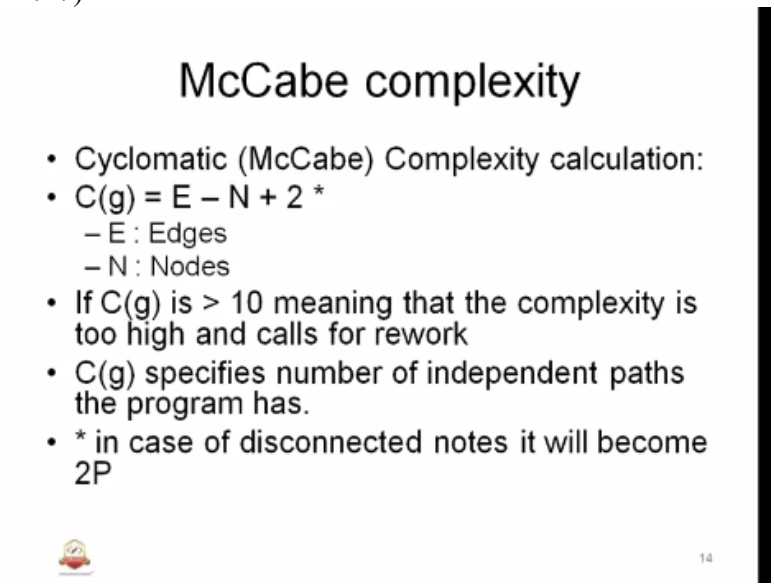
The next one is, the static metric that are used for software complexity, so we will go through static analysis with the help of the tools also the testing metrics with the help of mcCabe complexity etc, followed by that we will study the reviews inspection in the feather sessions. So static metric is used for static analysis with the help of tools as I said software complexity is one

of the important aspects of software static analysis mcCabe's complex secramatic complexity it is called, measure is one, lines of code is one, fan out and fan in, nesting levels so these are some of the metrics or matrices that are used. We will try to understand at least the McCabe complexity.

So what is the mcCabe complexity how it has come, so we know that a piece of program or the software as number of diamonds. Diamonds are within the process are the decision doxes. So in more decision are there in the code the more complexity it is that we know that so statement coverage, decision coverage, and mcCabe coverage all three have to be taken care in terms of coverage aspects you know have studied for white box testing, statement coverage, decision coverage and all that.

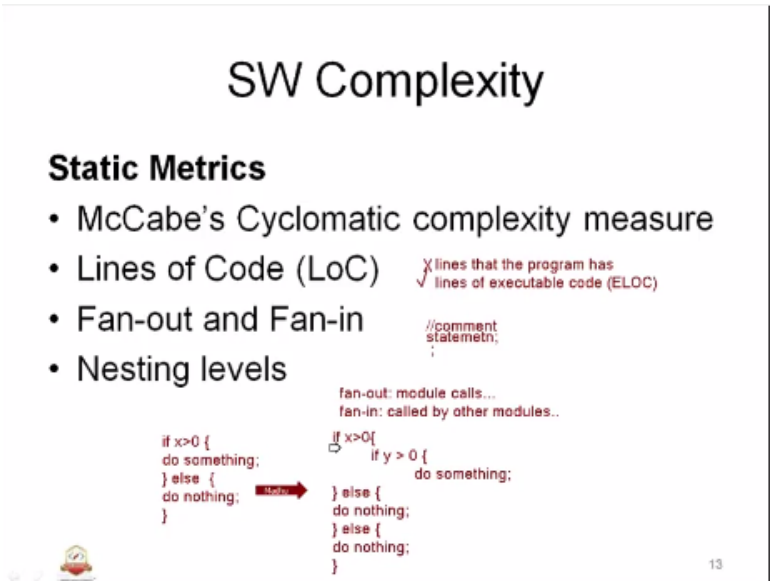
So mcCabe coverage is also very important in terms of whether we have covered them completely. All this will be followed and dependence taken care while doing the testing. So there are tools that can measure the complexity of the program, it also presents percentage of loops IF statements etc, I complexity we will have more problems in terms of maintenance and equations and it is a electron chance are high, electron software but extremely complex program we need sometimes in terms of close work control system. With the help of skill people it is getting developed.

(Refer Slide Time 44:27)



And mcCabe complexity so there are number of decision in a program or the control flow graph, so here you can see the edges the program as and nodes that the program as this is how we will give you the complexity measure that is how the calculation is done and we know that lines of code is also required. So basically a line of code is measurement of size of the program basically. A line of code is nothing but the executable lines of code which will not be the lines that the programmer has. It is the lines of executable code, it is also called executable lines of code so, you will not use the line of code just, but we can use the lines of executable codes such as statement, followed by semi clone I am talking about C language.

(Refer Slide Time 46:27)



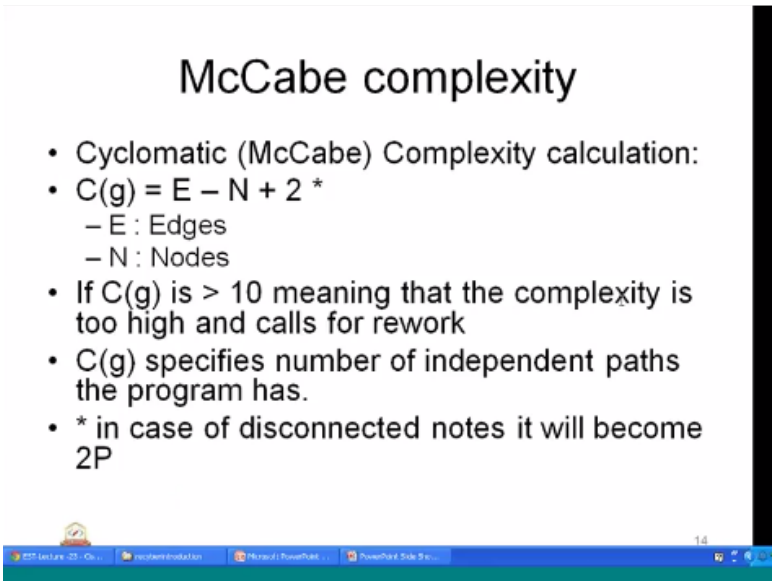
Above that we have a comment, so comment is also a line but it is not consider as an executor. Similar we have in a loop or for loop, semi clone itself as a one execute of a line that is also consider as one line under the LOC. Then we have the fan out and fan in, fan out is the amount of modules that is total number of module calls. Modules in the high fan out are often found on the upper part of the call tree that means the higher part of the call tree.

We will study about the call tree, fan out will be more, fan in, is the amount of module is that all it specific modules. So try to understand the difference fan out is the modules I have give module calls. How many calls it will make fan in, is the other way where the amount of modules that call a specific module that means current module is called by other modules. So this number of fan out and fan in also one of the static analysis metrics that is used in software testing.

So the next one is the nesting level we know that nesting for example many IF statements nested into each other getting into a deep nesting level. So if we will have another (IF) else again (IF) like this so it will go deep so this could be leading into a nesting level. So we have to understand what is the deep level, this will also be coming under as a edge in terms of mcCabe complexity so this will also be measured.

So code will be very difficult to understand so it is more difficult when the complexities are the cyclomatic complexity of the program is high. So simple nesting level is, if (X>0) do something else do nothing, so this is one nesting level. Suppose we have more IF's within that then it will become multiple nesting level, if (X>0) if we appropriately aligned if (Y>0)then do something else, or nothing and now we have come one deep, we have done. Next is done when (X>0) else do nothing, so this as complexity level as one, this nesting level is more than one here nesting is two, we have edges more edges, that is why it is more than one.

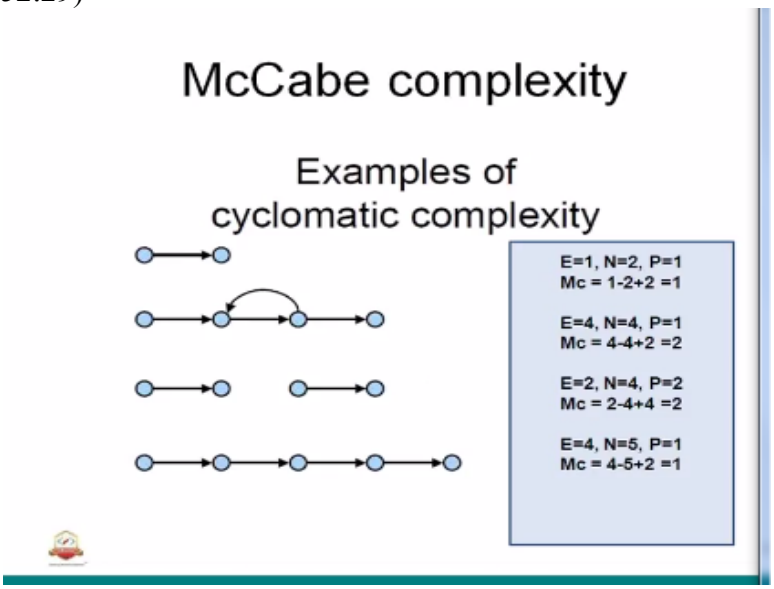
(Refer Slide Time 49:53)



We tried to understand that more in the next slide, mcCabe complexity is also called as a cyclomatic (McCabe) complexity how it is calculate is (E_N+2) so E- is mentioned as edges, N- is mentioned as nodes, and I pick this way, so we have program going this way and we have flow, functionality, nod1 then we have a nod2 then we lead into nod3, here we have three nodes and this can be also called as nod1, nod2, nod3 so the else part if it is a diamond and it can become a nod4, so here we have 1,2,3,4nodes and edges you can see1,2,3,(4_3+2) that is (1+3) so this is the complexity of this program is 3, likewise it is getting calculated.

So did you understand cyclomatic complexity is the McCabe cyclomatic complexity? In other words it is also can be consider as edges or the links in the pro graph and (_n) n is number of nodes in the pro graph plus number of disconnected parts of the pro graph or plus2 is the default number.

(Refer Slide Time 52:29)



Let us straight away understand a simple example of cyclomatic complexity this is from the reference for web I can just explain you in the first one we have two nodes and one edge, you can see edge is one, nod is 2, and the P is 1, that is the disconnected part there is no disconnected

part so only one connection we have, so we use something like mcCabe as (1_2+2=1) that is edges, 1 edge, 2nodes ,1edge_2nodes+2will become one, that is in other way it is also the in terms of parts it can be edges_nodes+2p, or simple edges minis nodes plus 2, because mostly the disconnection is going to be very less.

In the many of the flow, it will be usually connected considering that also, we can use this formula where 1 to P is used, P is the connections, so one connection is there, there is a default in 2, so that will become 1, complexity is 1, in this case the flow between different nodes we can see this will go this way it will reach here, for some condition it will come back here, it will go again, so like this flow is there.

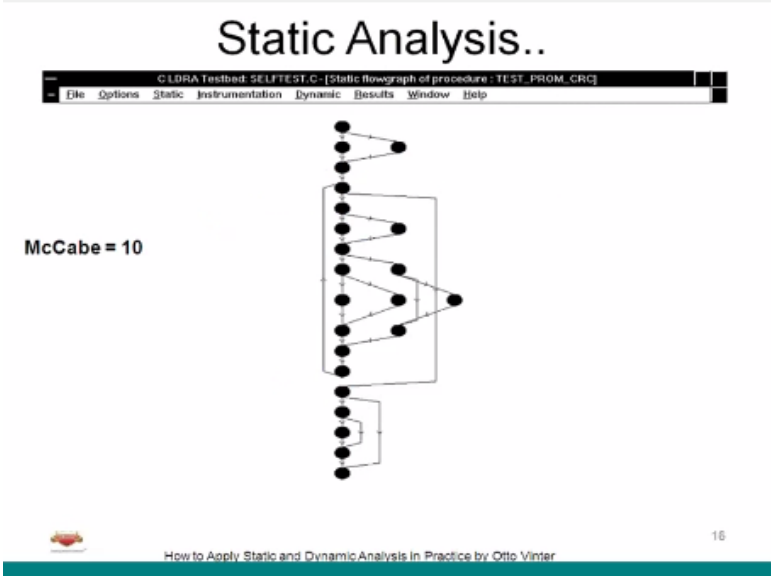
So what is the complexity of this program you can understand this see it is like a for loop for this condition whatever it could be, but the flow is something like this so what is the complexity here we can see 1 is her and the other one is like this, so 2 that means we have how many edges 1,2,3,4 edges, we have how many nodes 1,2,3,4 nodes, I will write it for your understanding 1,2,3,4 there are four nodes and edge1, edge2, I consider this as edge3, and edge4, so four edges are there, so what will be your McCabe complexity for this (N,E_N+2) here (4_4) because for this four nodes (+2) will become 2, so the complexity of this program is 2, similarly here another program you can consider this as a multi eternal task sort of a thing, where we have two parts.

So in this case the complexity is two, I leave to you how you're going to calculate because we know this formula, the last one is the is the straight forward you know that there are five nodes and four edges (4_5) to become (_1+2) so single 1 level of complexity is there in this type of program. So that is what is a definition understanding of the complexity, so we have the complexity in this the program is have more what does it called complicated or flow is more the control is more deeper so generally if the complexity is greater than 10, meaning that the complexity is too high, it calls for rework.

That generally in embedded industry automotive or aerospace, the complexity as to be around 10 or less than 10, considering the whole program if it is more than 10 means. The complexity is high so usually what they do is? They will ask to rework on that for reworking there are different methods and all that. That is how developer are work, they will split the functionality, or the condition that underneath the complexity to make sure that the mcCabe complexity is within the limit of 10, complexity specifies the number of independent paths the program has.

So you know that here the path is two, the first path is direct another path is from third nod it will come back to second, third, forth. So two paths it can execute so it specifies the number of independent paths, the program has. So in this case of here first and last nod the path is only one, so it is as simple as, so the star what is mention here is, in case of disconnected notes it will become 2P, so that is what the meaning of mcCabe complexity and its analysis, this as to analyzed and reported basically.

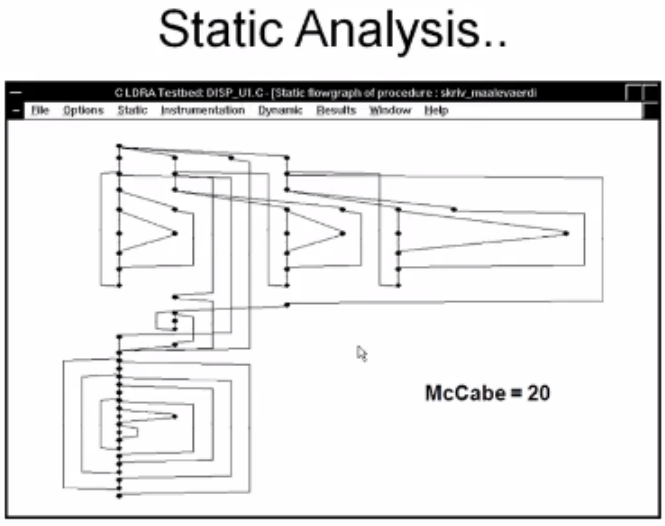
(Refer Slide Time 58:07)



The next one is few more in tried to put it based on a book how to apply static and dynamic analysis and practice by (auto winter) it is a good book you can study. You can see that and here some examples I have taken from one of the sample, so this report mcCabe complexity is been reported with a help of a tool called LDRA, LDRA is another tool they use it for finding the mcCabe complexity.

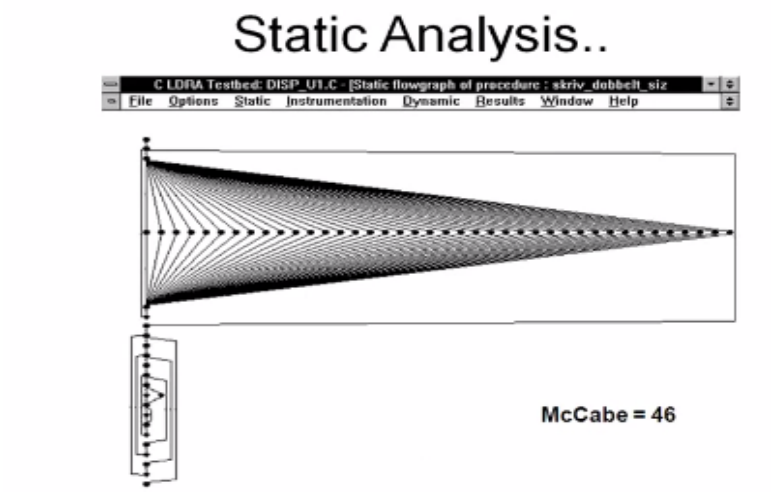
So here you can see mcCabe is 10 that means just complex a complexity is just in the normal change you can see number of independent paths so one is the, one here you can see two, three, four, five, this is six, this is seven here and eight, nine, ten like this, complexity is being arrived the tool itself will produce It you will have to calculate manually it is basically with the help of the formula which have spoken with in this slide.

(Refer Slide Time 59:35)



(Edge minis nodes +2) similarly we have more complex you can see here the flow of the different edges between the nodes and the complexity here it is very much the program is displayed program, so displayers of this be different complexity.

(Refer Slide Time 59:44)



So the complexity is 20 in these cases the more complexity you can see here it is very complex where the complexity is 46 definitely it is calls for a rework. So that is what is about the complexity, control flow analysis, mcCabe complexity, so in the next session we will study about the data flow analysis and the metric path, so that is the end of the session today.