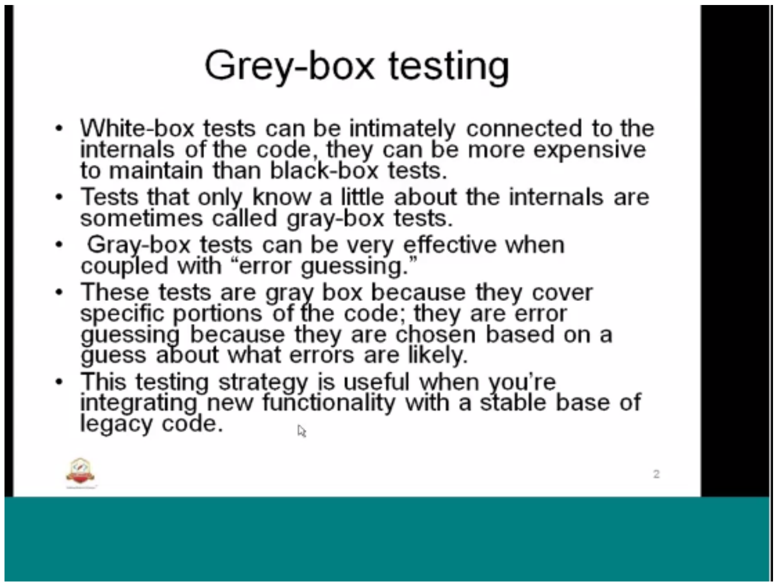
Hi, all welcome to the next session of embedded software testing with series with lecture 21 so today we study more about white box techniques and continue our white box testing understanding with more details and we will try to conclude this unit 2 in today session and in yesterday session we discuses about the web techniques of white box testing etc branch condition testing.
And before that we study that statement testing branch decision testing data flow testing so in branch conditions we know that the testing does source code which kinds of decision and the individual Boolean operands and the Boolean operands with the decision conditions will test with the help of decision conditions testing are also called as branch conditions testing the next of type of testing which is also called in terms of branch conditions.
In combinations testing where the testing aspects will be done on the source code which reorganize the decision individual Boolean operands within the decision condition all the possible line is that are going to be feed into the Boolean operands such decision will be tested so accordingly test cases will be define such way like that independence is achieved then we have studied about the modified conditions testing.
The outcomes especially the result whatever is going to arrive it will be tested also the last one the linear code sequence and jump testing here we assume the three types of TR testiness ratios where TR1 is number of statements, statements coverage basically TR2 is number of control flow statements versus control flow protocol branches and the last on is LCSAJ executed so that will be used its initial but no there a use the other type testing.
So we also went to some examples and there is a LCSAJ table and total number and the start line finish line and jump line addressed the last one is the D1 sincerity specific testing is called MCDC basically D1 process addressing to each life cycles activities such as planning development and testing, testing is called as integrate process and under development it has the requirements, design coding and integrations and we have studied about the few examples of the AND GATES the OR GATES table can be arrived so this typing of testing is also called as a truth table approach where all the possible combinations which we see the independences as well as the outcome of the testing will be tabled and according to the test will be prevent okay.
(Refer Slide Time: 03:58)


Grey-box testing

- White-box tests can be intimately connected to the internals of the code, they can be more expensive to maintain than black-box tests.
- Tests that only know a little about the internals are sometimes called gray-box tests.
- Gray-box tests can be very effective when coupled with "error guessing."
- These tests are gray box because they cover specific portions of the code; they are error guessing because they are chosen based on a guess about what errors are likely.
- This testing strategy is useful when you're integrating new functionality with a stable base of legacy code.

2

So in today session we will study about then gray box testing a I said in before class it will mix of both white box as well as the black box sometimes what will happen is the black may not sufficient to coverage or the distributions in the terms of testing similarly the white box sorry the black also may not be know so into balance between both of them that specially the integrations test cases system some of the system black.

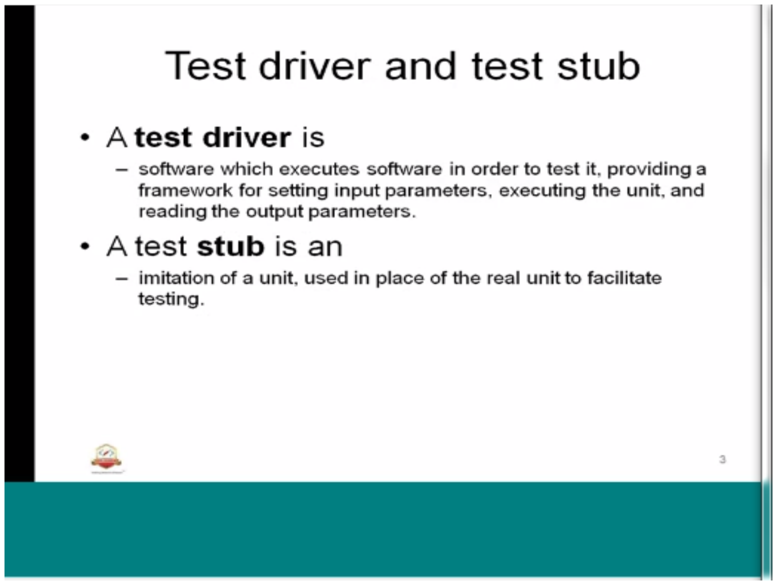
And some of the white box coverage we need to be balanced so with the help gray box system that is the addressed white box test can be intimately connected to the inputs of the code they can do more expensive to maintain number of approach test because of the complex screen other aspects as that one looking about the input of the control gray box thus here what I said is some knowledge of the internal will be more gray box testing can be very effective when coupled with error guessing so another type of matter called error guessing.

Where the error are guessed based on the knowledge unit under test and that will be implemented the test design will be applied accordingly so this test are gray box because they cover specify position of the code they are error guessing because their errors are based on the guess about what error are likely that means that test will be carried out as a pre test sort of thing the it deserve a knowledge about what could be the likely failures.

That could come based on the his knowledge about the system and about the code so he will balance between specify portions of the code as well as the black box features so he will apply certain test cases those test mechanism is called gray box testing so this testing strategy useful when you are integrating new functionality with a stable base of legacy code so that means what we have a base code which is working most of the time.

And which does not have much issues or it does not have unknown issues and they is a new cases of code or the functionality is being add so what we do is we will try to understand how the functionality is implemented with the help of the piece of code and with the help of the system understanding of the knowledge we will try to apply new test cases so this kind of new test cases mechanism is called as gray box testing okay.

(Refer Slide Time: 06:44)



Test driver and test stub

- A **test driver** is
  - software which executes software in order to test it, providing a framework for setting input parameters, executing the unit, and reading the output parameters.
- A test **stub** is an
  - imitation of a unit, used in place of the real unit to facilitate testing.

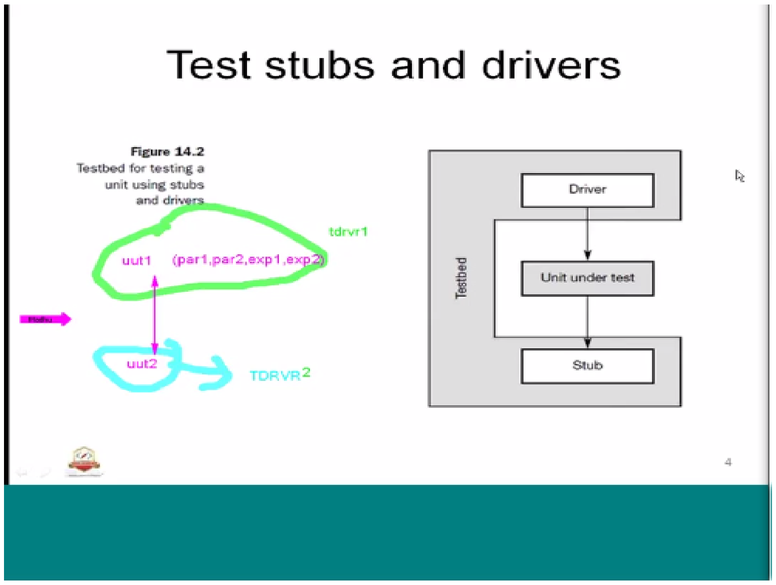
The next one is all are basically the additional details what I am trying to give you in terms of white box testing the main methods in white box testing coverage and all we have covered in earlier we will act to that they are different testing methods and test philosophy and the test details that is to be studied okay so we will study about it test driver and test stub so what is the test driver what is the test stub you might have eared about this so all the embedded systems having embedded system testing mechanism.

We will apply this test driver and test stub particularly useful for software testing where unit test is adopt so okay so the test driver software which execute software in add to test it that means test driver is again a piece of software that is test software which executes what the software that means the embedded software unit which is under test to test so providing a framework for setting input parameters executing the unit and reading the output parameters I have a diagram in next slide I will explain it you can understand it batter.

So what will happen is sometimes some of the input parameters executions and expected output may not be possible with the realistic inputs so what we need to have is development of the small driver which drives all this inputs and call the piece of software and drive test and express the result so with the help of the test driver this will be done a test stub imitation of a unit used in the place of the really unit to facilitate testing.

So this the complement basically of the test driver on the other side where the actual piece of software which is suppose what will replaced with the stub so that we know that the tester is working or not properly to see that what is expected and the same thing will be replaced actual piece of software to compare the expected result and arrive it conclusion result the test have been passed or fail so that is how test of the used.

(Refer Slide Time: 09:22)

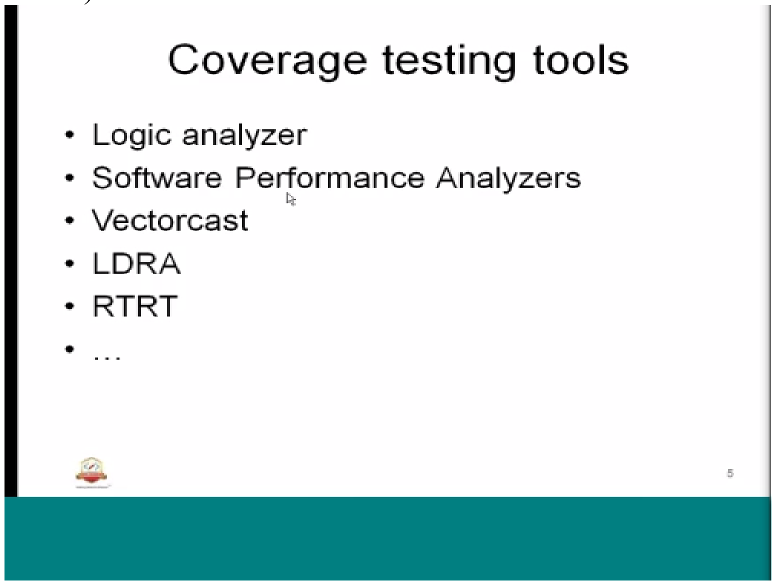


Test stubs and drivers

Figure 14.2
Testbed for testing a unit using stubs and drivers

You go through this diagram which will help a clear picture of all the test driver and the test have been used so basically the book refers to this diagrams test stub of course test particular test for set up for environment having both the driver on the even side and stub on the other side and the unit is used in the middle you can see the driver calls unit in test and unit in the test can be replaced with the help of stub is like a wrapper we can have for this.
So what does the wrapper do basically whatever the information that we need to get it from this unit you will be driven from this driver and wherever the possible expected output that is expected from this unit will be turn with the help of the stub so stub will be replaced the actual unit which is unit under test so that is the basic of test stub and drivers so interfaces between two systems, two system part can why it is used basically test stub.
And test drivers are basically for the interfaces so two system parts not to test the interface on the none side we use the driver because the other side w are going have the interface and when their closely in conjunctions we be not able to test it so for example so we have a unity one and suppose we have a use the unit two and we want to test the unity one with parameters set as parameters one parameter two and expected output as expected output one, expected output two and how does the unity one is going to intimate.
Is with the help of the unity two so what will happen is that is interactions between these and what we are going to do is we are going to replace this piece with the help of test driver and similarly vice versa while testing this piece of interface with the parameters whether going to use the test driver here will be one we can called likewise the interfaces between two systems parts can be tested with the help of this mechanism if both system are available here available means available for the tester to test it in depend so that is what is means or it is not available we are going to club both of them and drive it.
So this can have consequence for the testing time because the time is more required for us actually provide this and to start testing a system part as earlier as possible stubs and drivers are these that means still suppose some functionality in the embedded system come to the develop and other functionality is not being not developed how we are going to test that implemented are

ready functionalities with the help of the drivers, driver specifications details for known and with the help of inputs.
And the parameters the test drivers are developed and tested basically it is useful for testing the interfaces so established called the system and that test and provides the information the missing system for should have been given and a driver calls the system back standardization and use of a test stub actuators basically if you have this test bed actuators define in the under stage of the project it will be very good for each piece of the software unit under test.
So it will greatly input the effective use of stubs and drivers the test bed provides a standard interface for both the tester to construct the construct and executive test cases and for this stubs so each separate units must have a stub so we need to have a stub as well as the driver to test both of them inter change away and techniques for test automations such as data driven testing can be applied effectively where data is very important.
And all the type of data that we have sallied, like PK CK CUS and all that can be tested with the help of this there combinations the inputs that is need to be driven can be done with is actually so such a test, the testing of mean the use of such test during integrations testing and that automations of lower testing is easier basically so considering all this aspects to need to have test stub and drivers okay that is about test stubs and drivers and entire box testing with the white box testing mechanism.
(Refer Slide Time: 15:57)



Coverage testing tools

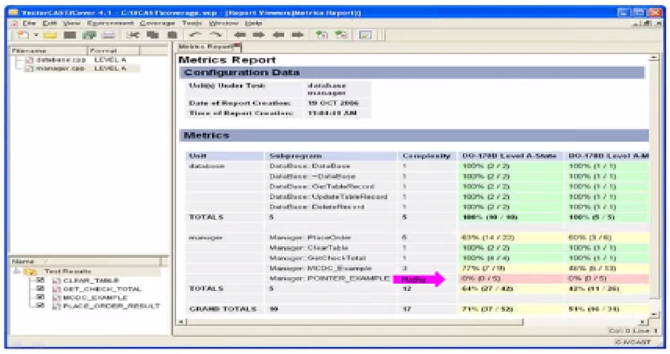
- Logic analyzer
- Software Performance Analyzers
- Vectorcast
- LDRA
- RTRT
- ...

5

Now we will come to the various coverage testing tools that are used in the industry in general so logic analyzer, software performance analyzer, vector cast, LDRA, RTRT there are lot of tools like this so which will help basically used for having the coverage and the instrumentation the unit test tools etc will be done with the help of this, it could be one tool are a multiple tools depending on the complicity of the embedded software that is been used okay let us try to study understand the basic thing about this tools okay.
(Refer Slide Time: 16:54)

Commercial tools contd.

- Vectorcast: (www.vectorcast.com)


VectorCAST/Cover 4.1 - C:\VCAST\coverage.vcp - [Report Viewers[Metrics Report]]
File Edit View Environment Coverage Tools Window Help
Metrics Report

Filename	Format
database.cpp	LEVEL A
manager.cpp	LEVEL A

Test Results
- CLEAR_TABLE
- GET_CHECK_TOTAL
- MCDC_EXAMPLE
- PLACE_ORDER_RESULT

Metrics Report

Configuration Data

Unit(s) Under Test:	database manager
Date of Report Creation:	19 OCT 2006
Time of Report Creation:	11:04:41 AM

Metrics

Unit	Subprogram	Complexity	DO-178B Level A-State	DO-178B Level A-M
database	DataBase::DataBase	1	100% (2 / 2)	100% (1 / 1)
	DataBase::~DataBase	1	100% (2 / 2)	100% (1 / 1)
	DataBase::GetTableRecord	1	100% (2 / 2)	100% (1 / 1)
	DataBase::UpdateTableRecord	1	100% (2 / 2)	100% (1 / 1)
	DataBase::DeleteRecord	1	100% (2 / 2)	100% (1 / 1)
TOTALS	5	5	100% (10 / 10)	100% (5 / 5)
manager	Manager::PlaceOrder	6	63% (14 / 22)	60% (3 / 6)
	Manager::ClearTable	1	100% (2 / 2)	100% (1 / 1)
	Manager::GetCheckTotal	1	100% (4 / 4)	100% (1 / 1)
	Manager::MCDC_Example	3	77% (7 / 9)	46% (6 / 13)
	Manager::POINTER_EXAMPLE	Radha	0% (0 / 5)	0% (0 / 5)
TOTALS	5	12	64% (27 / 42)	42% (11 / 26)
GRAND TOTALS	10	17	71% (37 / 52)	51% (16 / 31)

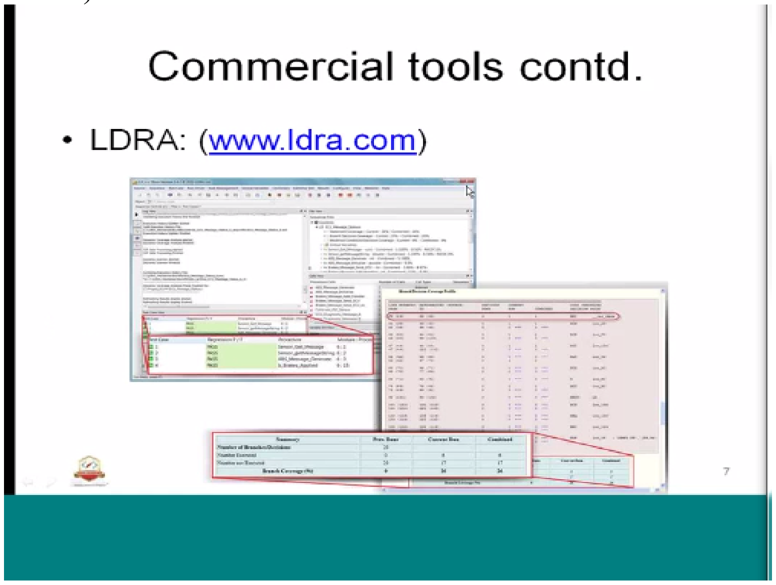
Vector cast this is the tool from vector caste corporations the aerospace is being used more where the define different levels and they do the instrumentation of the source code and they run the tool you will get the report such as this what is been showed below so for a data base sort of applications aerospace is being used we can the matrix what are the matrix it can generate so with the help of this matrix the conclusion will be done so as to see that whether the vector cast output is hundred percent coverage is done are not done you can see the data base total time number of data base have been interested here.

And complicity is five straight forwarding testing that means all the statement or the version for the conditions that are aware for each of this data ten, ten out of ten have been executed saying that the coverage is hundred percentage similarly you see another piece of software that is been tested with the help of the vector cast some functionality package called manager and desires 1, 2, 3, 4, 5 types of funct6ionallity like place order clear table get check total etc each of them have been tested with the help of a vector cast with the instrumentations mechanism.

And you can see the complicity it has the complicity of five we will try to understand what is complicity in the next unit three and complicity of other piece of 1, 1 with representing last one is two so there are total 12 complicity images that have been tested here with the help of vector cast and the coverage is some like 63 percent in cast in the first where the place order is been executed where the coverage is 14 out of 22 that means to say their the coverage is to be done for 22 executable statements or decision or whatever.

It is, that out of which only 14 of it have been covered saying that 63 percentage have been covered similarly for that next you can see 100 percent coverage and this one is 77 percent 7 out of 9 and the last one is zero percent saying that none of the statement have been executed or have been in worked with the help of this tools whatever instrument testing we have done so all together they generate the matrix of course we studied about the test matrix and all that in separate session in detail but to have a glance of tools of what tools commercial they are using I am just trying to present it the various tools the overall coverage is 71 percent and they cannot say the inner column introduced for a different piece of functionality or thing that will be showcased.

(Refer Slide Time: 20:33)

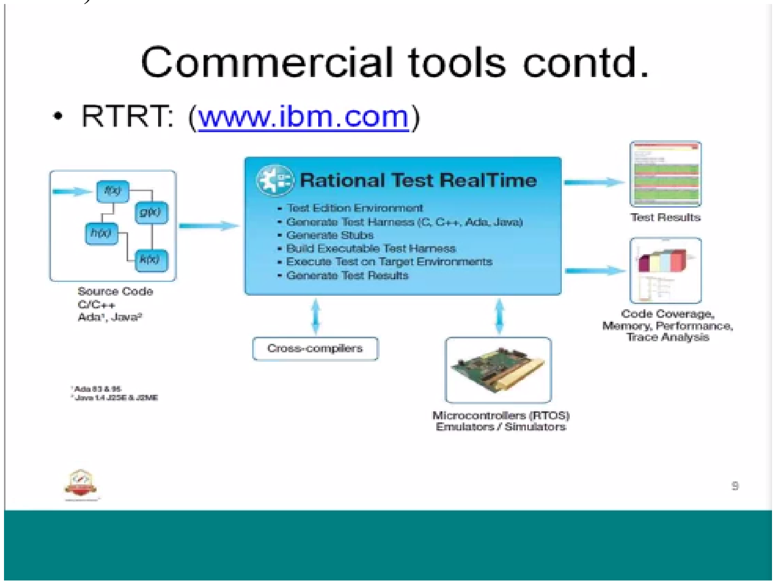


Commercial tools contd.

- LDRA: (www.ldra.com)

The next one is LDRA it is one LDRA and the one of the popular tool that is been used in the aerospace industry here also similar to what instrumentations we have seen so there input can be generated which will help in terms of doing the in testing you can see how many in test cases are used what are them are passed so what is the report likewise we have complete coverage of the unit under test okay.

(Refer Slide Time: 21:07)



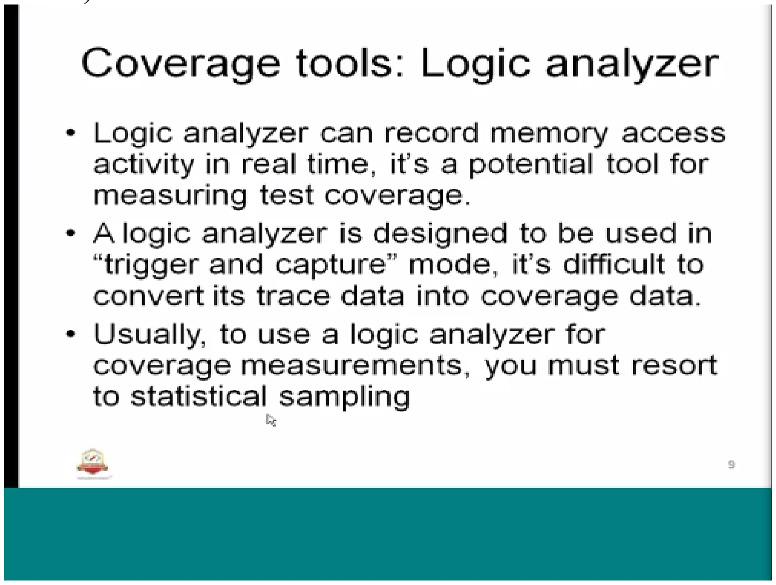
Commercial tools contd.

- RTRT: (www.ibm.com)

f(x)
g(x)
h(x)
k(x)
Source Code
C/C++
Ada¹, Java²
Rational Test RealTime
• Test Edition Environment
• Generate Test Harness (C, C++, Ada, Java)
• Generate Stubs
• Build Executable Test Harness
• Execute Test on Target Environments
• Generate Test Results
Test Results
Code Coverage, Memory, Performance, Trace Analysis
Cross-compilers
Microcontrollers (RTOS) Emulators / Simulators
¹Ada 83 & 95
²Java 1.4 J2SE & J2ME
9

So the next one is being RTRT this is also one of the good competitor and popular tool that is been used in the different industries including agriculture, finances, embracive automotive telecoms, aerospace etc widely nit have a different variance of this tool such as RTRT bed RTRT embed likewise so it is from IBM you can see that details in the backward side there is data sheet and all that which talks about this so the rational test really time is called as RTRT so what it

does means the source code such as C, C++ adra whatever it is head into that RTRT 2 so what it does is the steps are part of the RTRT.
In terms of configuring and using it so the environment will be define first the test harness will be created and with the help of test harness stub will be generated then we will have executables for the corresponding test harness then actual be test after the executions of the test on the target environment we will have the results and then result will be used to the report what is that coverage and all you can see on the right hand side is it is a code coverage or from lance memory or trace analysis all the aspects of the testing white box testing will be done.
And similarly test result will be reported to support this coverage or justify the coverage or justify the coverage to analysis it and it is a cross complier that has generate etc to complies the development harness and the bills and it uses the target such as micro controllers it will be altos or it could be with machines simulators or emulators on the target machines so that is how the RTRT will be structured to use on the white box testing methods okay that is what is about commercial tools that is used for white box.
(Refer Slide Time: 23:46)



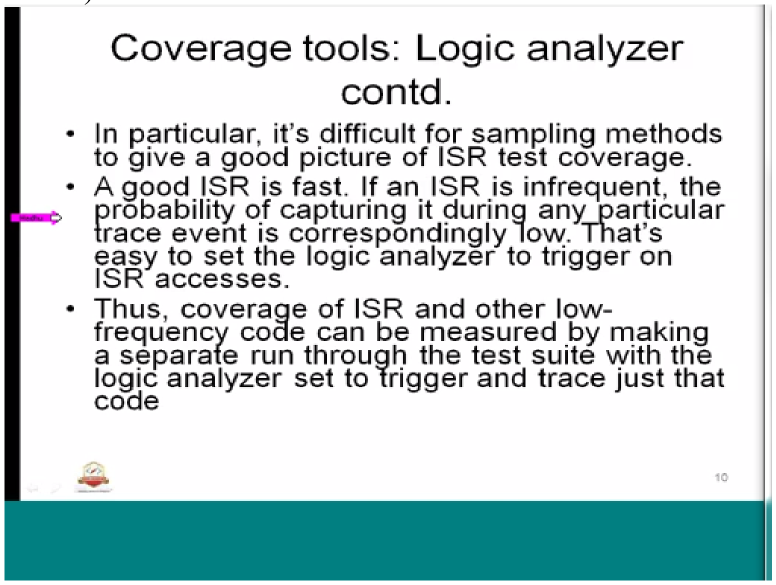
Coverage tools: Logic analyzer

- Logic analyzer can record memory access activity in real time, it's a potential tool for measuring test coverage.
- A logic analyzer is designed to be used in "trigger and capture" mode, it's difficult to convert its trace data into coverage data.
- Usually, to use a logic analyzer for coverage measurements, you must resort to statistical sampling

9

So let us study what is logic analyzer a basically logic analyzer can record memory access activity in real time, it is a potential tools for measuring coverage so what is does is we various test hooks using embedded system and the test hooks will record different data a dictated piece of memory and that can memory can be recorded with the help of analyzer that analyzer is called logic analyzer and there are logic analyzer probes.
That will be hook into the memory and with the help of probes it will accrue the data in real time and we will have the coverage and will provide results and coverage. A logic analyzer is designed and used in trigger and capture mode that means the logic analyzer can be used as trigger and capture mode for the memory or the interfaces that will be hooked for that piece of it is basically it is hardware and software both that logic analyzer has may not have a diagram to show how the logic analyzer
The logic analyzer is design in trigger and capture mode it is difficult to come out it trace data into coverage data so what is means the trace data whatever it got it we may not able to cover it

to in order to come convert that into coverage with the help of this but what it does is the overall measurement it does we do a sampling method is called statistical sampling with the help of this logic analyzer can be used to have a coverage on the trace data on the trigger mechanism in continuations of the logic analyzer.
(Refer Slide Time: 25:48)

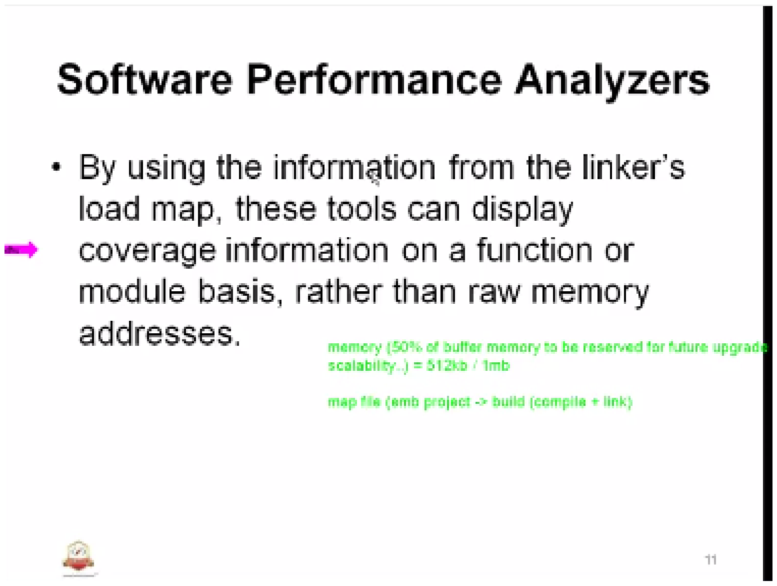


Coverage tools: Logic analyzer contd.

- In particular, it's difficult for sampling methods to give a good picture of ISR test coverage.
- A good ISR is fast. If an ISR is infrequent, the probability of capturing it during any particular trace event is correspondingly low. That's easy to set the logic analyzer to trigger on ISR accesses.
- Thus, coverage of ISR and other low-frequency code can be measured by making a separate run through the test suite with the logic analyzer set to trigger and trace just that code

10

In particular it is difficult for sampling methods to give a good picture of ISR test especially the piece of software having a ISR inter service pertain you must be aware of this so basically this the paten parts of the embedded software where an interrupt occurs for the normal show that interrupt has to be handled with the piece of control flow or functional flow whatever the action that needed that action and all will be part of the paten part interrupt services a good ISR is passed means say that ISR has to short and sweet that need to get passed and come out of that protein that means it is two flags.
So that it can come out very fast ISR cannot be bigger, it cannot be complex and it cannot take, refer to take more time. If an ISR is infrequent that means the frequency of the ISR happening in the embedded life cycles is as the probability of capturing it during any particular trace event is correspondingly low. That means capturing the trace data is lower because ISR is happening very less frequency. That is easy to set the logic analyzer to trigger on ISR accesses.
So, but easy to set the logic analyzer because the capturing the mechanism will be easier, this coverage for ISR and other local pencil code can be measured by making this supply control in to test suit with the value can basically set to trigger and trace just that code, so what can happen is we are trying to focus using the piece of software which are the part of the ISR, what we can do since ISR software is very fast.
And very frequently happening in the system and the system handling the complexity in terms of more number of ISR what best can be done with the logic and un logical which is there and we can commenter rest of the code and focus only on the ISR, what is in trigger that capture the data and what ISR can suppose to do and analyze the capture data with the help of logic analyzer and do the coverage there is ISR which is caravel of doing covering all the in time result,
(Refer Slide Time: 28:40)



Software Performance Analyzers

- By using the information from the linker's load map, these tools can display coverage information on a function or module basis, rather than raw memory addresses.

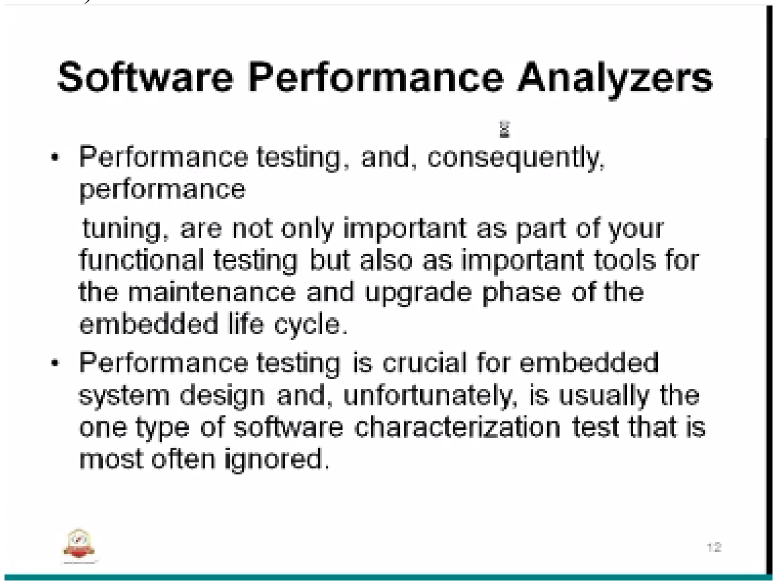
memory (50% of buffer memory to be reserved for future upgrade scalability..) = 512kb / 1mb

map file (emb project -> build (compile + link)

11

So there is another set of tool it is called software performance analyzer, so performance could be in the terms of,

(Refer Slide Time: 28:49)



Software Performance Analyzers

- Performance testing, and, consequently, performance tuning, are not only important as part of your functional testing but also as important tools for the maintenance and upgrade phase of the embedded life cycle.
- Performance testing is crucial for embedded system design and, unfortunately, is usually the one type of software characterization test that is most often ignored.

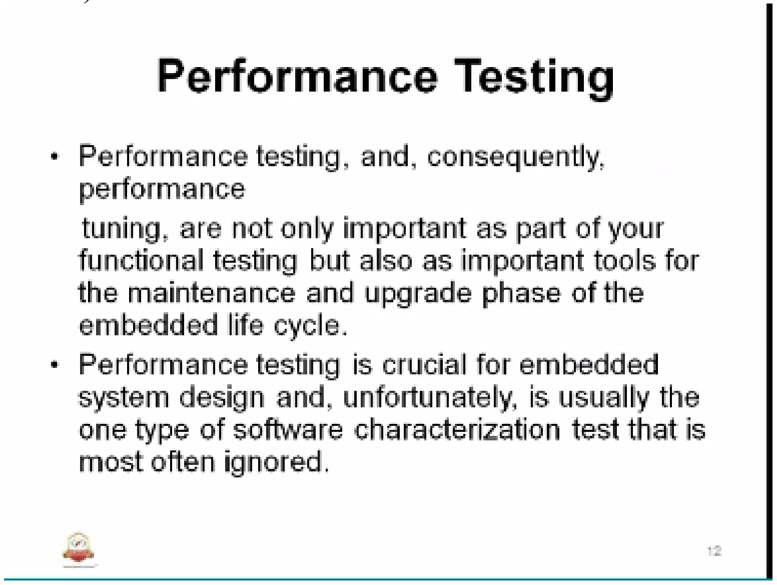
12

Memory, that means memory as to be accurate or intended for sudden portion of, suppose say the requirement can say 50 percent of buffer, memory to be reserved for future update or scalable etc, so what will happen is we need to say that, for example we have a one MB of memory we should have 512 kb of memory to be used one MB, so that is the requirement and how do we test? How much is the software is taking?

So there are lot of meditates and there are lot of tools also, so those all are coming under performance analyzer. We using the information from the linker and these tools can be displayed for a information on a function or modular bases and they are the raw memory address, so with the map file for each of the embedded project, this will be done with the help of builder, that means compile slaved with the help of this map filed will be generated, this map file will have of all the incoming tool such as the addresser.

The hop codes and BSS, stack and all those information with the help of that memory map and you know that how much it is going to take this build for that particular embedded project and with the help of that we should be able to arrive it with the performance of that particular tool, so whether there are tools, analyzers from the different vendors mostly it will be done manually or statistical. So performance testing,

(Refer Slide Time: 31:00)



Performance Testing

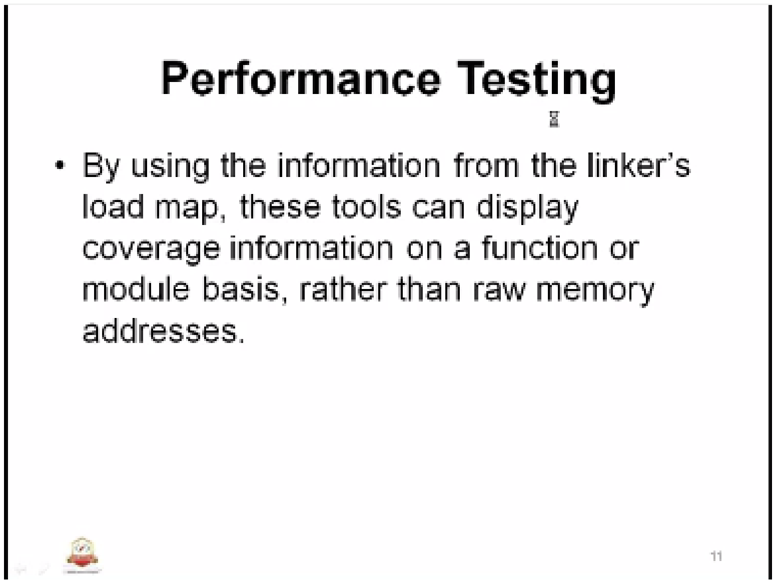
- Performance testing, and, consequently, performance
  tuning, are not only important as part of your functional testing but also as important tools for the maintenance and upgrade phase of the embedded life cycle.
- Performance testing is crucial for embedded system design and, unfortunately, is usually the one type of software characterization test that is most often ignored.

12

Performance testing and consequently performance tuning are not only important as part of your functional testing but also part of important tools for the maintenance and upgrade tools for the embedded life cycle, so I have in important aspect of connections memory you know, and speed, speed with load, so this is very important to have the performance of the embedded system, stable.

And continuous without the change and with the efficient, with efficiency having this scalability modularity, this sees one of the performance requirement they use it, not only enough for have a memory satisfied speed, speed means not fast basically with various conditions in the field should be performed consistently and scalability etc. the other thing is timing, it should be accurate and stable, and these also one of the performance measure they have it.

(Refer Slide Time: 32:24)



Performance Testing

- By using the information from the linker's load map, these tools can display coverage information on a function or module basis, rather than raw memory addresses.

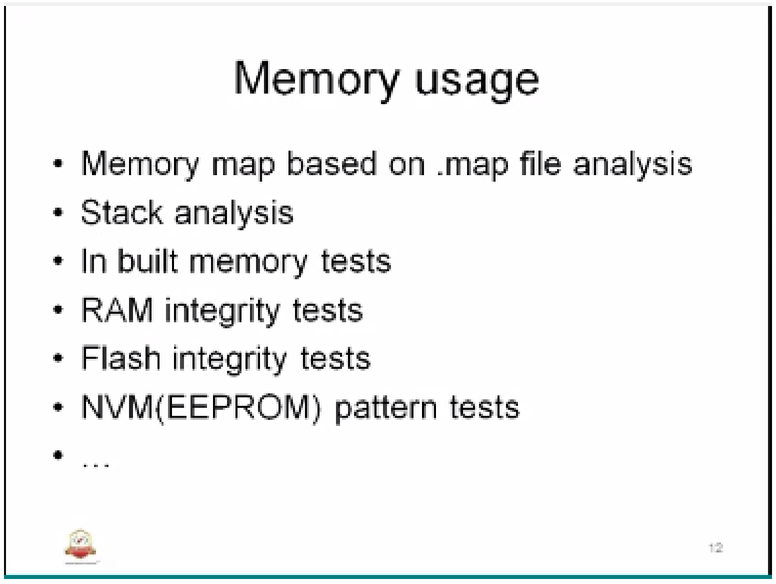
11

So, performance testing and consequently performance tuning, not only important tools for the maintenance and upgrade, upgradability is more and embedded system is ever living and growing, because of the fixes requirement, more requirement in another will be embedded lecture. Performance testing is crucial for embedded design and unfortunately it measures in the one type of software characterization that is most often ignored.

That means, assume and many of the embedded system involved in the industries they do less priority to performance testing in tuning especially in a begging or middle of the project, that they struggle in the end because they never been meet the criteria of the performance and lot of works and errors will occur due to performance issues, so that is why it is important to have in understanding of, what is the performance of the embedded system accordingly we need to have testing mechanism.

Specially on the memory, speed, timing and the load or the minute and we should use a analyzers, which is performance analyzer, map files and all, probably we try touch base simple map file and have a understanding of part of it does, basically it is part of the embedded system course, I will try to address it in the future.

(Refer Slide Time: 34:03)



Memory usage

- Memory map based on .map file analysis
- Stack analysis
- In built memory tests
- RAM integrity tests
- Flash integrity tests
- NVM(EEPROM) pattern tests
- ...

12

The next one is memory usage, what sort of memory usage is there in the embedded system testing, so basically we use a memory more with the help of memory map, and it will be able to analyze the stack the in build memory it could be RAM, ROM, the flash or we can have as small foot print in terms of false and all that, we will be restoring NVM, so that also called as electorally erasable programmable read on the memory.

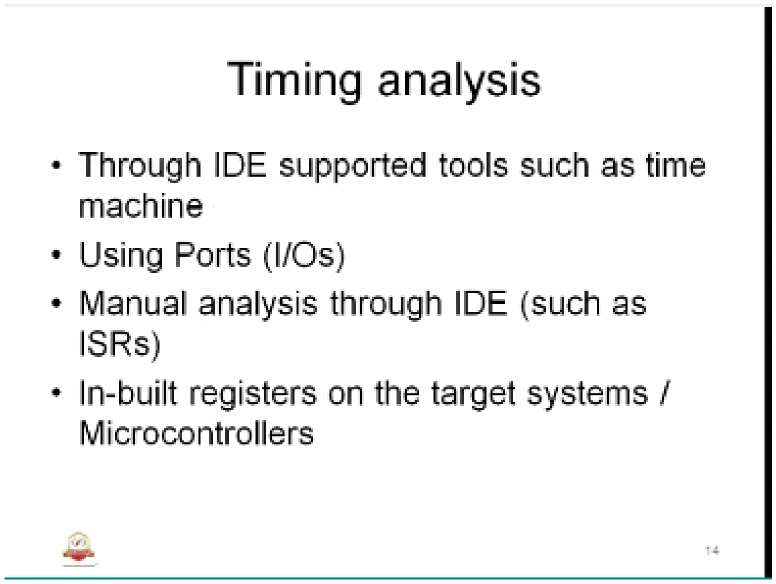
So there are various types of test for example for flash they use integrity test and NV test and can be done with the help of walking ones where each memory cell is tested with zeros and ones, that means whether each cell is capable of flashing or programming a zero a programming one. So that is also called as pattern test, usually they use it in the system with 5a, 5a, 5a; you can see 5,5a last 5a is nothing.

But I will write it on in binary minus 5 zero one, zero one. Whereas A is 1010 and it is 10, so we know that the cells, the first cell is addressed here mixed with zero and the same is each cell is addressed with one. Similarly the next cell is addressed with one and next same cell is filled with zero, so likewise we are going to have a pattern test or walking in test, that each cell in the memory will be tested.

So, on the memory like flash or RAM is sent testing one so mostly these test are basically build along with the code, because there are requirements which talks about this test how to be there frequently should be done in the embedded systems, so they will have a implementation in the embedded system itself but the certain frequency in the system is running but we need to log whether that we have any failures for such steps.

These tests are called as division test. There are different types of division test that is not something depending on the embedded systems of the memory and all these will be logged with the help of those case, so that is the part of the memory user and memory testing that in the embedded software.

(Refer Slide Time: 37:20)



Timing analysis

- Through IDE supported tools such as time machine
- Using Ports (I/Os)
- Manual analysis through IDE (such as ISRs)
- In-built registers on the target systems / Microcontrollers

14

So the next one is timing analysis, so there are stimulant timing requirements so test analysis timing how much in the embedded is taking, timing analysis have to be come back. And mostly the timing analysis has done with the help of time machines and trace machines which is available in the debugger itself. The IDE is nothing but integrated development environment, such as multi looter bag, code warrior etc.

All these debuggers have an a inbuilt time machines or trace machines that will be used that is help full in finding the timing requirements in the term of analysis code and doing with the rate points and measure the time, how much it is taking all these, so you can validate with the help of that, some of the embedded system will provide codes and that codes the validates I can say and they can instructed or IO ports.

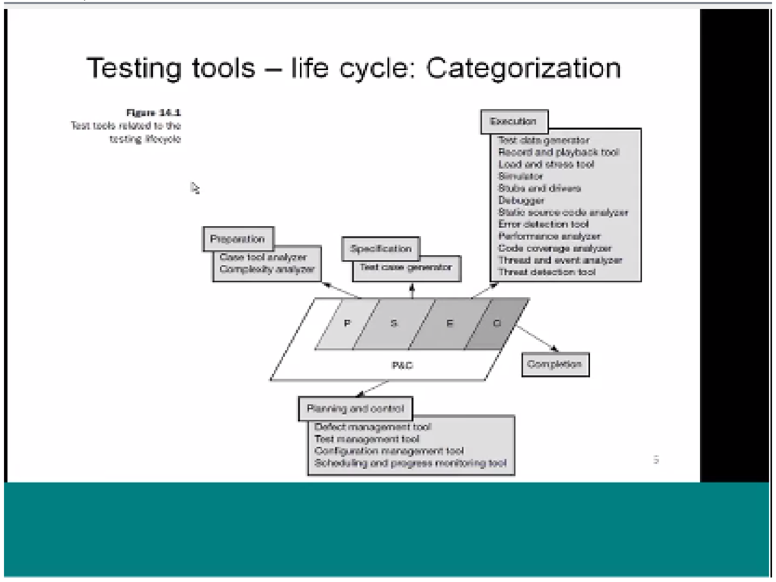
So which will toggle for sudden range or sudden frequency on that can be put in to the scope, scope means osscilar scope we have a multi channel osscillas scope on the Agilent, we can use in the any of that osscillas scope and measure it. So such embedded systems we will have to all ports available, but it may so happen that, that ports may not be there because it is a additional of hardware and they may not be able to.

They cannot or they may not be able to afford to have that because it is going to occupy a space and more current etc. for intermediate testing before the last build they may be having it within the part of the board or the target board of the SPGA whatever it is. With the help of that timing can be tested where the embedded software will be triggering upon certain events and that events can be captured in the oscilloscope.

And for ISR's we can analysis through IDE where ISR so we can have a counter and how much the counter is to be located. The counter also can have registers, the timing registers can be used along with the counters to arrive at on the time it has to come. So, manually it should be done to analyze the ISR timings etc. then we have In-built registers on the target systems and microcontrollers such as watchdog.

And also with the help of those we can state. Watchdog registers, timing registers, RTC that is written in a real time clock. All this can be used for doing the timing analysis. So, with the help of this we can do timing analysis. Okay,

(Refer Slide Time: 40:49)



Testing tools – life cycle: Categorization

Figure 14.1
Test tools related to the testing lifecycle

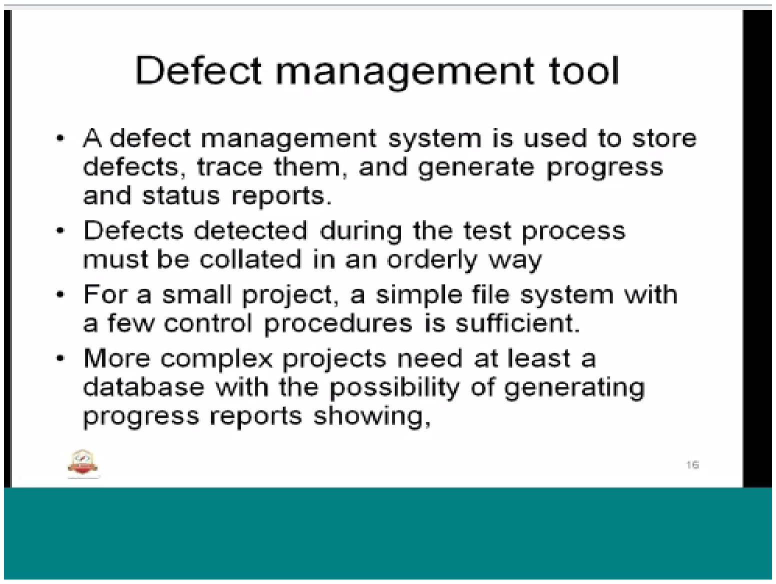
So, have you understood the various tools control that? The applicability you need to understand for the life cycle. Coming to the testing tools so there is a life cycle that is been categorized so we will try to understand in brief what this tools lifecycle is. Basically it is called as the various tools how they are getting used? How they are getting categorized? So what are the use and all that?

Typically we try to understand based on what is the type of tools that we need to have in the embedded software testing. Okay, so you can see a diagram here that depicts about the various tools related to the testing life cycle. So, we know that there are as per the TM method or the LITO physicals of the following behaviors of life cycle PSEC preparation execution and all those stuffs, completion hours.

So, we have various tools for the p preparation we have a case tools analyzer complexity analyzer for specifications we have test case generation or test case generator that will help in developing the test cases and for execution we have test data generator record and playback tool, load and stress tool, simulator, debugger etc. the number of tools given in that all need to be used but it is a categorization basically that need to be applied and planned. This all will be part of the planning.

Software verification planning is which we have studied in our earliest sessions. Similarly for completion we have a floating information number and over all for P&C planning and control we have different rules in terms of defect management, test management, configuration management, scheduling and progress monitoring tool. Okay, so this is how the lifecycle data or the life cycle aspects of the testing will be considered in terms of categorizing the testing tools. And we will try to study some of them in detail as we go through some of the sessions like defect management or test management, configuration management in the future process. Okay, that is about the lifecycle we will try to quickly understand each tool such as planning and control

(Refer Slide Time: 43:41)



Defect management tool

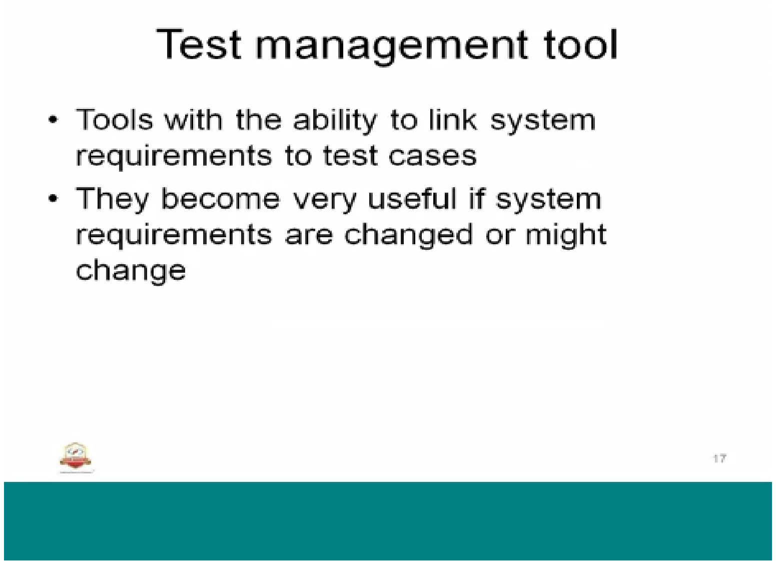
- A defect management system is used to store defects, trace them, and generate progress and status reports.
- Defects detected during the test process must be collated in an orderly way
- For a small project, a simple file system with a few control procedures is sufficient.
- More complex projects need at least a database with the possibility of generating progress reports showing,

16

Either defect management tool. A defect management system is used to store defects, trace them and generate progress and status reports. Defects detected during the test process must be collated in an orderly way. It should be organized properly. For a small project a simple file system with a few control procedures is sufficient. Where the more complex projects are need to have at least a database.

With the possibility of generating progress reports that tools should help basically which will help testing team to analyze where they are. So, defect management tools such as drugzilla can be used for management of the defects. So, that is part of the planning in control. The next one is the test management tool.

(Refer Slide Time: 44:40)



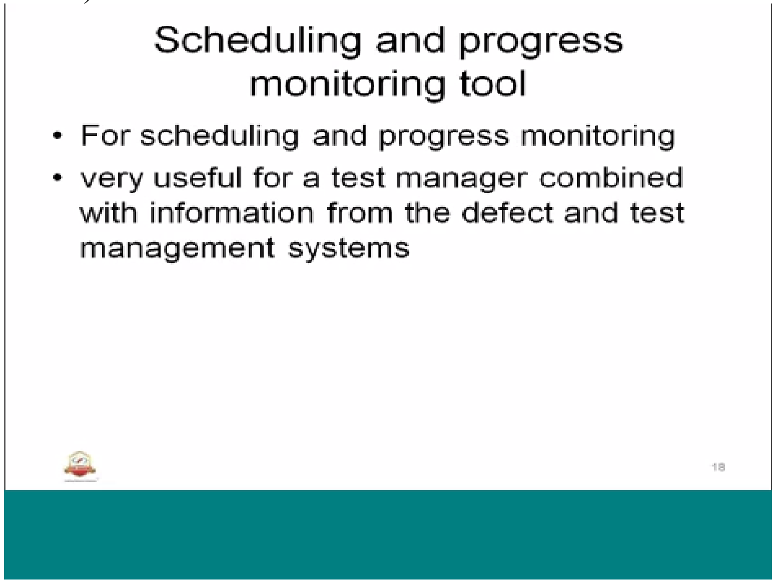
Test management tool

- Tools with the ability to link system requirements to test cases
- They become very useful if system requirements are changed or might change

17

So there is a tool with the ability to link system requirements to test cases basically the test are managed how they can be retraced all this will be part of this. So, the tools should have the ability to link the requirements into the test cases. They become very useful if system

requirements are changed or might change. Why this we need all these tolls is very important to have control of the changes.
So, how do we do? So with the help of these tools it is very easy to change it. To plug with the new requirements or function of the requirements etc. so, all this can be done with the help of test management tool.
(Refer Slide Time: 45:20)

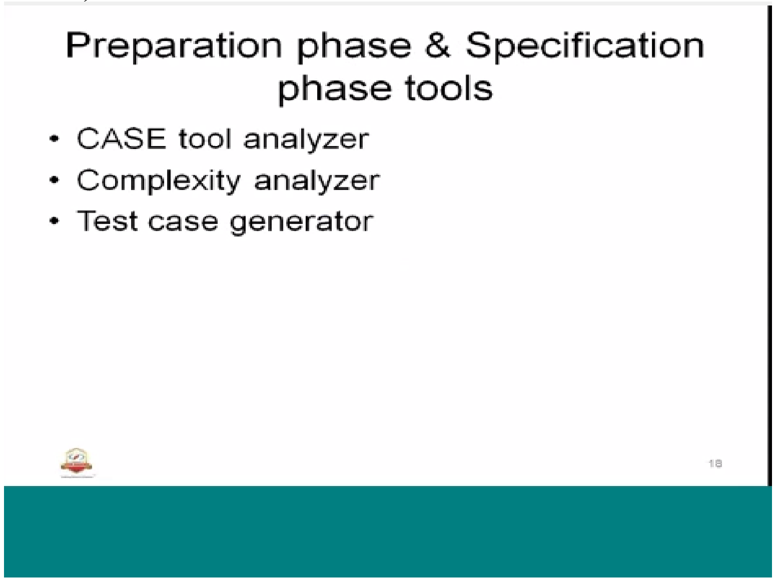


Scheduling and progress monitoring tool

- For scheduling and progress monitoring
- very useful for a test manager combined with information from the defect and test management systems

18

Then we have a scheduling and progress monitoring tool. There are number of tools available for scheduling and progress monitoring is used. Such as mp3 and all there are different tools also which can be integrated along with the test management tools such as test link or bugzilla for defect management etc. So, it is very easy with the help of these tools for scheduling and progress management is very useful for a test manager combined with the information from the defect and test management systems.
(Refer Slide Time: 45:55)

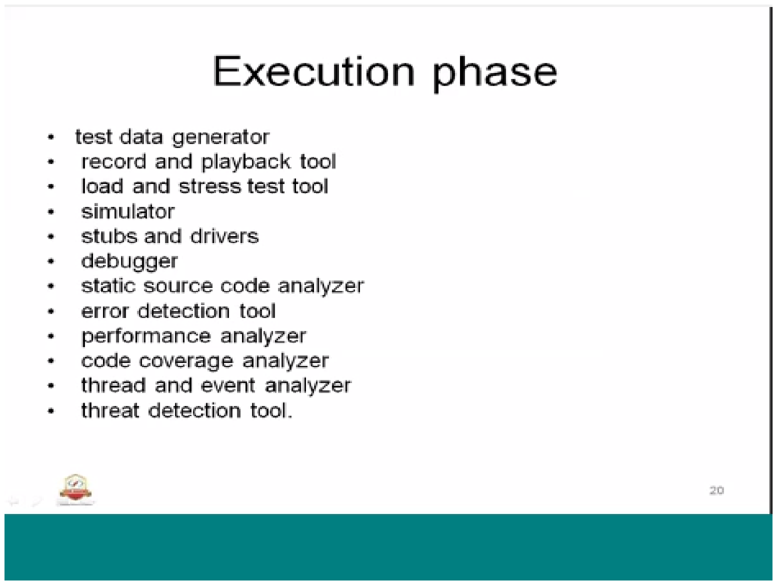


Preparation phase & Specification phase tools

- CASE tool analyzer
- Complexity analyzer
- Test case generator

18

And preparation phase and specification tools on the left hand side you can see it is a preparation phrase and specification phase. There are case tools analyzers, complexity analyzer, and test case generator. We will try to understand what those are. Probably we will try to detail it out the future sessions. Okay, case tool generator preparation phase they use it specially for those where we use object oriented model based systems, the tools such as union based tools are used or consistency checks and all that.
So, they will be able to help in terms of the preparation phase. So, they can be used to check whether are not be designed as omitted any things of embedded system management etc. so, this is basically test ability review of the test basis what is getting planned in terms of preparation. So, the next one is complexity analyzer which we try to study complexity, software complexity in future class.
Basically this complexity analyzer which is understood for physical purpose from sky tools if capable of giving indication about complexity of the software. The degree of the complexity is indicated of the chance of errors occurring and also the number of test cases how much we need and all. It has been in system thoroughly. So, for this the US standard called McCabe complexity. One of the important good complexity measure, generally they follow in the industry which identifies the complexity. This is basically getting identified within that in one of the next class with the formula and all that. We need to know about edge and nodes basically you know that software can have multiple edges and nodes by the fiction and the flow of the software with the help of that complexity is arrived.
Okay, next one is the test case generator which is for the specification, test specification or test cases. So, there are test case generators which are Mat lab, lab use and all that but sorry lab use and all used as the spit in all the stuff so using something like Excel sheet based tools or VC or python or Perl. These can be used from inputs such as requirements in test cases. So, with the help of this two test cases can be generated automatically and assistant it will be used for different requirement, filling requirements and all that.
So, that is how test case generators are used in the preparation and specification tools of the embedded system testing.
(Refer Slide Time: 49:52)



Execution phase

- test data generator
- record and playback tool
- load and stress test tool
- simulator
- stubs and drivers
- debugger
- static source code analyzer
- error detection tool
- performance analyzer
- code coverage analyzer
- thread and event analyzer
- threat detection tool.

20

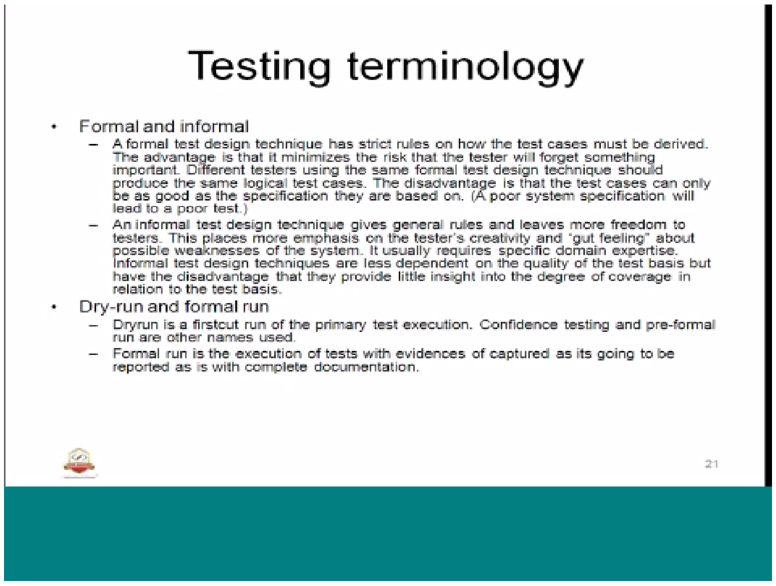
Next one we have is the execution. The execution can done with number of tools and various types of things we have definitely minimum 3 to 4 tools is used for any of the embedded system having a normal complexity and here is the list of type of tools that are used in the execution phase. Test data generators, record and playback tool, load and stress test tool, simulator we know we have tried about some simulators.

Stubs and drivers we studied this below, debuggers we know that IDE is survived as debugger for code analysis and all that. They have a static source code analyzer such as understand for simulators like wise and call the resource on the code analyzer, error detection tool in the code performance analyzers memory analyzers.

And the code coverage analyzers such as instrumentation analyzers –and LDRA so we have a thread unit analyzer were we use in altos, which we are living in multiple threads in all that, then other one is called thread protection tool, this specific tool were if there is a thread to the embedded software system due to enormous code or the inners way of having the crash etc, that can be identified with the help of this mechanism with this tool.

So these are the execution phase the tools are categorized, so, having understood this testing tools and all that.

(Refer Slide Time: 51:37)



Testing terminology

- Formal and informal
  - A formal test design technique has strict rules on how the test cases must be derived. The advantage is that it minimizes the risk that the tester will forget something important. Different testers using the same formal test design technique should produce the same logical test cases. The disadvantage is that the test cases can only be as good as the specification they are based on. (A poor system specification will lead to a poor test.)
  - An informal test design technique gives general rules and leaves more freedom to testers. This places more emphasis on the tester's creativity and 'gut feeling" about possible weaknesses of the system. It usually requires specific domain expertise. Informal test design techniques are less dependent on the quality of the test basis but have the disadvantage that they provide little insight into the degree of coverage in relation to the test basis.
- Dry-run and formal run
  - Dryrun is a firstcut run of the primary test execution. Confidence testing and pre-formal run are other names used.
  - Formal run is the execution of tests with evidences of captured as its going to be reported as is with complete documentation.

21

We will try to understand the testing terminology and all the stuffs in the next class. If the configuration management tool also introduced, we try to go through that in the next class.