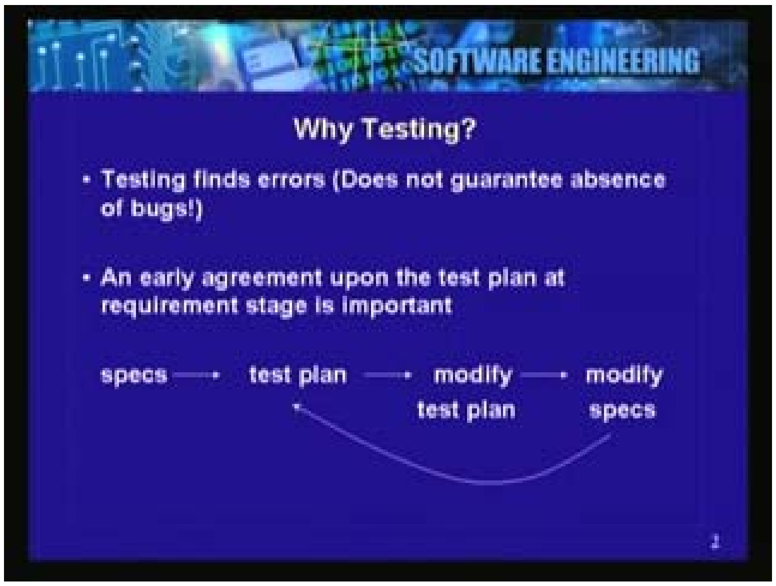
Software Engineering
Prof. Rushikesh K. Joshi
Computer Science & Engineering
Indian Institute of Technology, Bombay
Lecture - 18
Software Testing-I

Today we will talk about testing strategies. That is, for the given software, when to test, what to test, how to test and a general overview of different strategies that we use during testing phase.

(Refer Slide Time: 01:09)


SOFTWARE ENGINEERING

Why Testing?

- Testing finds errors (Does not guarantee absence of bugs!)
- An early agreement upon the test plan at requirement stage is important

$$\begin{tikzcd} \text{specs} \arrow[r] & \text{test plan} \arrow[r] & \text{modify test plan} \arrow[r] & \text{modify specs} \arrow[ll, bend left] \end{tikzcd}$$

Why do we have this phase of testing? As we know, testing finds errors. When we test a piece of software which could be a module, a function or the entire system, we find errors. Errors are reported or that the test goes through with the software, giving us the correct expected behavior. So testing helps in finding errors or different kinds of failures if there are any. But it can be noted that it does not guarantee absence of bugs. So if testing does not report a bug, it does not mean that there is a guarantee that there is no bugs in given software. It is not possible unless and until we prove that the software is correct and it will work as per the requirement specification. Hence testing is done to find errors. Then we debug the software and plug the holes and test the software again.

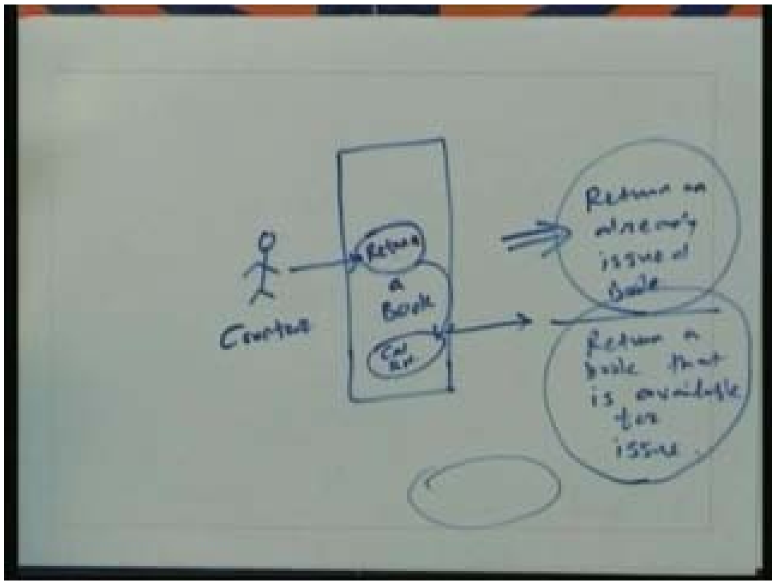
So testing is a very important activity in software lifecycle and it is also a profession. Testers need to be experts in testing and applying different testing strategy at different points of time in software lifecycle. A popular view is that testing starts after implementation. But that is not correct. The activities of testing do not start after the entire implementation is over. In fact it will start or it has to start as early as possible during requirement stage itself.

One can have an early agreement upon the test plan at the requirement stage and it is very important to have answers to the questions such as; what is going to be your test plan? How are going to test your software and finally when it will be built? So specs can drive a test plan. From the specs, you can generate a test plan and then the test plan can be validated or can be debated upon with the customer and with peers. The test plan gets modified, accordingly the specifications get modified. Then you go back and get a new test plan, if you have modified the specifications.

So an early test plan also helps in improving your specifications. The requirement stream is more clear about what the customer really wants and what is the ultimate goal when you point out or when you get a list of your test cases and the test plan. And if you know that ultimately this is what you want to test software for, you are more sure about your specification. So specifications can be used to drive the test plan.

For example, let us consider a specification in terms of a use case diagram. All of us know what a use case diagram is. Let us take a case of library system software, in which a use case such as return a book is identified.

(Refer Slide Time: 08:02)


Counter
Return a Book
Return an already issued book
Return a book that is available for issue

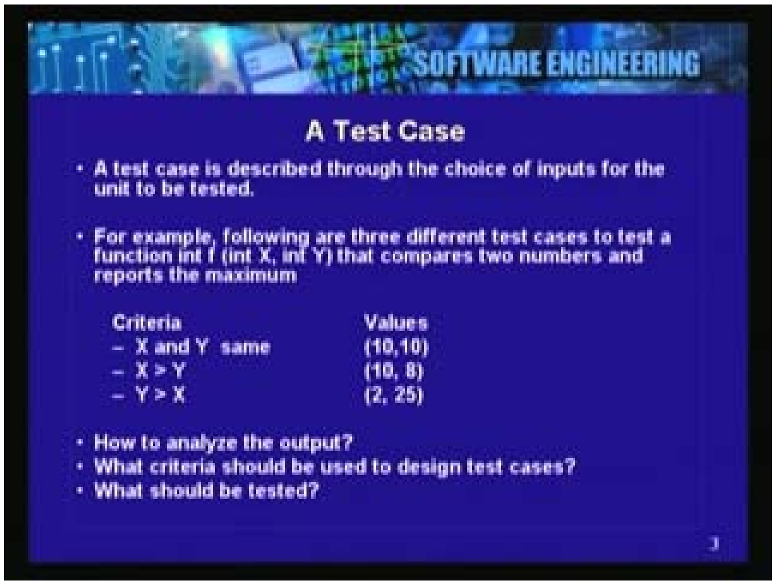
The user could be a person at the counter. Given this use case: “Return a book”, what should be a test plan? One might say that you can generate a test case where you have returned a specific book, that is an already issued book or you could have another test case where you try to return a book that is available for issue. That means in both these cases you are going to expect some behavior. What is the expected behavior? In case 1 when you return the already issued book, the status of the book should change from available to issued. In the second case, return a book from issue to available and if the book was already available then it remains available.

Now we have got a test plan with us and a test case or with a suite of say 2 cases for a given requirement specification captured in terms of use case. That is externally observable behavior of a system. Once this test plan is there, one might find out that when you return a book, you have to also calculate the fine that is to be paid by the user if there is any. That requirement is missing in this specification. So once you identify a test case, you might identify some missing features. This test case or the analysis of your test based on your high level needs, allows you to also think more about the requirements. So you could now add a "calculate fine" use case and say that return of a book uses the "calculate fine" use case. So after adding this, your test plan also changes. Now you want to also check what the fine was or what would be the accumulated fine on the user's account or on the account of the user who has just returned the book.

In this fashion, we can see that the specs drive the test plan and the test plan drives the specs. So modification in the specs causes a change in the test plan and a change in the test plan might also cause modification on the specs, if you are adding new features or if you are identifying something new from the specs point of view or new constraints and new features and so on. So testing is useful to find errors and an early plan of the testing can help refine your specifications. And afterwards, when the software is built, when your modules start appearing, testing has to be carried out. So the plan has to be implemented and carried out.

We will now see different aspects of testing, various high level strategies. This is an example test case. Here we are trying to find out what a test case is. A test case is described through a choice of inputs for the unit to be tested.

(Refer Slide Time: 09:04)


SOFTWARE ENGINEERING

A Test Case

- A test case is described through the choice of inputs for the unit to be tested.
- For example, following are three different test cases to test a function int f (int X, int Y) that compares two numbers and reports the maximum

Criteria	Values
– X and Y same	(10,10)
– X > Y	(10, 8)
– Y > X	(2, 25)

- How to analyze the output?
- What criteria should be used to design test cases?
- What should be tested?

3

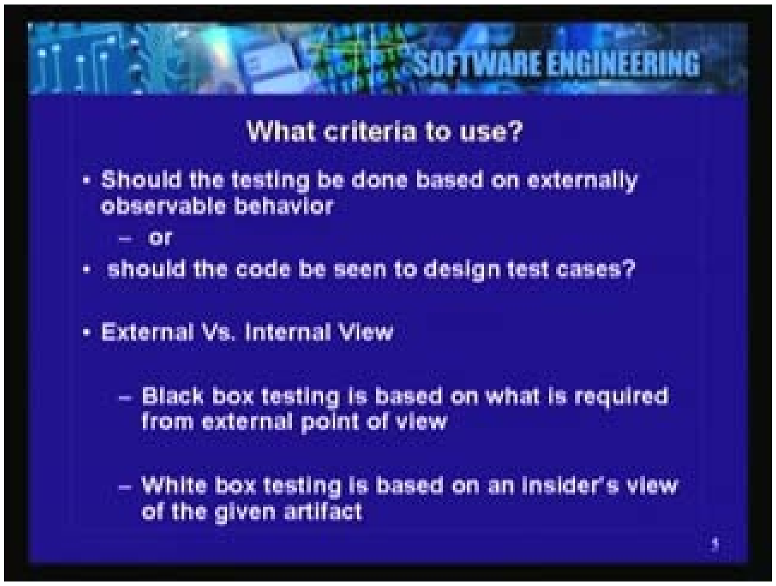
So given a unit, you want to test that unit and you want to identify what are the inputs for the test case. And along with the test case of course you have to also know what are going to be the expected results of the test case.

Now we have given a few examples on this slide about a suite of test cases for a given function. So you have a function "int f ()" that takes 2 input parameters x and y. This is C syntax. The function f takes two parameters x and y and the function compares the two input numbers and it reports the maximum. What should be your test suite? How are you going to test this function? For example you could have these three criteria: When x and y is same, x is greater than y and y is greater than x.

Are you able to get the right value as an output of this function? For these criteria, these are some chosen values. In your test case you must have some concrete values. For example x and y same: There could be many possibilities, but you have chosen one specific set of values. For x and y: (10, 10) For x greater than y, you have chosen x to be 10 and y to be 8. For y greater than x, you have chosen y to be 2 and x to be 25. Now given these three tests, your function works correctly, that does not mean that the function will work correctly for all set of inputs. But you have now validated the functions against these three cases and your confidence in the software or in the function will go up if it works correctly for these three cases.

Now we can ask these two questions: What criteria should be used to design test cases? How are you going to design these test cases? We identify three criteria here, which are with respect to the values, their types and also probably the semantics or the context, what should be the ranges of the values and so on. So what are those different criteria that could be used to design the test cases and what should be tested? In the above case the function has been tested. But what else can we test? Now we are going to look at some answers for these questions in the course of this lecture today. Let us look at the first question. What criteria should we use? Should the testing be done based on external observable behavior of the software or should one see the code to design the test cases?

(Refer Slide Time: 12:20)


SOFTWARE ENGINEERING

What criteria to use?

- Should the testing be done based on externally observable behavior
  - or
- should the code be seen to design test cases?
- External Vs. Internal View
  - Black box testing is based on what is required from external point of view
  - White box testing is based on an insider's view of the given artifact

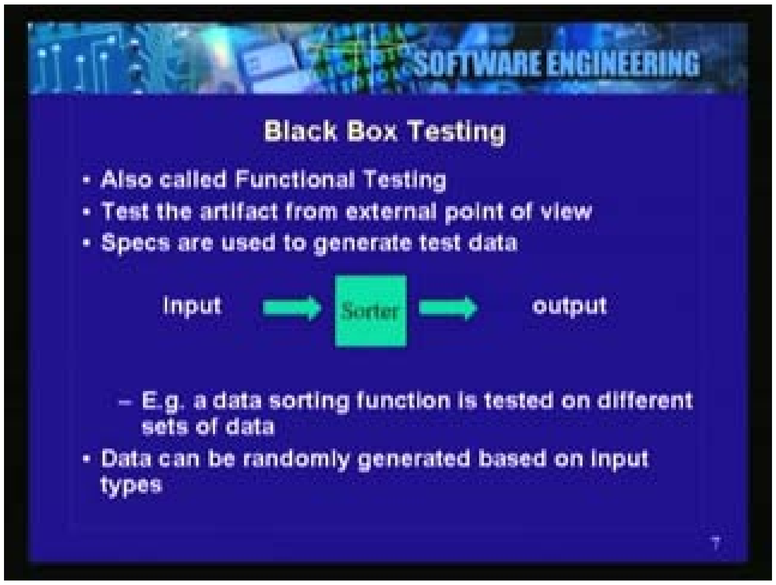
5

For example, we saw the use case based test plan. We had seen only, what is observable from outside and design a test case based on what is observable. But then should we not look into the actual code, the statements, and the structure of the code and design the test cases? This is an important question that can give us two major classes or major approaches to testing. So one looks at the external view of the software or of a given artifact and the other looks at the internal view, which means you are going to test the software based on what is there inside the software, what is the code, the specific statement, and it could be the conditions and so on and so forth.

The two major categories that we derive from this question are black box testing and white box testing. We can see here that black box testing is based on what is required from external point of view. So you are looking at the software units or artifacts as black boxes you do not see them inside. You are going to take an external view or a user's view. If you are looking at the entire software as a black box you are going to use the user interface and test the software. If you are going to look at one module inside the software as a black box, you are not going to see inside that module. But at the same time you have to test the module and you will have to generate some input parameters to the module and generate the context of the module.

So at any level you can apply this strategy of black box testing and this black box testing does not take into account the internals of the artifact that you have chosen at the given level. It could be the entire software or it could be a module or one class or one instance and so on. The second approach that we get based on the internal view is that of the white box testing. So white box testing is based on the insiders' view of the given artifact. Say for example, you have the code with you and you want to check whether all parts in the given code lead to correct answer or whether all branches are covered. You want to generate such test cases based on white box testing or the internals of given module or the implementation. So you get these two different approaches, black box and white box testing. We will look at these two a bit closely.

(Refer Slide Time: 15:21)


SOFTWARE ENGINEERING

Black Box Testing

- Also called Functional Testing
- Test the artifact from external point of view
- Specs are used to generate test data

Input
Sorter
output

  - E.g. a data sorting function is tested on different sets of data
- Data can be randomly generated based on input types

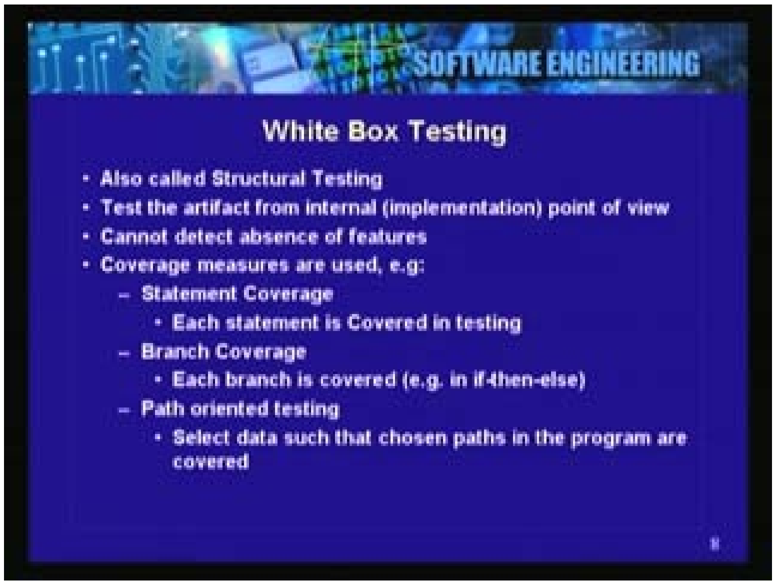
7

So here we see that black box testing is also called functional testing, that is because you are going to mainly look at the functionality of the black box, how it is to be used as a black box. You are not seeing inside or you are not looking at the internal structure. You are looking only the externally observable functionality and test the artifact from external point of view. Specs are used to generate test data. For example, specs of a given module: What is the contract that a given module supports? Let us take this sorter. It sorts data or data items. Sorting function or the sorter will be taking some input and produces sorted output.

How will you test this sorter? You might generate a test case based on this black box policy. That means look at only the input and the output, do not look at the code of the sorter. Do not try to test the integrity of the code itself. Only look at the input and output and generate your test. You are going to only sample the entire space through input and outputs of a given module or a black box. So here a simple test case could be a specific set of data as input and see whether the data item get sorted. Data can also be generated randomly based on input types. Once you know the types of inputs, you can generate random data and these random data can detect some class of bugs.

So black box testing essentially looks at the modules or the given artifacts as a black box. And you generate test cases based on the input to the black box and the output behavior expected from that black box. In contrast to black box testing we have white box testing, which is also called structural testing. Here you test the artifact from internal or implementation point of view.

(Refer Slide Time: 17:46)



SOFTWARE ENGINEERING

White Box Testing

- Also called Structural Testing
- Test the artifact from internal (implementation) point of view
- Cannot detect absence of features
- Coverage measures are used, e.g:
  - Statement Coverage
    - Each statement is Covered in testing
  - Branch Coverage
    - Each branch is covered (e.g. in if-then-else)
  - Path oriented testing
    - Select data such that chosen paths in the program are covered

8

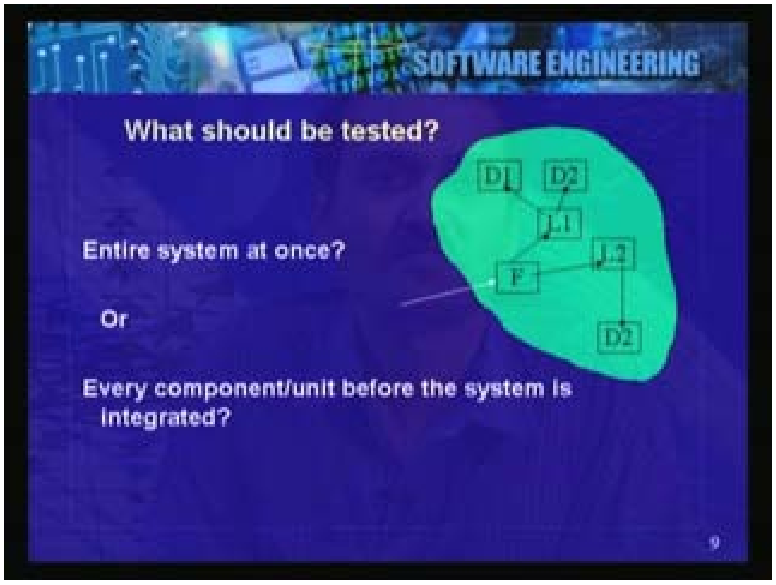
So we can see that though you are going to test the artifact from the internal point of view or the implementation point of view, you are going to look at every statement conditions and so on. You would not be able to detect the absence of features from the software because you are not going to see the externally observable functionality as such. You are going to look at the code and the different properties of the code itself. So white box testing is not advisable or it is not capable of detecting absence of high level features or functional features of software.

So different techniques that are used are as follows: Statement coverage: Each statement is covered in testing. So generate test data such that each statement is covered and then you see whether your code works or not or whether it results in a bug. You are going to check the robustness of your code. Branch coverage: Each branch is covered. For example when you have “if then else”, the “if” branch and the “else” branch both are to be covered and the code gets tested. So you are going to actually generate the input, so that both the branches get covered and then you are saying that both branches of this “if then else” statement have been covered.

Another approach is path oriented testing. You are going to select the data such that different chosen parts in a program are covered. So we look at the code more closely and generate the test cases and see whether your program works correctly. So I am not really basing your test cases on the externally observable behavior. You are not really basing them on different features. If there is something absent, you will not be able to detect. Whatever is present, that will get tested through your different inputs or through your different criteria or coverage measures that we use in white box testing.

Now the next question is that, what should we test? What should be tested as black box or white box? Should the entire software be tested at once after the entire implementation is over or should the software be tested as and when the programmers generate programs, modules, classes, tables, user interfaces, and business logic units?

(Refer Slide Time: 20:08)


SOFTWARE ENGINEERING
D1
D2
J.1
J.2
F
D2
9

What should be tested?

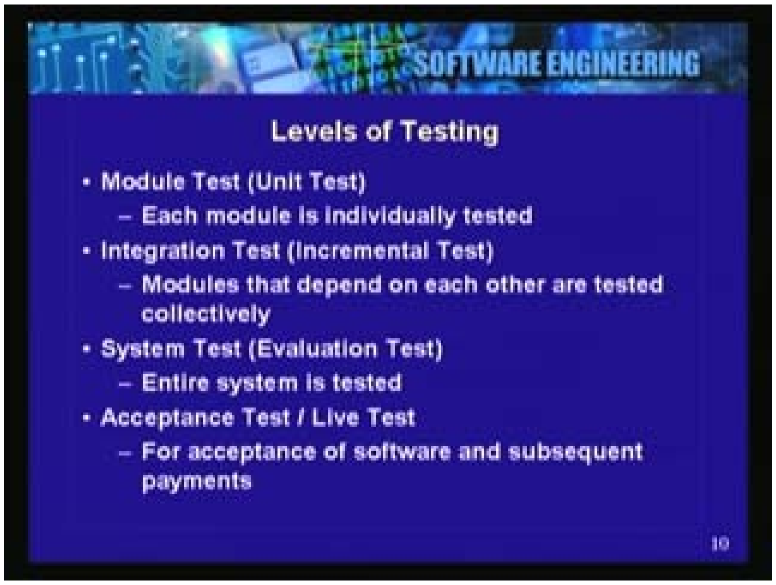
Entire system at once?

Or

Every component/unit before the system is integrated?

Should the software be tested in one piece at the end of entire implementation or should the testing go on during the implementation as and when the code is built? Should every small piece be tested before the whole is tested? That is what is put on this above slide: What should be tested? Should the entire software be tested at once through the user interface, once the whole software is built or should every small piece of artifact be tested before the whole is tested? So this is an important question and that is going to give us different levels of testing.

(Refer Slide Time: 21:23)


SOFTWARE ENGINEERING

Levels of Testing

- Module Test (Unit Test)
  - Each module is individually tested
- Integration Test (Incremental Test)
  - Modules that depend on each other are tested collectively
- System Test (Evaluation Test)
  - Entire system is tested
- Acceptance Test / Live Test
  - For acceptance of software and subsequent payments

10

One can carry out testing at different levels. So we can see the levels of testing here. The module testing or test of a module is at the individual module level. Each module is individually tested and it is also called unit test. As programmers build their modules, each of them are tested individually. A tester could be assigned for every module. So as soon as the module is delivered by the programmer, the tester tests that module. And if the bugs are found, it comes back to the programmer for fixing of bugs. So module test or unit test is carried out for every module, every independent module.

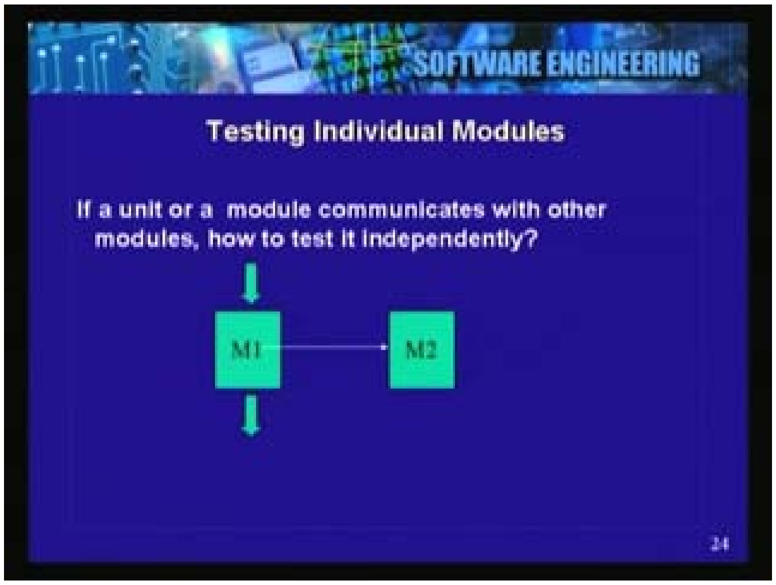
Then the second level of testing that you can get is the integration level or it is also called incremental testing. So modules that depend on each other are tested collectively. Say as modules get added into the system, as they get developed, the dependent modules can be put together and one can carry out an integration test. An integration test also could be carried out partially, for the partial view of the system. For example you may not have the entire software available. But still you can carry out integration testing for the modules which are available and built.

What is to be done for the modules that are not available when you do the integration testing? We will see that soon. Now, let us look at these different levels of testing. We have got module level testing, that is unit testing. We have also got integration testing that is incremental testing. As modules get added into the system you do the testing incrementally you can do the integration testing. Then the third level, you can have system test, that is evaluation test. You can test the entire system. So once the whole system is built, it is integrated and you test the entire system.

And finally you have the acceptance test or it could also be called live test. For acceptance of software, usually the customer does the acceptance test. It can be done at developers' side and also at the customers' side on live data. Based on acceptance test you have the subsequent payments and so on.

So we have these different levels of testing; modules, integration of modules, system and the acceptance test. Within modules you can go finer and you can look at statements, conditions and so on and so forth in white box testing. Let us look at testing of individual modules. So, if a given a unit or the module communicates with other modules, how to test that module independently? Given a module, if it is depends on other modules, say for example in this picture you have M1 and M2. M1 is ready, but M2 is not ready.

(Refer Slide Time: 24:20)



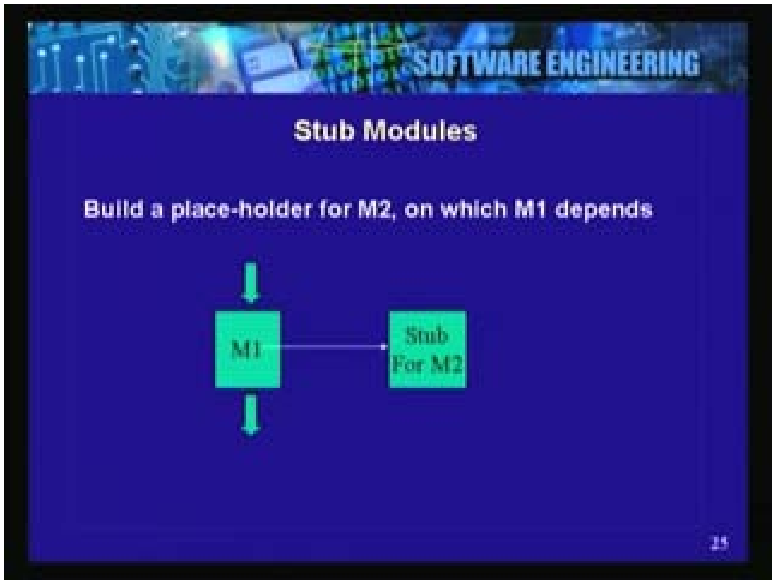
SOFTWARE ENGINEERING

Testing Individual Modules

If a unit or a module communicates with other modules, how to test it independently?

And now you want to test M1, but M1 makes a call on M2, how are you going to test M1? What will you do for M2? If M2 is ready, this becomes an integration test. That means you are going to test M1 by integrating M2 with it. Probably by linking M2 it could be a library module, you can link M2 with M1, the actual implementation of M2 and then M1 can actually call M2 if it is ready. But if M2 is not ready, how are you going to test M1? You still want to continue or complete the module level test for M1 even if M2 is not ready. So you can employ an interesting technique to test M1 in absence of M2, and that is to employ a stub for M2. Stub modules are the replaces for the actual. You have to write a small piece of code for M2, which will directly return the expected value or which will have the filler for the M2, so that M1 can continue.

(Refer Slide Time: 25:41)


SOFTWARE ENGINEERING

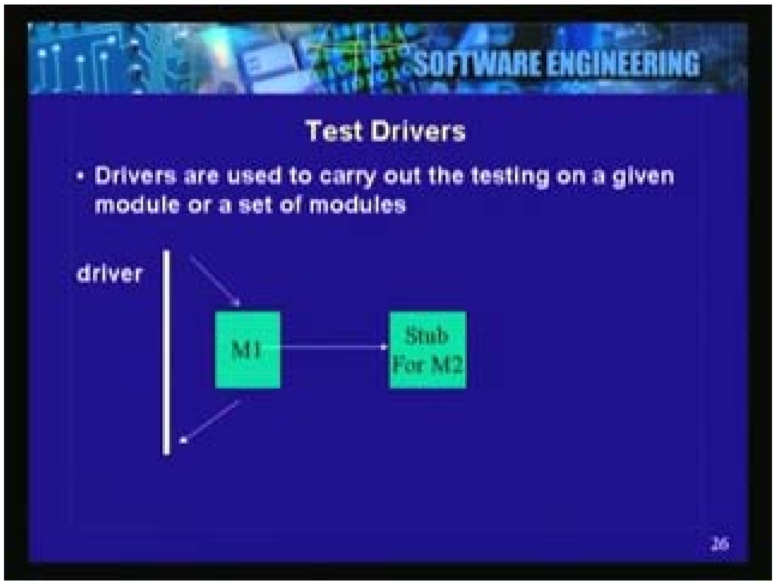
Stub Modules

Build a place-holder for M2, on which M1 depends

The return value should be in the expected range by M1. If a unit or a module communicates with other modules, then how can they be tested independently? You use stubs. In the above picture M2 has been replaced by stub for M2. Here the stub is not the actual implementation of M2 if M2 is not ready. But the tester writes this stub for M2 or the stub is available it can be integrated directly with M1 and M1 is tested. So M1 is independently tested at module level or unit level with the help of a stub for M2.

Then you also require driver for calling M1, for activating M1 itself. At least say, if you have a single module and you want to test that module, you will have to put that module inside a program, you will have to write a main, you will have to generate parameters and call that module.

(Refer Slide Time: 26:39)


SOFTWARE ENGINEERING

Test Drivers

- Drivers are used to carry out the testing on a given module or a set of modules

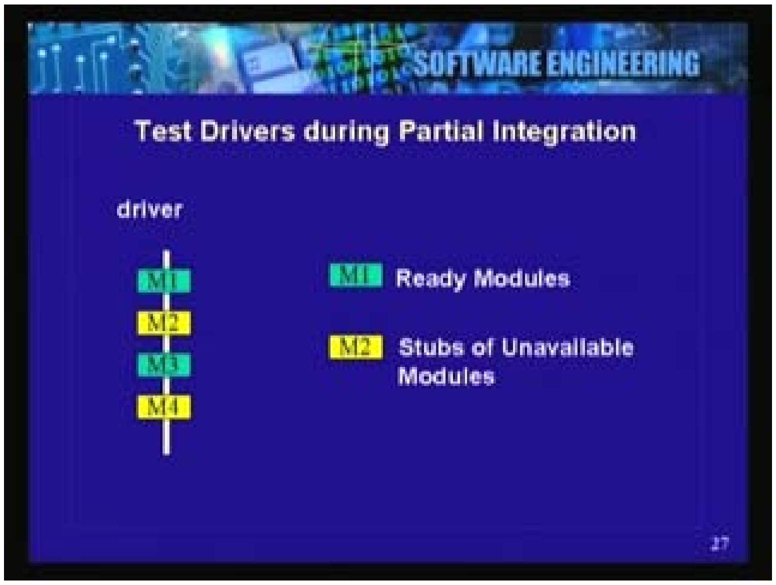
driver
M1
Stub For M2
26

So, that piece of code that actually makes it possible to instantiate this module M1. Say for example it could be a class or if it is just a function, you have to generate some input and call that function. So what is the code that does the actual calling of M1 or instantiation or creation of M1 and then testing of that M1? Now you have a stub for a M2, but for M1 how are you going to issue a call? How are you going to generate a call to M1 when the actual software that you uses M1 may not be available or the another module that uses M1 may not be available? We will have to create a small environment to invoke a call on M1 or you instantiate M1 if it is a class, and then invoke those test sequences on M1 and see its behavior.

So, that external environment or external entity that drives your test suite or the selected test cases on the given module for module testing is called driver. This driver is going to make a call on M1, send the input parameters as pointed out in the test case and then get the output and probably log the output or analyze it. This is what the driver is going to perform. As we have seen, M1 uses the stub for M2 when M2 is not available. So you have a stub and you have a driver. Drivers and stubs are the two very important mechanisms in testing of modules.

We can see on this picture that one can use a driver for partial integration or for partially integrated systems. Look at this driver you have this driver is going to test two modules which are ready.

(Refer Slide Time: 28:33)



Test Drivers during Partial Integration

🟩 M1 Ready Modules

🟨 M2 Stubs of Unavailable Modules

27

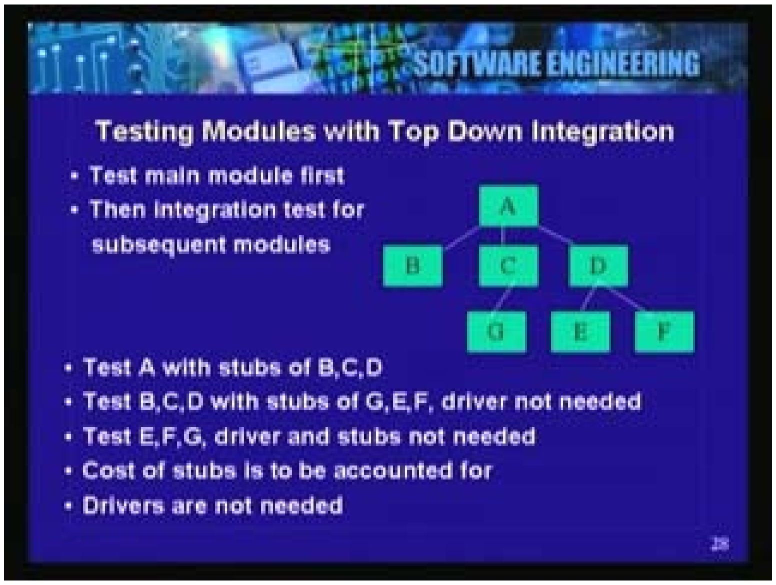
So the two modules which are not ready, we have put stubs in the driver. M1 and M3 are ready where as M2 and M4 are not ready. They have been combined by this driver. So probably you have an assembly line there or a sequence. The data flows from M1 to M2, M2 to M3, and M3 to M4 and comes out of M4. The input goes first to M1. So you have a sequence of instructions or sequence of modules which are executed one after the other, output of one module is coupled to input another module and so on.

In this particular sequence, M1 and M3 are ready but M2 and M4 are not ready. Still you can complete the testing by the stubs for the module which are not available and the driver to combine all the modules, to integrate all the modules, to actually pass on the output of one to the input of another. And the input identified in the entire test case for the first module in this particular assembly line. So this is one sequence or one possibility. There could be other structures in your module graphs and accordingly you have to choose drivers and stubs whenever required.

So the yellow ones are the stub modules and the green ones are the actual ready modules. We are able to do an integration testing with this partially complete system. Two modules are ready and two modules are not ready, but even if they are not ready we are going to develop stubs for them. The stubs are integrated along with the ready modules. The ready modules do not know that M2 and M4 are not complete. Because they still are able to accept the input and give you some output. That output from M2 is sent to M3. One has to write appropriate stubs for M2 and M4, so that you are able to test M1 and M3 as per your test plan.

Now there are some interesting issues regarding these drivers and stubs. Let us consider two different cases of top down and bottom up integration. So let us see how do we test, how do we carry out our testing plan, what should be our testing strategy for top down integration.

(Refer Slide Time: 31:03)


SOFTWARE ENGINEERING
A
B
C
D
G
E
F
28

Testing Modules with Top Down Integration

- Test main module first
- Then integration test for subsequent modules
- Test A with stubs of B,C,D
- Test B,C,D with stubs of G,E,F, driver not needed
- Test E,F,G, driver and stubs not needed
- Cost of stubs is to be accounted for
- Drivers are not needed

We have these seven modules in this particular structure. Module A uses module B, C and D. Module C uses module G. Module D needs modules E and F. Now when we want to do an integration from a top down point of view, we will first test A, then we are going to test B,C and D. Then we are going to test G, E and F after B, C and D gets tested. So in the top down fashion you are going to integrate everything. You test the main module first and then you are going to do integration test for subsequent model.

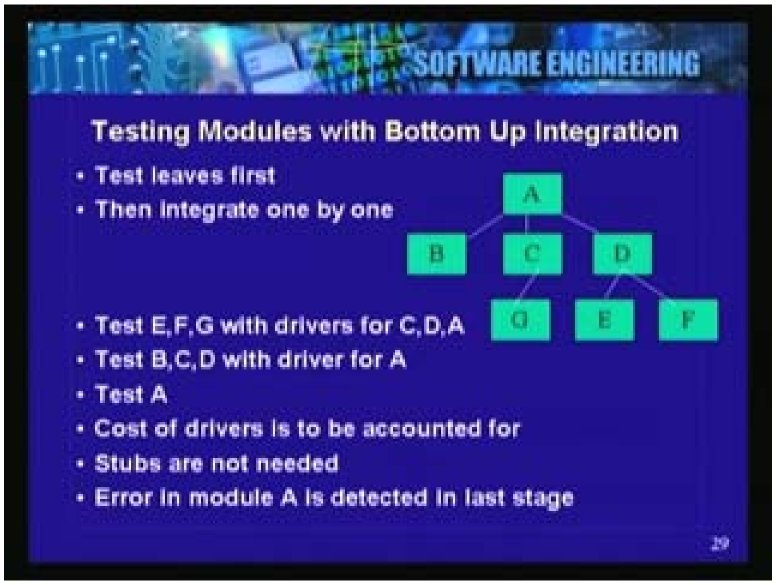
In top down integration first you develop the main or the central theme of the entire software. The main will be ready, but your remaining modules may not be ready. For example, only your main A is ready but B, C, and D are not ready. Meaning, when you are testing A, BCD may not be ready. But when you are testing A, A is completed, your main module is completed. But the function on which the calls are made from main are not yet ready. So how are you going to test A? You will test A with stubs of B, C and D. So make stubs for BCD and then test your module A.

Now when B is ready you can test B. Will you require a driver for B? You will not need a driver because A is ready. Similarly for C you won’t need a driver because A is ready. But you will need a stub for G because G is not ready. So you can test BCD with stubs of G, E and F. Drivers are not needed to test BCD because A was ready before BCD. So A itself is a driver for BCD. When the actual code that makes calls on BCD is ready, you do not need drivers for BCD.

Similarly when you want to test GEF, you do not need any stub and you also do not need drivers because actual modules of C and D are ready before G, E and F. So when you want to test with top down integration, you have to look into the cross sub stubs. You need the stubs for the lower levels when you want to test higher level module. But you do not need drivers because the high levels are ready. When you go to lower level, the higher levels themselves can be used as drivers.

So these are some interesting points or interesting features of top down testing through top down integration. Secondly, we can compare that with bottom up integration.

(Refer Slide Time: 34:14)


SOFTWARE ENGINEERING

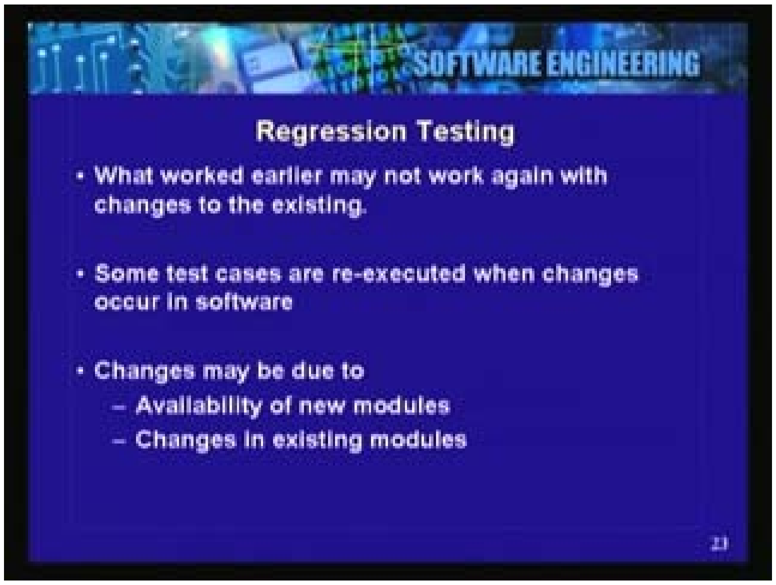
Testing Modules with Bottom Up Integration

- Test leaves first
- Then integrate one by one
- Test E,F,G with drivers for C,D,A
- Test B,C,D with driver for A
- Test A
- Cost of drivers is to be accounted for
- Stubs are not needed
- Error in module A is detected in last stage

In this, you test the leaves first and then you integrate them one by one into the higher levels. For example when you want to test G what will you need? You do not need any stub because G does not make calls to any low level modules. But C is calling G. C is not ready, only G is ready. You are going to test G when C is not ready. So you need a driver for C. Similarly for E and F you need a driver for D. In order to test GEF, you need drivers for CD and A. When you want to test BCD, you do not need stubs for GEF, because the low level modules are ready. But you need a driver for A. When you want to test A, you do not need any driver nor you need any stub, because all the low levels are ready and A itself is the highest level module.

You can test the highest level module directly. You do not need the driver only for the highest level module. Otherwise for all the modules, when you are coming from bottom upwards, you need the drivers for all the upward layers. So mainly you are going to account for the cost of drivers. Stubs are not needed when you have bottom up integration. So error in module A gets detected in last stage because you are going to reach the main thread in the software or the main control in software only at the end. There is another important testing strategy called regression testing. So we see that software undergoes changes. What worked earlier may not work again with changes to the existing.

(Refer Slide Time: 36:02)


SOFTWARE ENGINEERING

Regression Testing

- What worked earlier may not work again with changes to the existing.
- Some test cases are re-executed when changes occur in software
- Changes may be due to
  - Availability of new modules
  - Changes in existing modules

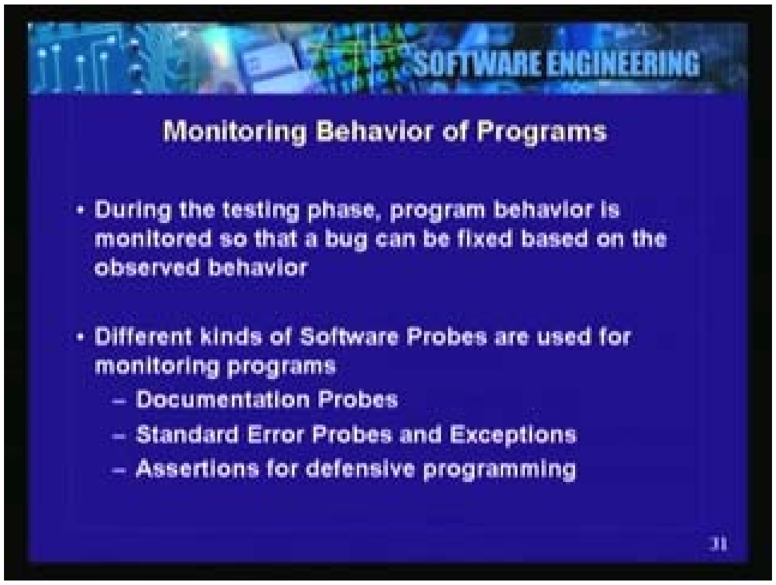
23

When we make changes to something that already exist, what we had tested earlier may not work again. So we have to carry out the test cases again if there is any change and the tested modules depend on what has undergone changes. It depends on the severity of changes and probably the variables that you change, the effects might be different. The effects or the dependencies get propagated to the modules which are dependent on the module that undergoes changes. Some test cases are re-executed when changes occur in the software. Changes may be due to availability of new modules or changes in existing modules. So regression testing allows you to capture these changes and ask you to redo the test cases for all the affected modules.

So you may have to carry out integration test again and again as the software changes due to availability of new modules and changes in existing modules. Now, we will look at some techniques that are used to monitor the behavior of the program. During the testing phase, the program behavior is monitored, so that a bug can be fixed based on the observed behavior.

When you are testing something, what are you going to monitor? Typically monitoring is important during the debugging phase. If you find that there are problems with your outputs for a given test case, you may insert these program monitors, so that you can catch a bug. Or during testing itself you add monitors and generate from the values, so that you don't need to repeat. You can save on some time. We are going to look at these monitoring techniques. Different kinds of software probes are used for monitoring of programs. They are classified as documentation probes, standard error probes and exceptions, and assertions for defensive programming.
We will look at them one by one.

(Refer Slide Time: 38:18)


SOFTWARE ENGINEERING

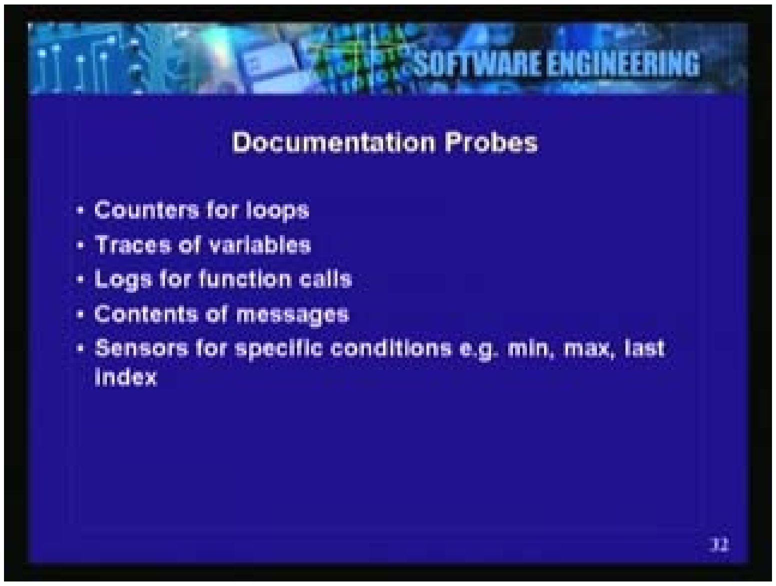
Monitoring Behavior of Programs

- During the testing phase, program behavior is monitored so that a bug can be fixed based on the observed behavior
- Different kinds of Software Probes are used for monitoring programs
  - Documentation Probes
  - Standard Error Probes and Exceptions
  - Assertions for defensive programming

31

So you could have different documentation probe such as counters for loops, traces of variables: how variables change their values, logs for different function calls: how functions are called, what is the sequence in which they are called.

(Refer Slide Time: 38:33)


SOFTWARE ENGINEERING

Documentation Probes

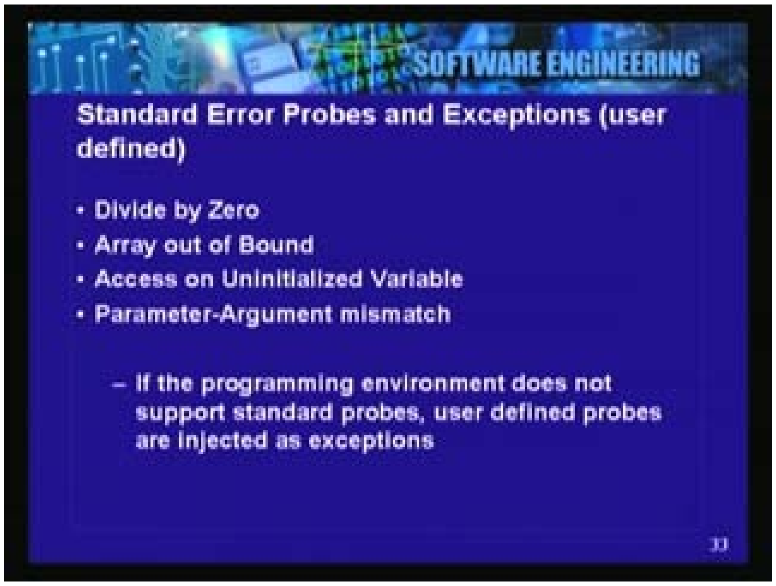
- Counters for loops
- Traces of variables
- Logs for function calls
- Contents of messages
- Sensors for specific conditions e.g. min, max, last index

32

Then contents of messages: if you are having say message passing between processes, objects, then we are going to look at what are the contents of messages that you are passing on.

Then there could be sensors for specific conditions. For example, when minimum is reached or maximum is reached or last index is reached and so on. These are different documentation probes which can be inserted into the program. And standard error probes and exceptions can be used. Standard error probes such as divide by zero, array out of bound, access on un-initialized variable or parameter argument mismatch, type mismatches might be available in a programming environment.

(Refer Slide Time: 39:20)


SOFTWARE ENGINEERING

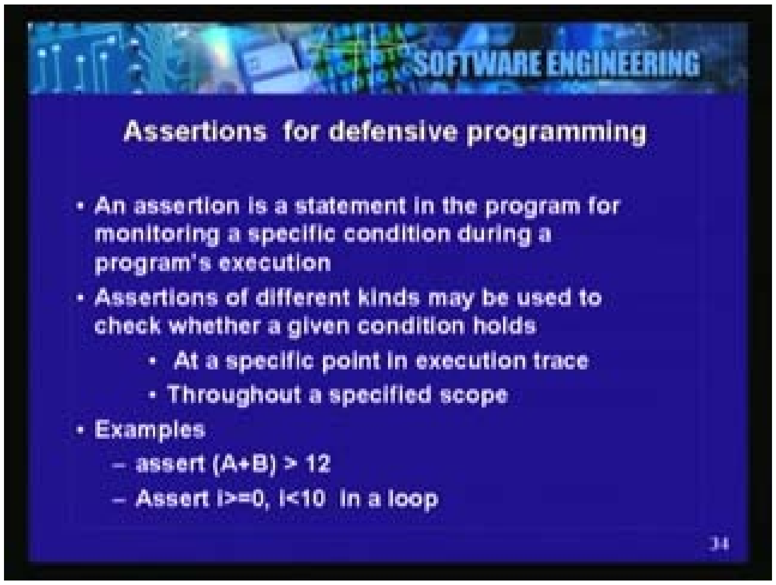
Standard Error Probes and Exceptions (user defined)

- Divide by Zero
- Array out of Bound
- Access on Uninitialized Variable
- Parameter-Argument mismatch
  - If the programming environment does not support standard probes, user defined probes are injected as exceptions

33

Some of these error probes are standard. They are given to you by the programming environment. But if they are not given one could implement exceptions which are user defined at higher levels. So if the programming environment does not support standard probes, user defined probes can be injected as exceptions. Then there is another important technique called assertions. It is used as a defensive programming technique. What is an assertion? An assertion is a statement in the program for monitoring a specific condition during a programs execution.

(Refer Slide Time: 40:00)


SOFTWARE ENGINEERING

Assertions for defensive programming

- An assertion is a statement in the program for monitoring a specific condition during a program's execution
- Assertions of different kinds may be used to check whether a given condition holds
  - At a specific point in execution trace
  - Throughout a specified scope
- Examples
  - assert (A+B) > 12
  - Assert i>=0, i<10 in a loop

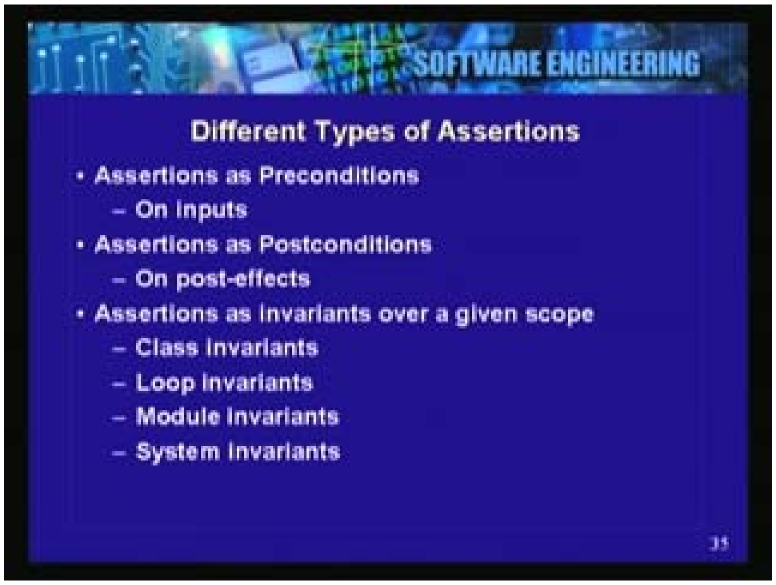
34

Assertions of different kinds may be used to check whether given condition holds at a specific point in execution trace or throughout the specified scope. So what does that mean? Let us look at an example. There are two examples at the end of this slide. First statement says that the value of A+B should be greater than 12. So one might say that when the program flow, during execution reaches this assert statement, at that point of time A+B should be greater than 12. If it is not, then there is some problem in the program. The error should be detected. Hence one could have this ‘assert’ statement when the programs are under development.

When everything is tested well, validated against expected behavior and bugs are fixed, assert statements could be flagged off or could be removed from the programs. Another example of assertions is the last line in the above slide, the second example. Assert i is greater than or equal to zero and i less than ten (i>=0, i<10) in a given loop. So value of i should be 0 to 9. If by some mistake, value jumps beyond this range, then there is a problem and failure must be reported. So assertions are used as defense against errors.

Defensive programming: When you are developing, if you first start with assertion, many of the bugs in the program will be caught early. So the idea is to start with assertions first, injective assertions at appropriate places in your code. And then you write the code for a given object or for a given module for a function. So we can have different kinds of assertions. For example assertions can be used as preconditions on inputs.

(Refer Slide Time: 42:10)


SOFTWARE ENGINEERING

Different Types of Assertions

- Assertions as Preconditions
  - On inputs
- Assertions as Postconditions
  - On post-effects
- Assertions as invariants over a given scope
  - Class invariants
  - Loop invariants
  - Module invariants
  - System invariants

35

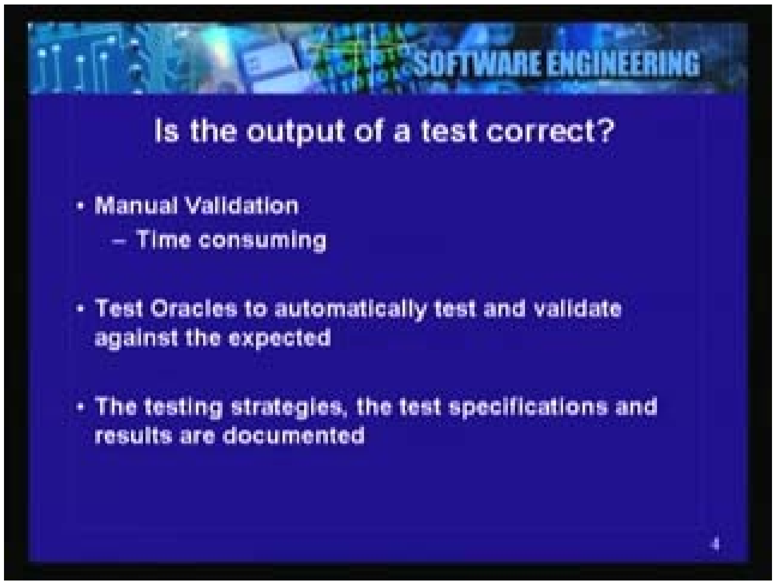
So when you have different inputs for a given function, you check whether the inputs are in a valid range. You assert for the validity of the input. You write an assertion or the assert statement in the program. Different programming languages provide certain support for assertions. For example ‘C’ has the macro called assert. You can use ‘assert.h’ and write assert statements, embed them in your program whenever required.

You can write an assert statement just at the beginning of a function, on the parameters that are sent into the function. So you are going to use assertions on the input, you can use them as preconditions. Or you can use them as post conditions, after the function is completed just before you return the value, check or validate the value. Write your assertion on the desired post effect. For example if you have an operation such as push on a stack. After the code for push is completed, you have to assert that the size of the stack goes by one and the element has gone on top of the stack. So we can write appropriate assert statements as preconditions and post conditions. Precondition is before the actual business logic is completed and post condition is asserted after that business logic is completed.

Then you could also have assertions as invariants over given scope. A specific condition which always is true for the entire life time of the object or of the module, for example: A stack with fixed size could have size less than or equal to its capacity. You cannot have more number of elements than the maximum capacity of the stack when the storage is static. These invariants over a given scope could be of different kinds. They could be class invariants, for a given class you can identify the invariant conditions. There could be loop invariants. There could be module invariants for a module or set of functions. There could be system invariants for the entire system. The conditions that always hold to be true for given scope; for a given class, for inside a loop, inside a module and for the entire system.

So till it could be a class invariant, object invariant and when the loop is executed or during the life time of that loop, the invariant holds true. So one can use these assertions very effectively in program design and also carry them over into implementation using the assertion support provided by a given programming language. All modern programming languages now support assertions through different mechanisms. There also higher level assertion techniques, higher level extensions to these languages which are available. There are many papers also available on assertions; how to use them, what different kinds of assertions are there and where should they be used. So this is an important technique for defense programming. If you use an assert statement one can catch bugs, when the assertion detects the failure or when we fail to assert a given condition at that point of execution. Now another question that we have to see is, how are you going to evaluate your output? Is the output of a test correct?

(Refer Slide Time: 46:03)


SOFTWARE ENGINEERING

Is the output of a test correct?

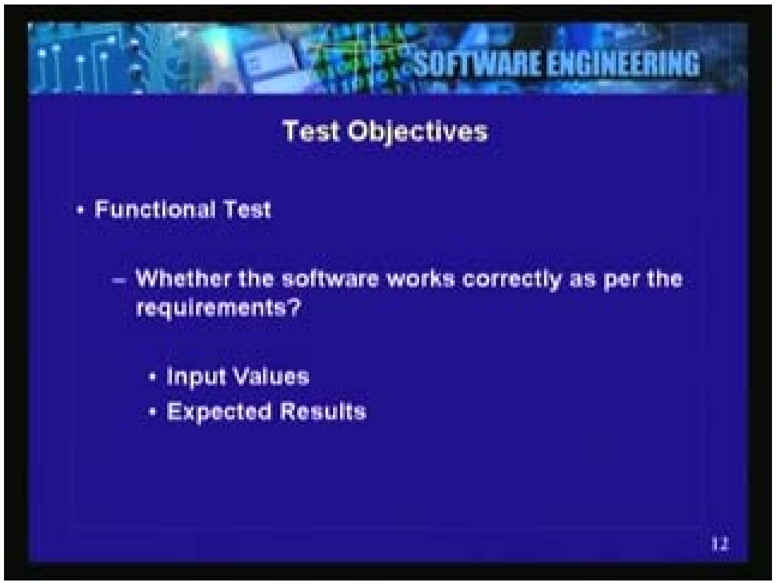
- Manual Validation
  - Time consuming
- Test Oracles to automatically test and validate against the expected
- The testing strategies, the test specifications and results are documented

4

One can either do manual validation, but it is a very time consuming task. You can generate all the data for a given test case and human being/testers or probably the evaluators can go through each of the outputs that have produced during the test. But it is a very time consuming task. You can also do the same automatically through test oracles. One can develop these test oracles to automatically test and validate against what is expected. The testing strategies, the test specifications and the catalogues are to be documented. The plans are to be very well laid out and one has to organize entire testing phase. As we have seen right from requirements to completion of the entire system delivery, you have to carry out some or the other activities that are related to testing.

So what are the objectives of testing? One can look at testing from functional point of view, whether the software works correctly or not. This is one of our important objectives. The most important objective is to look at the functionality of entire software.

(Refer Slide Time: 47:10)


SOFTWARE ENGINEERING

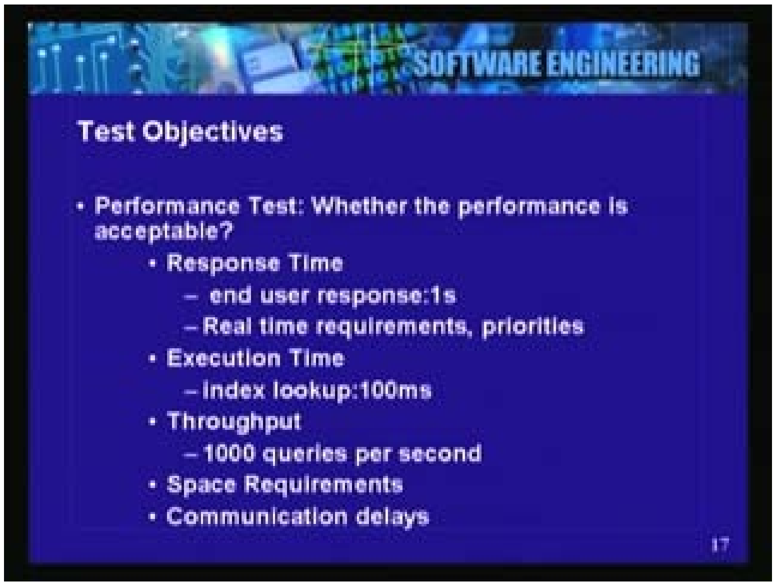
Test Objectives

- Functional Test
  - Whether the software works correctly as per the requirements?
    - Input Values
    - Expected Results

12

Hence one should look at the input values and the expected results. So functional testing is most important because we look at the basic functionality, the correctness in the software against the basic functionality expected out of the software.

(Refer Slide Time: 47:44)


SOFTWARE ENGINEERING

Test Objectives

- Performance Test: Whether the performance is acceptable?
  - Response Time
    - end user response:1s
    - Real time requirements, priorities
  - Execution Time
    - Index lookup:100ms
  - Throughput
    - 1000 queries per second
  - Space Requirements
  - Communication delays

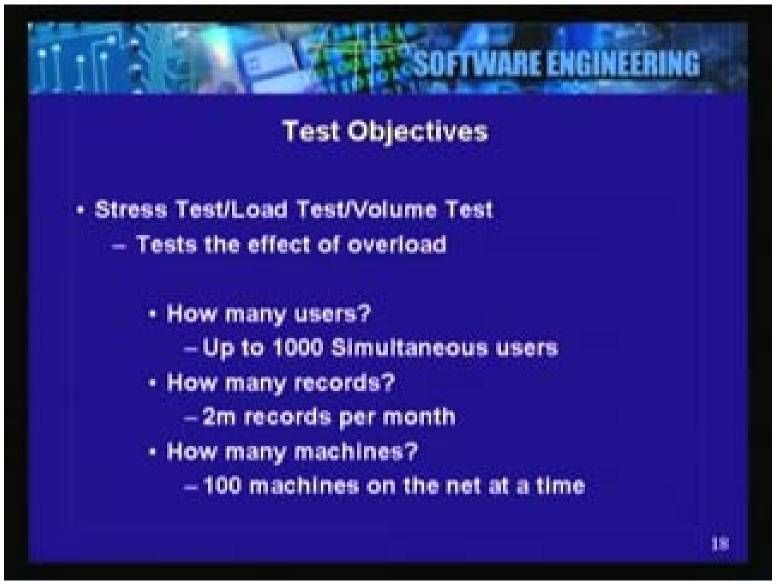
17

Then you look at the performance, whether the performance is acceptable or not. Different performance parameters have to be measured and tested against what is expected as per the requirements.

So response time; how much time does the software take in order to complete a specific feature. For example, when you want to return a book in the library, how much time does the software take? Does it take 3 seconds or 4 seconds? This is the response time. The execution time: This is typically the completion time of a given task. Through put: This is how many transactions are you able to carry out. So the difference between response time and execution time is that, in response time you are going to look at, when does the first response arrive at to you and execution time is how much time it takes in order to complete your given transaction. Throughput tells you how many transactions or how many activities are you completing in given unit time.

Then space requirements; how much space? How much memory is needed? What should be the disk capacity? So one can generate test and see whether you are exceeding your space limit or disk limit, spaces of different kind. Then delay in communication. If you have communicating systems there could be intranet or internet or different kinds of communication systems may be used. There could be hardware communication in your system. So all kinds of communication links are to be involved and you test for performance of your communication. Then you want to look at stress test. So you test for the effects of overload: How many users, how many records and how many machines, does the system work when it is overloaded etc.

(Refer Slide Time: 49:55)


SOFTWARE ENGINEERING

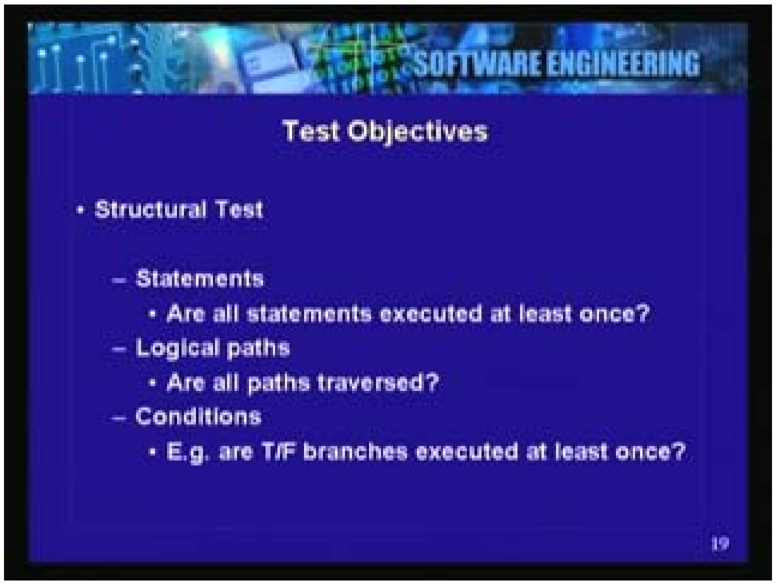
Test Objectives

- Stress Test/Load Test/Volume Test
  - Tests the effect of overload
    - How many users?
      - Up to 1000 Simultaneous users
    - How many records?
      - 2m records per month
    - How many machines?
      - 100 machines on the net at a time

18

Then you look at structural testing, you are looking at logical parts, branches etc. You traverse them and test your software against the structure. This is mainly the white box approach.

(Refer Slide Time: 50:15)



SOFTWARE ENGINEERING

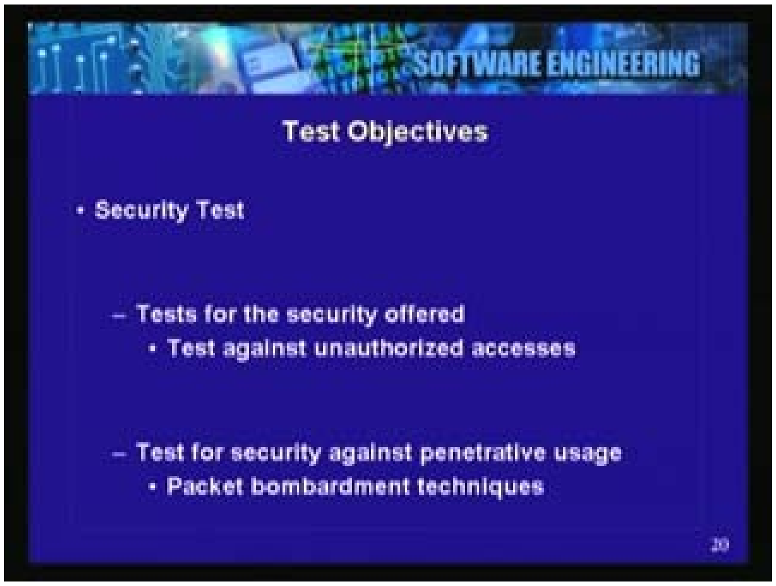
Test Objectives

- Structural Test
  - Statements
    - Are all statements executed at least once?
  - Logical paths
    - Are all paths traversed?
  - Conditions
    - E.g. are T/F branches executed at least once?

19

Then you have security test where you look at the security that is offered by the given software. Say for example, you may have password based security and you test whether it works.

(Refer Slide Time: 50:25)



SOFTWARE ENGINEERING

Test Objectives

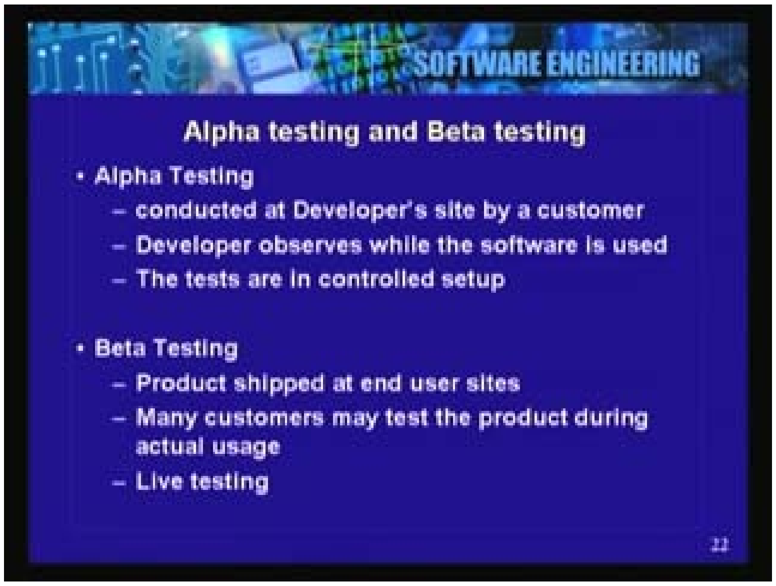
- Security Test
  - Tests for the security offered
    - Test against unauthorized accesses
  - Test for security against penetrative usage
    - Packet bombardment techniques

20

Then you also have to test against the penetrative usages. Is the system designed in such a way that the hackers are trapped or are there possibilities that there are trap doors? Are there any different loop holes in the system where the basic security can be violated? Hence different kinds of security test are to be conducted with these objectives, the test plan for module and subsequent integration test is made.

Then we hear these two terms alpha testing and beta testing quite often with respect to testing scenario. What is alpha testing? It is typically conducted at developer's site by a customer.

(Refer Slide Time: 51:15)


SOFTWARE ENGINEERING

Alpha testing and Beta testing

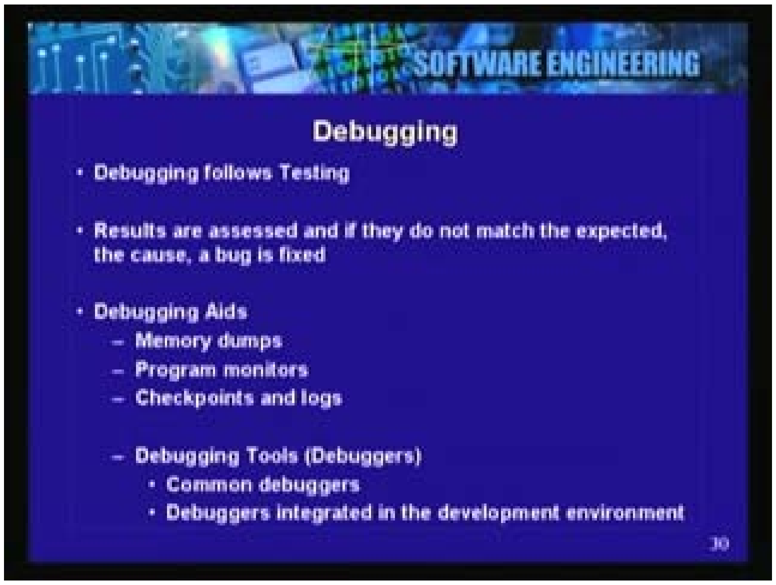
- Alpha Testing
  - conducted at Developer's site by a customer
  - Developer observes while the software is used
  - The tests are in controlled setup
- Beta Testing
  - Product shipped at end user sites
  - Many customers may test the product during actual usage
  - Live testing

22

Developers observes while the software is being used at developer's site and the test are in controlled setup at the developers site. Beta testing occurs at user's site. Software or the product is shipped at end user sites and many customers may test the product during actual usage. We have these two terms called alpha release and beta release associated with these two different phases in the products lifecycle. So one can say it is alpha tested or it is beta tested and so on and so forth. So beta testing is typically live testing at customer side on live data.

Then we will have a couple of points on debugging. Debugging follows testing and the results are assessed and if they do not match the expected, the cause of the bugs or the cause of this mismatch of expected from what was observed is to be fixed.

(Refer Slide Time: 52:10)


SOFTWARE ENGINEERING

Debugging

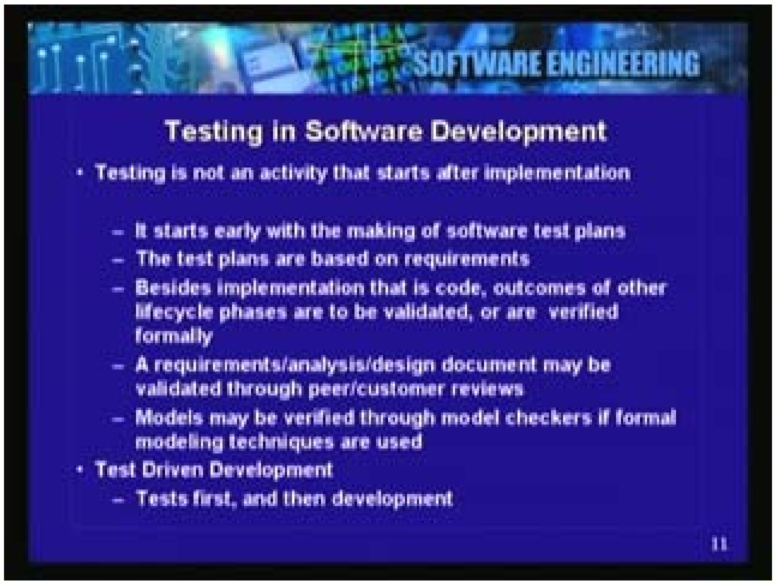
- Debugging follows Testing
- Results are assessed and if they do not match the expected, the cause, a bug is fixed
- Debugging Aids
  - Memory dumps
  - Program monitors
  - Checkpoints and logs
  - Debugging Tools (Debuggers)
    - Common debuggers
    - Debuggers integrated in the development environment

30

And various debugging aids can be used. For example, one can have memory dumps or program monitors which we have seen earlier or checkpoints and logs. There are tools available called debuggers. There are common debuggers which can be used on a specific executable format. But the executables can be produced through different programming languages. Or you could have debuggers which are integrated in development environment for the given language.

There are different debugging tools available and they provide some of these mechanisms. So debugging is an important phase where one has to carry out bug fixing after the testing is done. Then probably after debugging, again you carry out test and see whether you are now following all specifications, whether your test are generating the correct expected output. Some related issues in testing are listed below.

(Refer Slide Time: 53:23)



SOFTWARE ENGINEERING

Testing in Software Development

- Testing is not an activity that starts after implementation
  - It starts early with the making of software test plans
  - The test plans are based on requirements
  - Besides implementation that is code, outcomes of other lifecycle phases are to be validated, or are verified formally
  - A requirements/analysis/design document may be validated through peer/customer reviews
  - Models may be verified through model checkers if formal modeling techniques are used
- Test Driven Development
  - Tests first, and then development

11

Testing is not really an activity that starts only after implementation. It can start early with software test plans based on the requirements. Besides implementation that is code, outcomes of other life cycle phases are to be validated or are verified formally. For example, a requirements analysis or design document may be validated through peer or customer reviews. And models may be verified through model checkers if formal modeling techniques are used.

And there is another aspect of testing or another new view that software practitioners have found is driving development by test cases. That is called test driven development. What is advised there is that, one can create the test suite first and then you carry out the development. So first you get the test cases and then you carry out the development. Once you look at the test first, you are most sure about what you are going to expect. So this is called test driven development. These are some of the recent views on testing. In this talk, we have seen different testing strategies, we have gone through black box/white box testing, regression testing and different issues relate to modular testing, testing of modules, top down and bottom up integration strategies and we have also seen different kinds of program monitors and discussed various related issues.

Thank you.