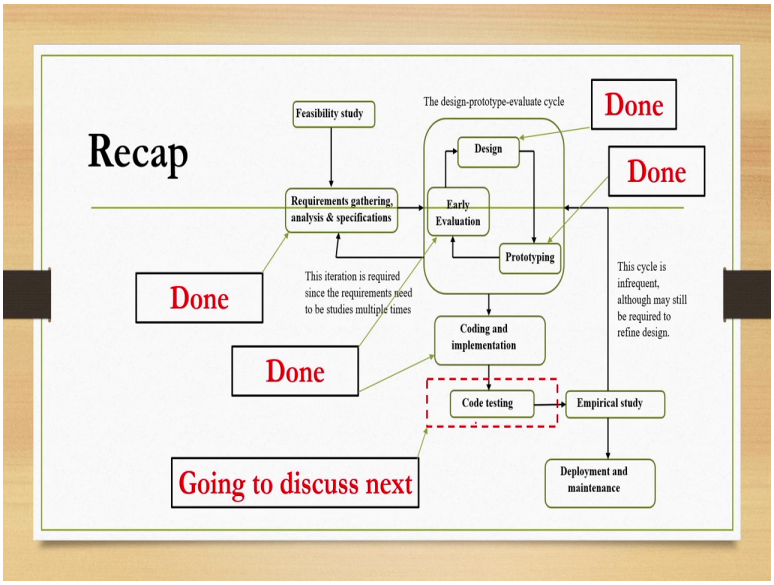
Design and Implementation of Human – Computer Interfaces
Prof. Dr. Samit Bhattacharya
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture: 28
Code Testing Basics

Hello and welcome to the NPTEL MOOCS course on design and implementation of human computer interfaces lecture number 25 where we are going to continue our discussion on implementation and testing. So, let us first quickly have a look at what we have covered. So, far and where we stand.

(Refer Slide Time: 01:02)



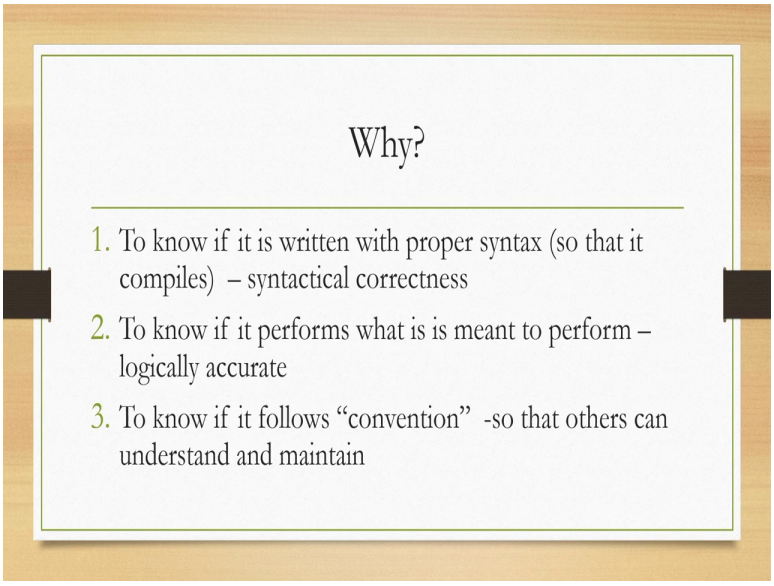
So, we are discussing the interactive system development life cycle it consists of several stages among those stages we have already covered some stages and currently continuing discussion on some other stages and in future also we are going to cover the remaining stages. So, which are the stages that we have covered. So, far let us have a quick look. So, we have covered requirement gathering analysis and specification stage then we have covered design stage prototyping stage, early evaluation of the prototyping stage.

Now this design prototype evaluation cycle is primarily meant for interface design keeping usability in focus. So, we have covered all the three substages of this cycle for interface design. Subsequently we covered design of the system or code design. Then we discussed in the previous

lecture the coding and implementation stage where we learned about coding practices what we should do what we should avoid while writing a program.

So, currently we are going to pay our attention to the code testing stage which is our topic of this lecture.

(Refer Slide Time: 02:29)



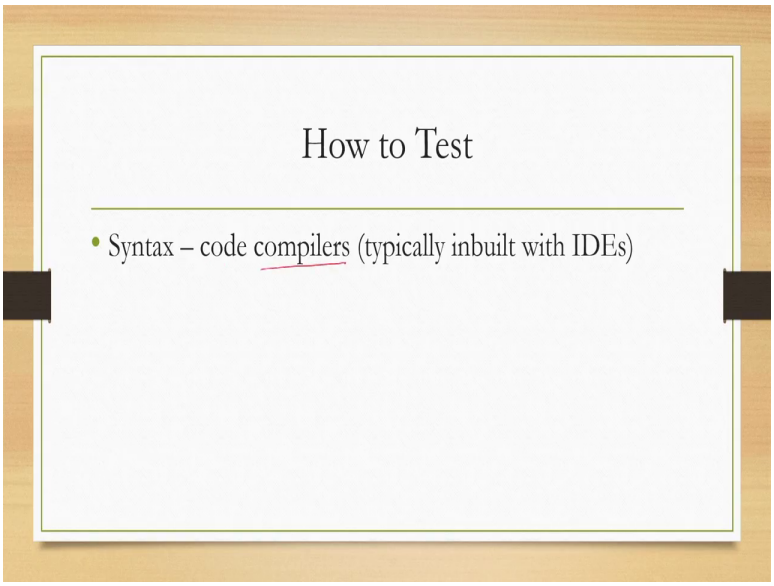
So, what is code testing first of all why we need testing as the name suggests testing means we need to test the code but why. So, primarily there are three reasons for testing our code first is to know if the code is written with proper syntax why that is required. So, that we can compile the code and create the executable program. So, essentially we need to check for syntactical correctness of the code.

Secondly we need code testing to know if the code performs what it is mean to perform. So, it produces the expected outcome after execution that means we need to test whether the code is logically accurate it does what we want it to do. There is another reason which is equally important that is we need to go for code testing to know if the code follows convention. So, that others can understand and maintain the code.

Now this is very important not only syntactical and semantical correctness we also need to test code specifically keeping in mind the maintainability of the code. Others can understand the

code others can modify the code in summary others can maintain the code this is required to implement a team effort and for that also we need to go for testing of the code.

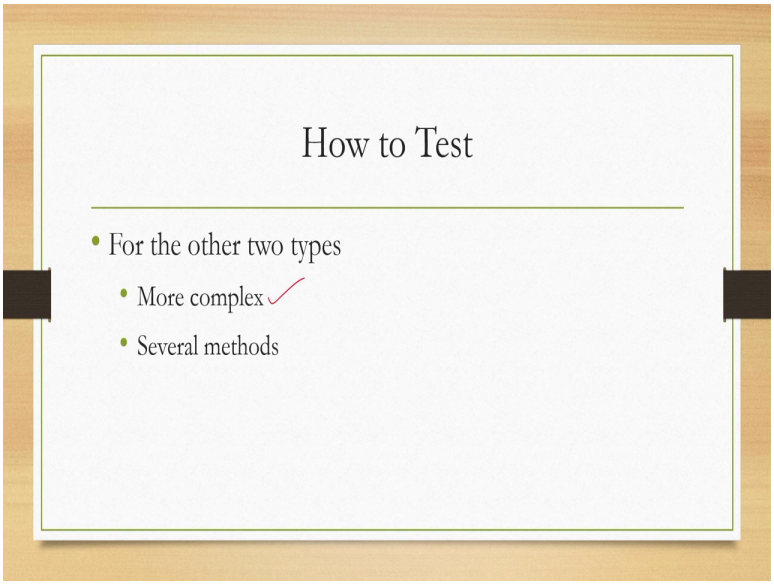
(Refer Slide Time: 04:11)



So, once we know the reasons the next thing that comes to our mind is how we can test our code keeping in mind these three objectives. First one is syntactical correctness how we can check for syntax of the code whether the code syntax is correct or not whether we have written the code correctly or not. That is simple generally that is the most simple part of code testing. We rely on the compilers which typically comes in built with the integrated development environments.

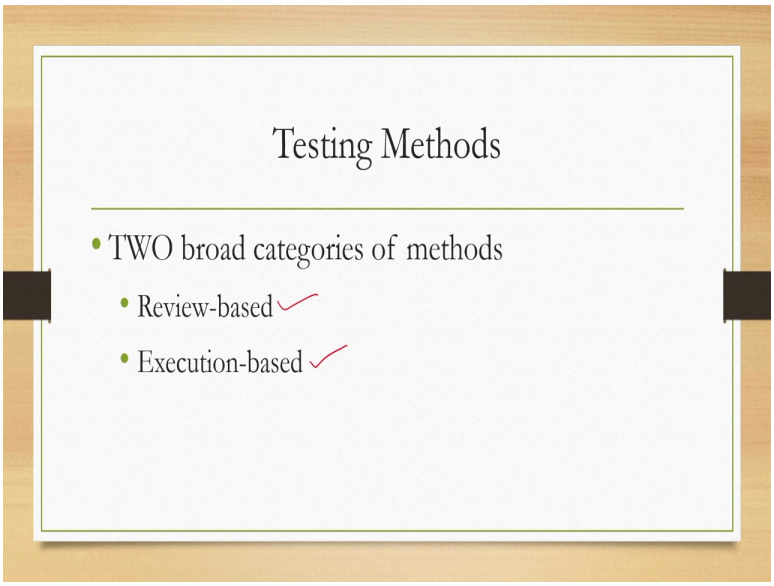
I think most of you have probably already used such IDEs and the inbuilt compiler to check for syntactical correctness of your code.

(Refer Slide Time: 04:58)



The other two testing situations namely testing for logical correctness and testing for compatibility with convention our IDEs or the inbuilt compilers cannot be of any help. These two are generally more difficult to test and more complex in nature and there are actually several methods to take care of these two types of testing. So, let us have a quick look at different testing methods.

(Refer Slide Time: 05:35)



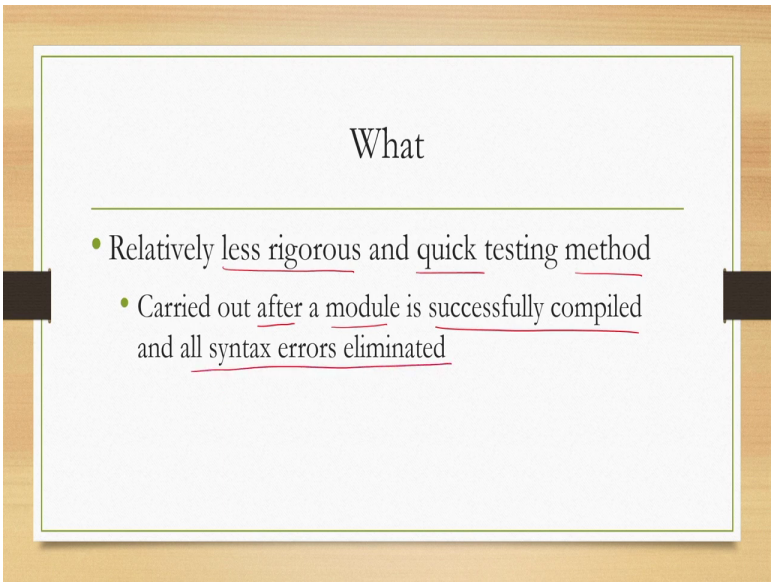
Now there are two broad categories of methods one is review based code testing other one is execution based code testing. So, whenever we are talking of code testing our primary emphasis is for testing of logical correctness and testing of conventions whether the code is following the

conventions. Now to do that we can apply any of the several methods available broadly all the testing methods can be divided into two categories one is review based testing one is execution based testing.

(Refer Slide Time: 06:24)

Let us first start our understanding with the review based code testing.

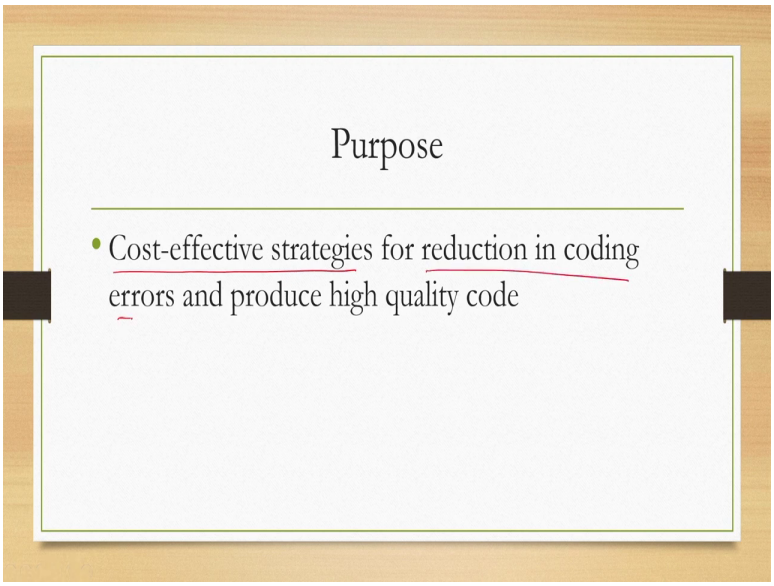
(Refer Slide Time: 06:27)



Now review based testing's are relatively less rigorous but they are also quick methods for testing of your code. So, although they are not very rigorous but they can be used to test code quickly generally such testing are carried out after a module or an unit is successfully compiled

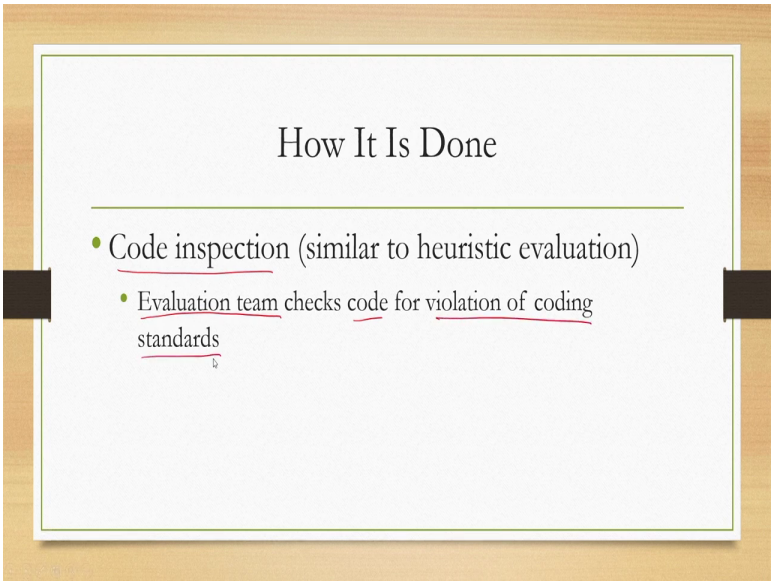
and all syntax errors are eliminated. So, once you implement an unit or module and compiled it and found out that there are no syntactical errors then we can go for review based code testing for that particular module.

(Refer Slide Time: 07:22)



Now review based testing offers a cost effective strategy for reduction in coding errors and production of high quality code. Of course that is the ultimate objective of any testing method that we identify errors address them and produce high quality code. Now review based testing provides a cost effective way of doing the same thing it produces high quality code to some extent with methods that can be applied quickly and with less cost.

(Refer Slide Time: 08:02)

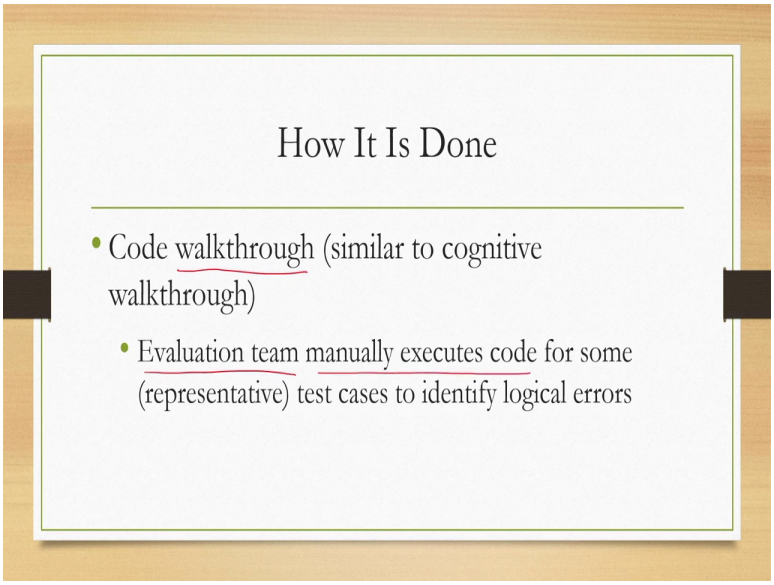


So, review based testing also can be done in different ways broadly there are two ways one is code inspection which is somewhat similar to heuristic evaluation that we have earlier seen during our discussion on evaluation of prototypes remember there we mentioned that heuristic evaluation is a kind of checklist type evaluation where we tick from a list after checking the prototype.

Similarly code inspection is kind of similar method where we have a set of predetermined things to check and we take the code and go through it and see whether the things that are there in our list in our checklist are satisfied in the code or not. So, here in code inspection method there will be an evaluation team just like heuristic evaluation for prototypes team members will check the code for violation of coding standards.

So, we have a checklist of standards to be followed while writing a code and the team members goes through the code and see if there are any violations.

(Refer Slide Time: 09:29)



The other method is code walkthrough again this is similar to the cognitive walkthrough method that we have seen or discussed earlier in the context of evaluation of prototypes. So, in case of code walkthrough again there will be an evaluation team each team member will manually execute the code this is often called hand execution I think many of us probably have done that during our code writing practices.

And the team members will do that for some test cases which are to be representative test cases to identify logical errors. So, like cognitive walkthrough there are some usage scenario which we call test cases which ideally should be representative test cases just like in the case of cognitive walk through. And those test cases will consist of input and output the members of the evaluation team will hand execute or manually execute the code for the given input and see what output is getting produced and will compare with the desired output.

And see whether there is any problem with the two outputs whether they are matching or not to find out if there exist any logical error in the code.

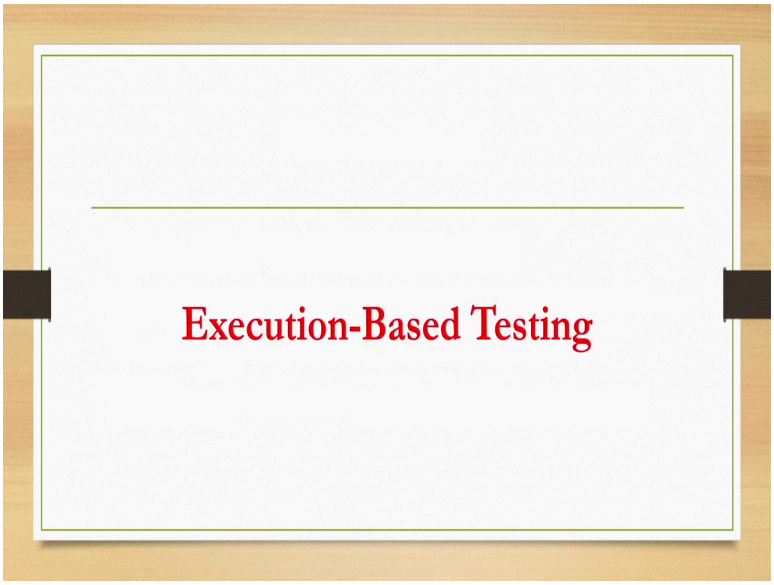
(Refer Slide Time: 11:04)

These are broadly two ways to go for testing of code in other words we review the code in case of inspection primary objective is to check if the code follows the standards. And in case of code walkthrough primary objective is to check if there are any logical errors with respect to some representative test cases. So, how these methods help these are quick methods similar to the prototype evaluation methods namely heuristic evaluation and cognitive walkthrough these methods can also be applied in a relatively quick way.

So, eventually they help us save development time of course they are not rigorous methods. So, after applying these methods it may happen. In fact it is quite likely to happen that some errors are still there which we could not detect through these methods but idea is that we identify as many errors as possible with this quick method. So, that the quality of the code improves. Code inspection helps to conform to good coding practices which in turn helps in maintainability and portability of the code.

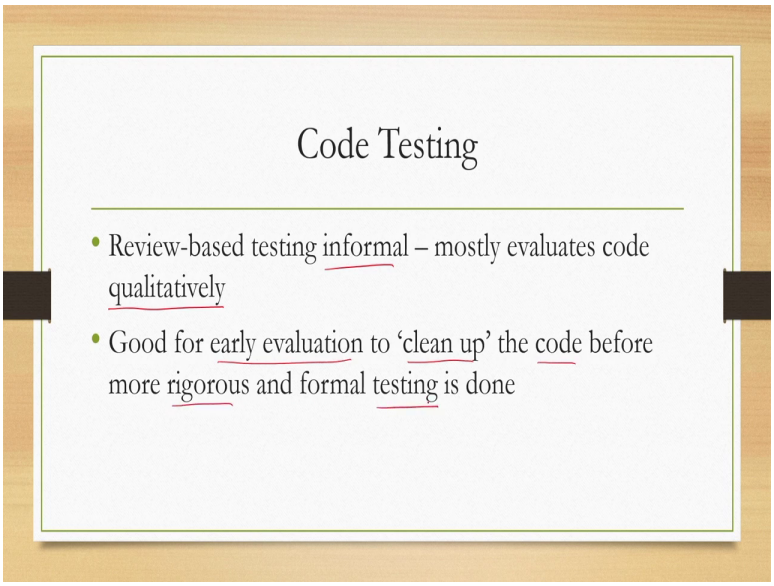
These are very important achievements when we are looking for team effort. Code walkthrough on the other hand allows the developers to quickly identify logical errors of course some errors may still be missed because it is not possible for a complex system to come up with all possible test cases. But if the test cases are chosen carefully if they are representative then major points where logical errors can happen major areas of the code can be quickly identified. So, that is all about review based code testing.

(Refer Slide Time: 13:16)



The other type of code testing is execution based testing. Unlike review based where we primarily go through the code manually and do the testing in execution based that is not the case.

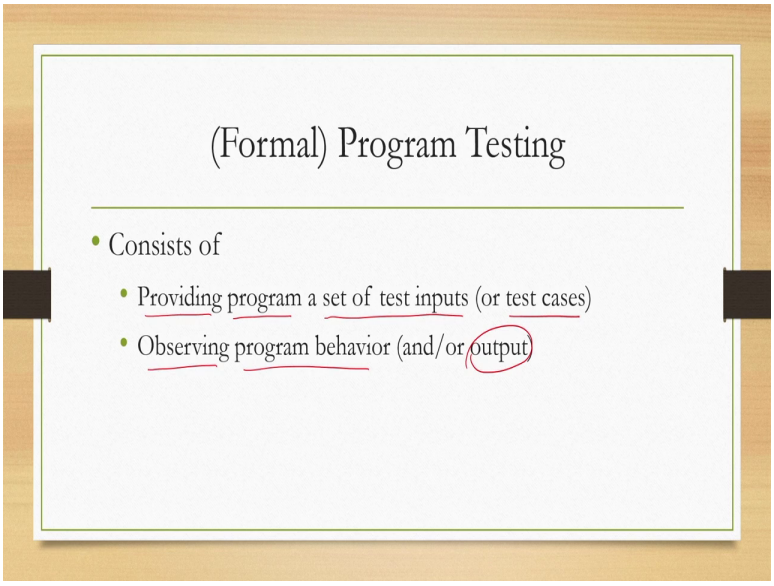
(Refer Slide Time: 13:31)



So, review based testing is somewhat informal they are primarily meant to evaluate the codes qualitatively that is 100% true for inspection based method but partly true for walk through based method. Although in walk through also one of the objectives is to identify violation of coding standards along with logical errors. So, if we are performing review based testing they may not be able to give us all possible errors.

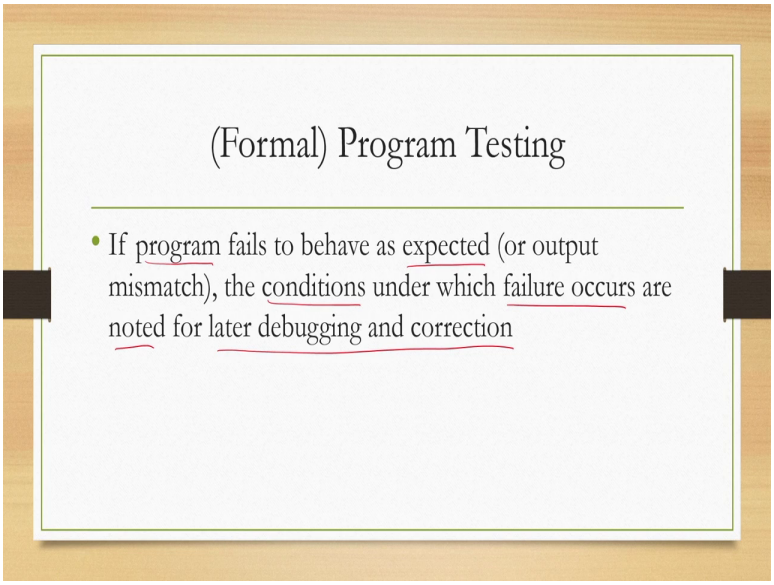
So, such type of testing's are good for early evaluation similar to prototype evaluation and testing. Now that early evaluation helps us to clean up the code before more rigorous and formal testing is done. So, eventually we have to go for some somewhat rigorous and formal testing but before that we can apply the code review methods to kind of preprocess the code and clean up some important errors.

(Refer Slide Time: 14:46)



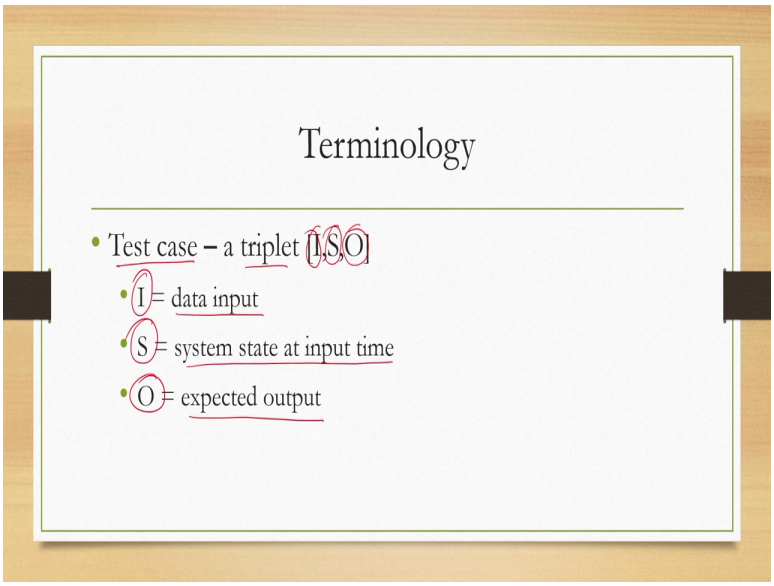
Now the formal or rigorous testing consists of providing the program a set of test inputs which popularly is known as test cases and then observing the behaviour of the program in other words the output primarily. So, these are the things that we generally do with more rigorous and formal program testing methods.

(Refer Slide Time: 15:21)



If the program or the code fails to behave as expected that means the output that it generates is not the same as the output that is expected for a given input the conditions or the inputs under which failure occurs are noted for letter debugging and correction that is the way to go for more rigorous and formal program testing. So, before we learn about the formal testing methods let us quickly learn about the terminologies that we are going to use in the subsequent part of the lecture and subsequent lectures.

(Refer Slide Time: 16:08)

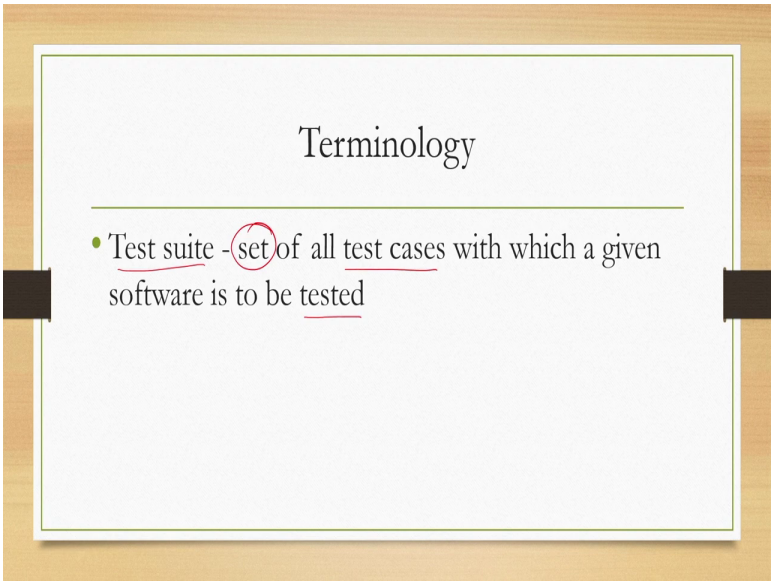


First term is test case this is a very common term that we are going to use throughout this lecture and the next lecture a test case can be considered to be a triplet I, S and O. So, here I stands for

the input data S stands for the state of the system at the time of providing input data and O stands for the expected output. So, that is the meaning of the triplet and this triplet is popularly called a test case. However for simplicity in subsequent discussion we will consider only the doublet I and O.

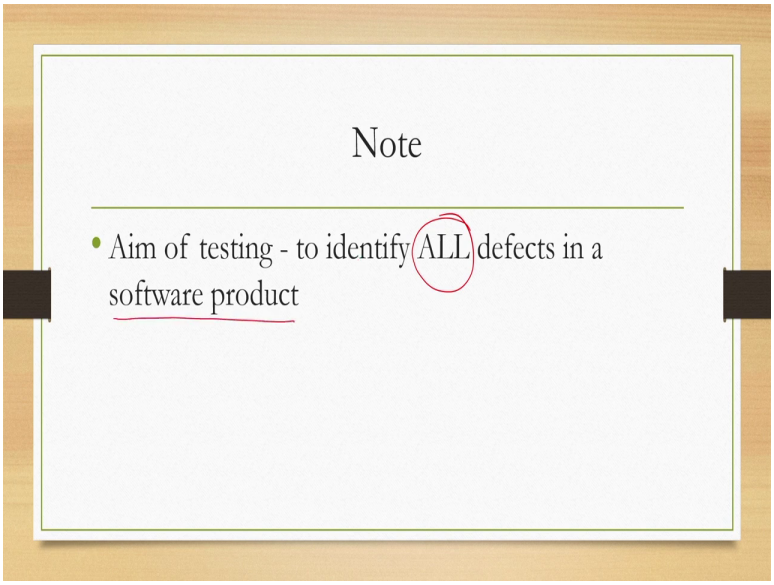
So, we will not generally consider S that will unnecessarily complicate the discussion without adding much value. So, we generally consider the doublet I and O to be representing the test case.

(Refer Slide Time: 17:12)



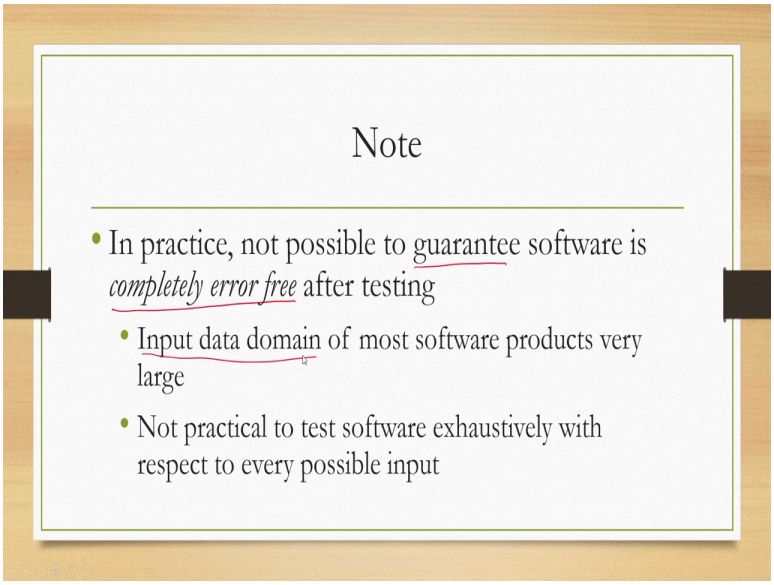
Closely related to the term test case is the term test suit. Now a test suit refers to a set of all test cases with which a given software is to be tested. So, generally we test a program or a software with a set of test cases rather than a single test case. Now this entire set is popularly called a test suit.

(Refer Slide Time: 17:40)



So, what is the aim of testing to identify all defects or problems in a software product but is that feasible is that viable.

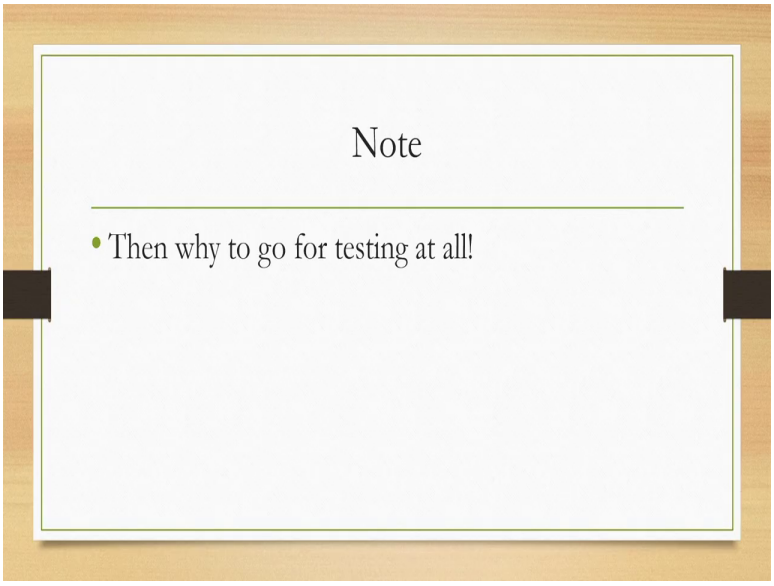
(Refer Slide Time: 17:51)



In practice it is generally not possible to guarantee that a software is completely error free after testing. No matter whatever way we try no matter whatever method we apply it is a practically impossible to guarantee that after testing we will be able to identify all possible errors and we will be able to rectify all possible errors that is not a practical way to think of testing. Instead what we can do is something different.

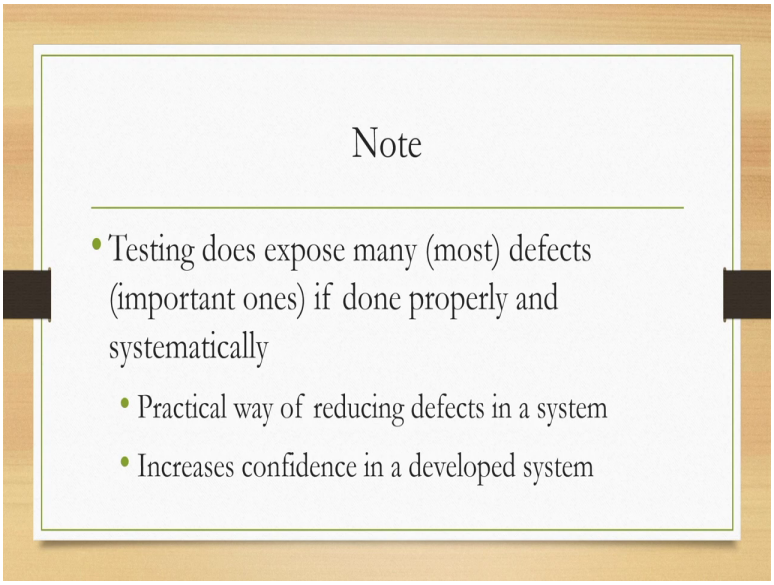
But before that why it is not possible because input data domain that means the set of values that can act as input for most of the real life software products are generally very large in size. So, the possible set of inputs is very very large in most of the practical systems then it is not practical to test a software exhaustively with respect to each and every possible input because that will consume huge amount of time if at all we managed to do that.

(Refer Slide Time: 19:14)



Now a logical question can be if it is not possible to test everything and it is not possible to identify all possible errors then what is the purpose of formal testing review based testing would have been sufficient why to go for testing at all.

(Refer Slide Time: 19:33)



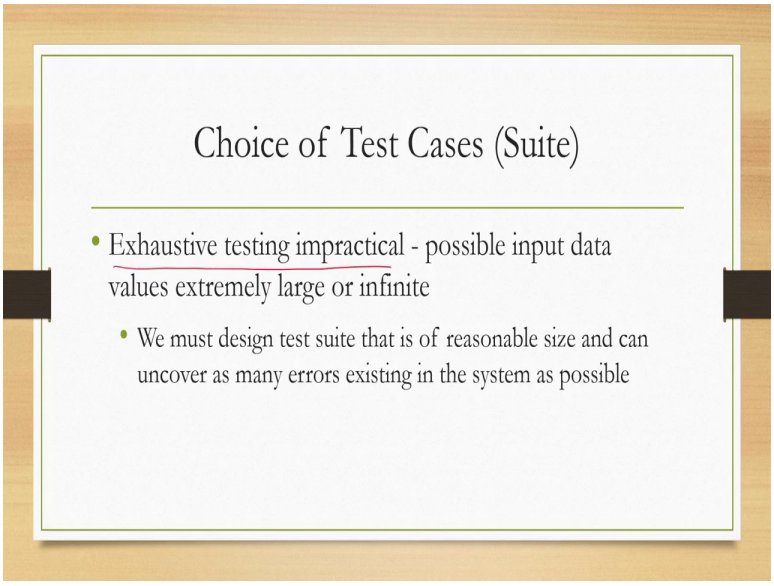
We have to look at this problem from a different point of view it is not binary that either we test and find out everything or we do not test rather our objective should be we test to find out major problems knowing very well that some of the problems may still escape our attention. However our objective should be to identify all major problems. So, testing does expose many rather most defects which are the important ones if done properly and systematically.

Now these two are very important. So, we cannot go for testing with random test cases and randomly created test suits rather we have to adopt a very appropriate method which has to be carried out in a very systematic manner. So, that we are able to identify the important problems or important defects with the system that is our goal rather than identifying all possible defects. In that way it is more practical to reduce important defects in a system and that in turn is likely to increase confidence in a developed system.

So, if we do not do testing then the end users or the clients may not have the confidence of using the system in real life situations and there may always be a thinking that the system may fail because we have never tested it. On the other hand if we test it and identify important issues then at least we know where it can fail and we can rectify for those scenarios which are likely to be the most important and most frequent ones.

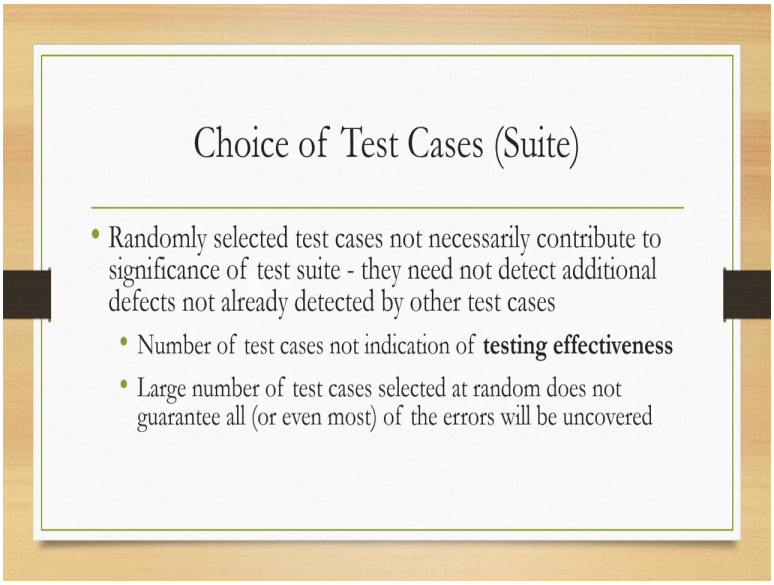
For some minor issues there can always be some bugs which we can always fix later. We have to always keep in mind that it is not possible to come up with a perfect system that does not exist whatever software we develop we have to have this knowledge that there will always be some bugs even at the end of a very rigorous testing phase. So, the key thing here is that we need to go for a proper method with systematic application of the method to identify the major defects that are most important ones.

(Refer Slide Time: 22:05)



Now exhaustive testing is impractical because possible input data values are likely to be extremely large or infinite. Then we must design test suit that is of reasonable size and can uncover as many errors existing in the system as possible that is our goal.

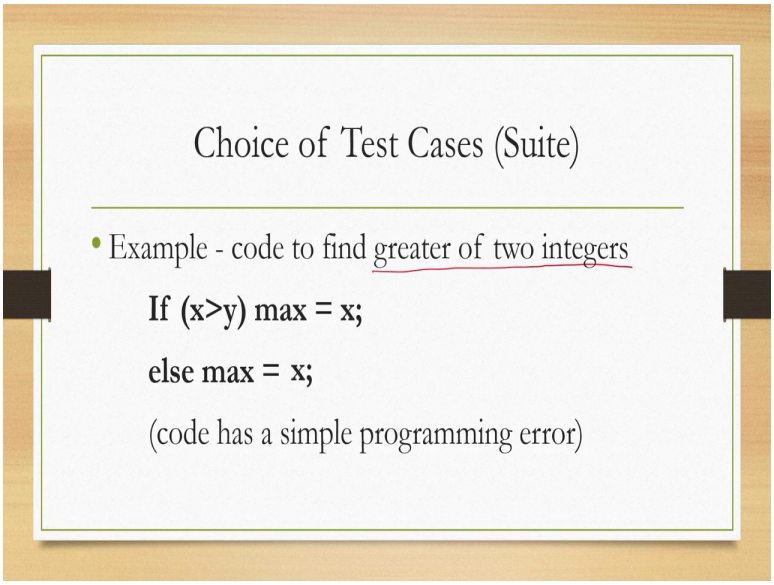
(Refer Slide Time: 22:29)



If we select the test cases randomly then they not necessarily contribute to the significance of a test suit that means they need not detect additional defects not already detected by other test cases. So, the total number of test cases in a suit not necessarily indicates the testing effectiveness where effectiveness is defined in terms of identification of major or important defects in the system.

Large number of test cases selected at random does not guarantee that all or even most of the errors will be uncovered. So, our objective should not be to go for a set of randomly decided test cases even if that leads to a very large test suit. Because it is very unlikely that that test suit will lead to uncovering of errors that are not already detected with a smaller suit or even it will lead to uncovering of important defects with the system.

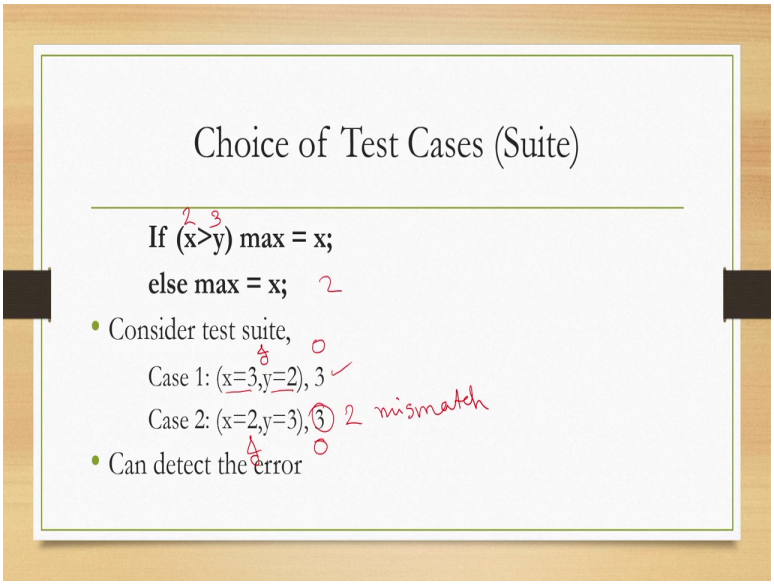
(Refer Slide Time: 23:52)



To understand this point let us take one example suppose we have written a code to find the larger of the two integers. So, the code snippet is given here if x greater than y then max equal to x else max equal to y. So, if x is greater than y then max equal to x and if x is less than y then max equal to y. So, this is likely to give us the larger or greater of the two integer values. Now here instead of y suppose originally it was written as x. So, the code has a simple programming error.

So, I although y should be the right one but instead while writing the code the programmer has written else max equal to x as shown here. So, the y is not present. So, this has a simple programming error. Now how do I identify this error?

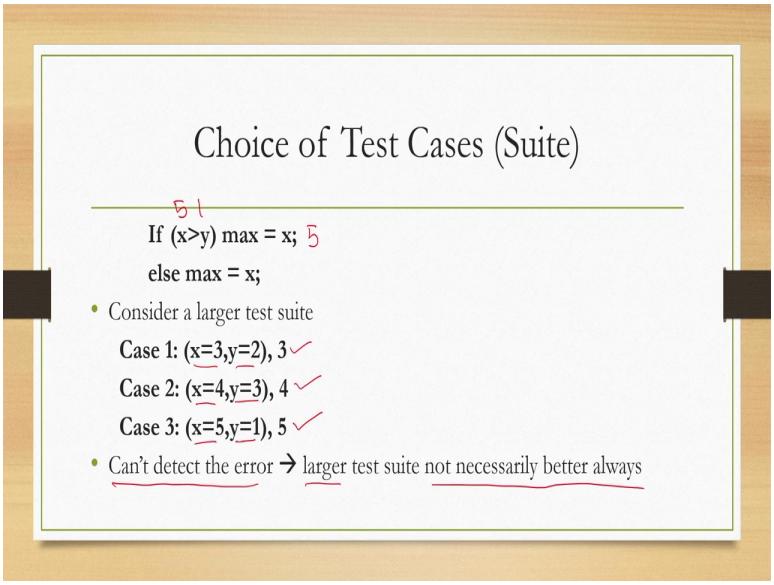
(Refer Slide Time: 25:07)



Let us see how we can design test cases to identify this error. Consider a test suit consisting of two test cases x equal to 3 y equal to 2 that is the input and output is 3. Similarly in the second test case x equal to 2 y equal to 3 and the output is 3. So, in the I O doublet form we have in each test case we have the input and the output mentioned. Now with this let us see how we can identify the programming error that is present there.

If we put x equal to 3 and y equal to 2 then we can see that the output will be 3. So, for the first case it produces the correct output. For the second case when we have x equal to 2 and y equal to 3 this condition fails it goes to next condition and then it produces 2. So, there is a mismatch instead of three it produces two. So, here we have a mismatch and then we know that there is some error once this error is detected we will be able to identify where exactly the error is and we can rectify it.

(Refer Slide Time: 26:53)

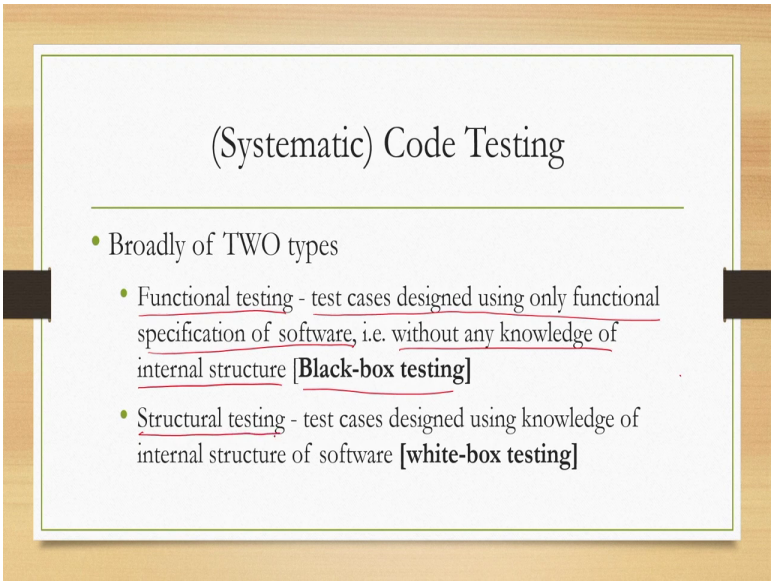


Now let us consider a larger suit consisting of three test cases like before x equal to 3 y equal to 2 output is three x equal to 4 y equal to 3 output is 4 x equal to 5 y equal to 1 output is 5. with this will it be possible to identify the error. Let us see when x equal to 3 y equal to 2 we get output 3 it matches when x equal to 4 and y equal to 3 again we get output as four it matches. So, we do not detect any error.

And finally when x equal to 5 and y equal to 1 again we get 5 as output. So, again we do not detect any error. So, eventually every time with the given input we get the desired output which is exactly what we want. So, there is no mismatch. So, we conclude that there is no error although we can see here that there is some error. So, this test suit cannot detect the error although this test suit is larger in size.

So, larger test suits not necessarily give us a better test suit. So, the moral of the story is if you do not choose your test cases carefully and create your test suit appropriately then no matter whether the test suit is larger or not you may not be able to identify the error that means you will fail in your objective of detecting errors.

(Refer Slide Time: 28:43)



So, the implication is test suit should be carefully designed it should not be decided randomly that is the implication of this discussion. Now this careful design of test suit requires some systematic approaches how we can do that. So, that systematic approach can be done in either of broadly 2 ways functional testing. So, when we are trying to perform a test of a code we need to systematically design the test cases and apply the testing method.

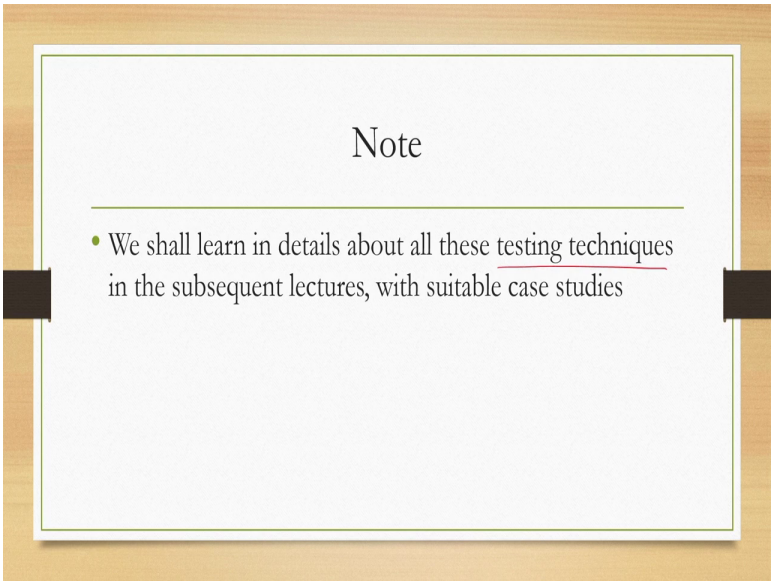
Now the systematic approach to testing can be done in broadly one of the two ways one way is functional testing here test cases are designed using only functional specification of the software. That means without any knowledge of the internal structure in other words suppose we have written a function at hint 1 in 2 and it returns an int there are some statements and this is the function. So, when we are testing we designed the test case based on only the definition of the function that means it takes as input two integers.

And produces as output one integer rather than what is written here inside the body of the function. So, this body we simply ignore and we test based on the definition of the function one popular way of going for this type of testing is black box testing. As the name suggests we treat each function as a black box only the declaration part is considered the internal or structural part is ignored. So, we simply assume add function to be a black box and what is there inside the box that means the body of the function is of no consent to us.

In that way we understand the function and with that understanding we create the test cases to test the function that is black box testing which is a type of functional testing. As opposed to the black box testing there is another method that is called structural testing. Here we decide the test cases based on the knowledge of the internal structure of the code. Unlike in case of functional testing or black box testing here we are concerned about the internal structure that means the body of the functions and the body of the code.

And based on that knowledge we design the test cases one example is white box testing which is a kind of structural testing.

(Refer Slide Time: 32:09)



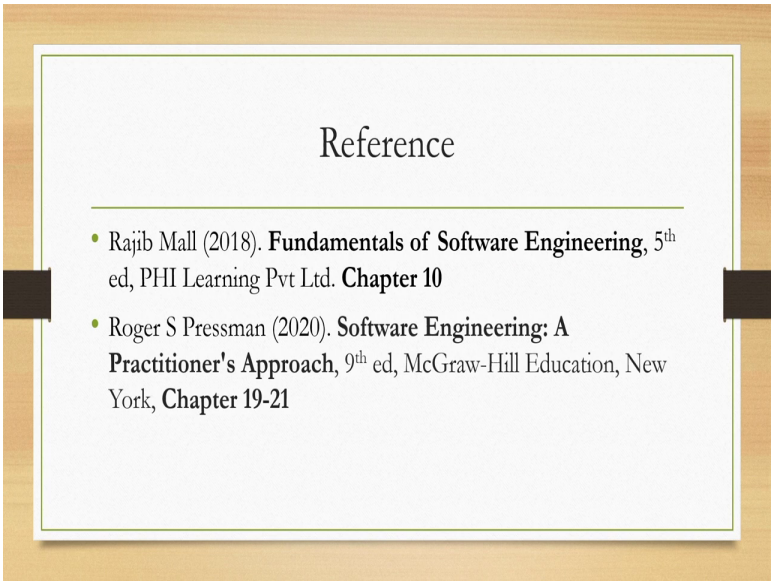
So, in summary we go for testing to uncover errors present in the code testing can be done in broadly two ways. One is a quick and cost effective way although it is not very rigorous or formal way that is the review based testing which also can be done in either of the two ways namely inspection based testing and code walkthrough. These are similar to the heuristic evaluation and cognitive walkthrough methods that we discussed in the context of prototype evaluation.

The other testing method is the execution based testing which is more formal and rigorous testing method with the objective of identifying more and more errors. Now in that case we go for actual execution of the system. Now there are broadly two ways to do that one is functional

testing one is structural testing. In functional testing we assume that the functions are black boxes and only the input and output matters accordingly we designed the test cases.

In case of structural testing we design test cases based on the internal structure of the code an example of functional testing is black box testing. An example of structural testing is white box testing. In subsequent lectures we shall learn in details about all these testing techniques namely review based testing black box testing and white box testing.

(Refer Slide Time: 33:54)



Whatever we are discussing in this part of the course you can find from this book fundamentals of software engineering chapter 10 or software engineering practitioners approach chapters 19-21. Of course there are the things are discussed in much more details than what we are discussing here but those are useful references where you can enhance your knowledge. I hope you understood the concepts and enjoyed the topics.

We will continue our discussion on testing in the next lecture looking forward to meet you soon in the next lecture thank you and good bye.