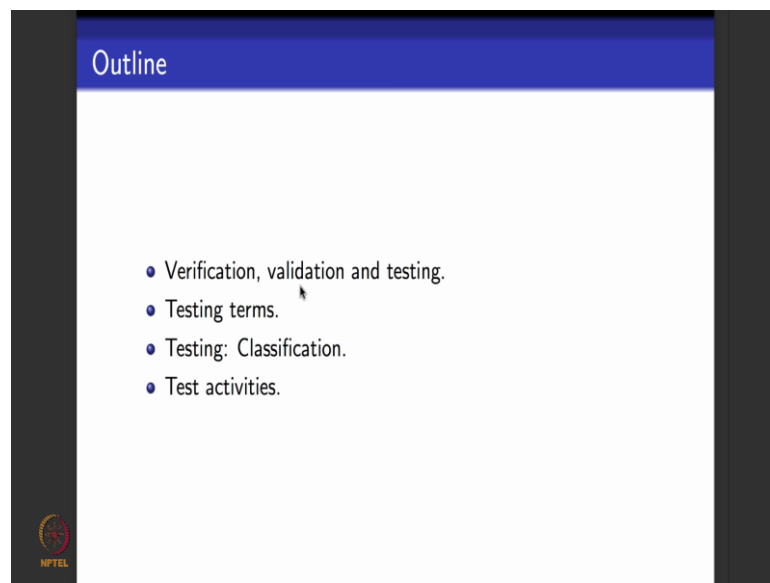


**Software Testing**  
**Prof. Meenakshi D'Souza**  
**Department of Computer Science and Engineering**  
**International Institute of Information Technology, Bangalore**

**Lecture – 02**  
**Software Testing: Terminologies**

Hello everyone. So, this is a first module of the first week. So, in this module what I will be doing is to introduce you to the various terminologies that exist in the area of software testing. What we would be using in this course - clarify the basic terms and tell you what are the various types of testing, the various methods in testing. We will also look at various activities in testing and see what this course will deal with. Here is an outline of the course.

(Refer Slide Time: 00:39)

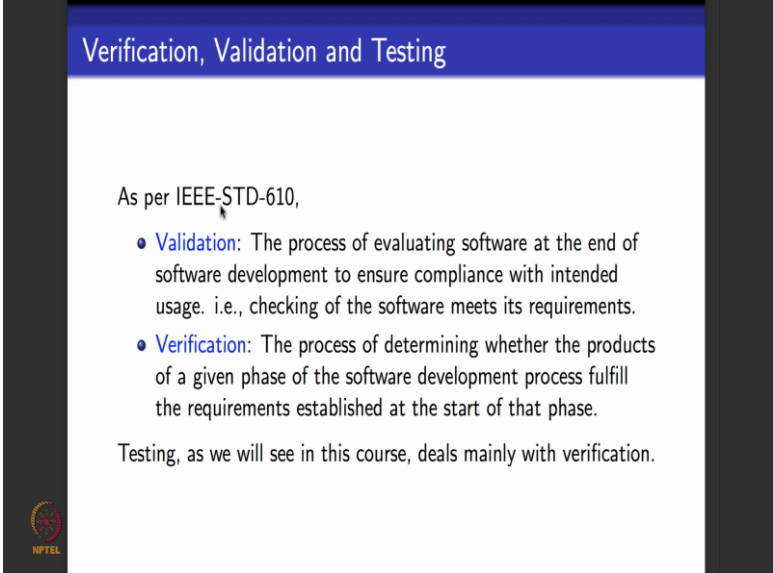


So, to begin with we clarify on these terms which you might have heard about verification, validation and testing and I will tell you what we will deal with in this course. We will also look at a whole set of related areas briefly and clarify what this course will cover. There are several terms and terminologies related to testing. What is the test case? What is an error? What is a fault? What is a failure? So, I will introduce you to all these testing terms and we will see what they mean.

And then in testing there are several methods of testing, several types of testing, several phases of testing. You might have heard terms like white box, black box, functional

testing, usability testing, performance testing. So, we will see what these various terms are in the classification of testing. I will end this module by introducing you to various testing activities and tell you what our course is going to focus on.

(Refer Slide Time: 01:36)




Verification, Validation and Testing

As per IEEE-STD-610,

- **Validation:** The process of evaluating software at the end of software development to ensure compliance with intended usage. i.e., checking of the software meets its requirements.
- **Verification:** The process of determining whether the products of a given phase of the software development process fulfill the requirements established at the start of that phase.

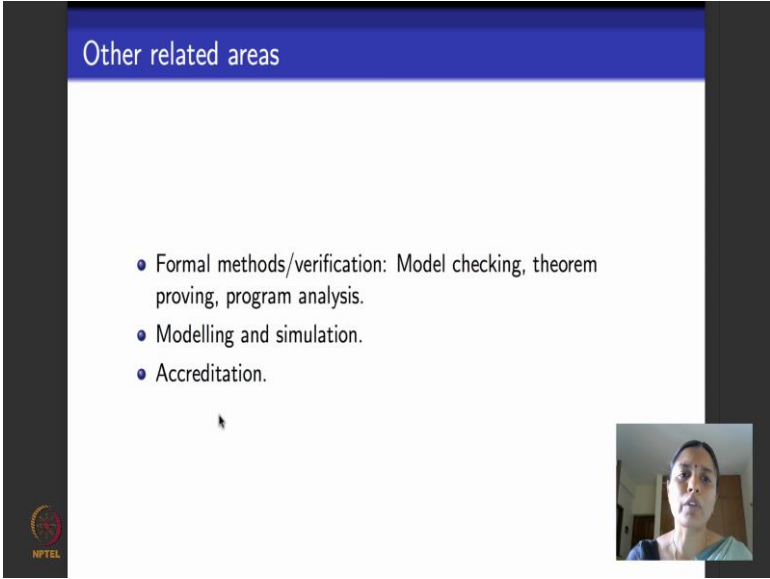
Testing, as we will see in this course, deals mainly with verification.



IEEE maintains a standard glossary of software engineering terms in this particular standard called STD-610. So, as per that standard what is verification and what is validation? Validation is something that you do at the end of software development or system development. You may have your own software or system developed ready in place and you want to be able to check if the software or system meets all its requirements when you do that at the end of the development then that is what is called validation. Verification deals with what you test or verify while developing software. So, while developing software you might go through various phases, phase where you define requirements, phase where you do design and architecture. Phase where write code and phase where you test. At each stage you are dealing with a set of software artifacts.

So, the kind of verification that you do where in you check whether a particular software artifact needs the requirements that were established for that particular set of artifacts then you do what is called verification. Testing as we will see in this course mainly deals with verification. There are several other related areas that you might have heard about.

(Refer Slide Time: 02:46)



The slide has a blue header with the text "Other related areas". Below the header, there is a bulleted list of three items: "Formal methods/verification: Model checking, theorem proving, program analysis.", "Modelling and simulation.", and "Accreditation.". In the bottom right corner of the slide, there is a small video inset showing a person speaking. The NPTEL logo is visible in the bottom left corner of the slide frame.

- Formal methods/verification: Model checking, theorem proving, program analysis.
- Modelling and simulation.
- Accreditation.

One of the most popular or my favorite ones is that of formal methods, formal verification. There are 3 broad areas that people work on within formal verification namely model checking, theorem proving, program analysis. Testing as we will see in this course, we will not deal with model checking or theorem proving or program analysis. Towards the end of the course we will see symbolic execution. A lot of symbolic execution is used in program analysis, but we will see it from the point of view of testing.

In fact, there are several NPTEL courses available in each of these areas and if interested you are more than welcome to go and see them.

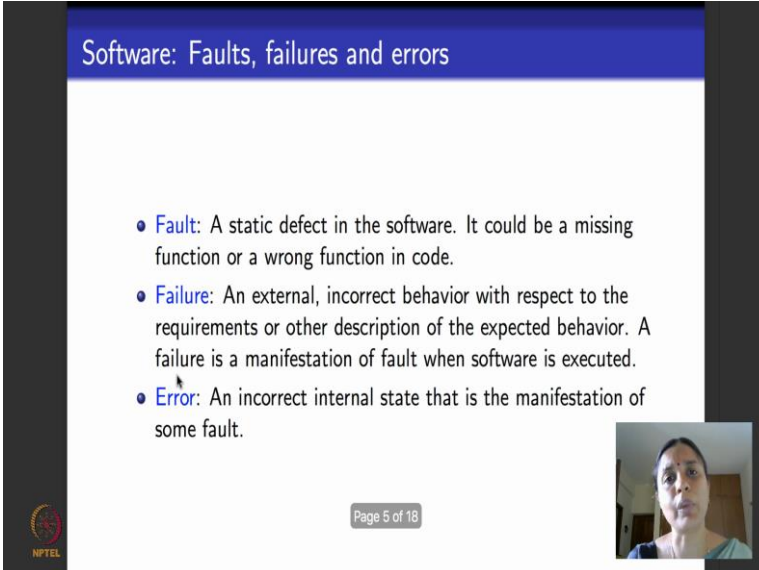
A related area where people also do verification is what is called modeling and simulation. Let us see if you have a piece of hardware design or a system design or a software design. For hardware design popular languages that you could use a VHDL very low, so you have your design and then you model your design in this particular language and you run simulations to see if the design model that you have done correctly does the design as per its requirements. So, you have a modeling language, which is usually proprietary or open source you modeled using that modeling language and run several different kinds of simulations to see if the design model needs its requirements.

So, that area is modeling in simulation and the testing that we would see in this course we will not deal with that also. Another related area is what is called accreditation. Now

what is accreditation? Accreditation deals with software that is safety critical and needs to be certified. Take for example, software that runs a plane, right, like typical autopilot software, any company you cannot say that I have written in autopilot software as per the normal requirements of autopilot software. So, here it is, take and run right. So, these autopilot softwares being safety critical because they cannot afford to fail they have to be audited and accredited by responsible authorities like FAA in the US, DGCA in India and they demand lot of additional testing there is a separate process of accreditation for which there is testing needs to be done.

But accreditation as it is, we won't look at, in this course what we will look at is the kind of testing that people do towards accreditation and the algorithm behind that kind of testing.

(Refer Slide Time: 05:02)



The slide is titled "Software: Faults, failures and errors" in a blue header. It contains three bullet points defining the terms. In the bottom right corner, there is a small video inset showing a woman speaking. The NPTEL logo is in the bottom left, and a page number "Page 5 of 18" is in the bottom center.

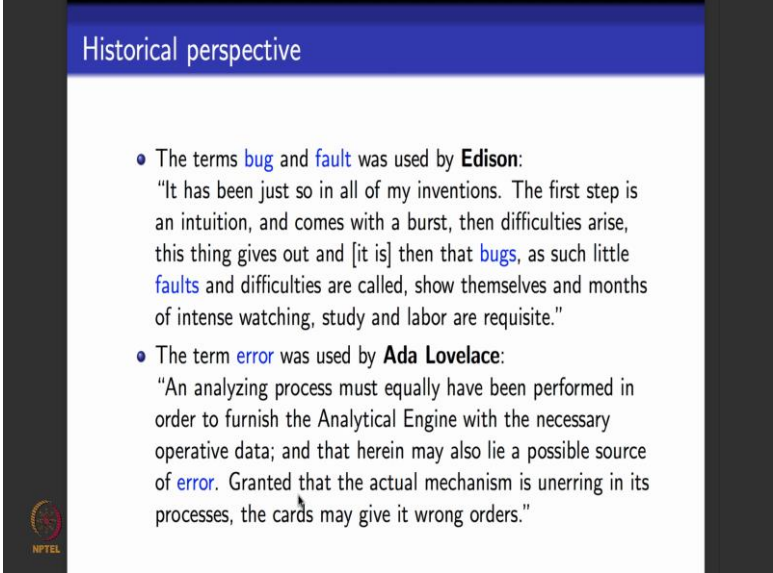
- **Fault:** A static defect in the software. It could be a missing function or a wrong function in code.
- **Failure:** An external, incorrect behavior with respect to the requirements or other description of the expected behavior. A failure is a manifestation of fault when software is executed.
- **Error:** An incorrect internal state that is the manifestation of some fault.

Page 5 of 18

We will move on to looking at various terms in testing, you might have heard of lot of these terms faults, failures, errors ... what do they mean? So, what is a fault? Fault is considered to be a static defect that occurs in software, a static defect in the sense the defect that occurs as a part of the software, not a defect that occurs when the software is executed. So, this defect could be a function that was wrongly written, function that was not written at all, it could be any of these, right and then when software with the static defect or fault is executed, there is a misbehavior or a wrong behavior on software that is observed, right. This wrong, observable behavior is what is called failure. When the

software has a fault to execute around the software you observe a failure, then the software is supposed to enter an incorrect state. The incorrect state in which software enters, the software that has a fault enters and a failure is manifested is what is called the error in the software.

(Refer Slide Time: 06:07)



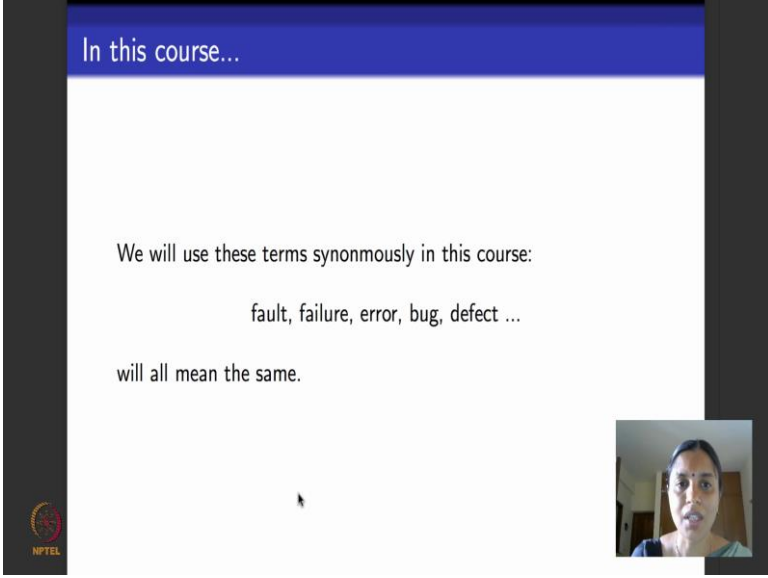
The slide has a blue header with the text "Historical perspective". Below the header, there are two bullet points. The first bullet point discusses the terms "bug" and "fault" as used by Edison, with a quote about his inventions. The second bullet point discusses the term "error" as used by Ada Lovelace, with a quote about the Analytical Engine. In the bottom left corner, there is a small NPTEL logo.

Historical perspective

- The terms **bug** and **fault** was used by **Edison**:  
"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise, this thing gives out and [it is] then that **bugs**, as such little **faults** and difficulties are called, show themselves and months of intense watching, study and labor are requisite."
- The term **error** was used by **Ada Lovelace**:  
"An analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of **error**. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders."

This is a small piece of history thanks to the book by Ammann and Offutt software testing. So, they connected the term bug and fault to Edison. So, apparently Edison use these terms first. So, he talks about his inventions and he says there is a lot of excitement when you initially invent something and then comes a bug or a fault, which he calls difficulties and he says they show themselves and it takes sometimes lot of effort to fix them. The term error is credited for its first use, to the first programmer who could have heard of namely the lady Ada Lovelace. So, when she was programming using Charles Babbage's analytical engine and punch cards, she says that, there could be an error. So, this is considered the first use of error and she says error could make the cards give wrong orders.

(Refer Slide Time: 07:00)



In this course...

We will use these terms synonymously in this course:

fault, failure, error, bug, defect ...

will all mean the same.

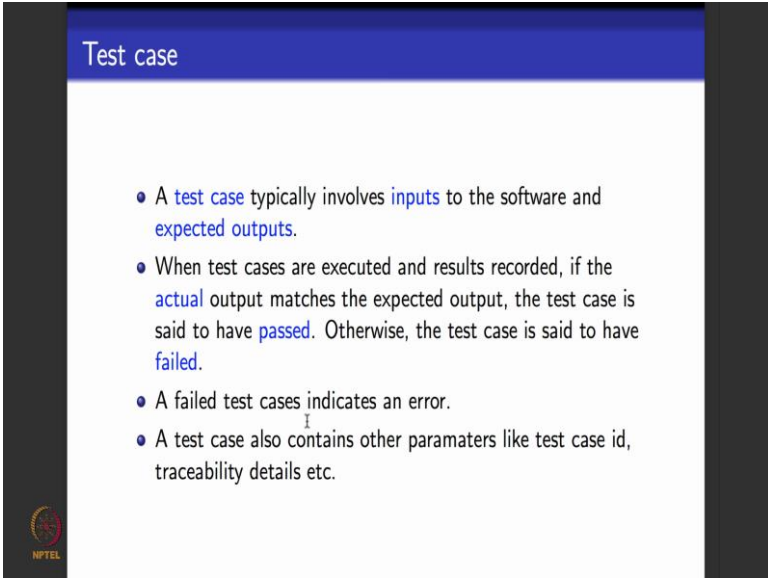
NPTEL

A small video inset in the bottom right corner shows a woman with dark hair, wearing a blue top, speaking.

As far as this course is concerned, we will use all these terms synonymously--- fault, failure, error, bug, defect they will all be used interchangeably and synonymously, they will all mean that they something wrong in the software artifact that I am testing.

Another important term that you need to know and get a clarified right now that we will use throughout the course in testing is when we say testing the first thing that we have to do is to write test case. What is a test case?

(Refer Slide Time: 07:26)



Test case

- A test case typically involves inputs to the software and expected outputs.
- When test cases are executed and results recorded, if the actual output matches the expected output, the test case is said to have passed. Otherwise, the test case is said to have failed.
- A failed test cases indicates an error.
- A test case also contains other paramaters like test case id, traceability details etc.

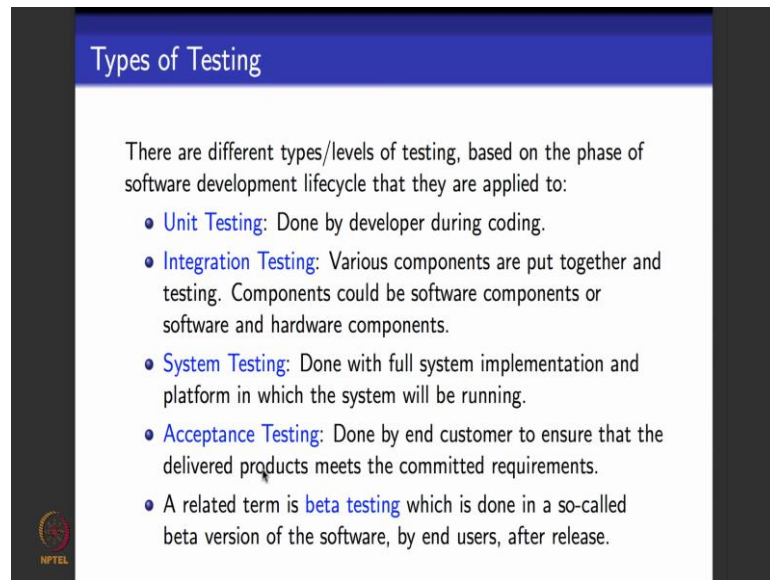
NPTEL

Test case typically has 2 main parts, it gives inputs to the software artifact that is concerned and it also says when I give this input to the software and execute the software on that, what is my expected output. So, test case has an input and what is the expected output, it is to be noted that if a test software artifact like code I always give only inputs to the codes and the part of the test case and be typically do not give values to internal variables to output variables and so on.

So, now I give this test case give inputs to the software and run the software, the software produces the output right. So, the output produced by software is what is called actual output. So, now, what we do typically, we check the actual output, compare the actual output to expected output and say whether they match or not. If the actual output matches the expected output, then you say the test cases said to have passed. If the actual output differs from the expected output then typically in testing this says that the test case has failed. A failed test case indicates that there is a fault in the software and the fault has manifested itself by resulting in an error, as per the terminologies that we saw.

Typically test case also contains things like an ID because you need to track and record the test case and people also maintain traceability information. What is traceability for a test case? Traceability tells you what are you testing this particular software artifact for. You might be testing a particular functionality in a piece of code. Who tells you to test for such functionality? Where did that functionality come from? That functionality could have come from a set of requirements. So, if it came from a set of requirements, which are the requirements that it comes from? So, all these information is maintained in what is called traceability matrix, traceability data which is also a part of the test case.

(Refer Slide Time: 09:34)



The slide is titled "Types of Testing" in a blue header. Below the header, it states: "There are different types/levels of testing, based on the phase of software development lifecycle that they are applied to:". This is followed by a bulleted list of five types of testing, each with a brief description. The NPTEL logo is visible in the bottom left corner of the slide.

- **Unit Testing:** Done by developer during coding.
- **Integration Testing:** Various components are put together and testing. Components could be software components or software and hardware components.
- **System Testing:** Done with full system implementation and platform in which the system will be running.
- **Acceptance Testing:** Done by end customer to ensure that the delivered products meets the committed requirements.
- A related term is **beta testing** which is done in a so-called beta version of the software, by end users, after release.

Now, we will move on to looking at the various terms that are popular in testing and trying to gain clarity on what is what. So, when I do testing, I do testing throughout software development and at each stage of software development I could do a different kind of testing. So, there are various levels of testing based on which phase of software development that I am testing on. The right first testing that I will do is called unit testing, unit testing is typically done by a developer. When he or she writes the code, they debug the code, they test the code then and there to see if the code is working fine as for as its requirements of concern.

These days because of the stress on agile methodologies developers are expected to do a lot of unit testing themselves, they do not really have the luxury to pass things to a tester and say that you do it, right?

After unit testing we typically do what is called integration testing. So, what is integration testing? A developer again can do integration testing. So, let us see a developer has written several modules of code and then they are trying to put together the code, what do you mean by putting together the code? Maybe the code is in different methods in particular method could call another method, a procedure could call another procedure. So, when you test these focusing on these calls, these function calls, procedure calls when you test the interfaces that occur between the various modules in code then you do what is called software integration testing.



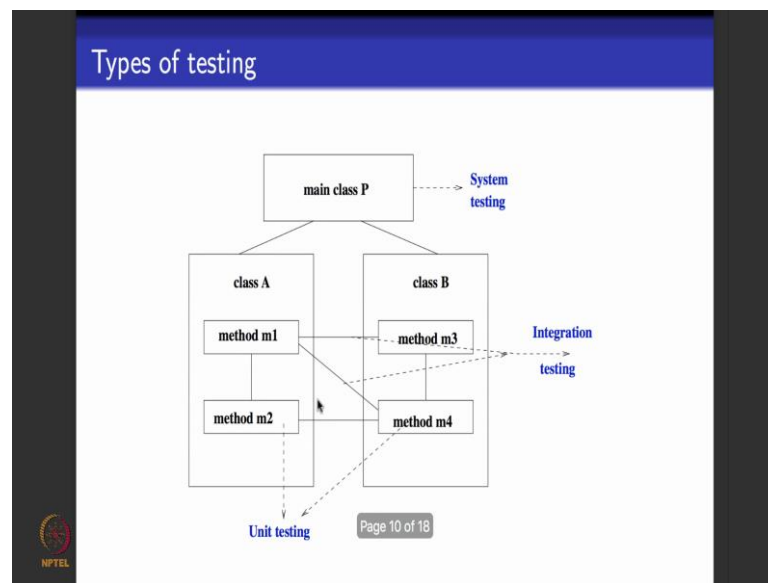
There is also another terminate integration testing, where people take the software as it is written and put it on the hardware that is supposed to work on. Let us say you have a fairly large server right you put it on the server that is supposed to work on you supposed to have integrated the hardware within the software. So, when you integrate the hardware with the software and test the integrated piece of hardware and software that is also called integration testing. So, post integration testing, the next step that people do is what is called system testing. So, in system testing the entire system is put together. Let us say take the case of an enterprise software, you will put together the server, the database, the web interface, the application server, the clients and all the interfaces between them and then you will test the end to end system from where the inputs come, what happens in the central system, what are the kind of main decisions, that are taken and how do they result in the output being produced.

In embedded software typically like a car or a plane you put the software as a part of the main system, which could be within the car or within the plane, the inputs will come from sensors the main control algorithms will run, from by picking up the sensor and puts from the bus and then they will produce actuator output. So, this end to end software as a part of the system with the inputs and outputs integrated from their original sources, the entire system when we tested we call it system testing. After system testing we believed that the system is more or less ready for reduce.

The last phase of testing the people do is what is called acceptance testing. Here you give the system to an end user and check if the system is working fine as per the end user committed requirements that the software or the system was supposed to meet. You might have also heard of a term called beta testing. What is beta testing? In beta testing people specifically release what is called beta version of software. When they release a beta version of software they roughly mean that this software is working fine, but I may not have tested it for mitigating all the involved risks.

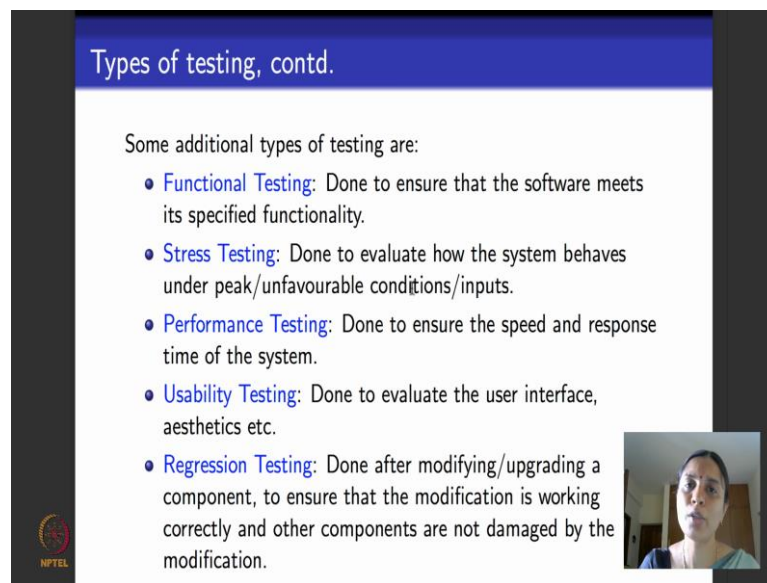
So, they are asking the users to start using the software and let them know if it has any bugs or defects. So, when you do this kind of testing then you are doing what is called beta testing right.

(Refer Slide Time: 13:27)



So, let us take an example to understand is a little better. So, let us say there is a main class called class P, which in turn has 2 classes A and B. Let us say class A has 2 methods m1 and m2, class B also has 2 methods m3 and m4. Let us see a particular developer is working on writing code for method m2 or m1 or m3--- just that method. So, when the developer finished his writing code or while writing code the kind of testing that the developer would do to test the functionality of just that method is what is called unit testing. So, now, if you look at this picture method m1 happens to call methods m3 and m4, m3 intern calls method m4. So, when a developer puts together these 2 classes all their methods and tests features specific to these calls, then the developer is doing what is called software integration testing.

(Refer Slide Time: 14:41)



The slide is titled "Types of testing, contd." and lists five additional types of testing. It includes an NPTEL logo in the bottom left corner and a small video inset of a woman in the bottom right corner.

Types of testing, contd.

Some additional types of testing are:

- **Functional Testing:** Done to ensure that the software meets its specified functionality.
- **Stress Testing:** Done to evaluate how the system behaves under peak/unfavourable conditions/inputs.
- **Performance Testing:** Done to ensure the speed and response time of the system.
- **Usability Testing:** Done to evaluate the user interface, aesthetics etc.
- **Regression Testing:** Done after modifying/upgrading a component, to ensure that the modification is working correctly and other components are not damaged by the modification.

Let us say this after this integration testing, the developer puts together the entire system and tests at the level of the main class B then that phase of testing is what is called system testing. You might have heard of all these additional terms in testing: functional testing, stress testing or load testing, performance testing, usability testing. What are they we will see them now. So, what is functional testing? Functional testing is typically done to ensure that the software meets its specified functionality. What do we mean by this? Let us take a software that runs as a part of an ATM machine of a bank right. What do you think is the core functionality of an ATM software? Core functionality of the ATM software could be the following. Once the user enters the password and the pin, if the credentials match and if there is sufficient balance left in the users account then, the amount that he or she has requested to be withdrawn should be provided to the user correctly.

So, ATM is meant to do just this and when I test a software to check if it needs its main intended functionality then I do what is called functional testing. Another kind of testing that is popularly done is what is called stress testing. So, here what happens is that the system is meant to be up and running in available lot of times and sometimes the system experiences what is called peak input conditions. For example, recently the class 12 marks were released right? So, they were released on the internet through a web application software. So, this system will experience what is called stress levels of input within 3-4 hours of the time in the date of release when all the students would want to

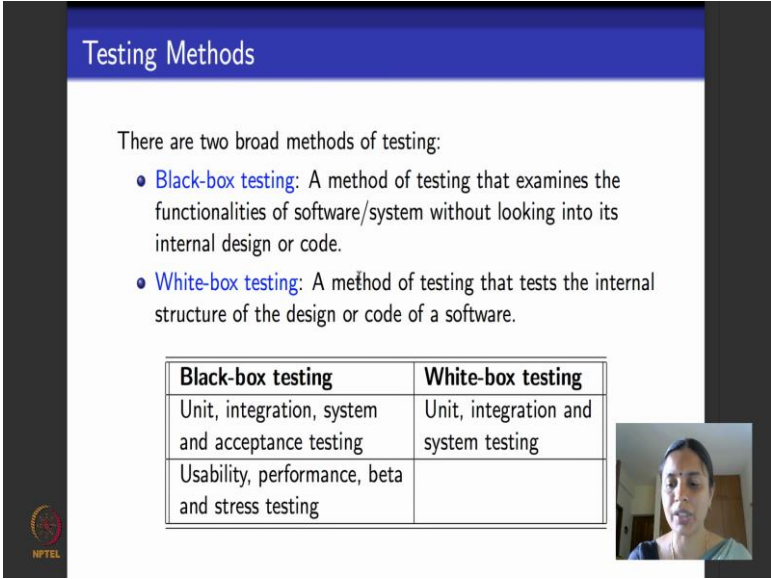
login and check what their marks are. So, when you test a system under this peak load conditions or input conditions then you do what is called stress testing.

The next kind of testing that you might have heard about is what is called performance testing. So, what will happens here is that here, people do testing to ensure that the system meets it is desired response time. It is fast enough, like for example, when we insert the card into an ATM machine, we want the ATM machine to be able to respond with the welcome screen almost immediately right, we cannot afford to wait for 10 seconds or minute or so. Performance testing specifies performance requirements on the system and checks whether all these performance requirements in terms of latency, speed, response, time, etcetera are met by the system.

Next popular kind of testing is what is called usability testing. It is not only useful to have a great software system, but the great software system is meant to be usable by a user which means the software should have a good user interface. It could be graphical or not, but the software should have a good user interface that is friendly. It should also be accessible, accessible in the sense by someone who is visually impaired, hearing impaired. It should be accessible. So, testing that is done to ensure that a software is usable, has a good interface, meets all the accessibility guidelines, is aesthetic enough---the kind of testing that is done to do all this is what is called usability testing.

Finally, one important term in testing before we move on is what is called regression testing. So, when does regression testing happen? So, let us see you have developed piece of software and usually released it. So, post release there is either change to the software that you do or you add a new functionality to the software. Now it is obvious that you might want it first check if the change or the new functionality that you have added is working fine and the second thing that you might want to do is that this change or the new functionality does not affect any of the other features that they are already present in the software before this was done. The kind of testing that is done to ensure that the software is working fine after modifying or upgrading it is what is called regression testing.

(Refer Slide Time: 18:45)



**Testing Methods**

There are two broad methods of testing:

- **Black-box testing:** A method of testing that examines the functionalities of software/system without looking into its internal design or code.
- **White-box testing:** A method of testing that tests the internal structure of the design or code of a software.

Black-box testing	White-box testing
Unit, integration, system and acceptance testing	Unit, integration and system testing
Usability, performance, beta and stress testing	

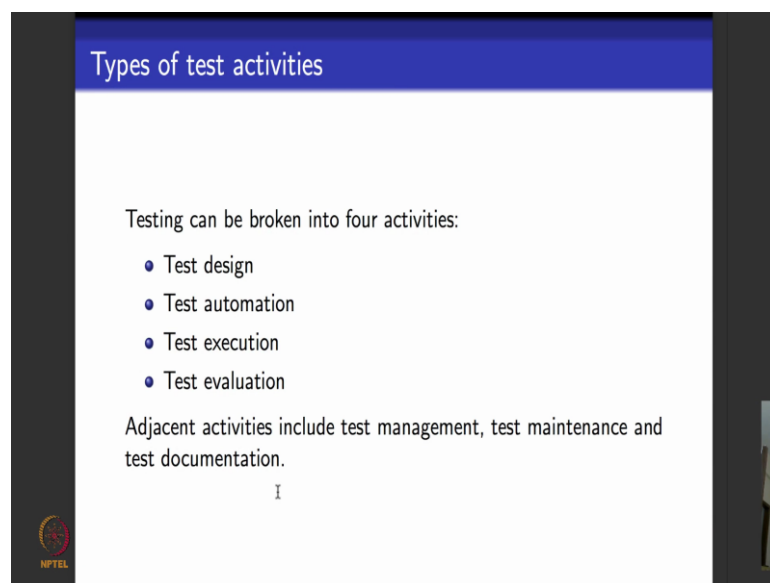
So, when all these methods of testing, all the various types of testing basically, 2 different methods in testing, you might have heard of these terms Black box testing and White box testing. So, what happens in black box testing? When you do black box testing you consider the software artifact that is being tested as a black box, which means you do not look inside it like, when you test the code you consider the entire code as a black box give inputs to the code execute the code and observe the outputs to check if the expected outputs match the actual outputs.

White box testing is the exact opposite. When I am testing a piece of code, with reference to white box testing I look into the code and I test for requirements like, suppose there is an if statement in the code can you write a test case that will cover the if statement. What is it mean to cover in the if statement? It means that can you write a test case which will execute the then part of the if statement ones which will execute the else part of the if statement once. So, similarly, when a white box test code I could insist that you cover loops in the code. So, let us say there is a while loop in the code, covering the loop would mean that you write test cases to skip the loop, to execute the loop in normal operations and execute the loop on boundary conditions.

So, in summary what does white box testing do? White box testing actually looks at the code, looks at the design at the corresponding software artifacts, looks at structure what it what is it have, what are the kind of statement, what are the kind of design elements it

has and then it tells you how to test it by looking at its structure. So, black box testing applies to almost all phases of testing, all types of testing it applies throughout the development life cycle, it applies for doing usability testing, performance testing, stress testing and so on. White box testing is typically done only during software development. Once software is ready put together the system is tested. Later on when we test for other non functional requirements like performance, stress, response time, good user interface, being accessible, we typically do not test it, as a white box testing then we consider the system as a black box and then test for all these quality features in the software right.

(Refer Slide Time: 21:12)

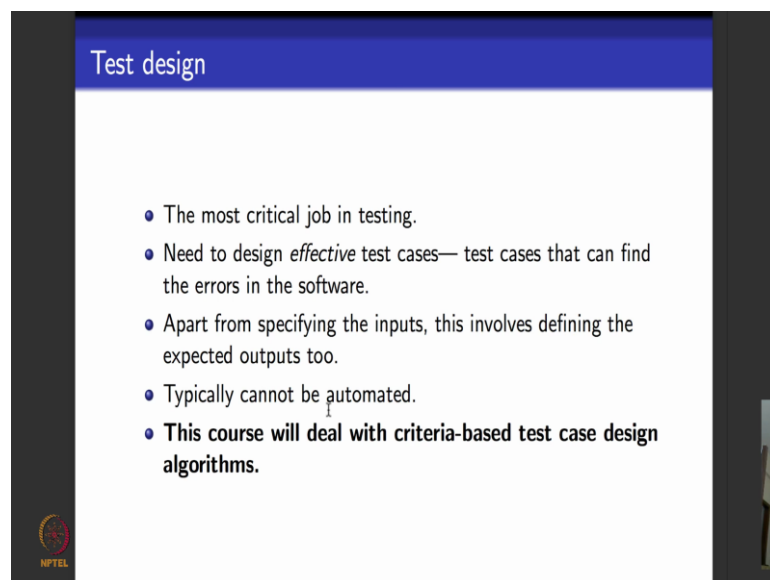


Now, we will move on to the last module, where we look at the various activities in testing. There are typically 4 raw activities in testing. So, we begin with design in cases because, if you do not have a test case then you cannot execute it and you cannot find errors in the software. The first thing is to be able to define or design a test case, once we design a test case I have to make it have to make it execution ready, that process is what is called test automation. After I make it execution ready, I can use a tool to actually executed and record the results. After I record the results then, I do what is called evaluation or analysis to check whether the test case is passed or failed and if it is failed and if there is an error where the error is and so on. These are the core technical activities and testing.

Apart from this there are several other umbrella activities. Like you have project management in software development, project management specific to testing is what is called test management. So, there are test managers to design test teams try to organize them.

Another important activity that happens is a part of testing is what is called test maintenance and test documentation. A lot of testing especially in enterprise software domain relies on reusing test , extensive number of test cases are reused again and again to be able to text software. So, how do I reuse a test case I should be able to document it and store it well to be able to make it amenable for use.

(Refer Slide Time: 22:59)



The slide is titled "Test design" in a blue header. It contains a bulleted list of five points. The first four points are in blue, and the fifth point is in bold black. The NPTEL logo is in the bottom left corner.

- The most critical job in testing.
- Need to design *effective* test cases— test cases that can find the errors in the software.
- Apart from specifying the inputs, this involves defining the expected outputs too.
- Typically cannot be automated.
- **This course will deal with criteria-based test case design algorithms.**

So, the kind of activities people do to ensure the test cases are well documented and well maintained and available for reuse as an when they are needed broadly constitute test maintenance and documentation.

So, now moving on to test case design this is considered the most critical job in testing. Why is it considered the most critical job in testing? Because, the Pareto principle applies to testing which means roughly 80 percent of the errors, a lot of the errors is focused on a very small percentage of the code. So, if I do not design my test cases effective what do you mean by effective in turn, if my test cases are effective then they will find errors faster, they will find all the errors or most of the errors areas that are

present in software, there is no point in saying that I designed so many test cases and I tested them for days and days and your software is doing fine right.

So, the effectiveness of test case design is in finding errors right and this needs domain knowledge. This needs knowledge about the system, about how it is developed and typically cannot be alternative. A lot of this course will deal with algorithms and testing techniques, that we will use to design test cases. I will give you more details about the precise kind of algorithms and artifacts that we will use in the next module.

After you designed your test case the next comes the converting this test cases in to executable script. So, you might have a test execution framework you could use JUnit, you could use Selenium. These are open source tools that will let you do test execution. So, you have to be able to make your test case ready for execution please remember, I told you that test case always talks about Inputs. So, suppose there is a requirement white box testing requirement, which says that you go to a while loop which is somewhere deep inside my code and you test for coverage of that while loop. It might so happen that the while loop has got no input variables, it directly deals with internal variables. Now it is up to the tester to be able to design test cases and give test cases input values such that this while statement is reached; not only it is this while statement reached this while statement is also covered under various coverage requirements.

So, how does a tester go about doing this? Software has to meet 2 important criteria called observability and controllability. We will look at these 2 concepts in the next module in this week. So, after your design your test case and it is ready for execution then comes the job of actually executing it and recording the results, this typically is almost always fully automated. There are several open source and proprietary tools that can do this really well for you.

After you have executed the test case, again, you need human intervention to be able to evaluate the result of the test case. So, let us say the test case is passed everything is fine, but if the test cases failed that indicates that the software is in an error condition. So, where exactly is the error? which is the erroneous components? Especially when you are doing system testing or integration testing, your software or system might be fairly large. It takes some amount of domain knowledge experience and human effort to be able to



isolate the fault and facilitate the development team to be able to debug and test it once again.

So, we would deal mainly with test case design algorithms in this course. So, in the next module I will tell you what are the algorithms are and what are the mathematical models modelling software artifacts that we would deal with in this course.

Thank you.