

Software Testing
Professor Meenakshi D'Souza
Department of Computer Science and Engineering
Indian Institute of Technology, Bangalore
Software Testing: Terminologies

Hello everyone, welcome back, we are still doing week one, I am going to use this module and the one that immediately follows this to tell you about certain standard glossaries of terms and terminologies that are used in software testing as a standard practice across the world. So, we will use two modules of lectures, this one and the next one to understand all software testing terminologies for this course and those that are standard in nature.

(Refer Slide Time: 0:52)

The screenshot shows a presentation slide with a blue header bar containing the word 'Outline'. Below the header, there is a bulleted list of topics. In the bottom right corner, there is a small video inset showing a woman, presumably the professor, speaking. The slide also features a logo of the Indian Institute of Technology Madras in the top right corner and navigation controls at the bottom.

- Verification, validation and testing.
- Testing terms.
- Testing: Classification.
- Test activities.

So, as an outline for this module, you might have heard of terms like Verification, Validation, and Testing, I will explain the difference between them, I will tell you what are all the testing terms that we will use, what are the types of testing as a classification and what are the typical activities that we will pursue during testing.

(Refer Slide Time: 1:18)

The screenshot shows a presentation slide with a blue header bar. In the top right corner is the IIT Madras logo and the text 'IIT Madras B.S. Degree'. The main text on the slide is as follows:

Software testing is the act of examining the artifacts and the behavior of the software under test by validation and verification.

Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.

Definition courtesy: Wikipedia.

At the bottom of the slide, there is a navigation bar with five items: 'Outline', 'Verification, Validation', 'Testing terms', 'Testing, Classification', and 'Test Activities'. Each item has a progress indicator below it. The 'Verification, Validation' item is currently selected, indicated by a black dot. To the right of the navigation bar is a small video feed showing a woman with dark hair wearing an orange patterned top.

So, this is the definition of software testing that I borrowed from Wikipedia it says, Software testing is the act of examining the artifacts, which is your code, your design, architecture documents, and requirements documents those are artifacts, you examine those artifacts and the behavior of the software when these artifacts are executed and test them by validation and verification.

We will see what these latter two terms are, what is the goal of software testing? It provides an objective independent view of the software and checks whether it follows the appropriate business capabilities that it is meant to satisfy and also is used as a means to evaluate the risks that are involved in code corresponding to the software.

(Refer Slide Time: 2:14)

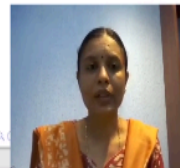
Verification, Validation and Testing



As per IEEE-STD-610, the IEEE Standard Glossary of Software Engineering Terminology,

- **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.



So, let us move on, I told you in the last class lot of IEEE standards apply for software engineering and software testing also, there is an IEEE standard glossary of software engineering terminology called STD-610 series, so IEEE STD-610 as per that glossary here are the definition of the terms Verification and Validation.

Validation is the process of evaluating software at the very end of the software development activity to ensure that it is compliant to whatever standards it is supposed to meet and compliant to the requirements that it was written for and it is meant and works correctly for its intended usage. So, validation is the process of evaluating software at the end of the software development stage to ensure compliance with intended usage for requirements specifically.

Verification, on the other hand is the process of determining whether the products produced throughout the phases of software development life cycle, each phase has a set of work products as we saw in the last lecture, with these products fulfill their requirements, their needs a process of checking for that is verification. A lot of the testing that we will do in this course will lean towards verification, towards the end we will talk about validation techniques also.

(Refer Slide Time: 3:51)

Other related areas

- Formal methods/verification: Model checking, theorem proving, program analysis.
- Modelling and simulation.
- Software quality assurance.
- Accreditation.
- Test driven development.

ITT Madras
B.E. Degree

Online 00 Verification, Validation 00 Testing terms 0000 Testing, Classification 0000 Test Activities 000000

A video inset shows a woman with dark hair, wearing an orange patterned top, speaking.

There are several other related areas, each one a subject meriting its own course as a standalone course that are related to software testing. One area that is very complementary to software testing but has similar goals to ensure correctness and to find bugs is that of formal methods of formal verification. It has three broad techniques model checking, theorem proving, and program analysis, there are several NPTEL and other courses in these that you could use to learn about them, we will not be doing formal methods in this course.

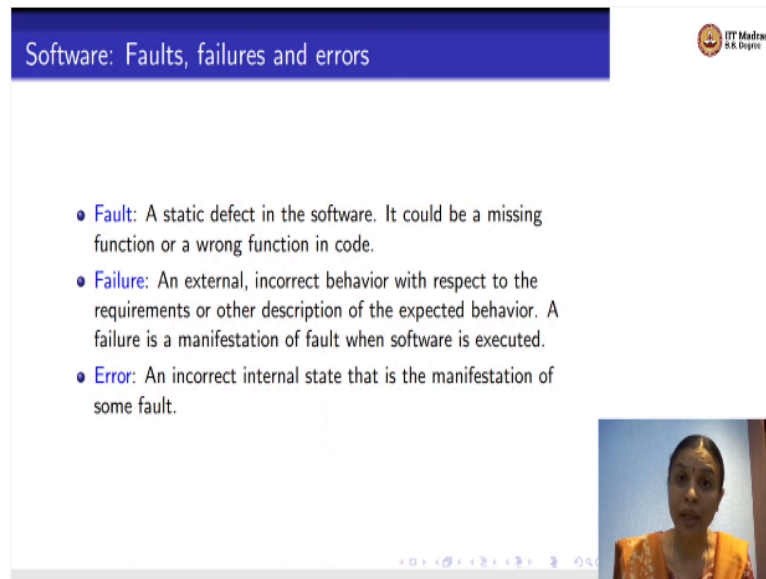
Another related area is to validate or verify design and architecture requirements using what we call modeling them by using modeling languages and running simulations on these models to check if they behave as we intend them to be, this is different from testing and this course will not cover modeling and simulation also.

An area that is very important for testing to succeed which goes along with testing is that of software quality assurance, I will briefly tell you what this is about some time in this course, another one is accreditation which is preparing software to meet certain standards of certification by a body that certifies the software which involves testing, we learn the techniques but we will not focus any other accreditation related activities in this course.

Agile software methodologies predominantly follow what we call test-driven development paradigm, even though this is not directly software testing, it is one important agile development

methodology that is related to software testing and promotes software testing, so we will see one lecture, we will devote one lecture for test-driven development in short TDD.

(Refer Slide Time: 5:47)



The slide has a blue header with the title "Software: Faults, failures and errors". In the top right corner, there is a logo for "IIT Madras 1959" and the text "IIT Madras 1959". The main content area is white and contains a bulleted list of three items:

- **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.

In the bottom right corner of the slide, there is a small video inset showing a woman with dark hair wearing an orange patterned top, looking towards the camera.

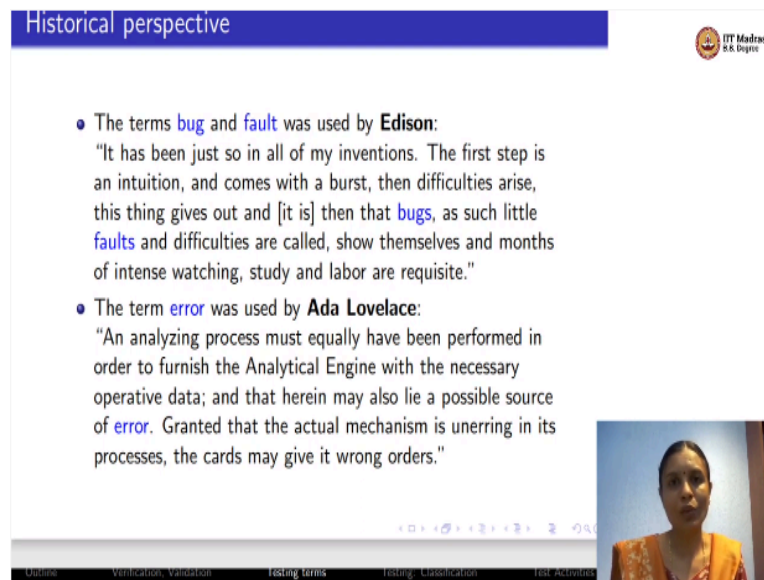
Let us move on and understand some testing terminologies. You must have heard in the first lecture I used the term error, you must have heard a synonymous term called defect, then you must have heard oh there is something faulty in my code or you must have heard saying this whole code is failing, this whole application is failing, so what are these terms, are they related, are they one and the same. So, let us see what the standard glossary has got to tell about them.

So, these are the definitions as per the standard glossary, a fault is a static defect in the software, which means that it is something that the developer has himself or herself missed, it could be a missing function or it could be a wrongly coded piece of code in the software, that is a fault in the software, when a fault occurs or it becomes visible during testing, it manifests itself as a failure, so a failure is an incorrect behavior of the fault that is visible to the external user, typically a tester or a developer, so a failure is an externally visible incorrect behavior with reference to some requirements that you are testing it for or some other expected behavior for your software, so a failure is an external manifestation of a fault.

The third one is an error, when a failure happens or when a fault is present the state of your program or your software artifact, the internal state of your program becomes incorrect, it enters

what we call an incorrect state, so when that happens then we say that there is an error in the software. So, these are the definitions as per the standard glossary.

(Refer Slide Time: 7:51)



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."


Few other terms bug, and defects all these are always there in the context of software testing. So, like last time I told you right testing is as old as coding is, how old it is, you will be surprised to know that people like Edison, the first programmer Ada Lovelace, they all talked about these terms, there is a quote by Edison where he uses the term bug and fault as follows.

So, he says it has just been so in all my inventions not related to software but more on inventions, the first step is an intuition and comes with a burst then difficulties arise and this thing gives out, and then that bugs, which he says are little faults and difficulties show themselves and it needs months of intense watching, study, and labor to fix them, right, so these are related to inventions failures and inventions but then they can also fail and then you have to rectify it.

Ada Lovelace who is believed to be the first programmer ever explicitly uses the term error, remember she used to do punch cards in Charles Babbage's analytical engine which is believed to be the first computer, so Ada Lovelace says an analyzing process must equally have been performed in order to furnish the analytical engine, the then computer, the first computer, with the necessary operative data and herein may also lie a possible source of error.

Simply speaking in current days terms what she is saying is that the program that you execute in a computer can have an error, granted that the actual mechanism is unnerving in its processes the cards may give it wrong orders, so errors are really old and software testing's goal is to find and fix them.



(Refer Slide Time: 9:45)



We will use these terms synonymously in this course:

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

will all mean the same.



Outline	Verification, Validation	Testing terms	Testing: Classification	Test Activities
00	00	000●	0000	000000

So, in this course even though we talked about this glossary of fault, failure, error, bug, and defect we will use all these terms synonymously, we would not really distinguish one from the other and in my lectures, you might hear one for the other, just remember that they all mean the same in this course, they all mean that there is something wrong with the software artifact that I am testing.

(Refer Slide Time: 10:13)

The slide is titled "Test case" in a blue header. It contains a list of four bullet points explaining the concept of a test case. In the bottom right corner, there is a small video inset showing a woman in an orange sari speaking. The slide also features a logo for "IIT Madras" in the top right corner and a navigation bar at the bottom.

- 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.

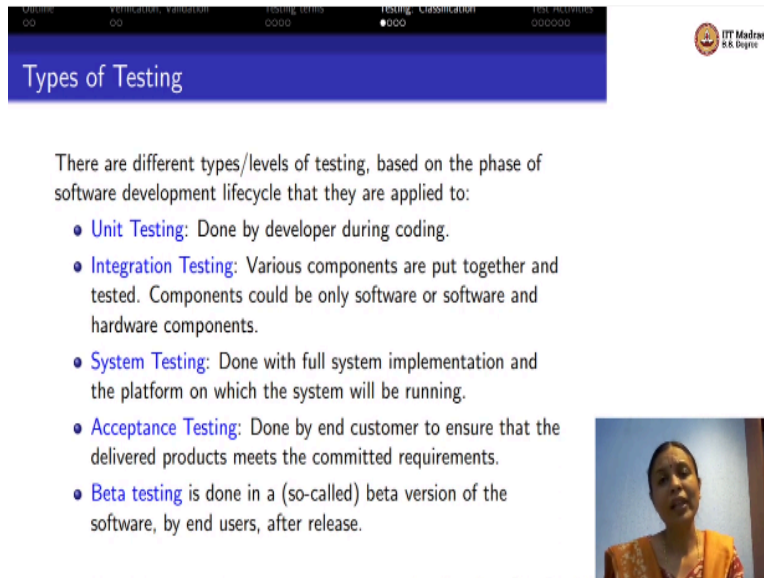
Moving on we know testing is all about writing test cases, write a test case, executing it, see what happens in the code. So, what is a test case, how does it look like, if you want to define what a test case is that is what is given here. A test case has two parts, two main parts, one is test case inputs which are given as the inputs to the software artifact that is being tested, you run the software artifact, typically you code on your inputs and then that one produces some outputs, so you take those outputs and you see if those outputs are right or not, how do you see that? You take those outputs, you call them actual outputs, and match it to what you have written out as expected outputs, if they match then you conclude that this test case has passed, if they do not match then you conclude that the test case has failed.

Failed test case could indicate an error, if your goal is to use failed test case to find an error. So, what is a test case? To repeat, a test case consisting of test inputs and expected outputs, after running a test case on your code if you match your expected outputs to the actual outputs produced by the code if they match then you say that the test case is passed, that is the outcome of the execution of the test case, otherwise you say that the test case is failed, a failed test case typically indicates an error.

Test case also has what we call test case ID and if you remember in the earlier lecture I told you about traceability, right, traceability basically says that what is this test case meant to test, this test case is meant to test such a such a requirement, so test case will also have traceability details

and id for retrieving it for later use and so on but from the purpose of learning algorithms for test case design you always have to remember just two things, test case consists of inputs and expected outputs.

(Refer Slide Time: 12:27)



Types of Testing

There are different types/levels of testing, based on the phase of software development lifecycle that they are applied to:

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

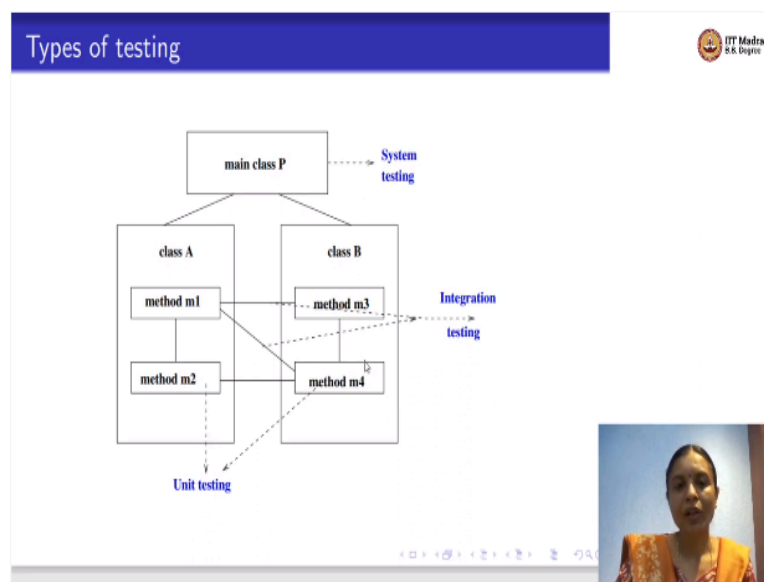
We will move on and understand various kinds of ways of classifying testing. So, the first way is types of testing as it goes down the phases of the software development life cycle what are the types of testing that are done. The first one is what we call unit testing, which I told you in the earlier module is done by the developer during coding itself, so it is a testing done by the programmer, programmer writes unit test cases, executes them on the code, checks whether the functionality that is written by him or her works correctly and if it does not, rewrites the functionality and ensures that the error is removed.

After unit testing comes integration testing where maybe two programmers wrote two different methods, they are putting together and testing if the two methods, one calling the other let us say work correctly along with the interface of calling, that is what is called integration testing, so this is software integration testing, when two software components are put together, at a later stage you could also do system level integration testing where you put the software with the hardware and the database, connect them to the server and then test.

So, after integration testing when the full system level implementation is ready, when you put it together on the platform and test it that is called system level testing, after system level testing is what we call acceptance testing which is typically done just before release or soon after release by the end customer to ensure that the delivered software product meets all the requirements that it was committed to be written for.

A related term is what we call beta testing, many times you would hear the term this is a beta release of a particular software, roughly it means that this is a release that is ready but it might have certain kinds of errors, if you happen to encounter one of them please report it back to us and we will strive to fix it. So, beta testing is done on what is called a beta version of the software, typically by end users soon after release.

(Refer Slide Time: 14:44)



So, I will illustrate these types of testing through a small example, I hope by now all of you are comfortable with reading a picture like this. So, this diagram says that there are three classes, there is a main class called P, which has two subclasses or child classes A and B and class A has two methods m1 and m2, and class B also has two methods m3 and m4.

Let us say us there are two developers, developer one is writing code for methods m1 and m2 in class A, developer two is writing code for methods m3 and m4 in class B, so when the developer tests whether he or she has written the functionality for m1 or m2 or m3 or m4, whether that has

been written correctly then that is what we call unit testing, a method is thought of as a unit of software and as a standalone entity the method is being developed and tested by the developer to ensure that it is working correctly then that is referred to as unit testing.

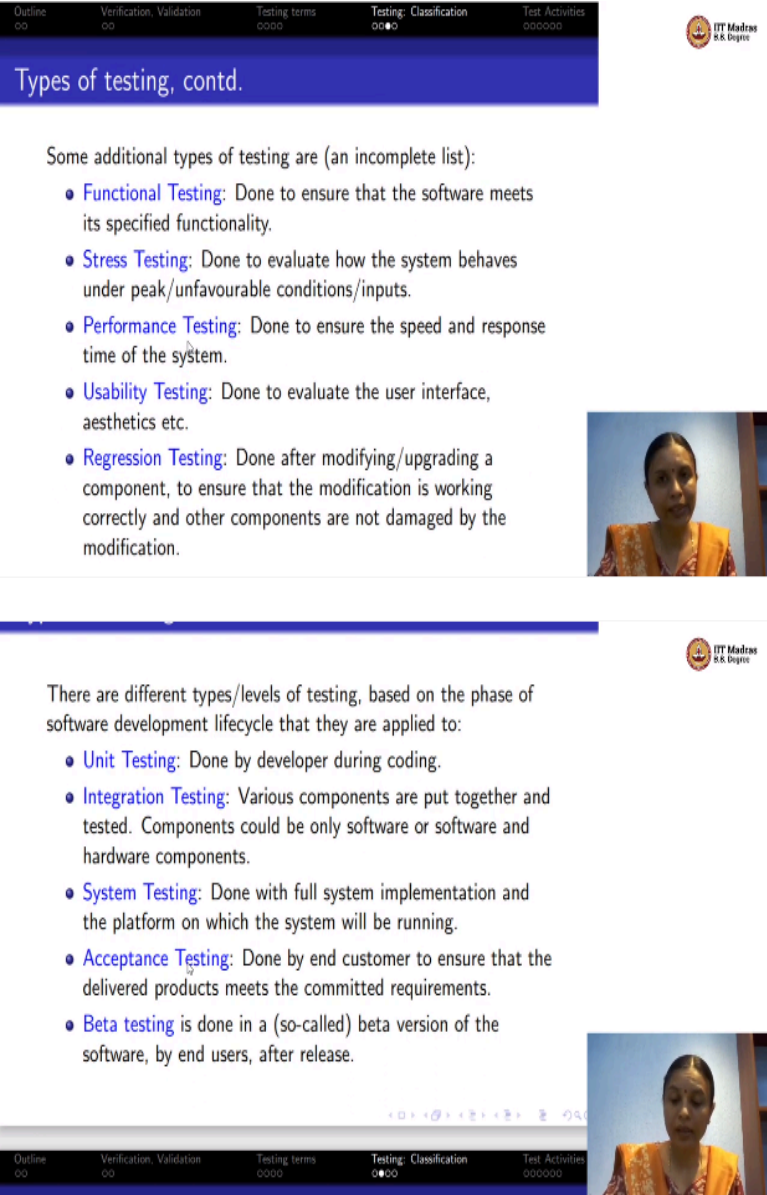
When the method interfaces, so if you look at this diagram method m1 calls m4, it also calls m3, method m3 in turn calls m4 again, method m2 also calls m4, so these are what we call interfaces or the call interfaces where one method calls the other. So, if you try to see method m1 put together with m4 do they together work correctly along with their interface, then you do what is called integration testing.

Similarly, if you check with the method m1 calling method m3 that is the interface between m1 and m3 is working correctly, then that is also called integration testing, that is the next stage after unit testing where two components, two different pieces of software components are put together and their interfaces are tested for correctness.

The next level above is you might want to test this whole class P, class P has this class A, class B, and child classes they have their respective methods but class P is meant to satisfy some functionality as per your design document. So, when you are testing the whole class P which includes the methods of classes A and classes B checking for its functionality to check whether it is fine, this is what we call system testing at a software level.

There is one more level of system testing that is not illustrated in this picture which talks about taking this whole piece of software, connecting it to the database, connecting it to the hardware, connecting it to the network and testing it at a system level, so that is also system testing but here I have focused only on the software side of the system testing. So, I hope you understand the types of testing after this.

(Refer Slide Time: 17:54)



The screenshot shows a presentation slide with a dark blue header bar containing navigation links: 'Outline', 'Verification, Validation', 'Testing terms', 'Testing: Classification', and 'Test Activities'. The 'Testing: Classification' link is highlighted. The slide title is 'Types of testing, contd.' in white text on a blue background. The main content area is white and contains the text 'Some additional types of testing are (an incomplete list):' followed by a bulleted list of five testing types. A small video feed of a woman in an orange patterned top is visible in the bottom right corner. The IIT Madras logo is in the top right corner. At the bottom of the slide, there is a navigation bar with the same links as the header, and a set of navigation icons.

Types of testing, contd.

Some additional types of testing are (an incomplete list):

- **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.

There are different types/levels of testing, based on the phase of software development lifecycle that they are applied to:

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

So, apart from these let me just go back by one slide apart from these which goes through the faces of SDLC towards the end you also test the software to see if it meets several quality requirements or not, so those are additional types of testing, what I have shown you here in this slide is a very incomplete list, as we move along in the course we will devote a couple of lectures to exclusively understand what these kinds of testing's are.

So, in addition to unit testing, integration testing, system testing, and acceptance testing, people also test the software for functionality, that is called functional testing, people test the software to

see if it meets its peak levels is stress testing, a classical example would be let us say class 10 results of a particular state board are coming out and you know for the first 2 hours every class 10 student, his or her parent and relative would be trying to access the system to check what the marks are, to check the results are, so for a certain period of time the software is subject to extreme peak load with several people accessing the software simultaneously, after a few hours, after a few days there may not be that high number of accesses.

So, a specific kind of testing done to meet, to evaluate if the software meets the requirements under peak conditions is what we call stress testing. Another kind of quality testing is performance testing, which is done to ensure that the software responds within a certain speed, see the minute we enter our credit card or debit card into an ATM machine we want the welcome screen and we want the menu, if it takes let us say even 10 seconds for it to load, we get restless, so performance testing tests for speed for the response time of software to see if it meets all the requirements.

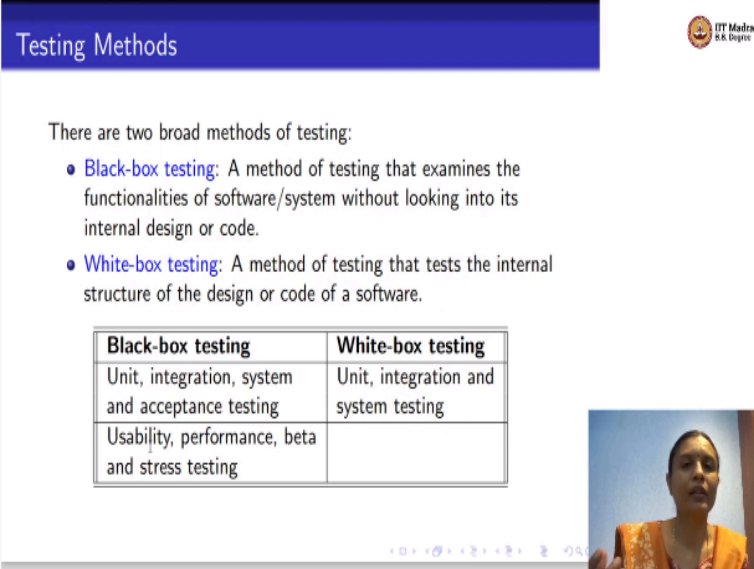
The next important kind of testing is usability testing to see if the software has a user interface that is usable by all, and if the software is aesthetic in terms of its availability to different kinds of users, a subset of usability testing is also called accessibility testing, like for example, we know that many of these, many of the software these days especially the web applications have what we call captcha, which is basically meant to check if a human being is accessing it.

So usually, they show a distorted image and you have to type the characters in the image or you have to recognize something about the image, but imagine a visually impaired person using the software, this part of the captcha will not be useful for them, so adjacent to the captcha logo there is usually an audio button which plays out a word or spells out a word that the visually impaired person can hear and type it and that's the captcha that validates the visually impaired person, so usability testing tests for all aspects of usability including this kind of accessibility.

Regression testing, I told you briefly in the last lecture, post maintenance, a lot of effort goes into maintaining and adding new features to the software, instead of testing the software all over again typically people reuse test cases and this whole process of reusing test case is a large area in testing called regression testing. So, we will devote one lecture to regression testing later, so

this is an incomplete list, there are so many other quality parameters that you can test for, later in the course we will talk a little more about these kinds of testing.

(Refer Slide Time: 21:41)



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	

The slide also features a small video inset in the bottom right corner showing a person in an orange shirt speaking.

Another classification on testing is the method that as a tester you would use for testing, you would have heard terms like Black box testing, White box testing, there is also something called Gray box testing which is between Black box and White box, so what are these? Black box testing is a method of testing where you examine your software artifact, let's say code without looking at its internal design or the actual code itself, you know the inputs to the software, you know the requirements of the software, you have the code as an executable artifact.

You design test cases purely based on inputs and the requirements that this software is supposed to meet and execute it on the code and see if the requirements are met or not, you do not look at the internals of the code, sometimes you may not even know how the code is written, what its design is. So, if you are testing by using the software artifact as a black box, that is without looking at its internals, testing purely based on its inputs and requirements then we call it black box testing.

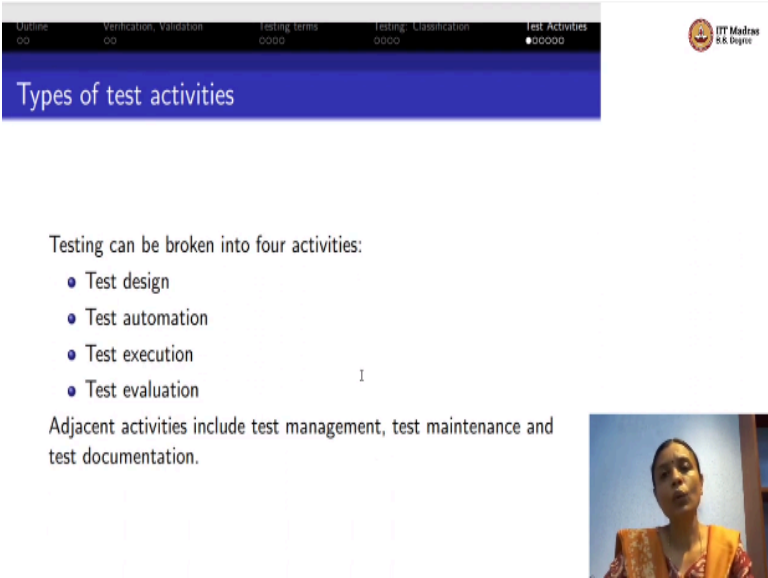
The other one is white box testing where you are as a tester aware of the structure of the code or the software artifact that you are testing, you know the language, you know its design, you know the classes, the methods, the hierarchies, all the details of the design and you design test cases by

looking into the structure of the code and then use them for testing that is called white box testing.

We will learn several techniques for black box and for white box testing in this course, if I now try to put the types of testing that we saw just a little while ago against the methods of testing that you see in this slide, this table will give you an overview of what is happening. So, unit integration system and acceptance testing can be black box or white box, not acceptance but unit, integration, and system testing can be both black box and white box.

When it comes to the other non-functional parameters, quality parameters most of them are black box, usability, performance, stress, load reliability, all of them are black box testing techniques. This part will become clear as we move on in the course you will be able to understand with better clarity what these differences are.

(Refer Slide Time: 24:20)



The slide is titled "Types of test activities" and features a progress bar at the top with five segments: "Define", "Verification, Validation", "Testing terms", "Testing, Classification", and "Test Activities". The "Test Activities" segment is highlighted with a blue background and white text. Below the progress bar, the text "Types of test activities" is displayed in white on a blue background. The main content of the slide lists four activities: "Test design", "Test automation", "Test execution", and "Test evaluation", each preceded by a blue circular bullet point. Below this list, it states "Adjacent activities include test management, test maintenance and test documentation." In the bottom right corner, there is a small video feed of a woman with dark hair wearing an orange and yellow patterned top, speaking.

Testing can be broken into four activities:

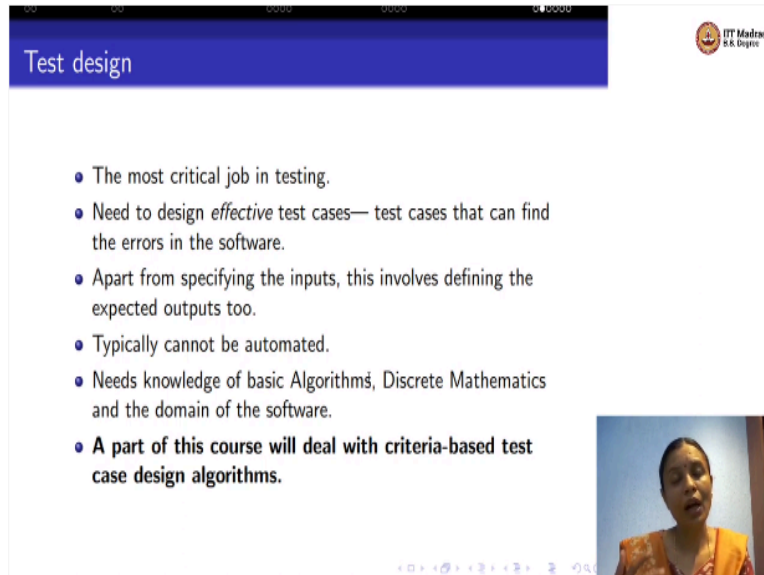
- Test design
- Test automation
- Test execution
- Test evaluation

Adjacent activities include test management, test maintenance and test documentation.

Now, we saw the types of testing, we saw briefly the methods of testing, now it is time to understand what are the typical activities that are pursued in testing. Broadly a tester throughout the life cycle of software development has four kinds of activities, first is to design test cases, which then is to actually automate, make them ready to be put in tool, the third one would be to execute these test cases and observe the result and the fourth one would be to make some inferences from the result and as we saw in the SDLC, there are lots of adjacent activities like,

like we have project managers, we have test managers, we have people who maintain regression, test case documentation that is needed for reuse later and so on.

(Refer Slide Time: 25:16)



The slide is titled "Test design" and features a list of six bullet points. The first five points describe the challenges and requirements of test design, while the sixth point states that the course will cover criteria-based test case design algorithms. A small video inset in the bottom right corner shows a person speaking.

- 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.
- Needs knowledge of basic Algorithms, Discrete Mathematics and the domain of the software.
- **A part of this course will deal with criteria-based test case design algorithms.**

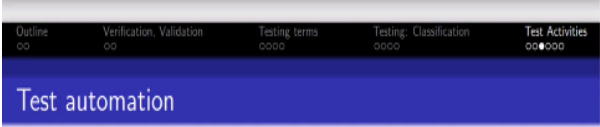
So, what is test case design, this is the most critical job in testing, and why is it important, because if I do not design effective test cases as a tester, then I might spend hours or even days doing testing and not find a single error. An effectively designed test case can catch defects or errors in the software reasonably fast, it is wrong to say that I spent several days and your software is so good that I could not find a single error, maybe it is true but many times the reason that people would give for such a thing would be that oh your test case has not been effectively designed.

So, it is important to know various ways or what are the right ways of designing test cases such that I can test a software for certain kind of feature, for a certain kind of coverage, or for finding certain kinds of errors, this part cannot be automated, needs human intervention, needs basic knowledge in computer science concepts like discrete maths and algorithms and a tester will also need to know the domain of the software that he or she is working on.

A testing domain like software for banking and finance sector is going to be different from testing software that works for the automobile industry, so domain knowledge, the basics of mathematics, and how to design effective test cases is what this critical phase of software testing

activity is going to be all about. A large part of this course is going to deal with teaching you algorithms for test case design.


(Refer Slide Time: 27:06)



The slide features a navigation bar at the top with five items: 'Outline', 'Verification, Validation', 'Testing terms', 'Testing: Classification', and 'Test Activities'. Each item has a progress indicator below it. The 'Test Activities' item is currently selected, indicated by a blue bar. To the right of the navigation bar is a small logo for 'IIT Madras'.

Test automation

- Involves converting the test cases into executable scripts.
- Need to specify how to reach *deep* parts of the code using just inputs: **Observability** and **Controllability**.
- Specific to the test execution framework or tool.
- **We will use a few open source test automation tools in this course.**



Once you design test cases you have to get them ready to be executed that process is what we call automation, testing automation involves converting the test cases into executable scripts. Typically, you might, for example, suppose you have a code which is 4000-5000 lines of code, quite standard to write a code like that.

But inside that your goal is to test if a particular method is working correctly or not, you have to first be able to reach that method and then you have to be able to test the correctness of that method using whatever way you want to and then if something is wrong you need to be able to observe it as a change in output. So, test automation evaluates how to prepare and reach the parts of the code or software that you want to test on and how to observe whether it has been erroneous or not and deals with concepts that we call as observability and controllability.

I will teach you these in the next week to follow what do we mean by this and throughout the course we will learn a few techniques that will help us in test automation along with test case design, we will see a few open-source test automation tools in this course.

(Refer Slide Time: 28:23)

[illegible]

Outline ○○	Verification, Validation ○○	Testing terms ○○○○	Testing: Classification ○○○○	Test Activities ●○○○○○
---------------	--------------------------------	-----------------------	---------------------------------	---------------------------

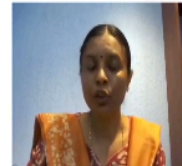
Types of test activities



Testing can be broken into four activities:

- Test design
- Test automation
- Test execution
- Test evaluation

Adjacent activities include test management, test maintenance and test documentation.



After automation comes execution, you need some entity a tool to be able to execute the test case, observe the results and see what is happening, that can be almost fully automated, there are several excellent open source and proprietary tools that are available if you happen to be working for a company after graduating, the company will most likely have proprietary or open source tools that they have standardized for use within themselves, in this course we will teach you how to use a few open source test automation and execution tools that you will use as you move on.

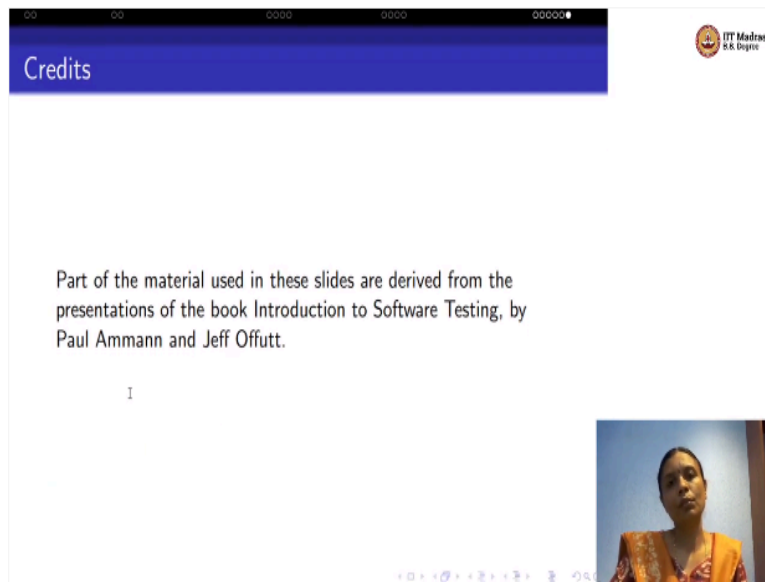
Finally, you have designed the test cases, you figured out how to package, you have executed them, and you have the results with you, how good is the result, does this indicate whether the software is correct or not, have I actually found the bug that I meant to find, is this test case passed or failed, how am I going to analyze it, all this is done in test evaluation this is again going to be very difficult to fully automate, you need manual help.

Manual help is also needed for another important reason, so let's say you have tested a piece of code as I told you, a class that contains several different methods, and let's say to put together there are more than 4000-5000 lines of code and you realize that something is wrong, there is an error but then where is the error, the code is pretty big, which method, which statement is erroneous this is the problem of isolating faults.

A fault has occurred how do I isolate it to a particular region in the code that needs experience, human knowledge, and human intervention, and cannot be automated. So, test evaluation is the

last step in software testing. So, just to go back briefly test case activities can be four parts, test case design, automation, execution, and evaluation. In this course, you will learn several techniques for test case design and use test automation tools to be able to execute them and maybe manually learn how to evaluate and isolate faults in these test cases.

(Refer Slide Time: 30:38)



All the material that I took in this course is taken from this textbook, there is an older version of the textbook freely available online if you would like to use them, thank you I will stop here for this module.