

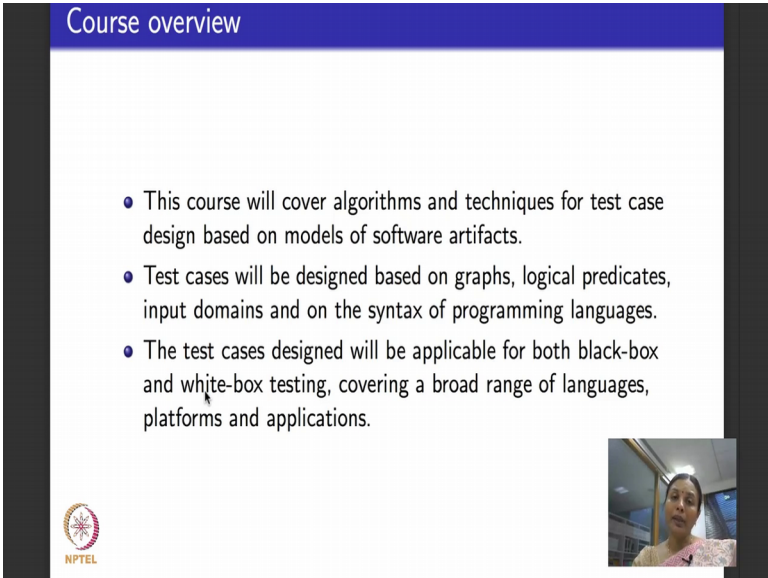
**Software testing**  
**Prof. Meenakshi D'Souza**  
**Department of Computer Science and Engineering**  
**International Institute of Information Technology, Bangalore**

**Lecture – 44**  
**Software Testing Course: Summary after Week 9**

Hello there, we are in the last lecture of week 9. What has we been doing for the past two weeks if you remember I have been doing syntax based testing or mutation based testing for the past two weeks. This sort of pretty much brings us to the end of the core of test case design algorithms that I wanted to teach you as a part of the course. At this point we are done with three-fourth of the course, we have three more weeks of lectures pending and it is a very good time to spend one video trying to look at what we learned from the beginning and where are we going to go from now.

So, that is what this lecture is going to capture, it is going to give you a summary of the course as it is after week 9.

(Refer Slide Time: 00:52)



Course overview

- This course will cover algorithms and techniques for test case design based on models of software artifacts.
- Test cases will be designed based on graphs, logical predicates, input domains and on the syntax of programming languages.
- The test cases designed will be applicable for both black-box and white-box testing, covering a broad range of languages, platforms and applications.

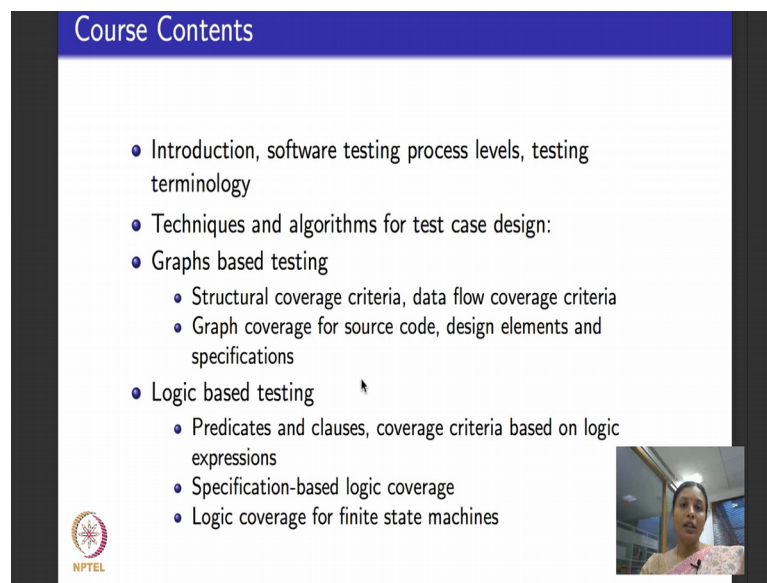
NPTEL

So, in the beginning when I started lecturing at the beginning of the semester what did I commit to. I told you this is exact replica of the slides that we had shared at that time, I told you that will cover algorithms and techniques for test case design how are we going to do it we going to take software artifact create models or structures out of them and based on these models or structures we are going to see techniques for test case design.

And then what are the structures that we see we saw graphs as models of artifacts, we saw logical expressions, then we only focused on the input space. And then finally, we focused on the syntax or grammar of the software artifacts.

The test cases that we designed can be used for black box when it is based on requirements and can be used for white box when it is based on source code.

(Refer Slide Time: 01:40)



**Course Contents**

- Introduction, software testing process levels, testing terminology
- Techniques and algorithms for test case design:
- Graphs based testing
  - Structural coverage criteria, data flow coverage criteria
  - Graph coverage for source code, design elements and specifications
- Logic based testing
  - Predicates and clauses, coverage criteria based on logic expressions
  - Specification-based logic coverage
  - Logic coverage for finite state machines

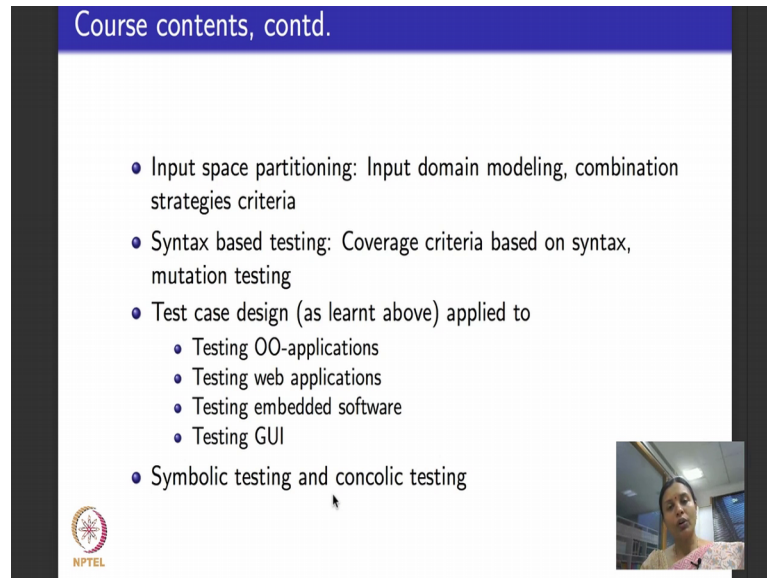
NPTEL

And then these were the contents of the course that I presented to you at the beginning of the lecture series right in the first week. So, we did begin by introducing what software testing is motivating it then I told you what are the process levels, what are the various set of (Refer Time: 01:57) terminologies that you will encounter in testing. After doing that initial bit the main part of the course the technical part of the course that we focused on was related to techniques and algorithms for test case design.

We did graph based testing we did two kinds of coverage criteria purely based on graphs - structural coverage criteria, data flow criteria along with this I gave you a good amount of basic graph traversal algorithms depth first search, breadth first search, strongly connected components topological sort and so on. And then what we did we took graphs for source code told you how to draw control flow graphs for various design entities we took graphs for design elements which were basically call graphs sequencing constrains and so on. We took graphs for specifications and then saw how to apply these coverage criteria on these graphs.

Then we moved on to a next structural model logic based testing. I introduced you to the basics of logic predicates clauses and then we looked at coverage criteria based on logic. We saw the most useful coverage criteria they have a predicate coverage clause coverage and active clause coverage then we applied logic based coverage criteria to source code for specifications and on finite state machines.

(Refer Slide Time: 03:11)



The slide, titled "Course contents, contd.", lists the following topics:

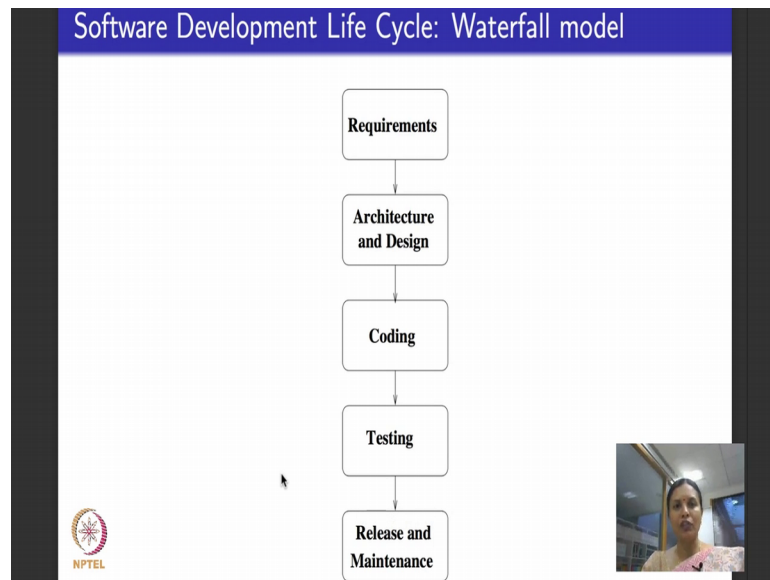
- Input space partitioning: Input domain modeling, combination strategies criteria
- Syntax based testing: Coverage criteria based on syntax, mutation testing
- Test case design (as learnt above) applied to
  - Testing OO-applications
  - Testing web applications
  - Testing embedded software
  - Testing GUI
- Symbolic testing and concolic testing

The slide also features the NPTEL logo in the bottom left corner and a small video inset of the presenter in the bottom right corner.

We moved on we looked at black box testing so to say, there we focused only on the inputs and outputs the space of inputs and then we modeled the inputs and wrote test cases for coverage criteria based on various characteristics of the input domain. Then in the past two weeks I have been focusing on syntax based testing which focuses on the underlying grammar of the programming language or the software artifact mutates or makes changes to that and writes test cases to kill the mutants.

Then the course in the beginning as I committed to you I said you learn how to test object oriented applications how to test web applications embedded software GUI graphical user interface with or without using these criteria, will see the popular state of the art issue mainly like a survey of each of these categories. Finally, I would like to end the course by presenting symbolic and concolic testing to you.

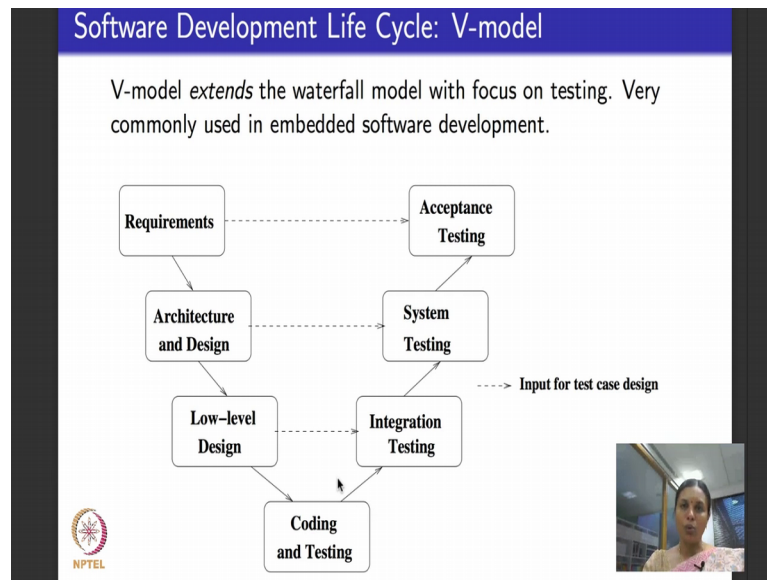
(Refer Slide Time: 04:09)



So, these were the course contents exactly as I had given you in the beginning of the course. Now I would like to spend some time looking back and thinking about what we have done how does it relate to software development. Here is a popular model of software development that is still predominantly used even in this agile development world it is called the waterfall model good old software development lifecycle model begins with requirements, goes on to do architecture and design, then people write code they also unit test their code then they put the things together and do integration testing system testing, collectively called as testing phase waterfall does not really distinguish between the kind of testing that you do. Finally, after the software is ready its released and then maintained.

In the waterfall model you assume that requirements a baseline before you do architecture and design you finalize the architecture and design before you write coding and when you have begun testing your code is more or less ready. There is not much going back up and down back and forth that does not happen.

(Refer Slide Time: 05:11)



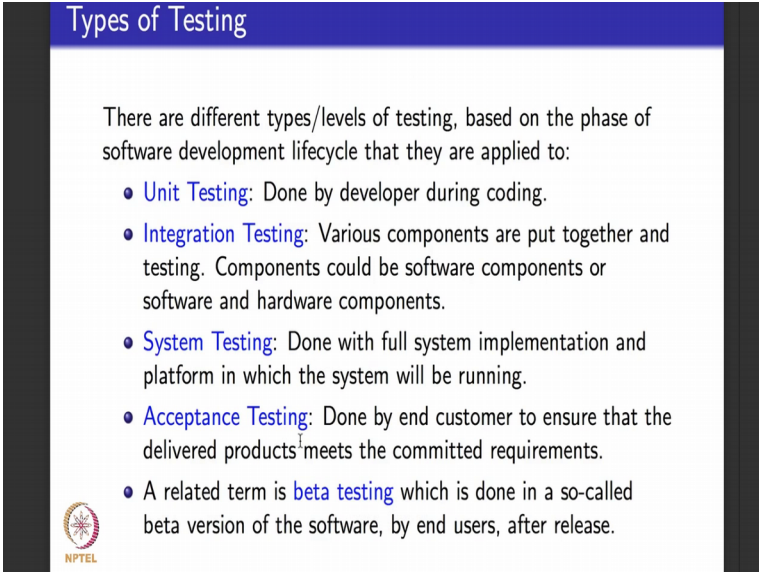
Testing as we saw in this course is predominantly related to another software development lifecycle model called V-model which is what is given in this cycle in this slide. So, what is V-model do it takes the waterfall model as it exists here and sort of bends it to look at the other side, it focuses into testing c splits the testing to see what are the various stages of testing that will go does not really particularly focus about release and maintenance its of course, there, but the model that depicts the level development lifecycle let us go this part release and maintenance. So, takes requirements architecture and design coding, takes testing and elaborates testing to let it span across other phases this is how it looks like. So, it always begins with requirements like in waterfall model followed by architecture in design again like in waterfall model, followed by I put this extra term where we model tries to focus on called low level design.

So, here we do system level design, here we design into the system components each individual things is almost like code then people do coding along with coding they do unit testing off the code after that they do integration testing where they put together the software components and test it, and after they put together and tested the software components they put the hardware along with the software and then do integration testing and then they put the whole thing in this system and do system testing. Finally, when the system is ready they do what is called acceptance testing we have seen all these terms before. What is the V-model say? Let us look at these dashed lines with arrows in the V-model.

It says that integration testing is related to low level design, system testing is related to architecture and design, acceptance testing is related to requirements. What is the legend for this dashed arrow? It says input for test case design that is if you want to design test cases for integration testing where you get data or inputs to design your test cases from you get it from the low level design. Similarly if you want to design test cases for system level testing the source set of software artifacts that give you inputs to design your test cases come from architecture and design documents similarly for acceptance testing which is mainly black box testing the source design for test cases come from your requirements doc.

What did we focus on in this course? We focused on the V-model, in particular we focused on how to use these various inputs from various software artifacts across the development lifecycle to be able to design test cases that we could use for coding and unit testing, we could use for integration testing, system testing or acceptance testing.


(Refer Slide Time: 07:58)



**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 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.

 NPTEL

This is again what I told you right in the first week is just a recap these were the various types of testing we discussed unit testing and integration testing just now and after that come system testing and acceptance testing which we are all discussed as a part of the V-model we also discussed what is called beta testing.


(Refer Slide Time: 08:15)

**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.

<b>Black-box testing</b>	<b>White-box testing</b>
Unit, integration, system and acceptance testing	Unit, integration and system testing
Usability, performance, beta and stress testing	



And I also told you at the beginning of the course that good old testing terminology says that there are two broad methods of testing black box and white box. Black box does not look into the structure of the design of the core assumes the design or the code to be a black box designs test case is purely based on inputs and requirements white box designs test cases by exploiting the structure of the design or the code. So, black box testing applies for unit testing integration testing system and acceptance testing. White box testing also applies at unit testing integration testing and systems testing phases in addition a whole set of other testing usability, performance, load, beta testing, stress testing, they are all black box testing techniques.

Back in the V-model they happen here at system testing or acceptance testing test cases for these kind of entities which test for non functional qualities of a software come from architecture and requirements and design. We did not look at any of those in this course I will teach you, tell you a bit of GUI but not the rest.

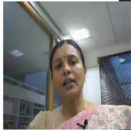

(Refer Slide Time: 09:20)

### Types of test activities

Testing can be broken into four activities:

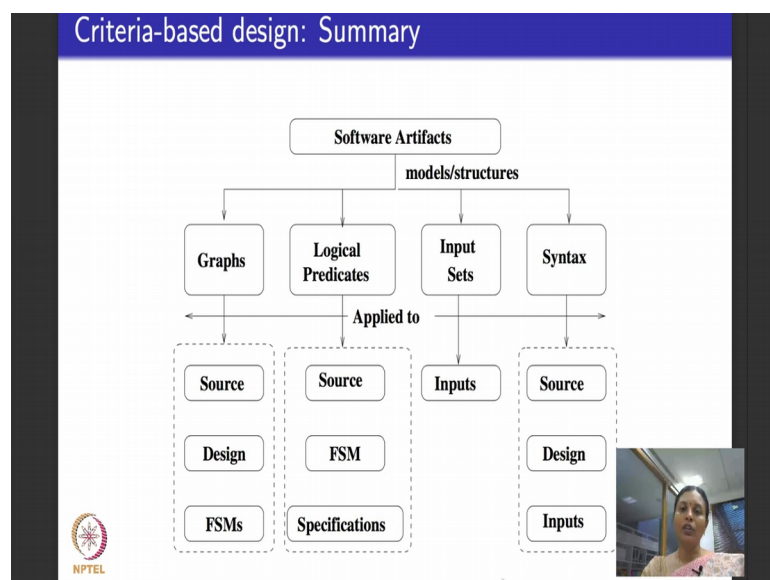
- Test design: Criteria-based on graphs, logical predicates, inputs and grammar.
- Test automation: JUnit.
- Test execution
- Test evaluation

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



We also discussed types of test activities just begin with test case design followed by automation, execution and evaluation. What did we do in the course as far as types of activities are concerned we focused a lot on test case design basically criteria based test case design and I told you how to do automation in execution using the tool called j unit which applies for java programs.

(Refer Slide Time: 09:44)



This slide gives a very good summary of what we have done exactly till now in the course. So, we took software artifacts which could be source code integrated source code

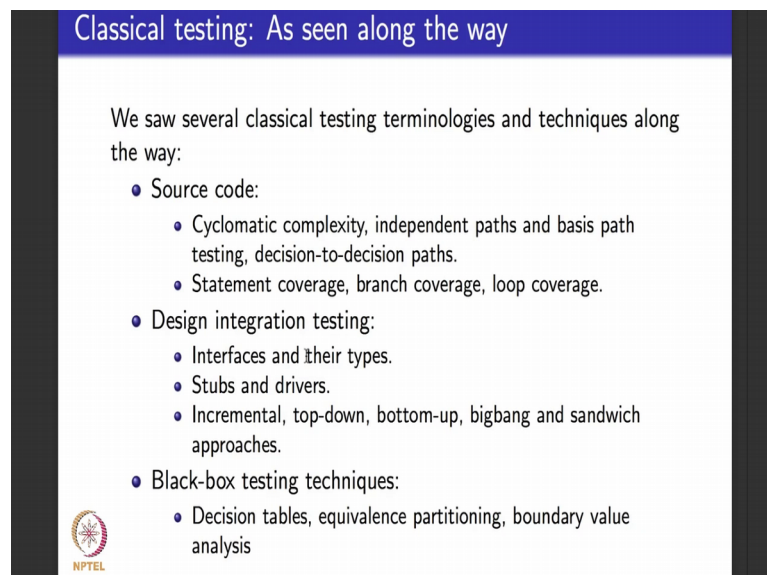


which is large code that includes all the modules put together methods put together classes put together, we took design elements, we took requirements either as documents or as finite state machines, then we took inputs details about inputs then we chose four different kinds of model or structures. We modeled these software artifacts as graphs, defined coverage criteria over graphs two kinds structural and data and then so how to apply it to source code to design elements and to finite state machines

After that we took logical predicates which come from decision statements in the source code and from requirements or guards in finite state machines and define coverage criteria on logical predicates clause, predicate, combinatorial coverage, active and inactive coverage criteria, we applied it again to test source code finite state machine and specifications. Then we focused on black box testing only on input spaces, define criteria that partition the input spaces and applied to test the software artifact based on the inputs.

Finally, we did mutation testing which was syntax based testing we saw how to apply to source code both within a method and at the software integration level, we saw how to apply it to design integration specifically we saw one module on object oriented design integration, finally, we saw how to apply to grammars for inputs.

(Refer Slide Time: 11:24)



The slide features a blue header with the text "Classical testing: As seen along the way". Below the header, the text reads "We saw several classical testing terminologies and techniques along the way:". This is followed by a bulleted list of three main categories: "Source code:", "Design integration testing:", and "Black-box testing techniques:". Each category contains sub-bullets. In the bottom left corner, there is a small circular logo with a starburst pattern and the text "NPTEL" below it.

Classical testing: As seen along the way

We saw several classical testing terminologies and techniques along the way:

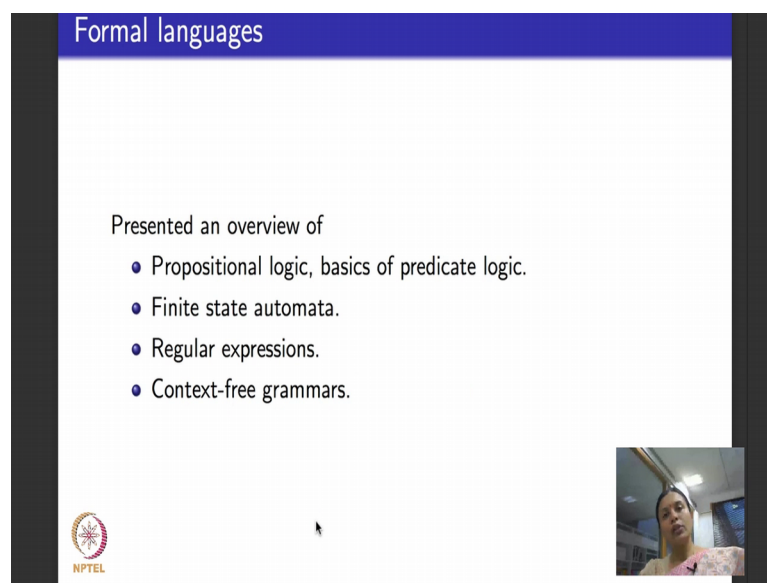
- Source code:
  - Cyclomatic complexity, independent paths and basis path testing, decision-to-decision paths.
  - Statement coverage, branch coverage, loop coverage.
- Design integration testing:
  - Interfaces and their types.
  - Stubs and drivers.
  - Incremental, top-down, bottom-up, bigbang and sandwich approaches.
- Black-box testing techniques:
  - Decision tables, equivalence partitioning, boundary value analysis

NPTEL

Along with this we also saw lot of the classical terms and testing that you would encounter in any old test textbooks related to source code these are the terminologies that I introduced you to cyclomatic complexity, independent paths, basis path, testing

decision to decision paths. I also told you in the context of white box coverage statement coverage branch coverage loop coverage, we saw them as node coverage edge coverage and prime path coverage. Then when we focusing on design integration testing we learnt about interfaces, types of interfaces, message passing shared, variable and so on. We learnt about what stubs and drivers are when it comes to integration testing we learnt various approaches to integration testing incremental top down bottom up big bang and sandwich. When it comes to black box testing techniques we learnt about equivalence partitioning, decision tables and boundary value analysis.

(Refer Slide Time: 12:17)



Formal languages

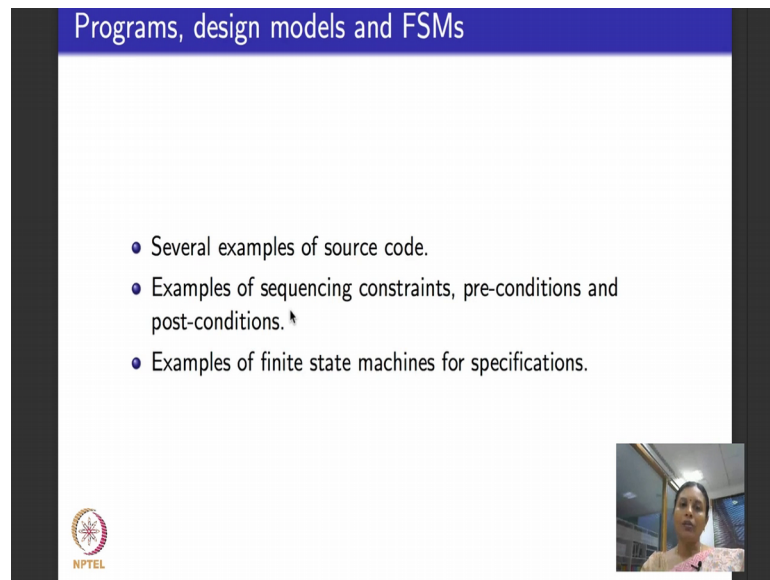
Presented an overview of

- Propositional logic, basics of predicate logic.
- Finite state automata.
- Regular expressions.
- Context-free grammars.

NPTEL

In this process of lecturing to you about test phase design and criteria we also saw a good number of formal models and languages we saw the basics of propositional logic and predicate logic. I introduced you to finite state automata, we saw regular expressions we saw context free grammars.


(Refer Slide Time: 12:41)



Programs, design models and FSMs

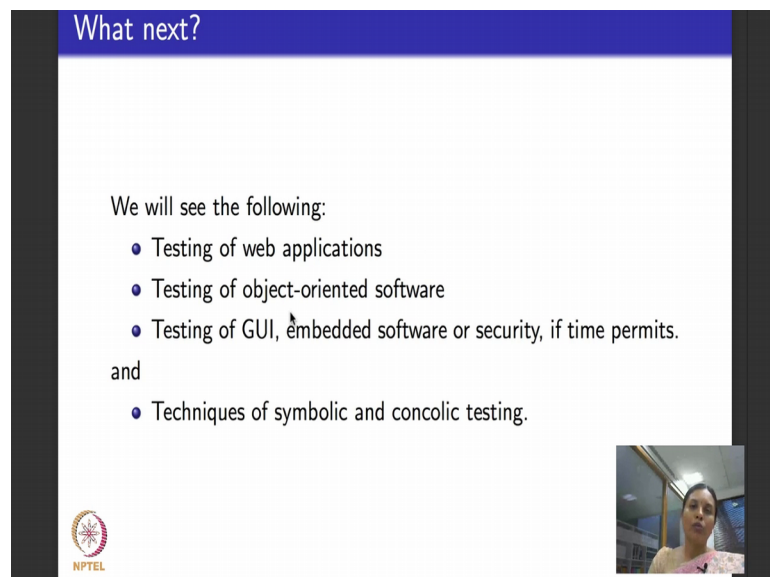
- Several examples of source code.
- Examples of sequencing constraints, pre-conditions and post-conditions.
- Examples of finite state machines for specifications.

NPTEL



Of course, not very rigorously at a shallow level, but we did introduce them formally and saw results in examples about them and we saw several examples of source code several examples of design constraints like sequencing constraints, preconditions, post conditions. We saw examples of finite state machines for specifications and XML for n.

(Refer Slide Time: 12:54)



What next?

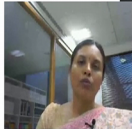
We will see the following:

- Testing of web applications
- Testing of object-oriented software
- Testing of GUI, embedded software or security, if time permits.

and

- Techniques of symbolic and concolic testing.

NPTEL



What are we going to do now from week 10 onwards for the remaining 3 weeks, I will introduce you to testing of web applications object, oriented applications wherever applicable will come back and look at how the coverage criteria we learnt applies here. I

will also introduce you if time permits to testing of embedded software, security testing, and about GUI testing. I am not confident at this stage that I will be able to cover all of them we will try to do as much as possible, but whatever we do we will do it rigorously and thoroughly and I would like to spend some time in the last couple of weeks doing these two techniques which are called symbolic testing or concolic testing which have been there for the past 30-40 years, but they have surfaced now in a big way and they give you a very nice set of algorithms to do path based coverage in testing and they also do the instrumentation for you.

So, this is where we stand and this is what we are going to do. I hope this overview lecture was useful for you to sit back and reflect where we are in the course, what we did till now and what are we going to do from now on. So, next week when I meet you the first video will be on testing of web applications.

Thank you.