

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

Lecture – 60
Software Testing: Summary at the end of the course

Hello everyone, this is the absolute last lecture of the software testing course. What I thought I will do in this lecture is to give you an end of the course summary. We will revisit all that we did in the course, find out what is not done and maybe look at some of the tools that will help us to do software testing. If you remember and if you listen to all the lectures of the course towards the end of 9th week, I had done a similar such summary because that time, we had looked at majority test case design and criteria and I thought it was apt at that time to give a summary through three-fourth of the course.



This lecture is essentially a repeat of that summary along with a few additional details that we have covered towards the end of the course and one of the things that I consciously took a decision of and did not do in this course is to teach you tools. So, what I thought I will do is towards the end of the course, I will give you an overview of about approximately ten tools that are widely used or believed to be widely used the extent I could gather information across different firms and institutions that to test their software.

So, this course will essentially; this lecture will essentially be an overview recap of all that we have done in the course similar to the one that we did in week 9 along with an overview of software testing tools that are currently been widely used.

(Refer Slide Time: 01:46)

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.



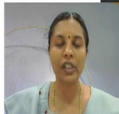

So, again I said it for week 9, I am starting with what was committed in the course that we began in the beginning. So, this course we said will cover algorithms and techniques for test case design by basing by modelling them, software artifacts through 4 kinds of models graphs logical expressions input sets and models based on grammars.

So, then we said these test cases could be applied to test code white box testing or could be applied to test based on requirements preconditions post conditions and invariance which would be considered as black box testing.

(Refer Slide Time: 02:30)

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



So, the contents of the course, as I recap from what was posted for you all to see and it will be in the website, read, just follows; we began the first week by introducing software testing; what are the process levels, the basic terminologies clarified a lot on them, then the bulk of the course from about the second week till about the ninth week, dealt with techniques and algorithms for basically designing test cases. So, we considered software artifacts which included requirements design source code elements of design like preconditions post conditions special models like finite state machines and so on.

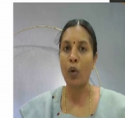
And module them first using graphs we learnt some graph algorithms for that we looked at structural coverage criteria purely based on graphs, then we looked at augmenting the structural coverage criteria with data flow which is the basis of a course and program analysis also which talked about definitions and uses of variables, then we applied the structural coverage criteria and the data flow coverage criteria to graph models derived from source code, from design elements finite state machines and from specifications. After this coverage on graphs, we started with logic I gave you a very very brief introduction to logic, we saw what predicates were clauses were, what is propositional logic, how difficult are the algorithms, checking satisfiability of propositional logic formulae and predicate logic formulae.

And then we looked at coverage criteria based on logical expressions, we looked at simple coverage criteria like predicate coverage, clause coverage, then we also looked at interesting criteria like active clause coverage, 3 versions of it, inactive clause coverage; 2 versions of it. These are widely used for testing safety critical systems, then we took this logical coverage criteria. First saw how to apply it to source code, if you remember, we saw 2 examples; example of a thermostat; an example that determines the type of a triangle, then we looked at how to apply logical coverage criteria for design elements where we looked at 3 conditions and how to apply logical coverage criteria for state machines. Moving on we looked at black box testing I introduced you to functional testing as you would find in many popular text books.

(Refer Slide Time: 04:56)

Course contents, contd.

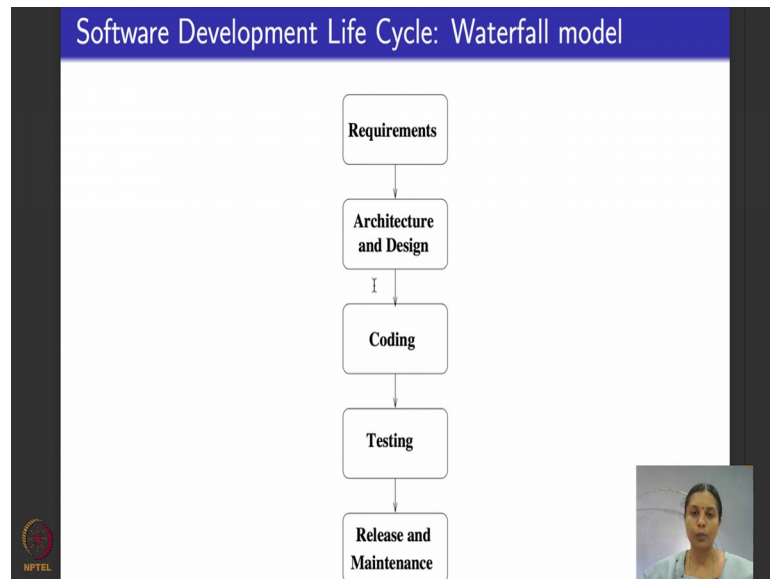
- Input space partitioning: Input domain modeling, combination strategies criteria
- Syntax based testing: Coverage criteria based on syntax, mutation testing
- Test case design (as learnt above) applied to
 - Testing OO-applications
 - Testing web applications
 - Testing embedded software
 - Testing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



We learnt equivalence partitioning, boundary value analysis, decision tables, random testing and so on, then we looked at a generic technique that partitions the input domain and derives test cases by considering a combination of these partitions. We saw how to model the input domain, how to partition the input domain and we saw about 6 different criteria to model combinations of input domains and write test cases for them, if you remember, we said all combinations coverage, each choice coverage pair wise coverage team wise coverage and for functional testing based input domain modeling. We looked at based choice coverage and multiple base choice coverage. So, that was a break from logic and graph based testing. We did black box input space partitioning based testing from there on, we moved on and looked at mutation testing which involved testing based on grammars.

I first introduced you to grammar as it occurred in programming languages to be able to do that; we saw regular expressions context free grammars, then we saw coverage criteria based on syntax mutation testing coverage criteria and then what it is say in the course, we said we are going to learn it to tests object oriented applications web applications embedded software GUI and then we said we will tests symbolic testing and concolic test.

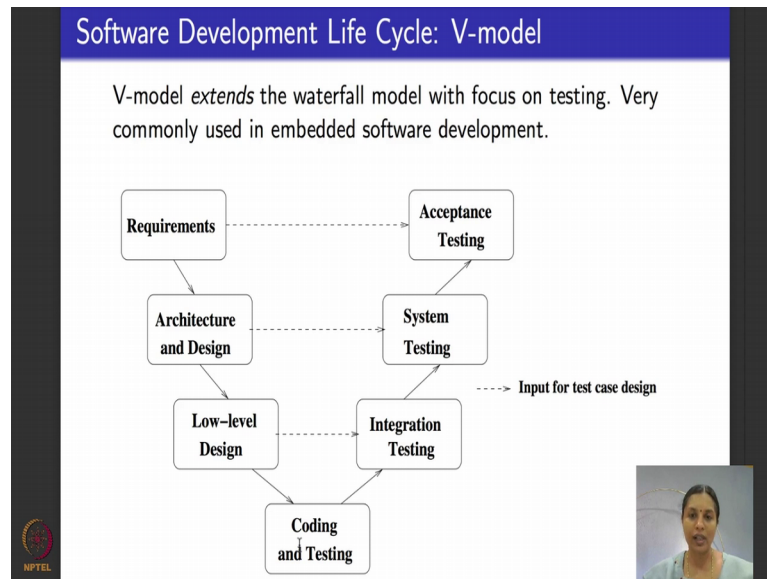
(Refer Slide Time: 06:35)



So, if you remember, this is how a software development lifecycle works. It is a classical waterfall model, we begin with writing requirements high level system level requirements, drill down to hardware software and specific requirements follow it up with architecture and design follow it up with coding and unit testing and then integration and system level testing and then release and maintenance.

Testing applies throughout this cycle and how does testing apply throughout this cycle? We bend the waterfall model to be able to get what is called the V model waterfall model is this part a part of it is depicted on the left hand side of the V model as they say, this is just a copy of what is given here requirements architecture low level design have refined it a bit.

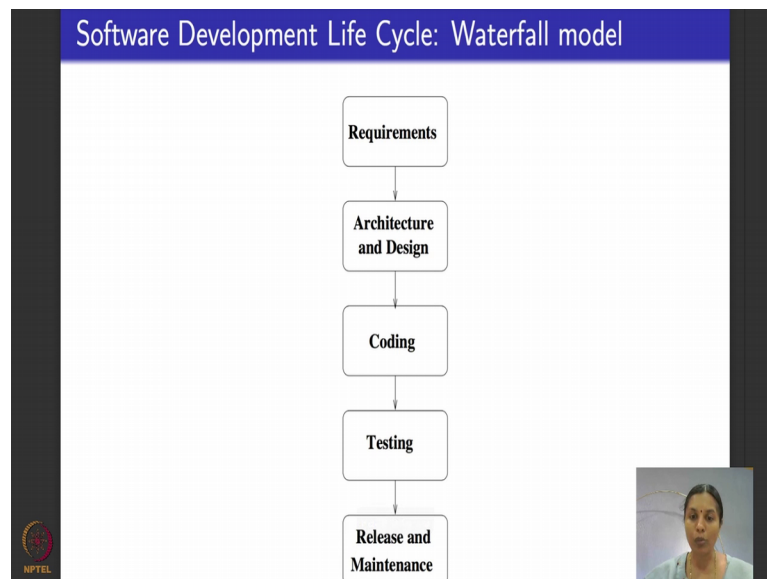
(Refer Slide Time: 07:06)



Because I want to be able to link it to integration testing coding and testing and what I did in the last phase here was to expand out testing comes along with coding which is unit testing followed by integration testing for which inputs or design for test cases come from the low level design because that tells you; how a software is broken up into components of modules, followed by system testing for functional to system testing inputs come from the design document. System testing involves putting the entire thing together and testing if the overall software integrated with the system needs this functionality.

As we saw in the last lecture, in this week's lecture or nonfunctional testing system testing also includes nonfunctional testing nonfunctional testing tests for various quality attributes like reliability security usability performance interoperability and we saw a whole set of nonfunctional testing attributes where does the majority of inputs for designing such test cases come from. They come from the architecture document which gives a static picture of the various components of the software, how they interact which is the processor in which, it is going to run on what is the platform care features and so on; after all this is done finally, users; end users of the software do acceptance testing to check if the software and the system needs its desired requirements or not in fact.

(Refer Slide Time: 08:56)



In this week, we saw some testing after the acceptance testing which is basically going back to the waterfall model deals with this phase of software development which is release and maintenance.

I gave you an overview of regression testing which people do when they maintain software to cater to patches upgrades or change requests that happen in the software.

(Refer Slide Time: 09:16)

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.

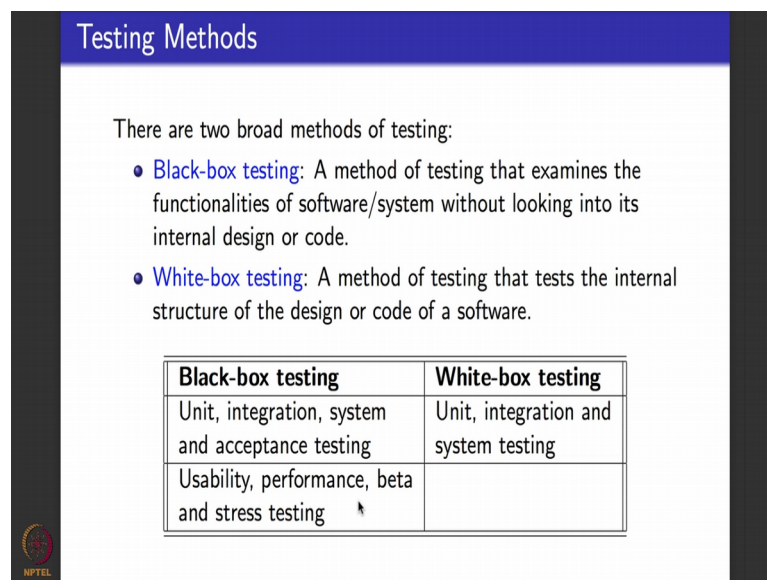
The NPTEL logo is visible in the bottom left corner.

So, these are the various sets of terminologies that we have seen. Each one tries to attempt testing and classify it into categories based on a particular feature or a phase that the testing is involved in the 5 types of testing that we saw are unit testing which is done

here by the developer especially methodologies developers are expected to do unit testing, you cannot rely and hope that there will be a separate tester who would do it for you followed by integration testing, here components are put together. Components could be software software components or software hardware components. We specifically saw integration testing related the software software components in this course and not integrating it with hardware, then system testing is done with the full system in place software running on the desired hardware platform.

Finally acceptance testing is done by customers and a related term is beta testing which is when the software is released, but with known bugs not 100 percent guarantee that it will cover and people do beta testing as a version of acceptance testing.

(Refer Slide Time: 10:27)



The slide is titled "Testing Methods" and contains the following text:

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.

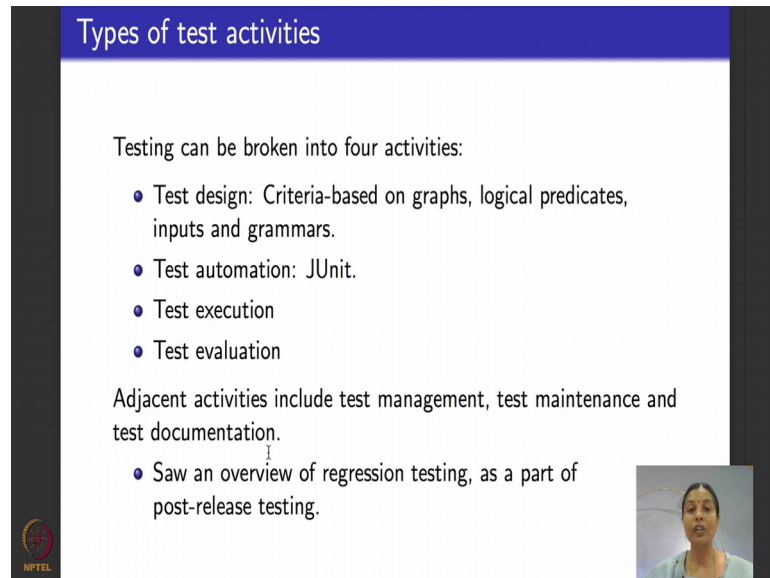
Black-box testing	White-box testing
Unit, integration, system and acceptance testing	Unit, integration and system testing
Usability, performance, beta and stress testing	

So, this spans the V model as we saw it another classification of testing are what is called methods in testing black box and white box. We saw these in detail black box testing considers the software or the system as a black box only the executable is given, we do not look at its internal details white box exploits the structure of the considered software to be able to design test cases.

Black box is widely practiced across several different testing phases and stages white box is typically restricted to unit testing integration testing at software software level and at the system level a lot of the course, we did see a mixture of white box and black box testing as it applies to the first row, here we did not spend too much time for black box

testing as it applies to the second role. Here, the last week in this lecture in this week, I gave you an overview of some of these techniques.

(Refer Slide Time: 11:23)



Types of test activities

Testing can be broken into four activities:

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

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

- Saw an overview of regression testing, as a part of post-release testing.

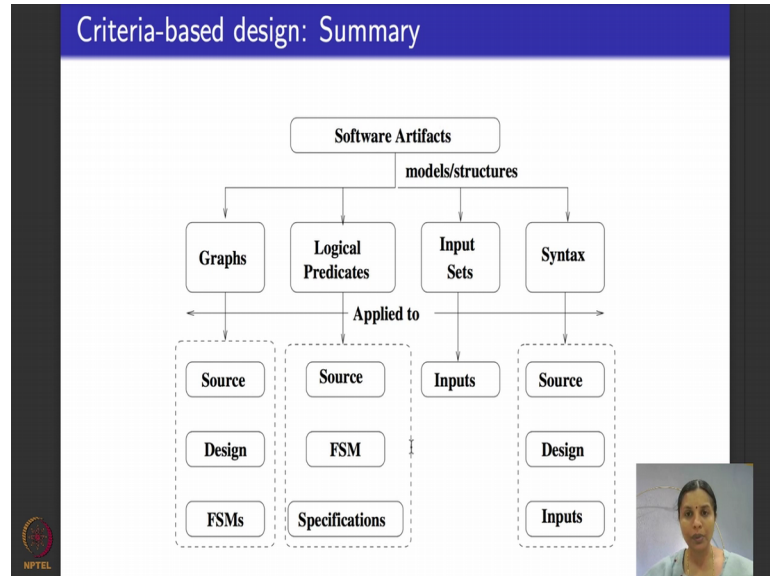
NPTEL

Another related thing is what are the activities or the processes of testing. The first thing is after you have your testing goals is to be able to design your test cases; that is what we did most of the course. We did an algorithmic style of designing the test cases and then once you design a test case you pick your tool of choice for execution. For this course, I had introduced you all to the tool JUnit because it was popularly used for java and the book that I followed for majority of the course also uses JUnit test cases. So, that is why I had introduced that tool to you, I hope you were able to try out some examples with JUnit once you have JUnit and you figure out; how to automate the test script on JUnit. The next job is to execute it and evaluate the test case to see if the test case is actually revealed an error in the program or not.

Typically lot of other activities happen along with testing test management test maintenance documenting the test cases typically if these activities have become very important after a software is released when it is being maintained here because that is when you do regression testing as we saw in this week. And the sources documents that are needed for regression testing are the documents that you keep with the role of test maintenance which is basically reusing with set of test cases for regression testing and documentation which talks about documenting a test cases which will help you to select

which test cases to reuse for regression testing then this is a summary of the criteria based design that we saw I just repeat it to you.

(Refer Slide Time: 12:55)



So, I will be brief here. Software artifacts could be code source, code design, preconditions, post conditions assertions, specifications models like finite state machines activity diagrams and so on.

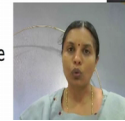
I take them based on what I want to test; I can model it as a graph model. It is a logical predicate consider only the inputs that drove the software artifact or work with the syntax of the artifact. Once I model the software artifact using one of these 4, I can use it to test source code design finite state machines, I can again use it to test source code finite state machine and specifications, but where I consider input space partitioning its only black box.

(Refer Slide Time: 13:55)

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.



So, I can use it only to test a software with reference with to its black box testing code this was the main bulk of the course as I taught you from the book by Amman and Offutt along with this, we also saw several classical software testing terminologies, typically when you pick up other books in software testing, when it comes to testing source code, you would hear of terminologies like cyclomatic complexity independent paths basis paths testing D to D paths statement coverage branch coverage loop coverage path coverage and so on.

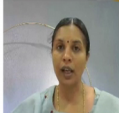

We did look at all these terms because these are classical standard terms and software testing. Now I hope it is clear; how these terms relate to the coverage criteria that we saw in the course, then when it comes to design integration testing, I told you about top down integration, bottom up integration, the various kinds of interfaces message passing interface, client server interface, shared variable communication, then we saw what test stubs and test drivers were how to create stubs, how to test using stubs and drivers and typically do top down or bottom up integration which are the 2 most popular integration testing techniques in black box testing techniques, I did a module on functional testing where you learnt about equivalence partitioning boundary value analysis and decision tables.

(Refer Slide Time: 15:09)

Formal languages

Presented an overview of

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

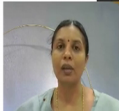



In the process, we also learned a bit of discrete maths and formal languages, I taught you graphs; graph algorithms DFS-BFS; topological stores strongly connected components, then we looked at logic specifically propositional logic and basics of predicate logic; how difficult is the satisfiability problem for propositional logic its N P complete for predicate logic is un-decidable. So, we work with a lot of heuristics bundled together and sat solvers to be able to solve the logical formulae, we also saw a detailed introduction to finite state machines to regular expressions to context free grammars as they were used to build grammars or the syntax of programming languages.

(Refer Slide Time: 15:50)

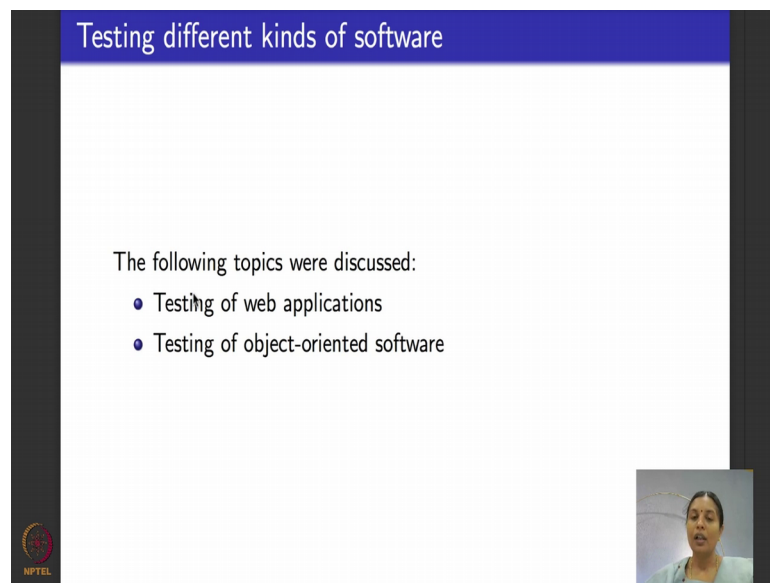
Programs, design models and FSMs

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



So, we saw several examples of source code, if you try to recap the examples that we saw throughout the program, we saw triangle types, then we saw thermostat, several different examples, then we saw examples of sequencing constraints, preconditions, post conditions, if you remember calendar method Q example file open, file close, then we saw examples of finite state machines for specification sub wave microwave oven and so on.

(Refer Slide Time: 16:27)



The slide has a blue header with the text "Testing different kinds of software". Below the header, the text "The following topics were discussed:" is followed by a bulleted list:

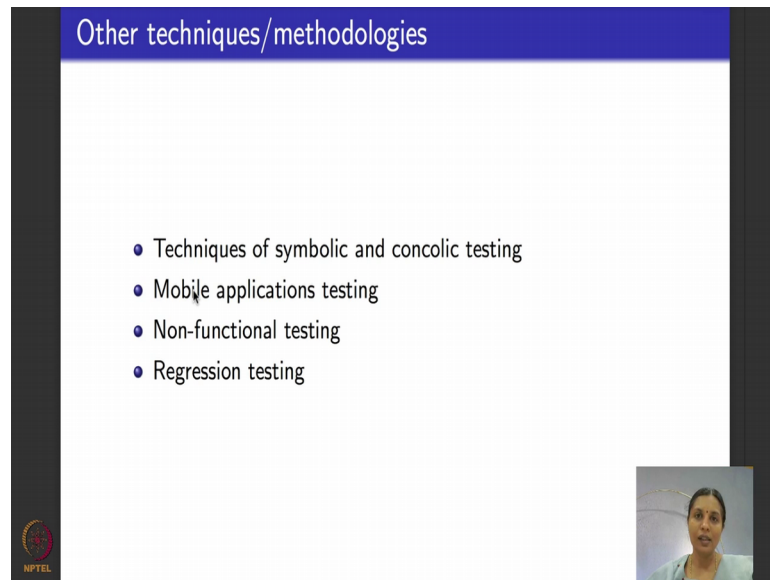
- Testing of web applications
- Testing of object-oriented software

In the bottom right corner of the slide, there is a small video inset showing a woman with dark hair, wearing a light blue top, speaking.

So, we also after week 9, this was what I did newly after week 9, a lot of what I told you now was a repeat of the overview that I had given at the end of week 9. After week 9, we mainly concentrated on the following topics. We saw how to test web applications static hypertext websites, dynamic hypertext websites from the client side and from the server side I had taught you one technique each for server side testing and for client side testing bypass testing, user session, data base testing and testing based on graph models which were derived from the code corresponding to server as atomic sessions. And then we also looked at testing of object oriented software because typically object oriented software have lot of features like inheritance polymorphism we did a sort of a crash introduction to all these object oriented features, we saw how these features can lead to lesser static determinism and you do not know which version of the variable is being called and how to determine the state.

We saw the models of yo-yo graphs, we saw various object oriented faults in detail and towards the end in this week I defined object oriented integration testing coverage criteria for you all last week; week 11, we saw symbolic and concolic testing techniques.

(Refer Slide Time: 17:41)



The slide features a blue header with the text "Other techniques/methodologies". Below the header, there is a white background with a bulleted list of four items: "Techniques of symbolic and concolic testing", "Mobile applications testing", "Non-functional testing", and "Regression testing". In the bottom right corner of the slide, there is a small, square video inset showing a woman with dark hair, wearing a light-colored top, speaking. In the bottom left corner of the slide, there is a small logo for NPTEL.



Which was a distinctive departure from the coverage criteria based techniques that we have seen until now in concolic testing, we particularly saw the dart tools algorithm then as requested by the users of this course I gave an overview of these topics.

Mobile applications testing; testing non functional things with specific focus on performance testing and regression testing, while I did this, I also gave you references to some tools, open source and proprietary that you could have access to or use it to try it out and what we did not do in the course what else is there in software testing that we did not do over the course, first of all I hope you do believe that we have covered a lot in the course and I hope it was useful learning for you.

(Refer Slide Time: 18:22)

What we didn't do?

- Non-functional testing, in detail.
 - Security testing, software reliability analysis, usability testing.
- Testing of GUI.
- Testing of embedded software.
- Tools, apart from JUnit.
- Acceptance testing.



But because like any other course in any other topic we have to leave out a lot of things, so, what we did do is nonfunctional testing in detail. I did one module of nonfunctional testing, we covered performance testing a bit low stress testing, but security testing is a very beautiful theoretically rich area which we did not cover at all then there is something called reliability analysis which I told you when we did nonfunctional testing typically reliability analysis for both software and hardware work with models that involve probability like discrete mark of decision processes hidden mark of models and they need a good amount of statistics and probability to be able to determine the reliability, right which talks about measuring parameters like mean time between failures average down time average up time and so on.

We did not do any of those because those are quite involved theoretically and algorithmically and need you to pick up that kind of maths another big area in testing which we have all together completely left out is usability testing usability engineering itself is a big area inside that testing for usability is very important catching up a lot because of all the apps and games and other interfaces that we are dependent on to use our software specific part of usability testing is testing of graphics user interface which I did list as a part of the contents of the course, but I did not do because of lack of time. I wanted to there is a particular style of techniques by which you work with graph models that look a lot like finite state automata using which you can test graphics user interface which I did not do.



Testing of embedded software is very critical because typically embedded software is a safety critical system and it goes through very rigorous testing process. So, I should say that all the coverage criteria based testing that we did so far can very well be used to test embedded software for GUI applications and so on but specifically for things like embedded software people do; what is called real time testing also because such software come with lot of real time constraints like interrupts timeouts and exception handling features. So, there is specifically a lot of testing done for these kind of software which we did not look at, but what were we looked at can be applied to test embedded software and can also be applied to tests GUIs also and tools I had introduced you to JUnit mainly because the programs that we dealt with they are all compatible with JUnit it is a nice open source testing tool for java thing, but apart from JUnit we did not do any other tools.

I will do some tools now as a part of this lecture, give an overview of several other tools and the other thing that we did not do. If you go back to the V model, here after system testing comes acceptance testing which is a software is released into market and users basically check if a software is working or not, there is not much algorithmic content in that thing typically, you know for an acceptance testing like a car; what they do is to see if the brakes of the automatic braking system is working fine, they try to crash a car and check the brake system. Similarly for the software that controls how the airbags are handled they try to create a scenario to check if the car controller is bringing up the airbags properly or not. So, these things are either done using exhaustive simulation setups or in the real life. So, there is not much algorithmic mathematical content that can be taught as a part of such a course.

(Refer Slide Time: 22:33)

Why we didn't look at tools?

- Tools help in automation of testing.
- Automation is possible only for execution, documentation and maybe, analysis.
- Test case design is to be manually done.
- Learning a tool is basically learning its test interface, notations etc., can be picked up easily.



So, we did not do acceptance testing because of that reason. So, now, for the rest of this lecture, I want to spend some more times on looking at tools. Before we do that I want to justify and tell you why we did not look at tools at all throughout this course, it might be a very natural question that you have tools are needed for testing they are indispensable in fact for testing; testing cannot exist without automation with the help of tools. Automation is necessary only for executing a test case, for documenting a test case and maybe use the results for analyzing the test case to see if there is an error in the software or not, but who is going to design the test cases that has to be done by the tester or a programmer that is what this course focused on algorithms and techniques, for test case design that is manually done, once that is done, you should be able to use any tool for automation and to be able to execute it.



Also the other reason why I did not really teach tools is basically what is the teaching a tool, mean you download the tool; you learn its interface you learn the commands of the tool or if it has a web interface you learn the web interface or the IDE. So, basically all our focus will go on figuring out the then trying to learn its interface and how it interacts with the user, yeah, I am sure you would agree with me that that can be picked up on your own and you really do not need a teacher to be able to teach you that. So, typically in a classroom course, we do not focus on tools, it is only if you have a lab component attached to the classroom course that you would do the tool and that also you sort of try to pick up on your own, we typically do not teach it as a part of the routine lectures.

(Refer Slide Time: 24:13)

Broad classification

Testing tools can be classified as those that do

- Functional testing
- Non-functional testing: Load, performance, security etc.
- Test management tools.
- Bug tracking tools.



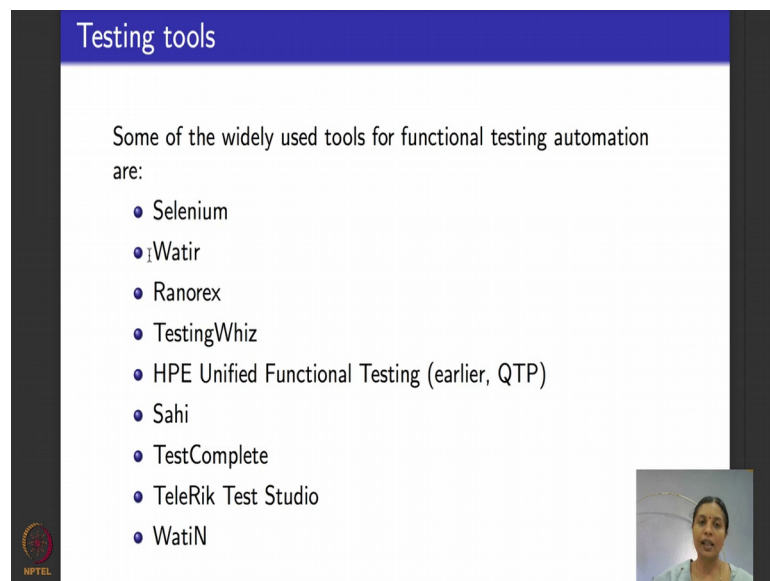
So, what I will do for the rest of this course; this lecture is to give you an overview of tools. So, tools that involved in testing can be broadly classified into the following categories tools do functional testing automation tools that do nonfunctional testing typically, load testing, performance testing, security vulnerability analysis and tools that are used for test management which is basically to archive record test case to track the progress of testing to plan your testing project managers who focus on testing project managers use these tools.

Another important category of tools are what are called bug tracking tools you might have heard of tools like Jira and all that once you do a defect, right find, in effect using testing the idea is to be able to record that defect take it back to the developers see if the developer is able to fix that bug or defect and re-test it again to see if the defect has been mitigated. So, once you find the bug or a defect it needs to be tracked from the time it was found to the time that it is rectified.

So, there are specific tools that help you to track these bugs or defects those are called bug tracking tools. As a part of my modules on various non functional testing I had given you some of the nonfunctional testing tools that are popular and please remember, when we did that web testing, I had given you the most exhaustive webpage that I am aware of that contains several different nonfunctional testing tools. So, what I will do now and I also give you tools for regression testing and mobile apps testing that I am familiar with what I will do; now is I will try to focus on testing tools that will do functional testing some of them will also test for load and performance we will see them as we go on.

Here is what is believed to be the most widely used set of tools for test automation Selenium Watir, this is pronounced water tool called Ranorex which I already introduced you this week in another module testing whiz then there is something called HPEUFD; this tool was recently renamed it was popularly known as QTP you might have heard about it, there is another tool called Sahi Sahi and Selenium are used by many companies test complete Telerik test studio WatiN which was motivated by Wtir.

(Refer Slide Time: 26:22)



The slide is titled "Testing tools" and lists several widely used tools for functional testing automation. The tools listed are Selenium, Watir, Ranorex, TestingWhiz, HPE Unified Functional Testing (earlier, QTP), Sahi, TestComplete, Telerik Test Studio, and WatiN. There is a small video inset in the bottom right corner showing a person speaking.

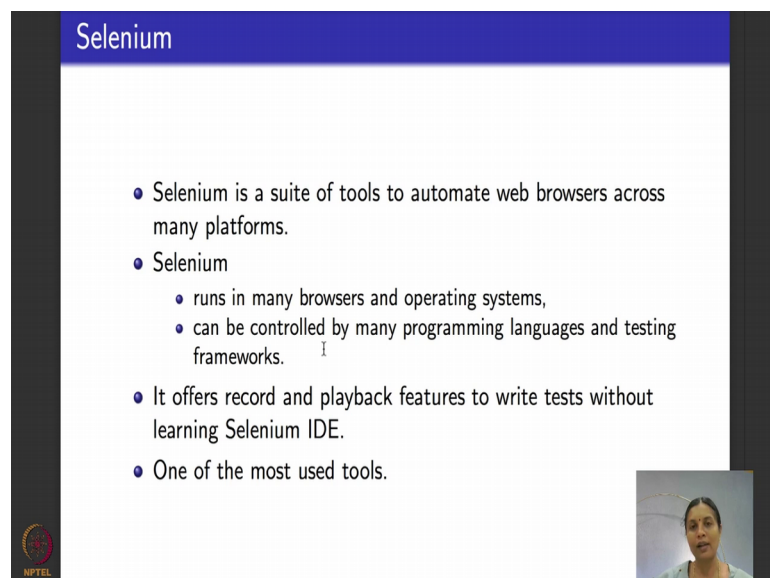
Testing tools

Some of the widely used tools for functional testing automation are:

- Selenium
- Watir
- Ranorex
- TestingWhiz
- HPE Unified Functional Testing (earlier, QTP)
- Sahi
- TestComplete
- Telerik Test Studio
- WatiN

So, well just look at a few of these tools and see what they can do.

(Refer Slide Time: 26:41)



The slide is titled "Selenium" and describes it as a suite of tools to automate web browsers across many platforms. It lists several key features: it runs in many browsers and operating systems, can be controlled by many programming languages and testing frameworks, offers record and playback features to write tests without learning Selenium IDE, and is one of the most used tools. There is a small video inset in the bottom right corner showing a person speaking.

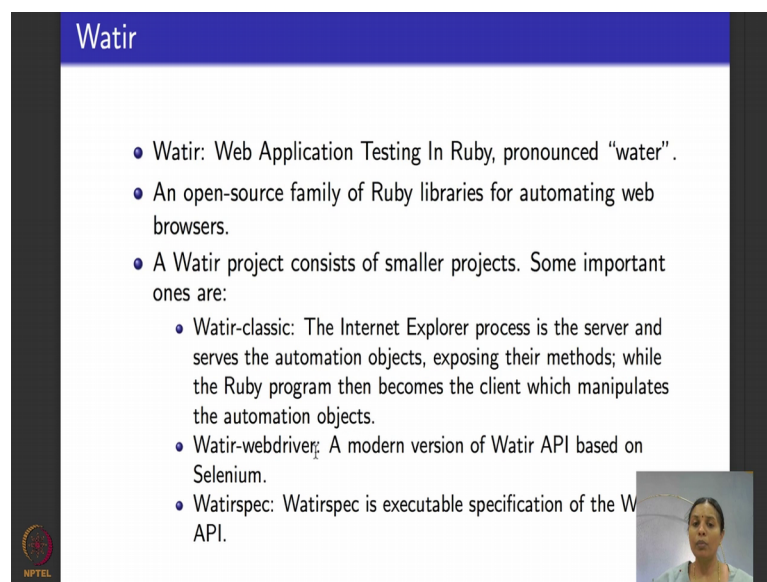
Selenium

- Selenium is a suite of tools to automate web browsers across many platforms.
- Selenium
 - runs in many browsers and operating systems,
 - can be controlled by many programming languages and testing frameworks.
- It offers record and playback features to write tests without learning Selenium IDE.
- One of the most used tools.

So, selenium is basically a suite of tools to automate web browsers its or web browser based test automation. So, it automates web browsers across several different platforms windows based Linux based and so on selenium as I told you runs on many different browsers many different operating systems can be controlled by several programming languages and testing frameworks it basically gives you a web interface to handle and automate your test execution.

It records; it offers record and playback features which I told you in the context of regression testing to write tests without learning the IDE and it is considered to be one of the most used tools in the industry.

(Refer Slide Time: 27:24)



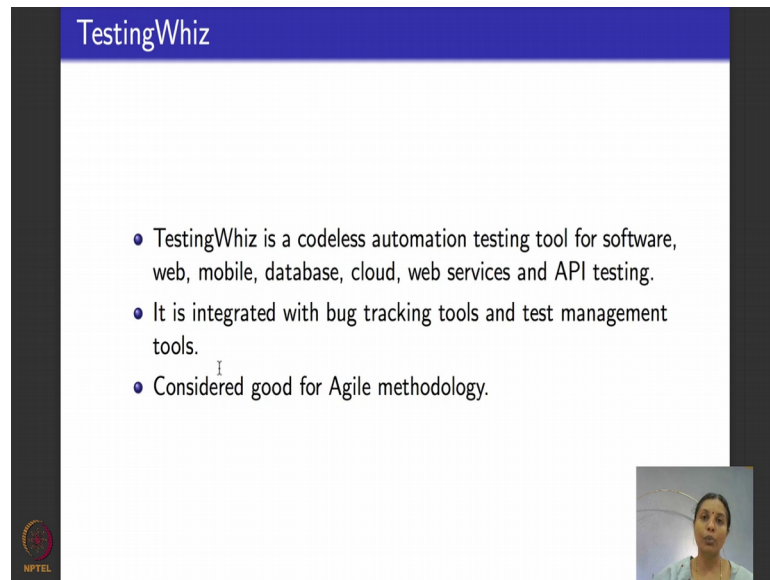
The slide is titled "Watir" in a blue header. It contains a bulleted list of information about Watir. In the bottom right corner, there is a small video inset showing a woman speaking. The NPTEL logo is visible in the bottom left corner of the slide.

- Watir: Web Application Testing In Ruby, pronounced "water".
- An open-source family of Ruby libraries for automating web browsers.
- A Watir project consists of smaller projects. Some important ones are:
 - Watir-classic: The Internet Explorer process is the server and serves the automation objects, exposing their methods; while the Ruby program then becomes the client which manipulates the automation objects.
 - Watir-webdriver: A modern version of Watir API based on Selenium.
 - Watirspec: Watirspec is executable specification of the W API.

Then there is a tool called Watir, pronounce it as water, it stands for web application testing in ruby, ruby is this programming language. It basically is an open source family of libraries that are written in ruby again for web browser automation for automating web browser automation and text execution through web browser automation a Watir project typically consists of smaller pieces Watir classic is the IE process where the server is the IE process and what does it serve the clients, they are the objects that you want to automate and execute as a part of your test case execution how does it do it exposes their methods and then the ruby program in which Watir is written becomes the client then it manipulates the automation objects.

So, that is how the classical version works. There is a web driver Watir which is a modern version of Watir API based on selenium tool. So, selenium can be integrated with Watir then there is something called Watir spec.

(Refer Slide Time: 28:28)



TestingWhiz

- TestingWhiz is a codeless automation testing tool for software, web, mobile, database, cloud, web services and API testing.
- It is integrated with bug tracking tools and test management tools.
- Considered good for Agile methodology.

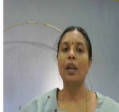

NPTEL

Which is basically an executable version of the Watir API, then another popular tool is testing whiz it is a codeless automation testing tool for software testing web applications testing mobile apps testing; testing for databases like SQL cloud testing; testing by hosting on the cloud web services which form a specific part of web apps and for API testing, the nice thing about testing whiz is that its integrated with several different bug tracking and test management tools. So, you could actually use the same tool to be able to track your defect, if you have found any and manage it later, instead of switching over to a separate bug tracking tool and because of this testing whiz this the belief is that is considered good for people who follow agile methodologies.

(Refer Slide Time: 29:16)

HPE Unified Functional Tester

- Popularly known as QTP (Quick Test Professional).
- UFT is primarily used for functional, regression and service testing.
- Using UFT, the following can be done:
 - automate user actions on a web or client based computer application,
 - test the same actions for different users, different data set, on various Windows operating systems and/or different browsers.



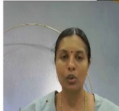

The next 1 is QTP; earlier it was known as QTP short form for quick test professional, but now it is called HPE unified function tester, it is primarily used for functional testing regression testing and testing of service apps you can automate user actions on a web browser. This what always happens in most of the automation tools or you can automate it on a client based computer application that you write for it.

(Refer Slide Time: 29:52)

Sahi

- An open-source test automation tool to automate web applications testing.
- Sahi provides following features:
 - Performs multi-browser testing
 - Supports ExtJS, ZK, Dojo, YUI, etc. frameworks
 - Record and playback on browser testing

I



You can test the same actions for different users, for different datasets on various windows operating systems and various browsers. The next popular tool in our list is Sahi; it is the nice thing about it is that it is an open source tool. Again for test automation, it automates web application testing you can use it with several different

browsers, it supports all these frameworks that are listed here and it also does record and playback like most other automation tools.

(Refer Slide Time: 30:12)



The slide is titled "Telerik TestStudio" and contains a bulleted list of features. In the bottom right corner of the slide, there is a small inset video of a person speaking. The NPTEL logo is visible in the bottom left corner of the slide.

- Telerik TestStudio is another tool to automate desktop, web and mobile application testing.
- Includes UI, load, and performance testing.
- Offers various compatibilities like:
 - Support of programming languages like HTML, AJAX, ASP.NET, JavaScript, Silverlight, WPF, and MVC.
 - Record and playback
 - Cross-browser testing
 - Manual testing
 - Integration with bug tracking tools

Telerik studio is the next in our list. This again automates desktop web and can be used for mobile applications testing, I do not think I listed it in my mobile applications lecture, but I am not sure this also does nonfunctional testing, it can test for graphical user interface, load stress and performance, it offers supports many programming languages like HTML, AJAX, ASP dot NET, JavaScript and so on, it offers record and playback features. In fact, most of these tools offers record and playback features supports several browsers can do manual testing and it is integrated with bug tracking tools. So, this was an overview, I have did not give you URLs because I realized that they keep changing and once I give a URLs for such a long list, it is going to be difficult for me to keep track of when the website gets updated, but feel free to Google for them, you will be able to get the URLs well.

So, this will give you a basic set of test tools open source and proprietary, ones that are available for free trials for you to start your testing exercise, if you are interested; if you want to call yourself as an expert complete expert in software testing, apart from this course, you could also learn one of the popular tools like selenium or Sahi or one of those and the get to familiarize yourself with their interfaces and what their capabilities are and how to do automation of test case execution in them that will definitely help you

if you are interested specifically in job interviews and so on to be able to get a job as a quality expert.

So, I hope you enjoyed this course and we hope these set of lectures were useful for you and you learnt a bit in software testing all the best for your upcoming exams hope to see you all again in NPTEL sometime.

Thank you.