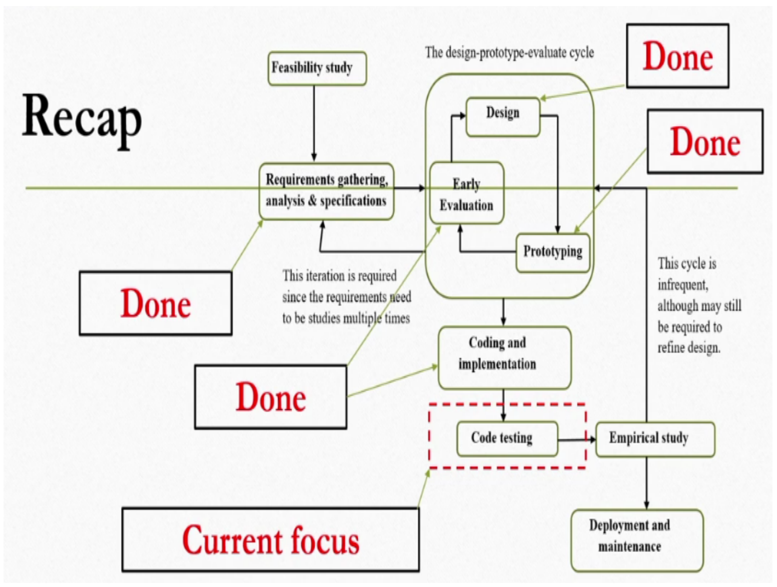
Design and Implementation of Human-Computer Interfaces

Dr. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology – Guwahati

Lecture – 36
System Integration and Testing

Hello and welcome to the NPTEL MOOCS course on design and implementation of human-computer interfaces, lecture number 30 where we will continue our discussion on how to test the system. As is the practice we are following before we start the lecture, let us quickly recap the things that we have learned and where we are currently.

(Refer Slide Time: 01:09)

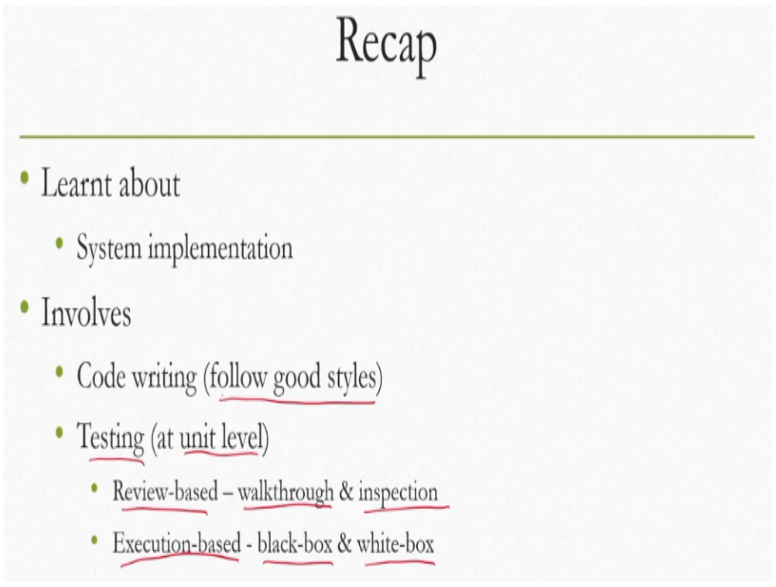

Recap
Feasibility study
The design-prototype-evaluate cycle
Done
Done
Design
Requirements gathering, analysis & specifications
Early Evaluation
Prototyping
Done
This iteration is required since the requirements need to be studies multiple times
This cycle is infrequent, although may still be required to refine design.
Coding and implementation
Done
Code testing
Empirical study
Current focus
Deployment and maintenance

So, if you may recollect, we are discussing interactive system development lifecycle. In this lifecycle there are several stages. We have covered many of the stages and we are currently covering the implementation stage. So, among the stages we have covered requirement gathering analysis and specification stage. Outcome of the stages the SRS document as we have noted earlier.

Then we have covered design, prototyping and evaluation stages which together constitute the design prototype evaluation cycle. In the design we have also covered the system design as well as the interface and interaction design. Outcome of the design stages the design document, both for interface design as well as system design. Outcome of the prototyping stage is the prototype that is created and outcome of the evaluation stages the evaluation report of the prototypes.

We have covered the coding and implementation stage as well. In the coding and implementation stage, we have learned about good coding practices and do's and do not’s for coding., the outcome of the stages the code. Currently we are discussing the code testing stage. So, in this stage we are discussing how to test the code that we have written to implement the system.

(Refer Slide Time: 02:43)



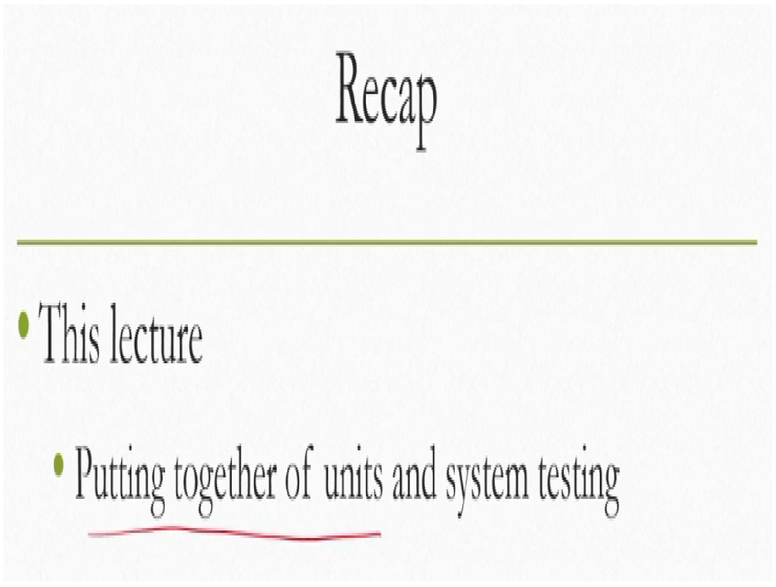
Recap

- Learnt about
  - System implementation
- Involves
  - Code writing (follow good styles)
  - Testing (at unit level)
    - Review-based – walkthrough & inspection
    - Execution-based - black-box & white-box

So, we have learnt about implementation using code and we are discussing the testing. Code implementation involves code writing which involves following good coding styles and testing can be done at different levels. At unit level, we have learnt about the review-based testing and the execution-based testing. Review-based testing involves code walkthrough and code inspection methods which we have discussed in details.

Execution-based testing involves black box testing and white box testing, which also have discussed in details in the earlier lectures. Outcome of each of these testing methods can be testing document as we have seen earlier. Now, all these testing that we have discussed so far are done at the unit level. Recollect that while talking about design of a system, we talked about designing it at module level, creating a module level hierarchy. So, the unit level is basically referring to the implementation and testing at the module level.

(Refer Slide Time: 03:56)

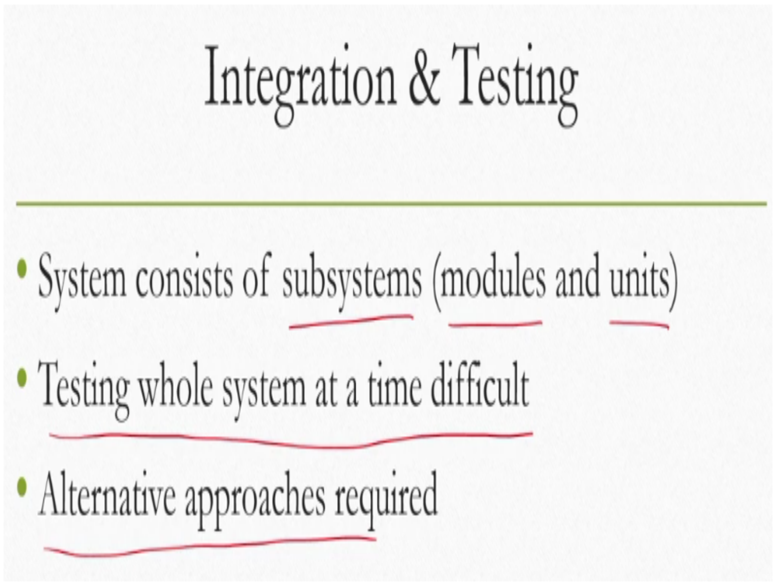


Recap

- This lecture
  - Putting together of units and system testing

So, in this lecture we are going to talk about how to test the whole system once we put together all the modules or the units. So, putting together of units and testing of the whole system together is the subject matter of this lecture.

(Refer Slide Time: 04:27)



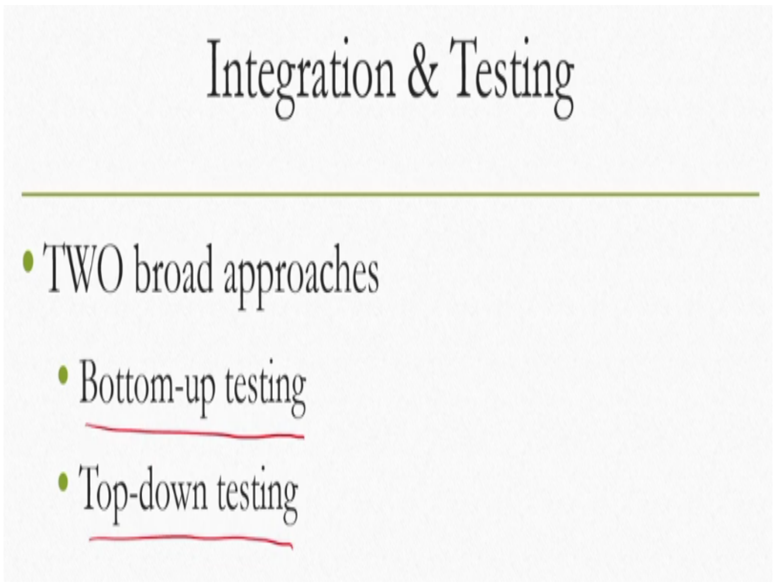
Integration & Testing

- System consists of subsystems (modules and units)
- Testing whole system at a time difficult
- Alternative approaches required

Now, as we know or we have already noted in the earlier lecture, particularly during our discussion on system design, the whole system is not a monolith. So, it consists of subsystem subsystems, in other words modules and units. So, we are supposed to create a module hierarchy, hierarchy of subsystems to go for the design of the whole system that makes it easier to understand and maintain the system.

Therefore, while implementing we also go for implementing one unit at time rather than trying to implement the whole system together at the same time. So, testing the whole system at a time becomes difficult because we have a modular hierarchy and we require some alternative approaches for testing the whole system once all the units are implemented and linked together to form the whole system.

(Refer Slide Time: 05:24)

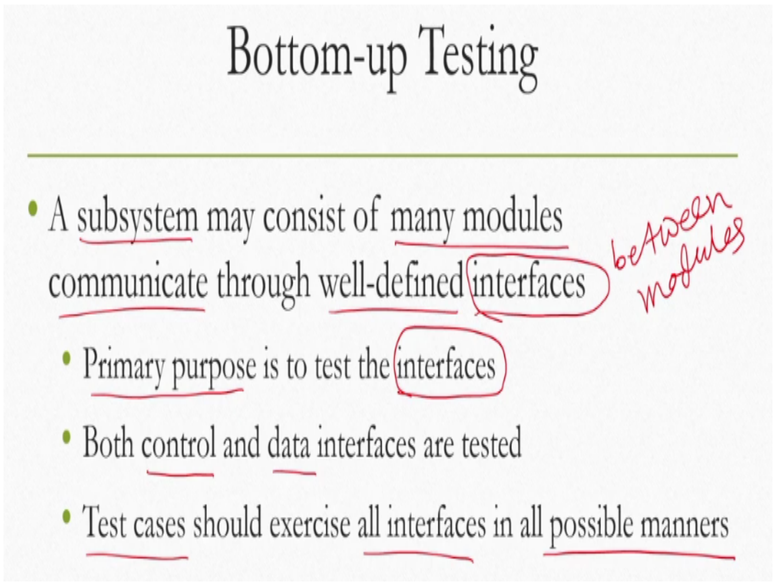


Integration & Testing

- TWO broad approaches
  - Bottom-up testing
  - Top-down testing

In order to test the whole system, we follow broadly two approaches. One is bottom-up testing and the other is top-down testing. So, what are these concepts? What is bottom-up testing and what is top-down testing? Let us try to understand these different testing approaches.

(Refer Slide Time: 05:50)



Bottom-up Testing

- A subsystem may consist of many modules communicate through well-defined interfaces between modules
  - Primary purpose is to test the interfaces
  - Both control and data interfaces are tested
  - Test cases should exercise all interfaces in all possible manners

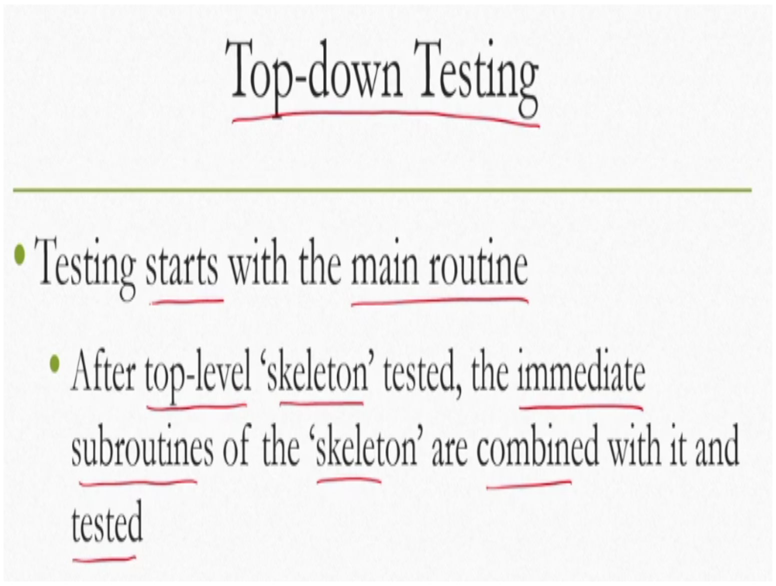
In the bottom-up testing what we do? Each subsystem is tested separately and then the full system is tested. That means as the name suggests we go from bottom to top. At the bottom level, there are subsystems and at the top level there are subsystems put together to form the whole system. So, we test the subsystems or units and then test the whole system. Now, a subsystem may consist of many modules, which communicate with each other through well-defined interfaces between modules.

So, do not confuse it with user interfaces, these interfaces between modules that form the subsystem. So, while we go for bottom-up testing, our primary purpose is to test the interfaces whether the interfaces are working properly. Now, there are two types of interfaces, control interfaces and data interfaces. So, we test both types of interfaces in the bottom-up testing.

The test cases that we designed to test the interfaces should exercise all interfaces in all possible manners that is the general principle that is followed while we use bottom-up testing method for testing the whole system. So, then in the bottom-up method what we do, we first test the units and then join them together and test the whole system. In the testing, the primary emphasis is given on how the interfaces are performing because already we have done testing at the unit level.

No need to test them separately, but whether their communication is taking place as per expectations that is what we need to test, so it tests the interfaces. Both control and data interfaces we test and our test cases should be designed in a way such as that all these interfaces are tested in all possible manners that is the general principle guiding the bottom-up testing approach.

(Refer Slide Time: 08:13)



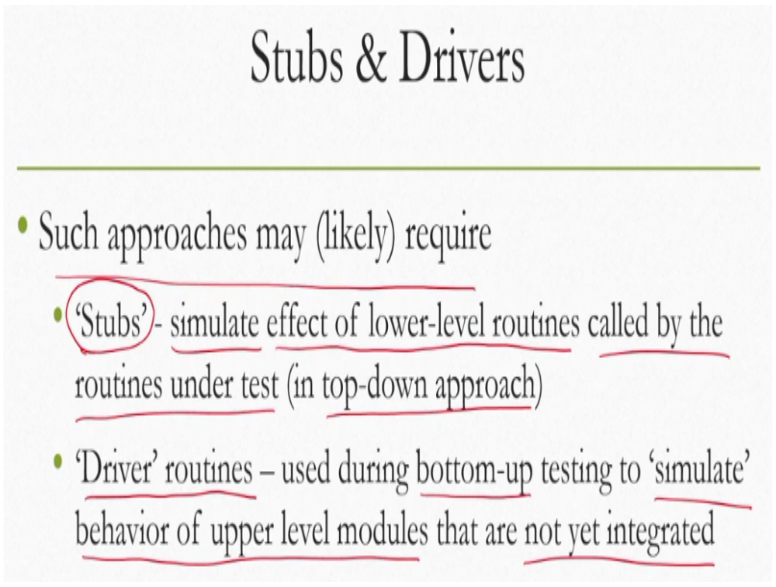
Top-down Testing

- Testing starts with the main routine
  - After top-level 'skeleton' tested, the immediate subroutines of the 'skeleton' are combined with it and tested

Next is the top-down testing. What we do in this top-down testing? Here the testing starts with the main routine. So that means we are starting with the main routine at the top and then slowly going to the bottom. After top-level skeleton is tested, the immediate subroutines of the skeleton are combined with it and tested and this process continues till we cover the whole system.

So, first we start with the main routine, a skeleton version of it, then we combine it with the immediate subroutines and test it and this process continues till the entire system is tested. Now, the problem with this is that while we are testing at the top, we may not have access to the lower-level routines. So, in order to implement this approach, we require additional support.

(Refer Slide Time: 09:20)



Stubs & Drivers

- Such approaches may (likely) require
  - 'Stubs'- simulate effect of lower-level routines called by the routines under test (in top-down approach)
  - 'Driver' routines – used during bottom-up testing to 'simulate' behavior of upper level modules that are not yet integrated

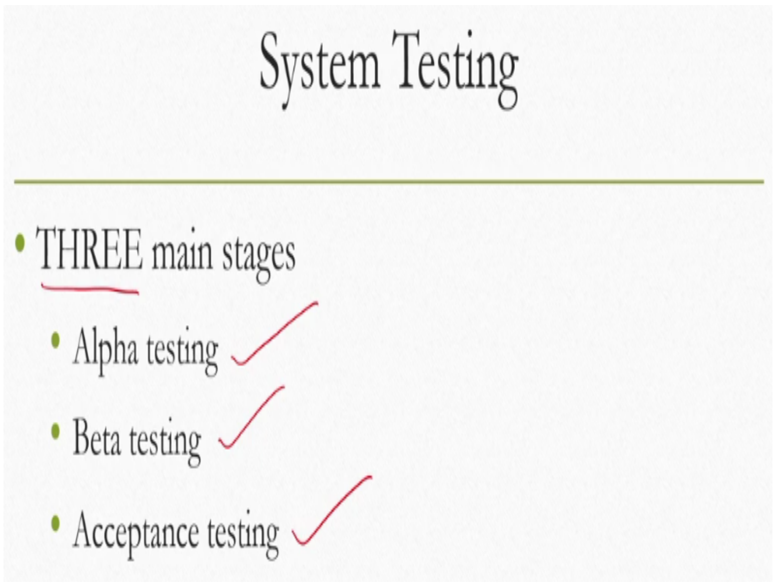
So, these approaches may require a concept called as stubs. Stubs simulate effect of lower-level routines called by the routines under test, particularly when we are following a top-down approach. So, we require stubs which simulate the effect of lower-level routines while we are following top-down approach. So we start with main routines, but the lower routines are not available.

So we use stubs instead of the actual routine so that the effect of the lower-level routines for the given test cases can be simulated, but that is not only restricted to the top-down approach, in bottom up approach also we require some additional support. We require driver routines. These routines are used during bottom-up testing approach to simulate the behaviour of upper level modules that are not yet integrated.

So, in bottom-up approach, we start at the bottom and slowly go up. While going up all the routines that are available at the higher level may not be available to the person have the system under test, so at that time we use driver routines to simulate the effect of those routines that are not yet available. So, for full system testing either bottom-up or top-down, we require some additional things to do.

If we are following a top-down approach, we require the help of sub modules. If we are following a bottom-up approach, then we require the help of the driver modules. So, those are the two broad categories of approaches we use for system testing. Now, let us quickly have a look at different stages of systems testing. How many stages are there and what they do?

(Refer Slide Time: 11:26)

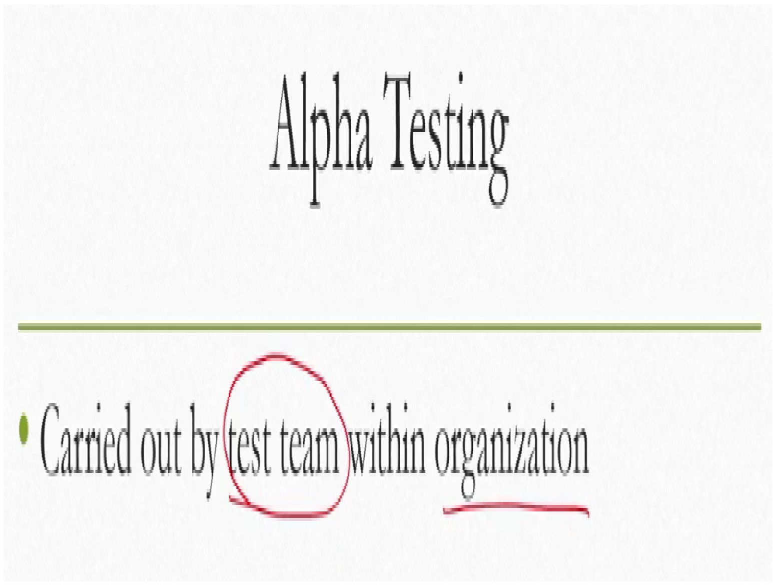


System Testing

- THREE main stages
  - Alpha testing
  - Beta testing
  - Acceptance testing

There are mainly three stages; alpha testing, beta testing and acceptance testing. Let us see what is what.

(Refer Slide Time: 11:38)

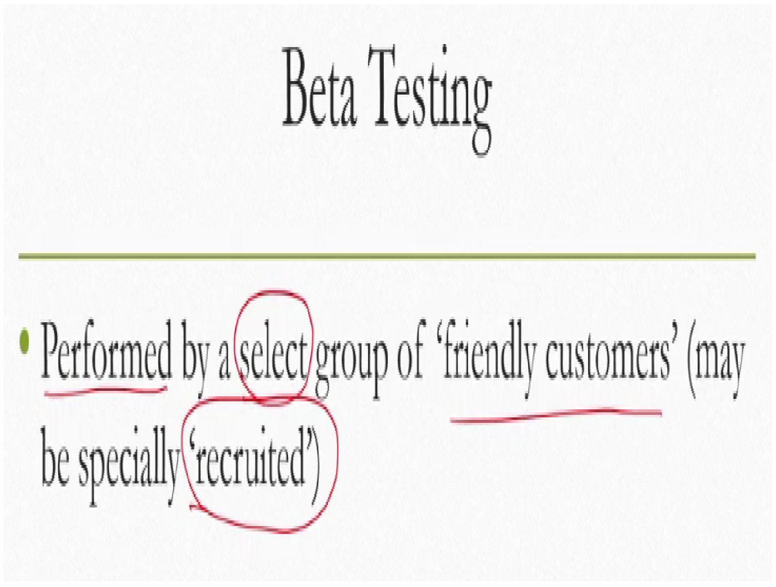


Alpha Testing

- Carried out by test team within organization

Alpha testing, this is the first stage of system testing. In this stage what happens is that the full system testing is carried out by a test team which is part of the organization. So, within the organization the testing takes place.

(Refer Slide Time: 11:58)

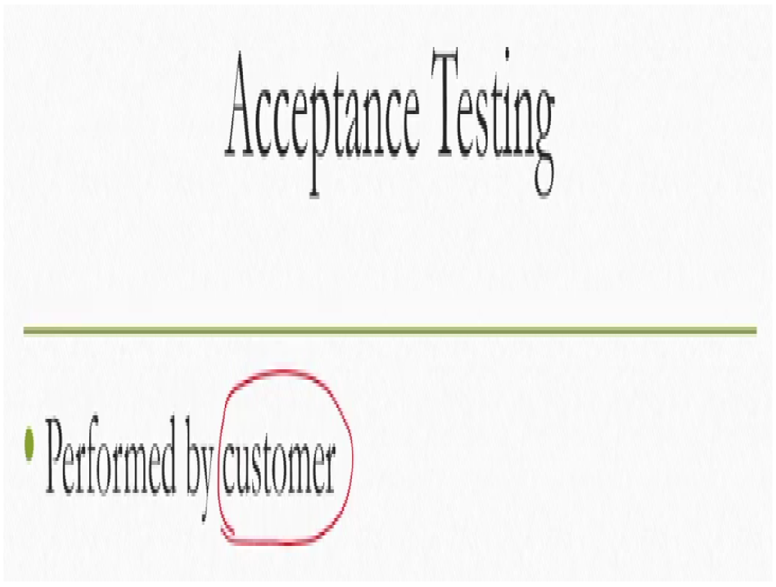


Beta Testing

- Performed by a select group of 'friendly customers' (may be specially 'recruited')

Next is beta testing. Unlike the alpha testing, beta testing is performed by a select group of friendly customers which may be specially recruited. So, in case of alpha testing all the members of the test team's are part of the organization, so no outsiders are involved. In case of beta testing that is not the case, select group of friendly customers are specially recruited to perform the testing. Note that here we are talking about customers, not the end users.

(Refer Slide Time: 12:43)

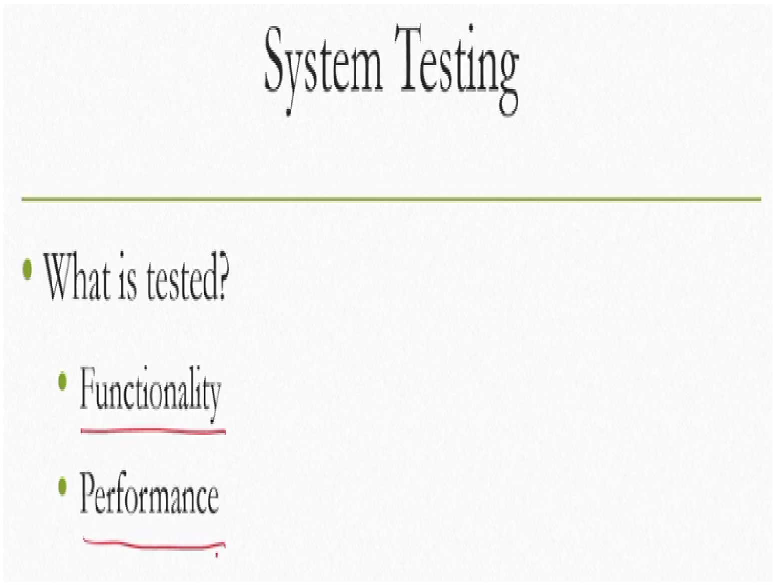


Acceptance Testing

- Performed by customer

The third and final stage of testing is acceptance testing. This is entirely performed by customer. It can be any customer, not necessarily a select group of friendly customers specially recruited. So, this is the last stage where customers provide feedback on the performance of the system. So, we have learned about broad approaches, stages of testing, let us focus on types of testing. What are the different types of system testing?

(Refer Slide Time: 13:24)

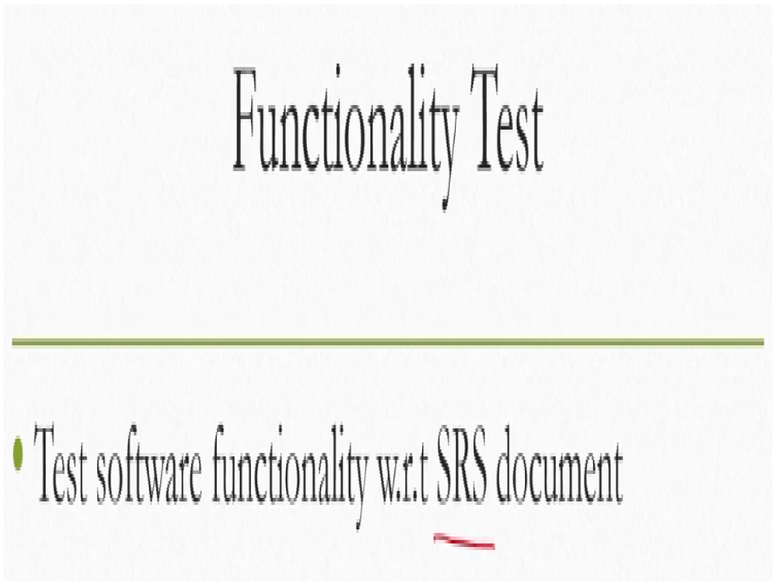


System Testing

- What is tested?
  - Functionality
  - Performance

So, in system testing what do we test? We test functionality and performance. These are the two things that we test.

(Refer Slide Time: 13:31)

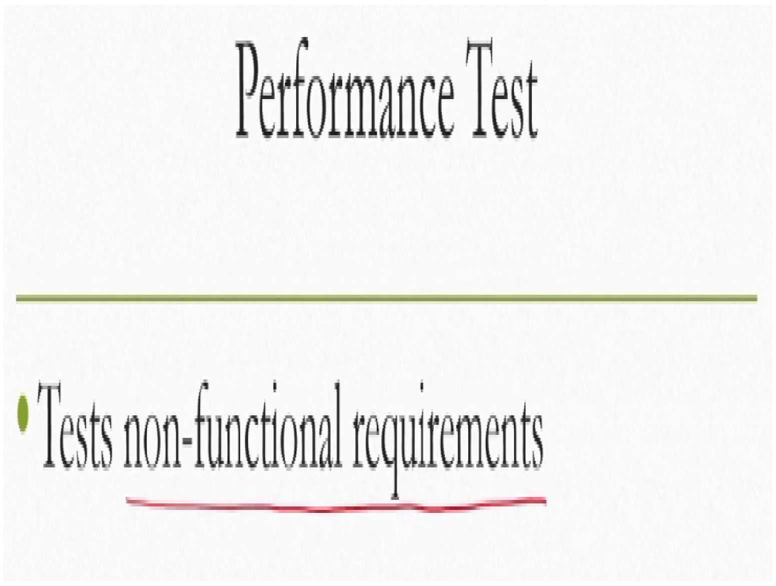


Functionality Test

- Test software functionality w.r.t SRS document

When we test functionality, SRS is our primary document. Whatever is listed in SRS those things we test to see if the system satisfies the functional requirements that are specified in the SRS.

(Refer Slide Time: 13:54)

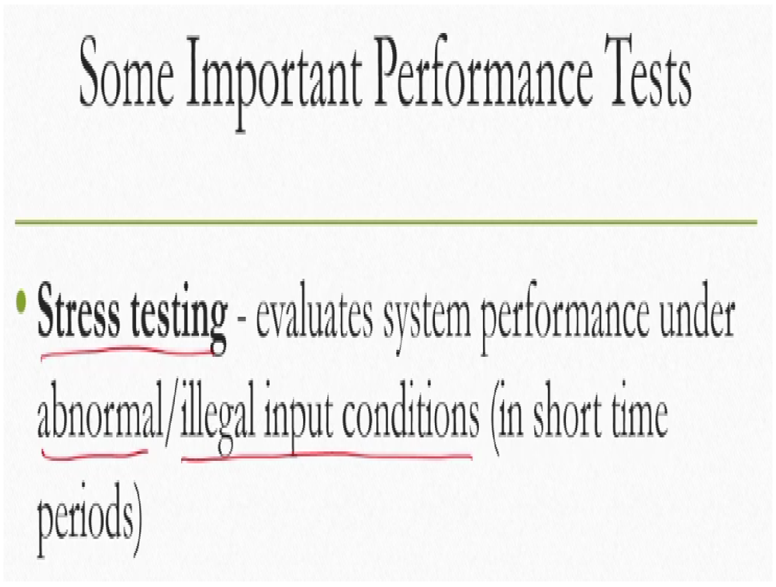


Performance Test

- Tests non-functional requirements

In the performance test, that is the other broad category of tests, what we do is perform non-functional requirement testing. So, in the performance test testing is for non-functional requirements. What are those requirements and how those are tested, let us have a quick look.

(Refer Slide Time: 14:13)

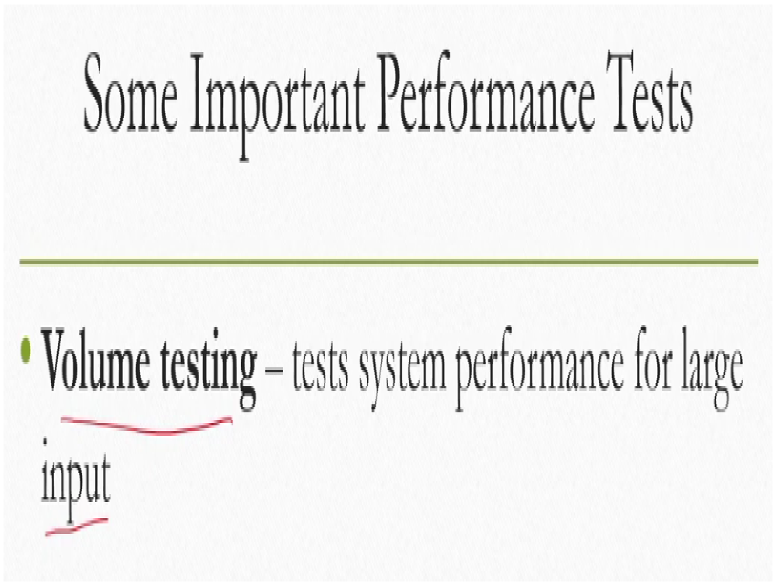


Some Important Performance Tests

- **Stress testing** - evaluates system performance under abnormal/illegal input conditions (in short time periods)

First is stress testing. In this testing system performance is evaluated under abnormal or illegal input conditions in short time periods. So, that is one kind of performance test that we popularly perform or frequently perform during non-functional testing.

(Refer Slide Time: 14:41)

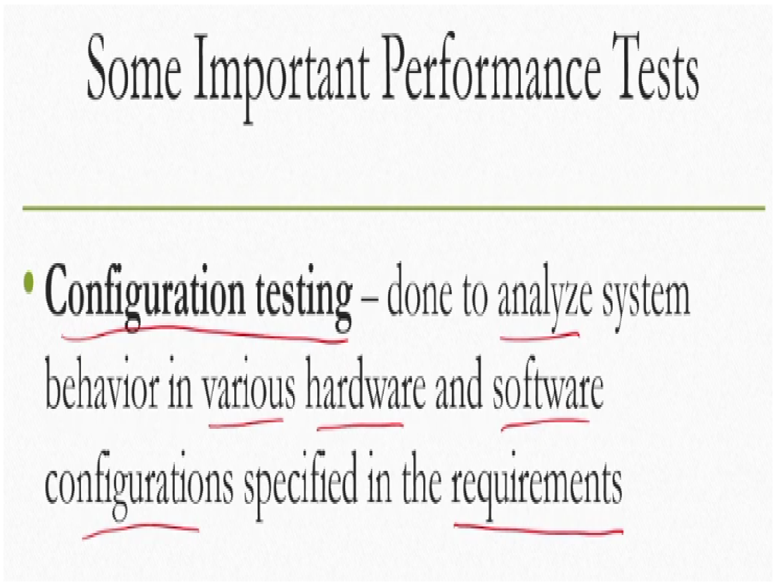


Some Important Performance Tests

- **Volume testing** – tests system performance for large input

Then comes volume testing. Here what we test? We test system performance for large input. So, in the previous case stress testing within a short time span, we provide large amount of input or illegal or abnormal input and test. In case of volume testing, timespan is not the issue rather the amount of data that the system is being asked to process is the issue. So, we test how much data it can process, the volume of the data.

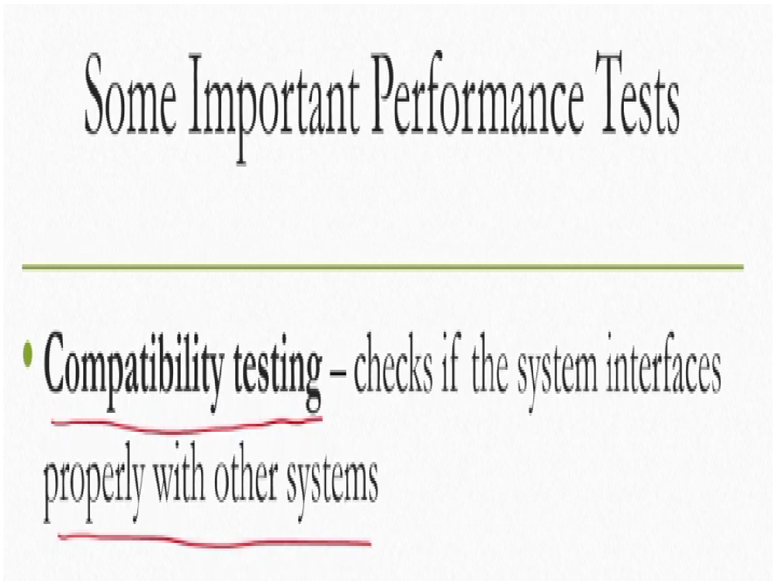
(Refer Slide Time: 15:18)


Some Important Performance Tests

- **Configuration testing** – done to analyze system behavior in various hardware and software configurations specified in the requirements

Then comes configuration testing. So, you have stress testing, volume testing, then comes configuration testing. This testing is done to analyze the behaviour of the system in various hardware and software configurations specified in the requirements. Typically, when you specify the requirements for the system, you specify operating environments including hardware and software. Now, in configuration testing those specifications are tested, whether the system is actually performing under those specifications that we test in configuration testing.

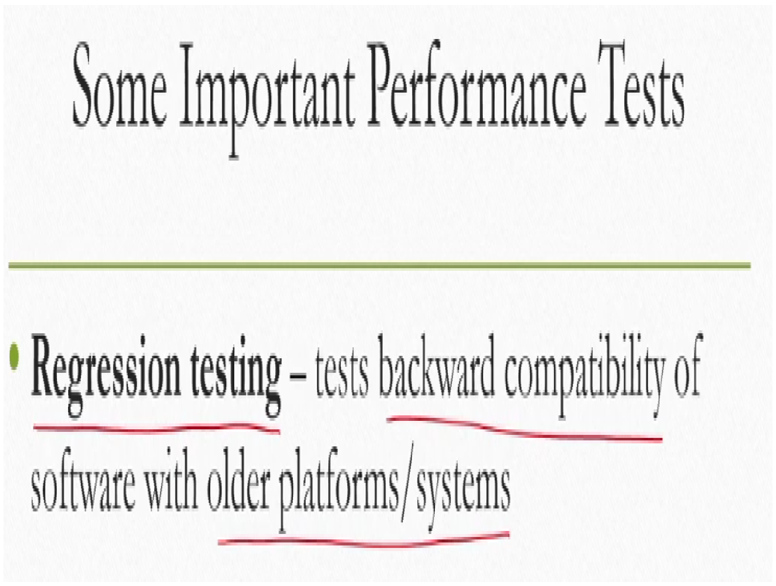
(Refer Slide Time: 16:08)


Some Important Performance Tests

- **Compatibility testing** – checks if the system interfaces properly with other systems

Then comes compatibility testing. So, in this testing we check if the system interfaces properly with other systems. This is another special type of testing, sometimes it may so happen that the system under test needs to work with some other system for that some specific interfaces may be defined, and in compatibility tests we check if those interfaces are working as per expectations.

(Refer Slide Time: 16:45)



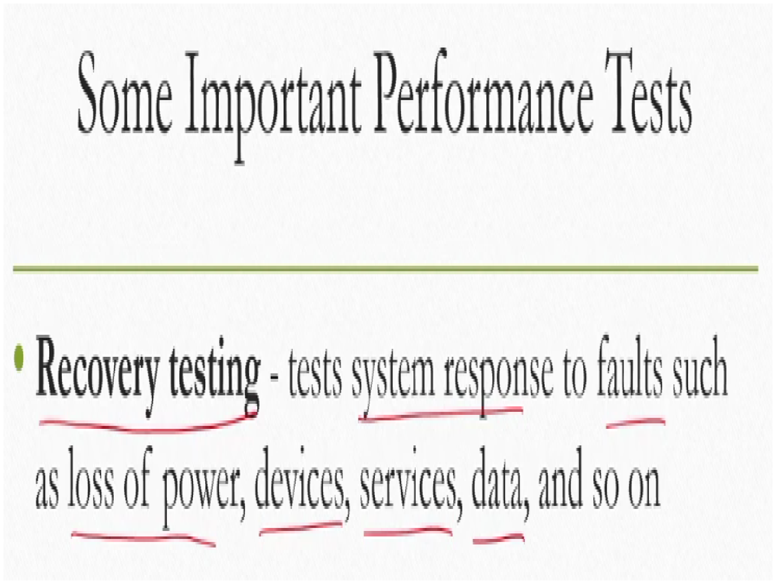
Some Important Performance Tests

- **Regression testing** – tests backward compatibility of software with older platforms/systems

Another testing is regression testing. In this case, we test the backward compatibility of the software with older platforms or systems. Sometimes softwares are upgraded to newer versions. Now, whether those newer versions work with compatible software of older versions those things we need to test and those testing are done under the broad concept of regression testing.

So, in compatibility testing we do not check for backward compatibility, we check for current compatibility whereas in regression testing, we check for backward compatibility.

(Refer Slide Time: 17:34)

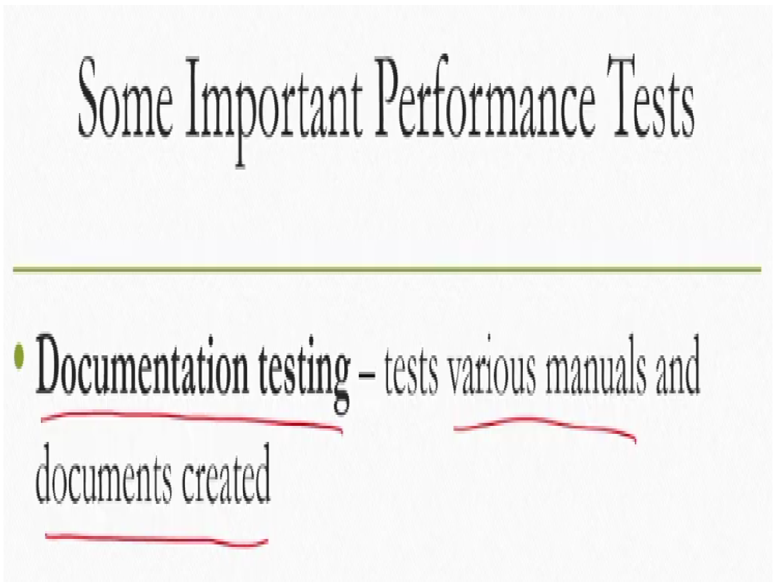


Some Important Performance Tests

- **Recovery testing** - tests system response to faults such as loss of power, devices, services, data, and so on

Then comes recovery testing. Under recovery testing what we test? We test system response to faults, what are those faults? Loss of power, faulty devices, faults in services, faulty data and so on. So, when we perform these types of tests that is recovery testing, so how the system behaves or how the system responds when faulty environment is there. Now, fault can happen in any one or multiple of those things such as loss of power, devices, services, data and so on.

(Refer Slide Time: 18:30)

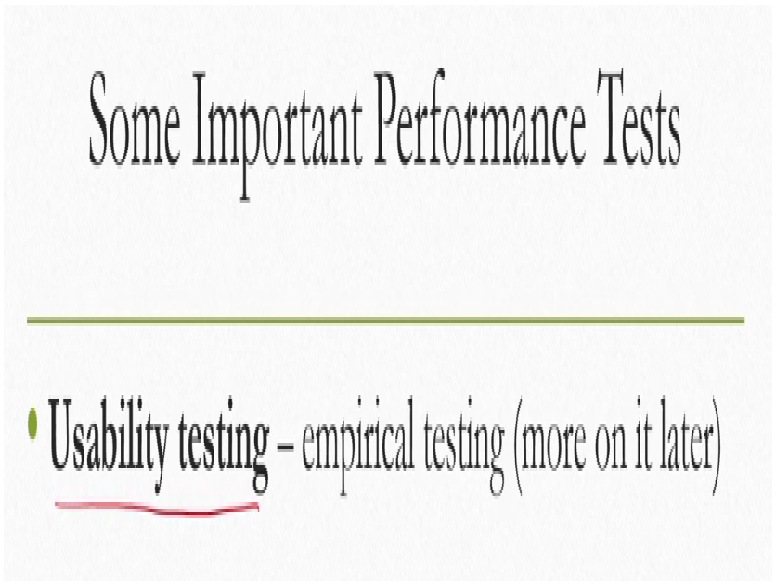


Some Important Performance Tests

- Documentation testing – tests various manuals and documents created

We also have documentation testing, what happens here? As the name suggests in this testing, various manuals and documents are created and tested for understandability. It may appear that this is not a very significant test and any document created should be understandable, but that is not the case. When we create documents, it should be thoroughly proofread and tested for understandability, readability and maintainability, those things are also tested as part of performance test of a system under documentation testing.

(Refer Slide Time: 19:18)



Some Important Performance Tests

- Usability testing – empirical testing (more on it later)

Finally, comes usability testing which is the most important non-functional requirements in the context of our focus that is interactive systems or human=computer interfaces. So, far when we talked about different types of testing at unit level or system level such as review based testing, black box testing, white box testing or these top-down and bottom-up testing. primarily these were meant to test functionalities.

So, the starting point of system design was SRS and these testing are done to see how much the implementation confirm to the requirements specified in the SRS document. Now, as we have noted earlier in case of interactive systems one important consideration is usability. So, functional requirements are not the only thing, usability requirements are also very important to make a successful human-computer interface.

So, far in our testing we did not discuss much about usability requirement testing except in the design prototype evaluate cycle where we mentioned that interface designs are prototyped and evaluated with experts to check for usability issues. But that is a very quick way of testing, we require more rigorous systematic and scientific testing methods which is a major stage in interactive system development lifecycle.

In the subsequent lectures, we are going to talk about usability testing in details. Other performance testing that we have mentioned like stress testing, volume testing, compatibility testing, regression testing, configuration testing and so on we will not devote much time on those and we will have only the brief idea that we have discussed here. However, on usability testing, we are going to devote several lectures next to understand in details how usability testing can be performed with end users in a very systematic and rigorous manner.

So, that is in summary what we need to test when we are testing the full system. Just to recap, we mentioned about two broad approaches to test the whole system, top-down and bottom-up. Earlier we have seen unit level testing in top-down and bottom-up approaches along with unit level testing we test the whole system. When we talk about bottom-up approach, we essentially test the unit individually and then put them together.

And we test the interfacing between the units so that the whole system is tested. In top-down approach we start with the main routine and progress in an iterative fashion by adding immediate subroutines to the main routines and this process goes on the next immediate subroutine to the subroutine attached to the main routine and so on. And this process goes on till we manage to test the whole system.

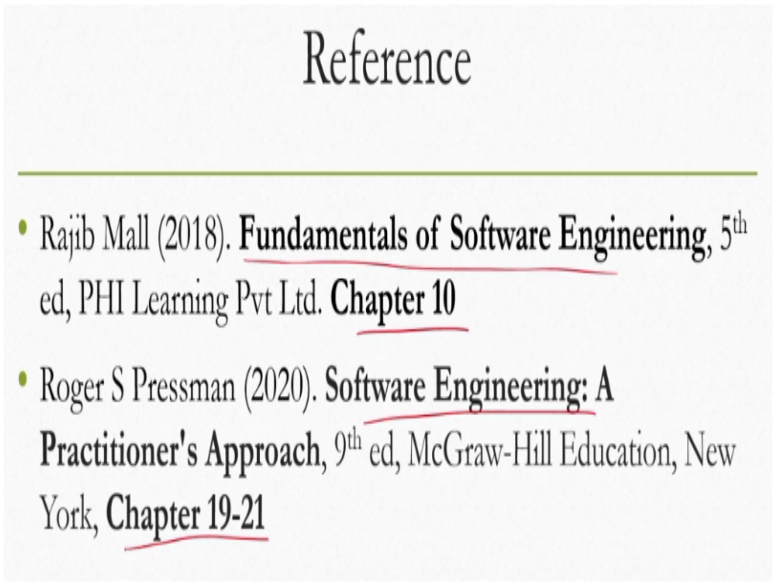
In both the cases, we require additional implementations to support the testing because your whole system is not available at a time. So testing whole system at a time is difficult, so we require additional support. If we are following a top-down approach, then stub modules are used to implement or simulate lower-level responses which are not immediately available to the main routines or the higher routines.

Similarly, if we are following bottom-up approach, then we need to implement driver modules which are used to simulate response of the program for higher-level modules which are not immediately available for the subsystem under test. Now, these top-down or bottom-up approaches are primarily meant for functional testing. Along with that, we also need performance testing. In functional testing or the top-down and bottom-up approaches, there are primarily three stages of testing.

Alpha testing where the testing is done within the organization, beta testing where some selected customers are part of the test team and acceptance testing which is done by the customers. Along with functional testing, we also perform performance testing. Several performance testing’s are there including stress testing, volume testing, compatibility testing, configuration testing, regression testing, documentation testing as well as some other testing’s.

One of the important testing's here is the usability testing, which is our primary focus in this course. In usability testing, we test for usability issues with end users. So, in subsequent lectures we are going to learn about such testing methods and the process involved.

(Refer Slide Time: 24:56)



Reference

- Rajib Mall (2018). **Fundamentals of Software Engineering**, 5th ed, PHI Learning Pvt Ltd. **Chapter 10**
- Roger S Pressman (2020). **Software Engineering: A Practitioner's Approach**, 9th ed, McGraw-Hill Education, New York, **Chapter 19-21**

Whatever we have discussed today can be found in these books excluding details of usability testing of course, but the broad ideas of these different testing methods can be found in these books; Fundamentals of Software Engineering chapter 10 and Software Engineering A Practitioner's Approach chapter 19 to 21. Of course, in these chapters you will find the concepts in more details than how we covered.

For our purpose, this brief description is sufficient to explain the concepts. Next, we will focus more on details of usability testing. Hope you have enjoyed the lecture and understood the concepts. With this, we have come to the end of our discussion on the code testing stage. As you can quickly recap in this code testing stage, we have learned several concepts mostly focused on unit level testing that includes code review, execution-based testing.

And also, we have covered in brief ideas that are important for system level testing. Next, we are going to talk about usability testing, looking forward to meet you all in the next lecture. Thank you, and goodbye.