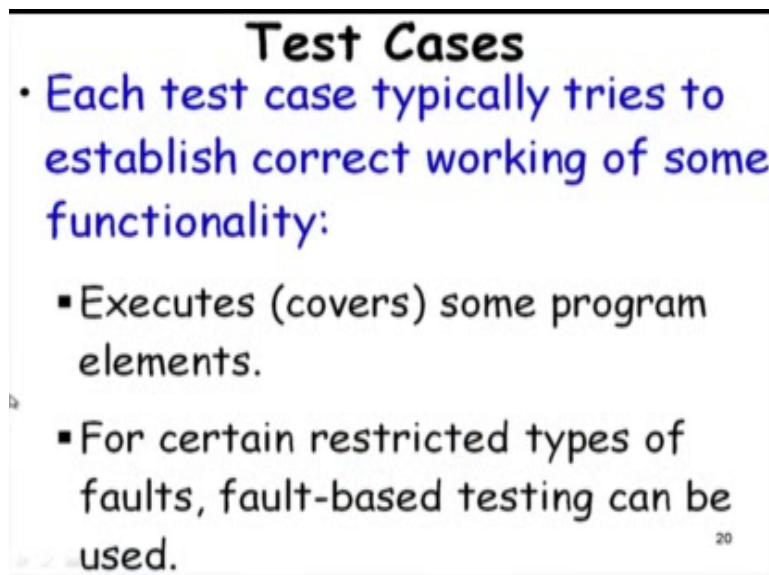


Lecture – 04
Basic Concepts of Testing (Contd.)

Welcome to this session. Now let us continue with some of the basic concepts that we were discussing in the last session. So, we were discussing about test cases. Software is tested with a large number of test cases, and this set of test cases is called as Test Suite.

(Refer Slide Time: 00:53)



Test Cases

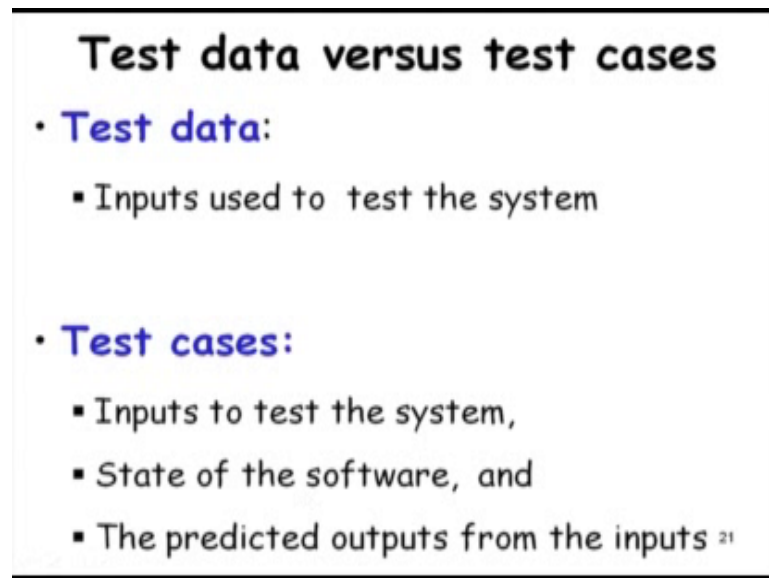
- Each test case typically tries to establish correct working of some functionality:
 - Executes (covers) some program elements.
 - For certain restricted types of faults, fault-based testing can be used.

20

Each test case basically executes some functionality of the software. And as it executes the functionality, it covers some program elements; it executes or covers some program elements. The program elements can be statements, or conditionals or some jumps etcetera. But then we measure the coverage that is achieved by a number of test cases and check whether the required number of the required elements have been covered. So, these are called as coverage base testing, where we try to ensure that the targeted elements are covered.

On the other hand, we also have few fault based testing, where we do not target really coverage, but these try to expose that whether certain types of bugs have been tested and removed.

(Refer Slide Time: 01:58)

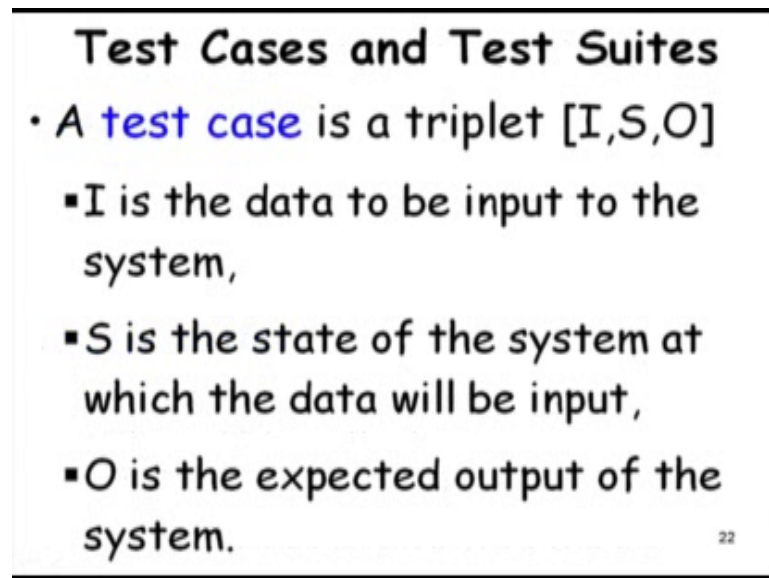


Test data versus test cases

- **Test data:**
 - Inputs used to test the system
- **Test cases:**
 - Inputs to test the system,
 - State of the software, and
 - The predicted outputs from the inputs ²¹

Now, let us distinguish between test data and test cases. A test data or the data which we input for testing, but that test case not only have test input that is test data, but also a test case denotes the program state at which we apply the data. For example, we might have chosen certain menu item or may be logged into the system and so on, so that is the state of the software and also the expected result. So, once the test case has been designed and documented in this form that, what is the state at which we will apply the input that is logged in, chosen some menu item. And then we input the test data and then what will be the result, so that the tester during testing can just carry out this procedure and check.

(Refer Slide Time: 03:07)



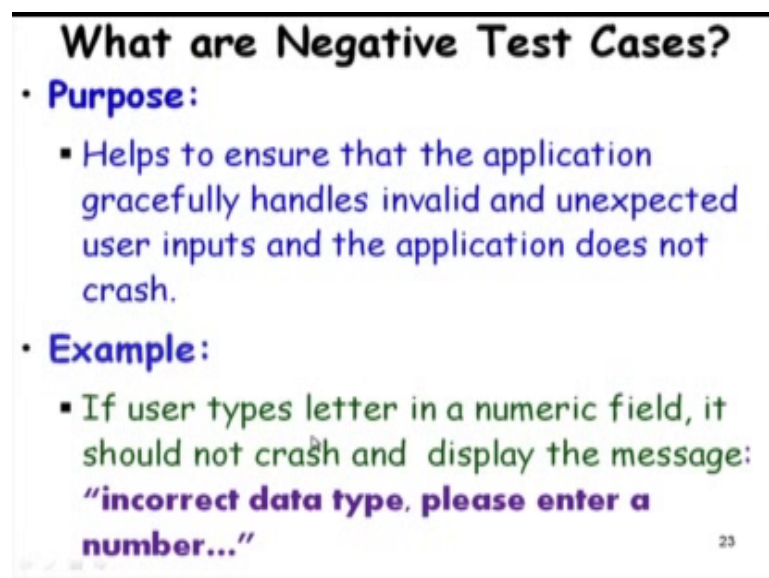
Test Cases and Test Suites

- A **test case** is a triplet [I,S,O]
 - I is the data to be input to the system,
 - S is the state of the system at which the data will be input,
 - O is the expected output of the system.

22

So, we can say that a test case at the minimum is a triplet. Input - the test data, input is the test data; S is the state of the software at which the input is to be applied; and O is the expected output.

(Refer Slide Time: 03:29)



What are Negative Test Cases?

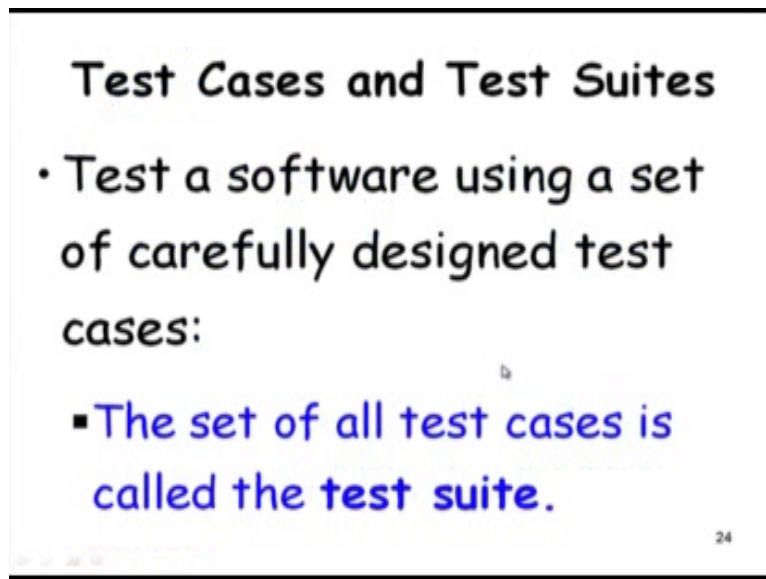
- **Purpose:**
 - Helps to ensure that the application gracefully handles invalid and unexpected user inputs and the application does not crash.
- **Example:**
 - If user types letter in a numeric field, it should not crash and display the message: **"incorrect data type, please enter a number..."**

23

But before we proceed further, let us look at another very basic concept. We design both positive test cases and negative test cases. Positive test cases, they check that given a valid input, whether the program is working satisfactorily, but what if, the user gives input which is not really intended to be given.

For example, let us say the programmer typed the character where a number was to be written; so instead of 23, he wrote some a, b, c will the software crash so that we will call as negative test cases. A negative test case handles whether the software is well behaved when invalid and unexpected inputs are given. So, the positive test cases check the working of the software, when the inputs are valid. Whereas the negative test cases, check that the program behaves gracefully when invalid inputs or unexpected inputs are given.

(Refer Slide Time: 04:56)



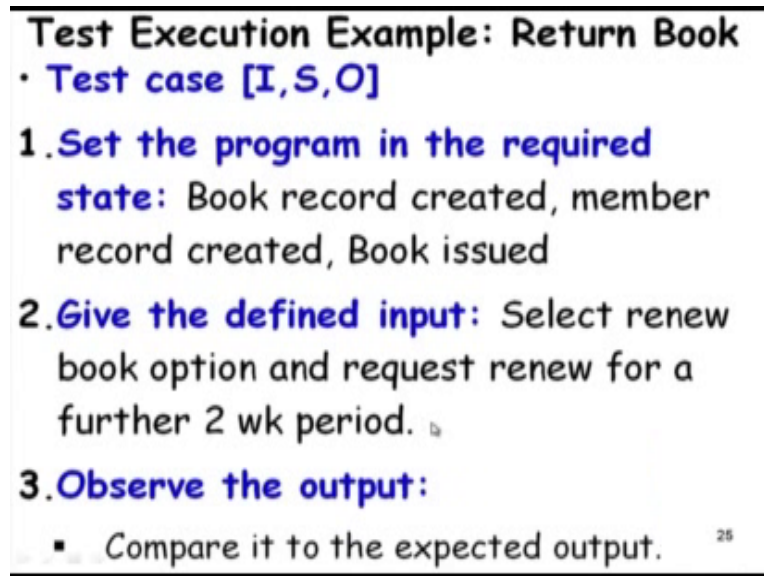
Test Cases and Test Suites

- Test a software using a set of carefully designed test cases:
 - The set of all test cases is called the test suite.

24

Software is tested using a large number of test cases, which are designed based on some tests strategy. We will look at many tests strategies, many black box strategies, and white box strategies. And these test cases, the set of all test cases that are designed is called as the test suite that is the terminology that is typically used.

(Refer Slide Time: 05:29)



Test Execution Example: Return Book

- **Test case [I,S,O]**

1. **Set the program in the required state:** Book record created, member record created, Book issued
2. **Give the defined input:** Select renew book option and request renew for a further 2 wk period.
3. **Observe the output:**
 - Compare it to the expected output.


25

Let us look at one example test case. Let us say we have library software; and we are trying to write one test case for return book case; taken the library example, because everybody is familiar with such a software. We have taken book and they are trying to return it.

So, what will be the test case? Here, in the test case, we write the state of the test case; the state of the software is the book has been created, member record has been created and then after that the book has been issued to the member, so that is a state at which the input data - the test data is to be applied. And the test data is renew request for a 2 week period. And then the output is, observe the output which is intuitive that whether if the book is renewed or not.

(Refer Slide Time: 06:52)

Sample: Recording of Test Case & Results	
Test Case number	
Test Case author	
Test purpose	
Pre-condition:	
Test inputs:	
Expected outputs (if any):	
Post-condition:	
Test Execution history:	
Test execution date	
Test execution person	
Test execution result (s) : Pass/Fail	
If failed : Failure information	
: fix status	



There are prescribed formats that have been designed with lot of thought for documenting the test case. So, we just discussed a triplet, which was the very least to describe a test case, but just see here a more elaborate format for recording test case.

For example, writing the test case number, test case author, test purpose, the precondition that is the state at which the inputs are to be applied, the test input, what are the outputs and then what is a post condition, what would be the program state after the test has completed, and then test execution date, the person conducting. The test result - pass or fail; and if fail, what are the details of the failure, what really happened, and then what is the fix status, which is once this is given to the developer; they would try to fix it and they would write the fix status.

(Refer Slide Time: 08:06)

Test Team- Human Resources

- **Test Planning:** Experienced people
- **Test scenario and test case design:** Experienced and test qualified people
- **Test execution:** semi-experienced to inexperienced
- **Test result analysis:** experienced people
- **Test tool support:** experienced people
- May include external people:
 - **Users**
 - **Industry experts**




Now, in the test team, there are various categories of testers, who do different activities. Now let us look at what are the different types of testers that can be there. For example, we might have test case planning, so for which, we need very experienced testers. Test scenario and test case design; a test scenario as you will see is that a very high level description of a test case, where is a test case is the actual test case where we write all the details like what exactly is the input data, what state and so on. Now we fill up that template, just we showed the previous slide. Here also for test case design, we need experienced and qualified people.

For test execution, actually taking the test cases and executing manually first time, second time of course, we can record and replay. So, the first time, semi experienced to inexperienced persons, they can give the input, take the software to a specific state give the input and observe whether the software is behaving as expected. Test result analysis, so this also need experienced people, what really happened during the testing; was it a failure and so on. And then test tool support, so if test tools are used supporting those we need experienced people; and not only these we might need external persons for example, users for performing usability tests and so on and also the industry experts.

(Refer Slide Time: 10:02)

Why Design of Test Cases?

- Exhaustive testing of any non-trivial system is impractical:
 - Input data domain is extremely large.
- Design an **optimal test suite**, meaning:
 - Of reasonable size, and
 - Uncovers as many errors as possible



But one basic question that we need to answer at this point before proceeding further is that why designed test cases, what is the advantage compared to random input of test cases. Why not give random inputs 10,000 values and that would be much more time effective; in an hour, we can give 1,000 inputs may be we will ask 5 different testers to give keep on giving inputs. So, why do we have to design test cases? The answer is that even if we test with 10,000 random inputs, we might not have tested it well; the main reason is that many of those random inputs might be trying to test the same type of bugs or they might be covering the same type of program elements.

(Refer Slide Time: 11:05)

Design of Test Cases

- If test cases are selected randomly:
 - Many test cases would not contribute to the significance of the test suite,
 - Would only detect errors that are already detected by other test cases in the suite.
- **Therefore, the number of test cases in a randomly selected test suite:**
 - **Does not indicate the effectiveness of testing**

29

The test cases may not be effective. So, just saying that we tested with 10,000 randomly generated test cases may not mean much.

(Refer Slide Time: 11:24)

Design of Test Cases

- Testing a system using a large number of randomly selected test cases:
 - Does not mean that most errors in the system will be uncovered.
- Consider following example:
 - Find the maximum of two integers x and y .

30

Just to give an example - that we will show it with a small program, finding maximum of two integers x and y , very small program, just two line.

(Refer Slide Time: 11:34)

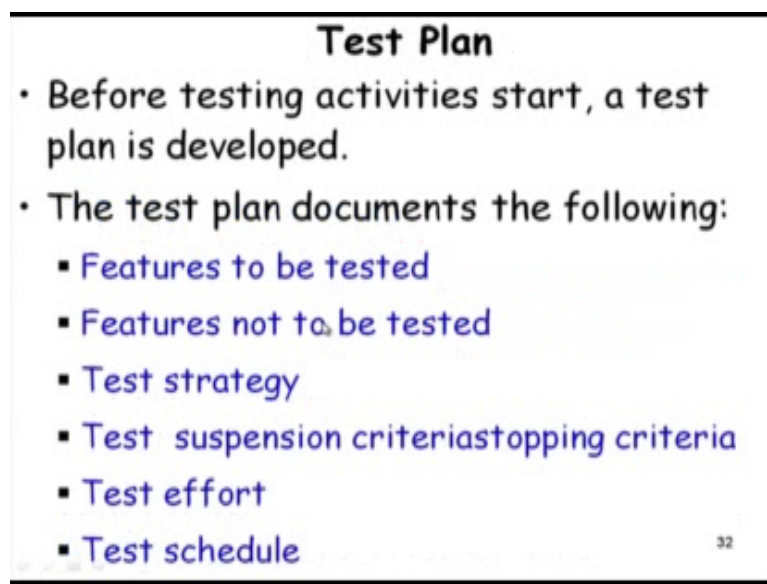
Design of Test Cases

- The code has a simple programming error:
- If $(x > y)$ $\text{max} = x$;
 else $\text{max} = x$; // should be $\text{max} = y$;
- Test suite $\{(x=3, y=2); (x=2, y=3)\}$ can detect the bug,
- A larger test suite $\{(x=3, y=2); (x=4, y=3); (x=5, y=1)\}$ does not detect the bug.

31

If x greater than y , max is equal to x ; else max is equal to y , this is what the programmer wrote, but then the bug is here should have written max is equal to y , else max is equal to x . But then let us see the test cases with which it tested. So, somebody tested with 3 2, 4 3, 5 1, etcetera, etcetera; hundreds of test cases where used, but then that would not expose this bug, because each time only one statement is executed; the other statement is not even executed. So, just saying that we tested with 1,000 test cases and each time x is larger than y does not bring out the bug; whereas, just two test cases would bring out a bug in either of these statements.

(Refer Slide Time: 12:34)



Test Plan

- Before testing activities start, a test plan is developed.
- The test plan documents the following:
 - Features to be tested
 - Features not to be tested
 - Test strategy
 - Test suspension criteria/stopping criteria
 - Test effort
 - Test schedule

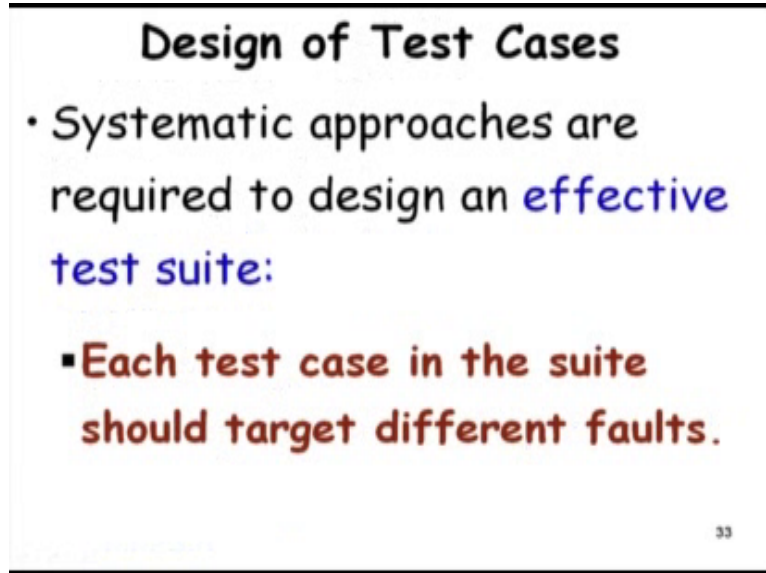
32

Another basic concept that we will discuss before looking at different testing strategies and test case design is a Test Plan - before the testing starts, a test plan is produced. So, what does a test plan contain, a test plan is what are the features to be tested, what are the features not to be tested, may be for a specific release, we are not planning to release some features, they have been included, but then not planning to release to the customer. So, we need not test that.

What are the different test strategies that will be employed? So in black-box testing, do we do equivalence testing, boundary value testing, condition testing, and combinatorial testing etcetera, etcetera? So, what are the different types of testing strategies that we will use? And test suspension and criteria, so here this is basically a stopping criteria. So, if we say that if some core features do not work, then we do not test it further, we just give it to the developer to fix the core functionality before the other functionalities can be tested. And there is estimate of test effort and also test schedule how many days expected to test. So, this is a typical test plan that is developed before the

testing starts.

(Refer Slide Time: 14:13)



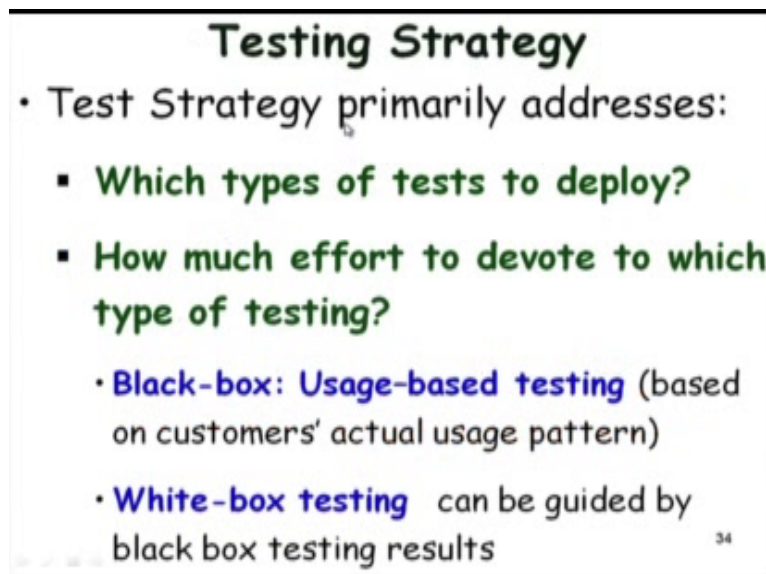
Design of Test Cases

- Systematic approaches are required to design an **effective test suite**:
 - **Each test case in the suite should target different faults.**

33

Now, let us look at the design of test cases. So, in designing test cases, one thing is clear that as far as possible the different test cases so target to detect different types of bugs, because hundreds of test cases, trying to detect the same bug will be a waste of effort. They would not achieve too much in exposing the other types of bugs that are present.

(Refer Slide Time: 14:43)



Testing Strategy

- Test Strategy primarily addresses:
 - **Which types of tests to deploy?**
 - **How much effort to devote to which type of testing?**
 - **Black-box: Usage-based testing** (based on customers' actual usage pattern)
 - **White-box testing** can be guided by black box testing results

34

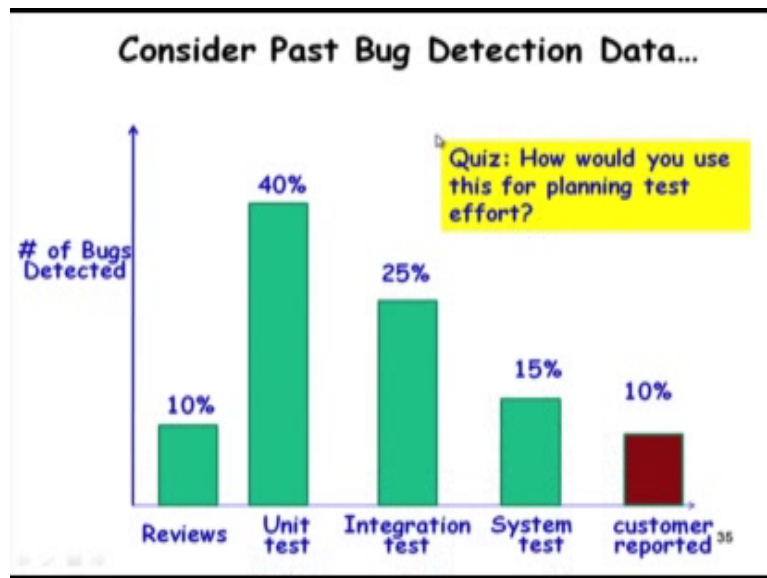
So, we will have many tests strategies. Each strategy is actually a bug filter, and we need many of these strategies. So, during test planning, we decide which types of tests, which strategies to deploy, how much effort to be given for each strategy. So, do we look at the equivalence tests, 100 hours, and then the condition test 5 hours or is it vice versa. The equivalence test will give 5 hours and the condition testing 100 hours, so not only which strategy is to deploy, but how thoroughly or how much time we give to that strategy during testing.

And then for black-box testing do we use a usage-based testing, so that is based on how the customer actually uses. So, some features usages are used by the customer very heavily and other features not so much. So, do we test in proportion to how the in what frequency the users use it. Actually, it is a good idea to give as much effort to a feature as the users use it.

Some features is used very heavily, for example, in a library software, book issue and book return are features that are used heavily, but a book lost report feature may not be used that much; reporting a book lost may not be used that much compared to a book issue and book return. The idea here is that the book issue and the book return features, if there even small problems, bugs are present. This will be noticed by the users, but the book lost feature is used very rarely; and if there are few problems there that may not be noticed immediately. So, the usage-based testing is one thing that to be considered while doing the test planning.

And then the white-box testing, which is carried out after the black-box testing. So, this we might get guided by the black-box testing result. So, what do you mean by the guided by black-box testing result. During black-box testing, we might find out the specific components the software which have bugs; the specific types of features. And then in white-box testing, we might spend more time testing those features which were showing bugs, because one very peculiar thing bugs is that they occur in clusters. If some part you detect bugs, there will be more bugs there. They live in community large number of bugs will be living there. Whereas the parts that are relatively very few bugs are getting reported, there will be actually less bugs to look for.

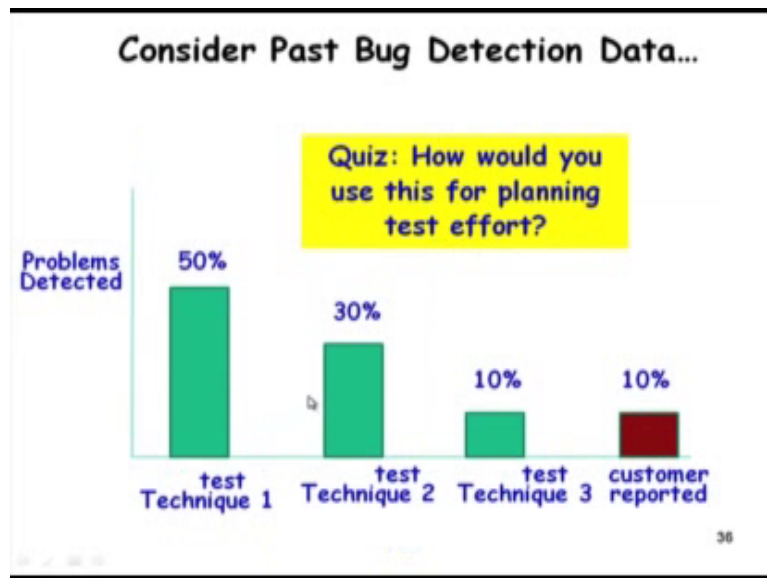
(Refer Slide Time: 18:08)



Now, based on that let me just ask this question that our passed data indicates that reviews detect about 10 percent of the existing bugs, I mean this not real situation, just an example that reviews detected about 10 percent of the bugs. Unit testing exposed about 40 percent of the bugs; integration about 25 percent; system test 15 percent and customers finally, when the software was shift to them, they reported 10 percent. So, this is the data for a company which has been shipping software to its customers based on may be 30 or 40 software that has shipped it has collected the data, how much bugs are getting exposed by each of this techniques.

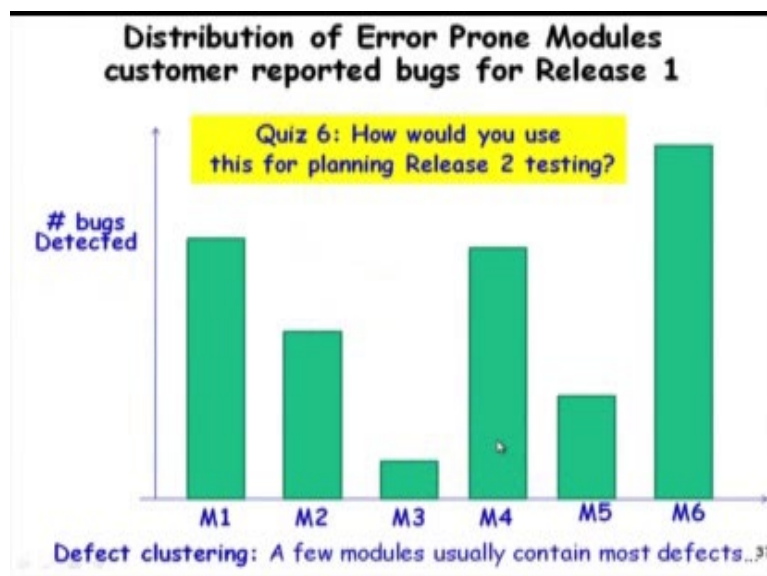
So, how would the company or the test manager plan the test effort? The answer is not very difficult. Since unit testing is exposing more number of bugs, it is a very effective technique, and therefore, we should spend more time on unit testing compare to let us say reviews, system testing if this is the data, but then it is not real data, a hypothetical data. In a real data reviews, actually expose lot of bugs.

(Refer Slide Time: 19:50)



Now, let us look at another one, a test planning that we are using many test strategies. So, strategy 1, which is may be equivalence partitioning, boundary value and may be condition testing. So, the first one detected 50 percent of the problems, second one 30 percent, third one 10 percent. So, how would we plan the test effort? Here also the answer is easy. If test technique one is effective, we should do it well. So, it should give enough time for test technique one and then we should spend time on test technique two and finally, much less time on test technique three.

(Refer Slide Time: 20:50)

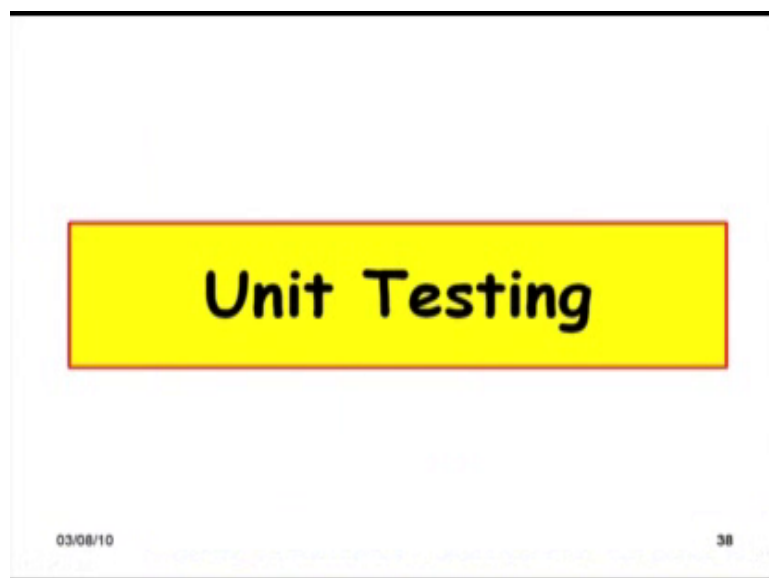


This is another data that while testing doing the black-box testing; we found that different modules had different bug contents. Module 1 had much more bug then module 3, but module 6 had the highest number of bugs. So, how do we do the white-box testing? So, we have to do this much more thoroughly. And the situation may occur when we have released software, and then we are trying to release the version 2 of that software. So, there also we might do similar planning.

The main reason here as we are discussing is defect clustering. A few modules usually contain most of the defects so that is the observation what is the reason, the reason is that possibly that module was very complex. And therefore, most of the defects are there in that module the other modules which are either straightforward, there are less number of problems or may be one module is written by very experienced programmer who understood the problem very well, and there were very few problems.

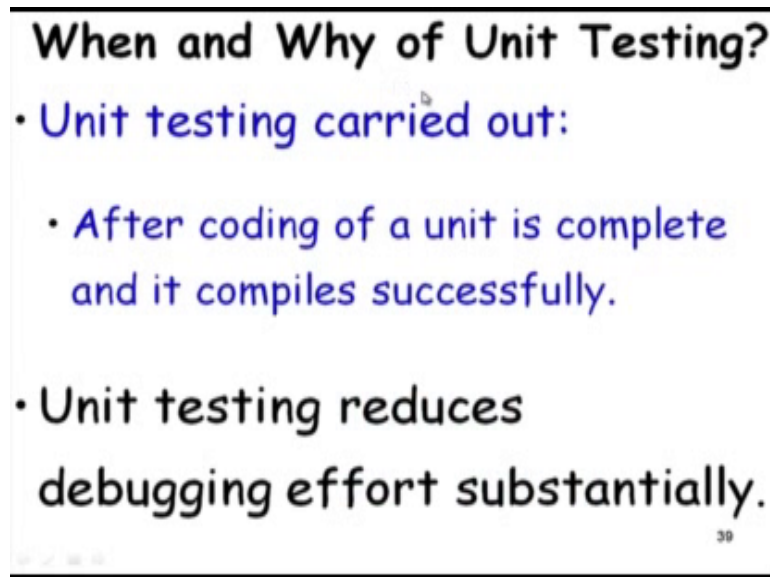
But there is another module written by a first time programmer, did not understand because was not very conversant to the language and neither with the algorithm and so on, so there will be large number of bugs. So, we need to spend more time testing that module, we should not spend uniform time in testing all the modules.

(Refer Slide Time: 22:47)



Now, let us start our discussion unit testing. We just said earlier that unit testing is undertaken once the code for a unit is complete; a unit we said can be a function; it can be a module; in object orientation, it can be a class; it can be a component in a component base development. In unit testing, we start the testing activity as soon as the code for that unit has been written and has been compiled. So, compilation errors have been removed, syntax errors.

(Refer Slide Time: 23:36)



When and Why of Unit Testing?

- Unit testing carried out:
 - After coding of a unit is complete and it compiles successfully.
- Unit testing reduces debugging effort substantially.

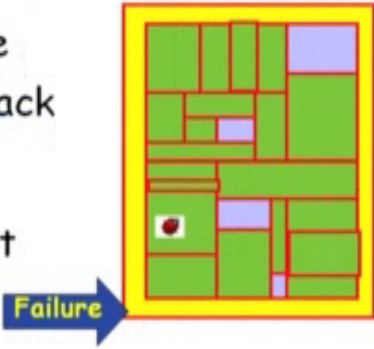
39

Now, let us look at some basic concepts in unit testing before we start looking at the test case designing. So, unit testing is carried out after the unit coding is complete and it compiles successfully.

(Refer Slide Time: 23:52)

Why unit test?

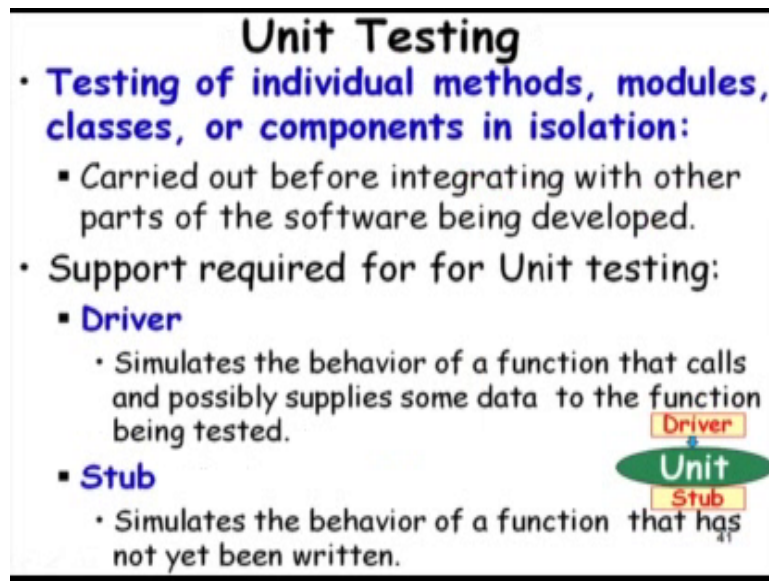
- Without unit test:
 - Errors become difficult to track down.
 - Debugging cost increases substantially...



But before we look at how really to design unit test cases, let us answer a basic question that cannot we just do the system testing well, why tested the unit level, cannot we just spend all our effort on doing a good system testing, why do unit testing and then finally, do integration and system test.

Let us look at the answer here. This is an example where we are there is a bug and while doing the system testing failure was reported. Let us assume that unit testing, we did not spend time, we are doing a thorough system testing. And each, we do a test, and just find a failure, and then there are some bugs which cause the failure. Now, have this entire thing to look at, may be the code is tens of thousands or fifty thousand lines long, and we have to look through there somewhere trying to find out where the bug is.

(Refer Slide Tie: 25:11)



On the other hand, if the bug is found after the unit testing, and when we observed the bug, we might have a very small number of code to look at to easily identify the bug. So, unit testing can think of that it reduces the testing effort, debugging effort, because system testing same bug may be caught, but then it will be very expensive to correct it. Whereas unit testing we can easily localize the bug, because the lines of code to look for are very small compared to system testing.

Now, in unit testing, we test the units in isolation. What it really means is that if the unit has a driver or some other unit needs to call it, and it needs to call some other units to get the result then we have to write the driver in stub before we can do the unit testing. So, driver is the one which calls the unit, provides some data to it; and the stub is the one which the unit needs to call and uses the result of the other unit to produce its results.

Before we start unit testing, we might have to write the drivers and stubs for the unit. So, we will just stop at this point, and we will continue from this in the next session.

Thank you.