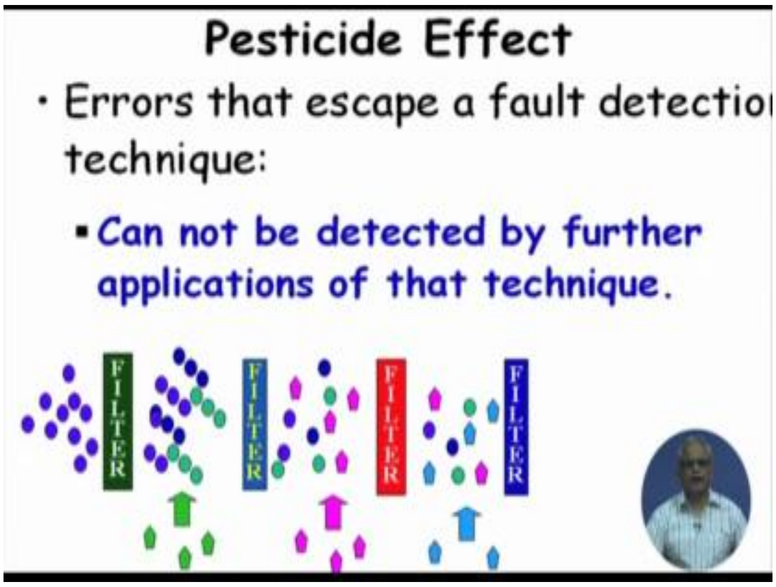
Software Testing

Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 03
Basic Concepts of Testing

Welcome to this session. We will continue from what we were discussing last time. Last time we were looking at different levels of testing and few basic concepts. So will continue with the basic concepts that we are discussing, before we look testing in some detail.

(Refer Slide Time: 00:46)



Pesticide Effect

- Errors that escape a fault detectio
  technique:
  - Can not be detected by further applications of that technique.

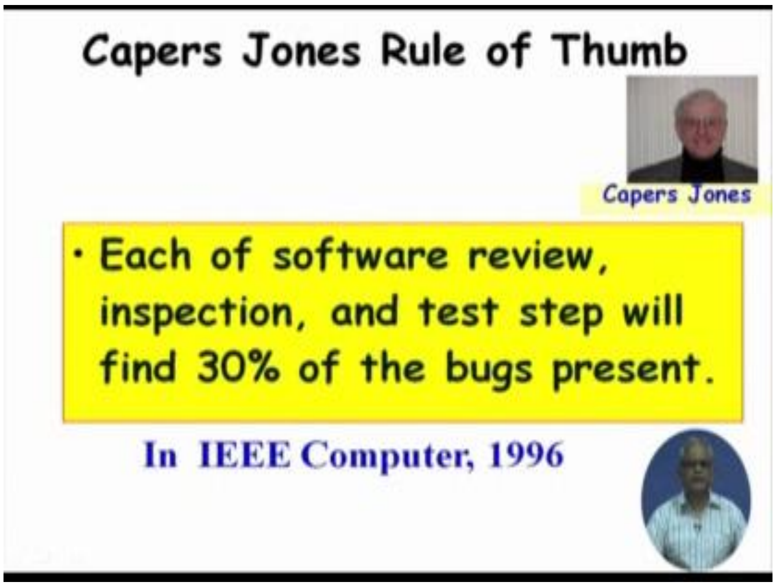
Last session we started discussing about this very important concept of the Pesticide Effect. We were saying that there is a good analogy of testing technologies, use of testing technologies with use of pesticide by a farmer. We said that when bugs infests a crop, farmer uses ( ) pesticides and bugs get killed, but then the bugs which survive a pesticide, develop resistance to the pesticide and the same pesticide is not effective any more, need other types of pesticides. There is a rough analogy here in testing.

As we were saying that there are many testing techniques, there are a dozen black bugs testing and may be more than two dozen white box testing techniques, and also there other bug removal techniques like, review, simulation and so on. Each of this bug removal technique whether it a review, stimulation, black box testing, white box testing, unit, integration, system testing, we can consider all of them as different type of bug removal filters.

I have colored these different types of bug removal techniques as different colored filters. Meaning that, they are effective against some type of bugs. So, just see here initially there were some bugs and then we used a bug detection technique, may be review. Then those bugs have got reduced, but some have survived. And those which are survived a filter further applications of the same filter is not very effective.

So that is what a written here, bugs that escape a fault detection technique cannot be detected by further applications of that technique, very very fundamental concept. And, when we fix the bugs that have been eliminated and when we make other changes to the software, new types of bugs get introduced. After each filter some bugs get eliminated, new bugs appear and then we use a different type of filter and then, the bugs that survive this filter would not get eliminated by further applications of the same filter we keep one using new filters or new bug removal techniques.

(Refer Slide Time: 03:55)



Capers Jones Rule of Thumb

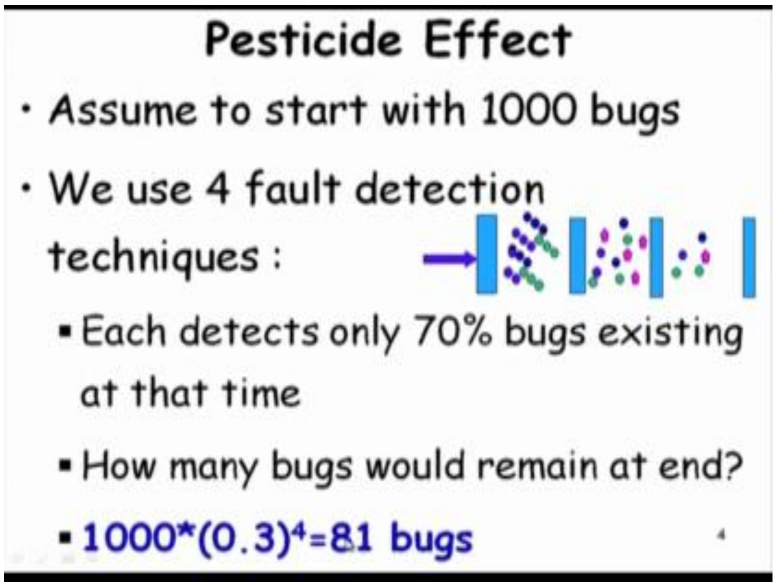
Capers Jones

- Each of software review, inspection, and test step will find 30% of the bugs present.

In IEEE Computer, 1996

Capers Jones, who is an authority in this area, published a landmark paper, IEEE Computer 1996. He said each of software review inspection and test step finds about 30 percent of the bugs present. So, 70 percent bugs escape every bug filter that was his observation in IEEE Computer, 1996.

(Refer Slide Time: 04:25)



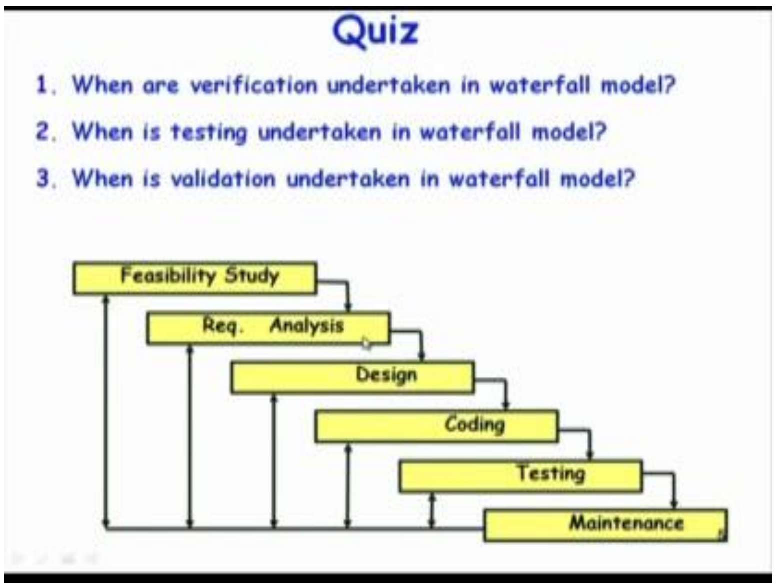
Pesticide Effect

- Assume to start with 1000 bugs
- We use 4 fault detection techniques :
  - Each detects only 70% bugs existing at that time
  - How many bugs would remain at end?
  - $1000*(0.3)^4$=81 bugs

Now, let us do a small problem. Let us assume that 1000 bugs are present before we start any bug detection technique. We use 4 ( ) bug detection techniques, and each is able to detect 70 percent of the bugs. So very effective filters, removing 70 percent of the bugs and as we observe Capers Jone saying that only 30 percent of the bugs get filter by bug detection technique, but here we are considering 70 percent bugs are detected in eliminated under each bug detection technique.

After we apply all the 4 techniques, how many bugs will survive assuming that no new bugs are introduced? To answer this question is not very difficult we say that 1000 was the initial bugs and each time we apply a bug filter, 30 percent survive. At the final, after 4 bug detection techniques have been applied $1000*(0.3)^4$ roughly equal to 81 bugs will survive.

(Refer Slide Time: 05:57)



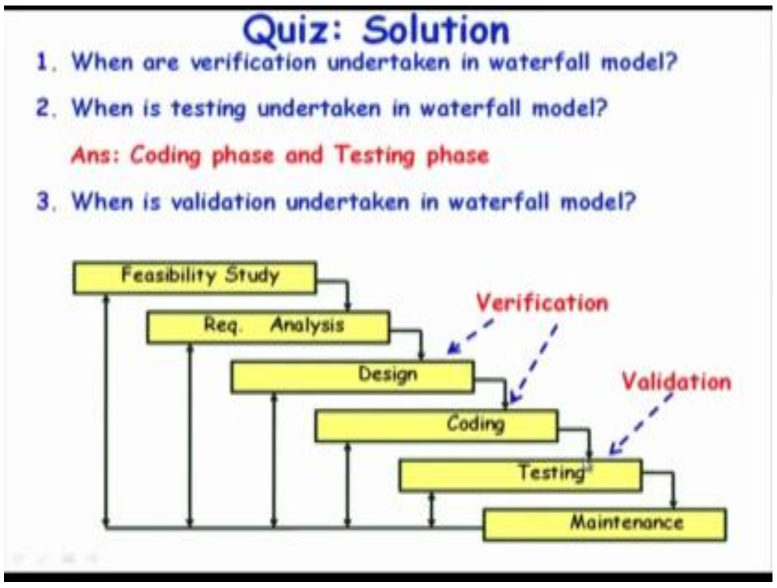
Quiz

1. When are verification undertaken in waterfall model?
2. When is testing undertaken in waterfall model?
3. When is validation undertaken in waterfall model?

Feasibility Study
Req. Analysis
Design
Coding
Testing
Maintenance

Now before we look at further concepts small quiz based on what we have discussed in the last two sessions. We have shown here a water fall model, iterative water fall model. And the question is that, in which phases verification activities are undertaken? The answer to this question is that verification is undertaken during Requirement Analysis, Design and Coding. All stages verification is undertaken to check whether they conform to the previous step.

(Refer Slide Time: 06:25)



Quiz: Solution

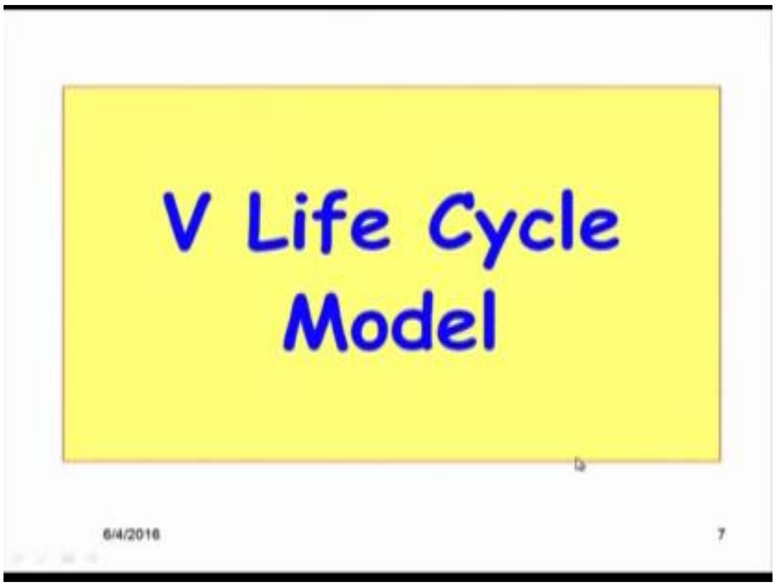
1. When are verification undertaken in waterfall model?
2. When is testing undertaken in waterfall model?

   Ans: Coding phase and Testing phase
3. When is validation undertaken in waterfall model?

Feasibility Study
Req. Analysis
Design
Coding
Testing
Maintenance
Verification
Validation

When is testing undertaken in water fall model? There are two types of testing, one done by the programmer himself, the Developer, which is unit testing; which is undertaken during coding and unit testing. So, testing is undertaken in coding phase and also integration in system testing takes place in the testing phase. Both coding phase and testing phase, testing activities are undertaken. When is validation undertaken? Validation essentially is system testing where we test the software for conformance ( ) to the requirement specification and that is undertaken in the testing phase.

(Refer Slide Time: 07:46)



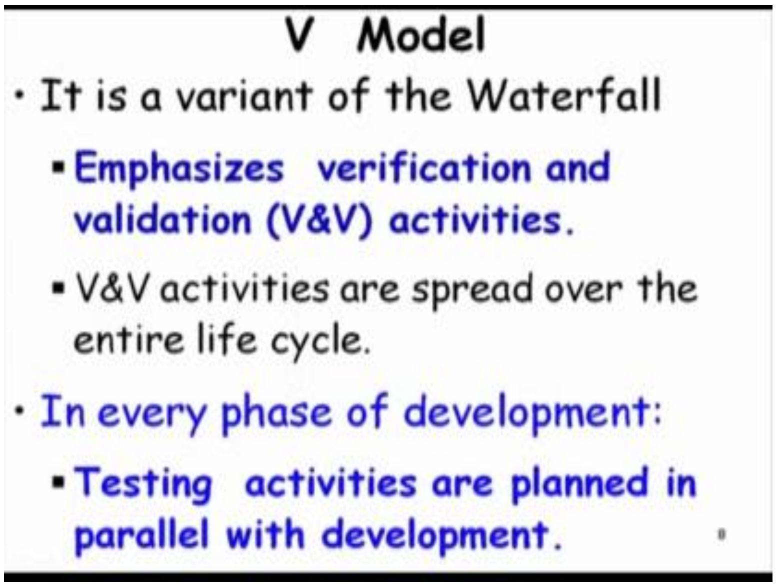
V Life Cycle Model

6/4/2016

7

Now, let us look at the V Life Cycle Model. The V life cycle model is special for testing.

(Refer Slide Time: 07:57)

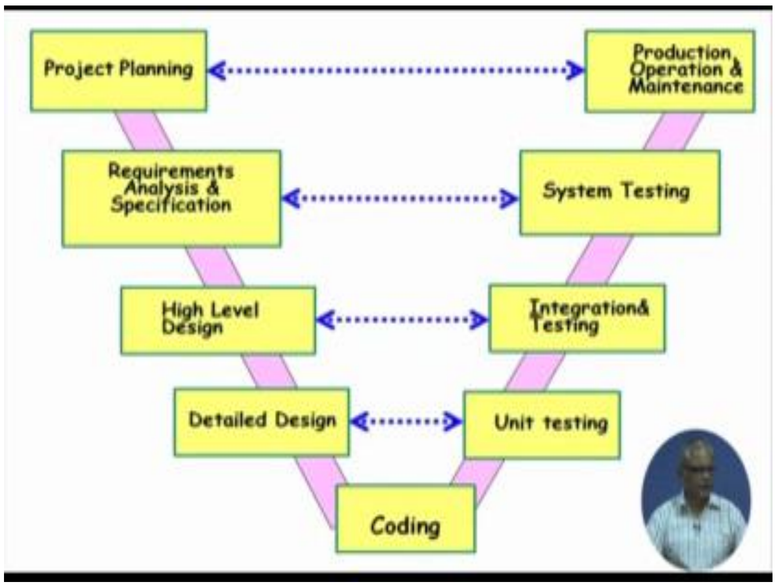


V Model

- It is a variant of the Waterfall
  - Emphasizes verification and validation (V&V) activities.
  - V&V activities are spread over the entire life cycle.
- In every phase of development:
  - Testing activities are planned in parallel with development.

It is a variant of the waterfall model, but then to as one of the early models to recognize the importance of carrying out verification and validation though out development cycle. Here, unlike waterfall model, in V model testing activities are spread all over the life cycle, that is in every phase of development test. is planned. The test cases are designed in parallel with development.

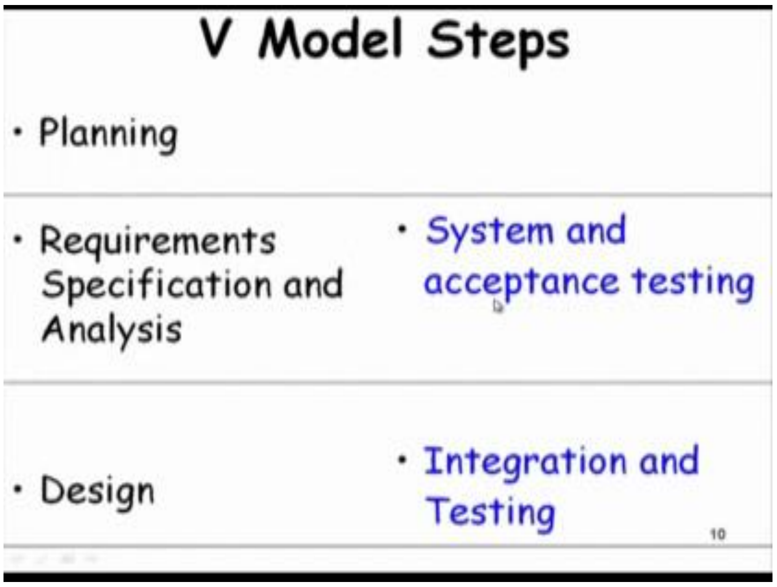
(Refer Slide Time: 08:42)



During requirement specification, we plan the system testing activities. Because system testing as we will see require only the functional and non functional descriptions of the system. And that is available in this requirements specification documents.

As the requirements specification document is developed the system test cases are planned, as the high level design is done the integration testing is planned, and as the detailed design is done the unit testing is planned. So, one very prominent advantage is that it spreads the testing activity throughout the development life cycle. And also when we due a development stage we plan out the testing and therefore the planning of test activity itself makes the artifact testable and leads to good quality artifact to be produced.

(Refer Slide Time: 09:56)



V Model Steps

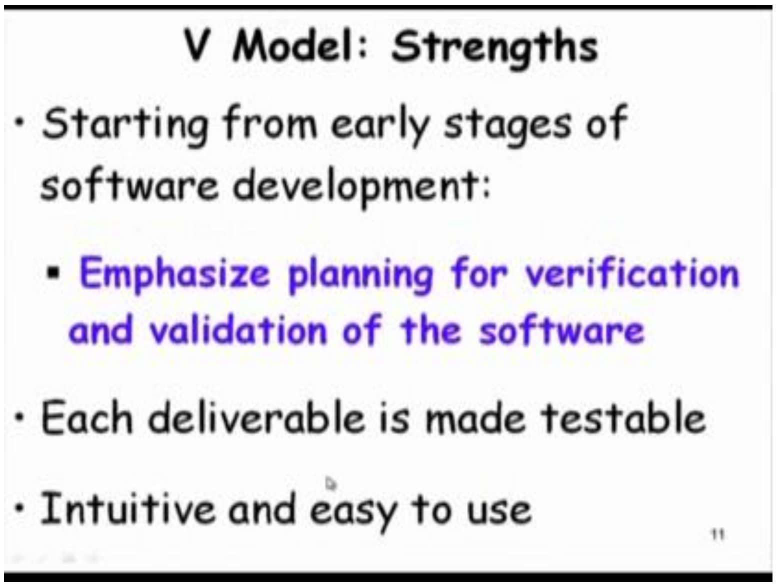
- Planning

• Requirements Specification and Analysis	• System and acceptance testing
• Design	• Integration and Testing

10

As we are saying the development activity and the corresponding test activity here, analysis and specification system and acceptance testing is carried out. During design integration testing and during detailed design or coding the unit test is planned.

(Refer Slide Time: 10:11)

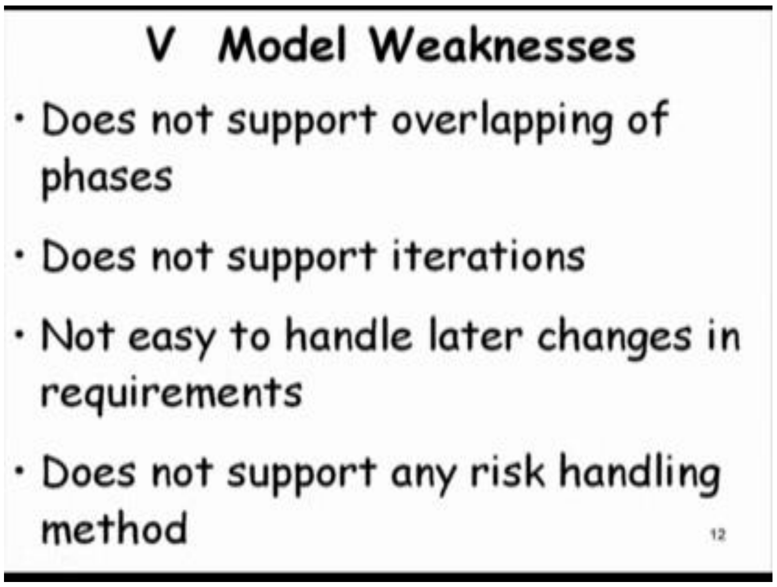

V Model: Strengths

- Starting from early stages of software development:
  - Emphasize planning for verification and validation of the software
- Each deliverable is made testable
- Intuitive and easy to use

11

The V model strengths we already mentioned that it emphasize planning for verification and validation of the software throughout the life cycle. The test activities, that is; planning and testing spread over the life cycle. Each deliverable is made testable; it is intuitive and easy to use.

(Refer Slide Time: 10:48)


V Model Weaknesses

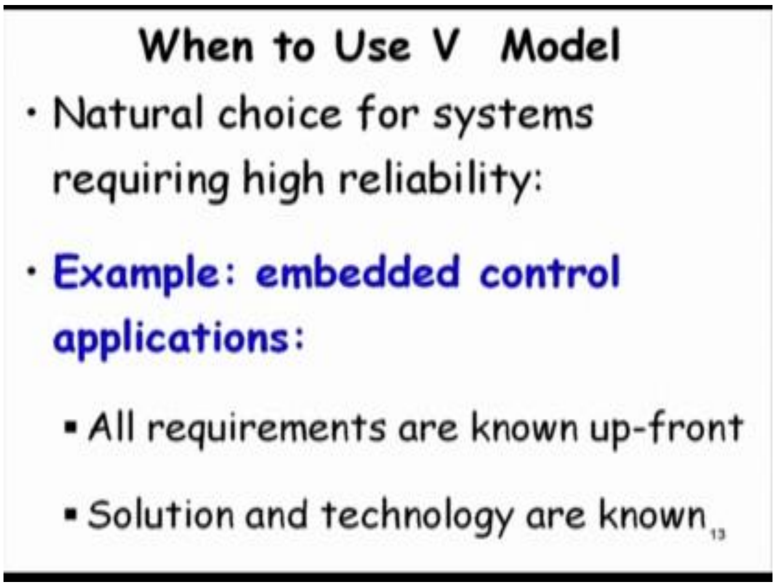
- Does not support overlapping of phases
- Does not support iterations
- Not easy to handle later changes in requirements
- Does not support any risk handling method

12

But then, the V model being a variant of the waterfall model it suffers from some of the same short comings as the waterfall model itself. For example, it does not support over lapping of the phases. We know that that is a major set coming of the waterfall model. There is a clear demarcation between different phases, and when one phase stops the next phase begins. But then in a practical situation we need over lapping activity of different phases as has been done in recent development methodologies such as Agile programming. The V model does not support iterations. Actually, iterative development is one of the very important it has been recognized has one of the important principles.

So the product, the software is developed over a large number of iterations and each iteration is actually a mini project where specification, design, coding and testing is carried out, so testing is there in every iteration. But V model does not support that. And because it is not iterative it does not handle the change requests. Here, initially the requirements are frozen and based on that the development is carried out and therefore very little scope for changing the requirements. And also there are no explicit mechanism per risk handling like the waterfall model.

(Refer Slide Time: 12:49)



When to Use V Model

- Natural choice for systems requiring high reliability:
- Example: embedded control applications:
  - All requirements are known up-front
  - Solution and technology are known

13

Now, we saw that the V model actually is a small adaptation of the waterfall model, where the testing is given importance and is spread over entire life cycle. But what kinds of software developments is V model suitable. One thing is that, the software which require very high reliability, the V model is preferred to be used their and the other characteristics are there. We should know the requirements upfront, the requirements should not be likely to change and also the solution should be proven.

Let us say we have one version working and we are just going to develop a newer version. We know that the technology works, the solution works and we are just trying to develop in different situation. In these cases this is a good model. For example, embedded control applications, where very high reliability is required and the solution might have already implemented one solution and trying small variation of that, we will use the V model.

(Refer Slide Time: 14:05)

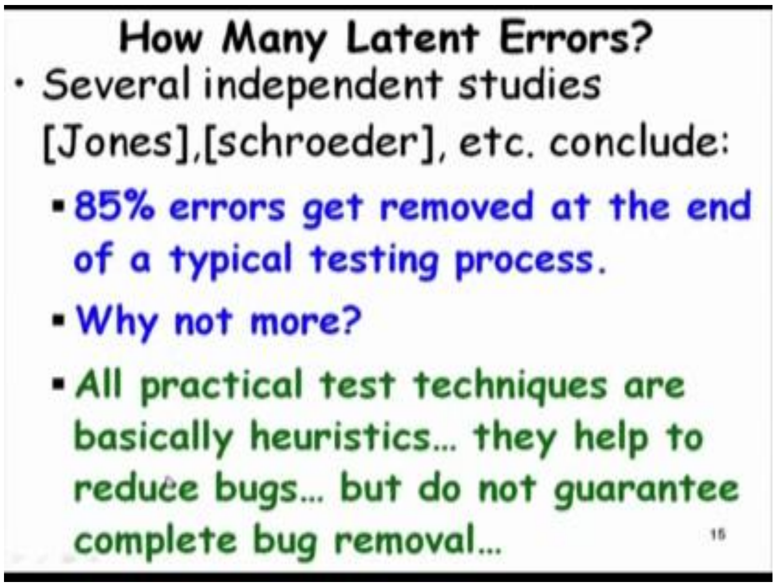
A Few More Basic Concepts on Testing...

6/4/2016

14

Now, let us look at few more basic concepts in testing before we look at details ( ) of test case design, test coverage analysis and so on.

(Refer Slide Time: 14:18)



How Many Latent Errors?

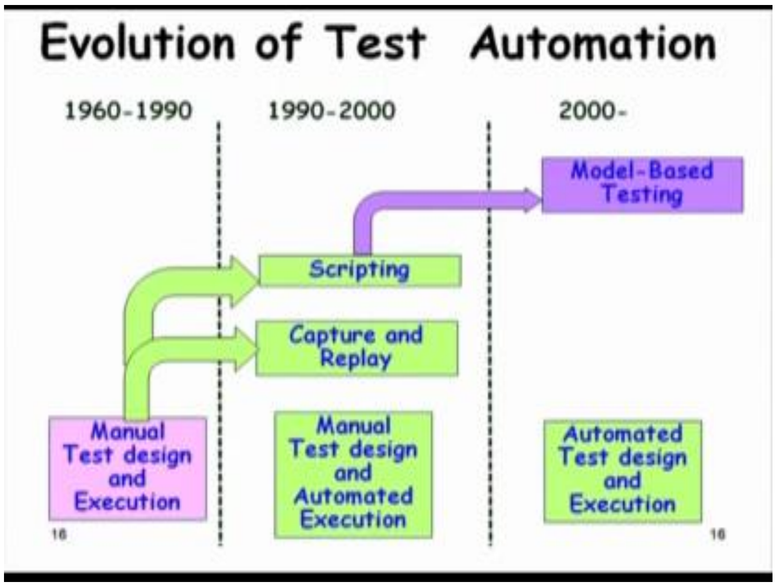
- Several independent studies [Jones],[schroeder], etc. conclude:
  - 85% errors get removed at the end of a typical testing process.
  - Why not more?
  - All practical test techniques are basically heuristics… they help to reduce bugs… but do not guarantee complete bug removal…

15

Now, one interesting question that is often raised is that suppose, software is developed using the available processes, reviews are carried out, available testing techniques are carried out, and ( ) after that how many bugs survive. And the version that is delivered to the user, how many do the bugs are present their? Some studies they conclude that about 85 percent of the bugs are removed. 85 percent of the bugs leave in the code when they are handed over to the customer. But then why is it that we are not able to remove more than 85 percent. Cannot we remove ( ) 100 percent?

The answer is that, is very difficult becomes extremely expensive to remove much more defects then 85 percent, because we would have to use many more filters and each filter or bug detection technique is actually heuristic. The only way 100 percent bugs can be removed is by a trying out all possible test inputs. And for practical situations the test inputs are infinite. Therefore, guaranteeing removal of 100 percent bugs is not possible and in a realistic situation about 85 percent bugs have been removed. As it is used by the customers and bugs are reported, more bugs get removed but then bugs also are introduced due to the changes. So, we will look at those aspects later.

(Refer Slide Time: 16:22)


Evolution of Test Automation
1960-1990
1990-2000
2000-
Model-Based Testing
Scripting
Capture and Replay
Manual Test design and Execution
Manual Test design and Automated Execution
Automated Test design and Execution
16
16

As we are saying that, last 30-40 years we have more and more automated tools that have become available. As you can see in this diagram till about 1990s, the test case design and execution was very well manual just keep on giving random inputs and possibly ( ) look at what is the coverage achieved and then the tester uses discretion to check whether it is a pass or fail. Basically, till about 90s the testing was more or less manual. But then after 1990s slowly test tool started to appear. One of the tools is capture and replay.

Capture and replay tools essentially do not really generate test cases or do automatic testing in that sense, but then as the tester inputs the test cases they capture the input. As the tester inputs the test data the tool here, the capture and replay tool captures the test input and also captures the result. Next time the test has to be executed he just replay that. So, it just captures the testers test input and later just replays.

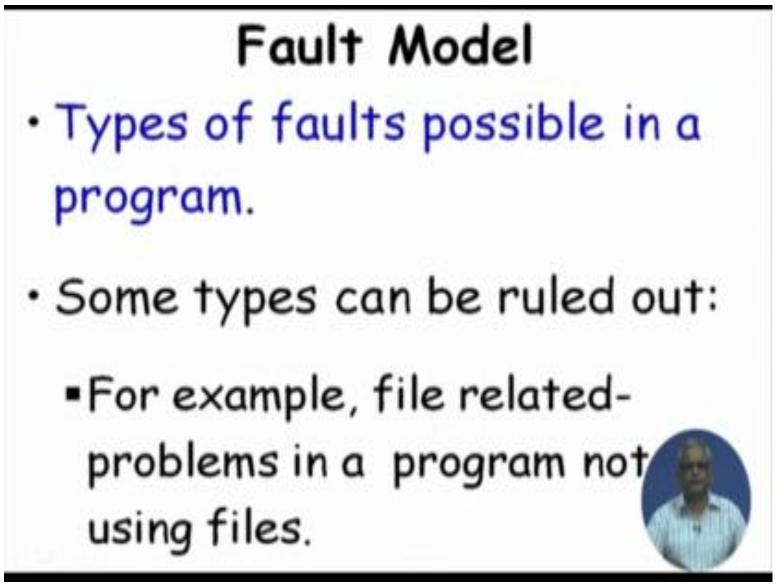
It does not really help design test cases or does not decide whether the test execution is right or wrong. It basically, the tester has to first time design the test cases and also decide whether it is a pass or fail and based on that the test will captures the test input and also can judge whether another time it is played whether it is pass or fail. But then, this is a big help, the capture and replay tools are a big help in testing. Can I ask why, why these are big help in testing, because it appears that the tester has to anyway design test cases input them, judge whether it is pass or fail.

So, how does a capture and replay tool help in testing? To answer this question in a typically software development scenario the same test case is executed hundreds or thousands of times. Why is that? That is due to the regression. Once some test cases have passed and we change something to the software we need to run those test cases which have passed just to check that still those parts of the software are working all right. So, capture and there is another category of tools called a Scripting tools.

In the scripting tool the test cases are actually small programs. The tester takes time to really ( ) write the test cases as programs. These small scripts they run and test the software, and these have some advantages over the capture and replay. In the sense that the scripting test cases are much more reusable. In a capture and replay if a feature or let us say one of the input just change little bit, the entire test case becomes useless and has to be thrown out. Whereas the script with the small change the same test can run. The scripting based tools these are much more reusable test cases, the produce much more reusable test cases even though initially they take more time to write the scripts.

So, these are the two major categories of testing tools, the scripting best tools and the capture and replay tools which evolved in the 1990 and 2000. And slowly we have more and more tools are which are appearing. For example, the model base testing tools base has done certain program model like, control flow, graph, dependency graph, and so on. We will try to see tools later in this course, but scripting and capture and replay tools will look at them first and other tools later.

(Refer Slide Time: 21:30)



Fault Model

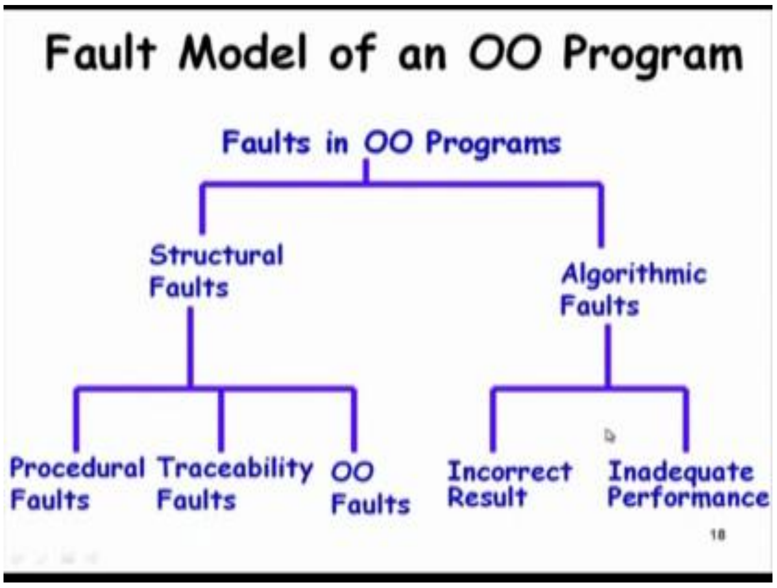
- Types of faults possible in a program.
- Some types can be ruled out:
  - For example, file related-problems in a program not using files.

Now, another basic concept that we want to discuss is a Fault Model. Actually, when a program is developed there are certain types of faults that get introduced. Let me repeat that word, that there are certain types of faults that get introduced. For example, one type of fault may be algorithm make fault. The programmer misunderstood the algorithm and his code where syntactically correct, and also possibly the code way what really he wanted to do but then he had misunderstood the algorithm. Maybe the sorting algorithm it does not sort properly, some parts of the input space.

Another type of bug may be Programming bug where, let us say the programmer inter changes some variable, instead of using i at some expression he use some k. Or maybe his look conditionals were not properly formed and so on. We can imagine that there are certain types of faults and the number of types can be also very large, because a programmer can do many types of mistakes while writing code. But this concept is very important that there are different types of faults that can be introduced by the programmer.

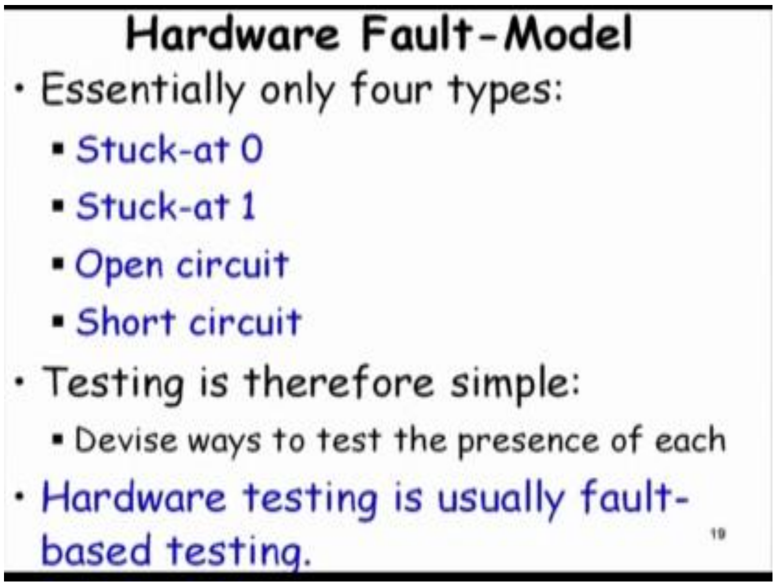
Now depending on the kind of program, certain kinds of faults can be ruled out. For example, if the program does not use any files, a file related programs can be ruled out. It does not use network communication related problems can be ruled out and so on.

(Refer Slide Time: 23:40)



If we roughly look at the high level categorization of the types of faults, we will say that there are Structural Faults. The structural faults will might have traceability fault that is while the programmer was coding the design just misunderstood the design, maybe left out some part of the design or misunderstood some design those the Traceability faults and so on. Similarly, we have Algorithmic Faults where we might have incorrect result wrong implementation of the algorithm or may be inadequate performance and so on, and we can sub categorize.

(Refer Slide Time: 24:21)



Hardware Fault-Model

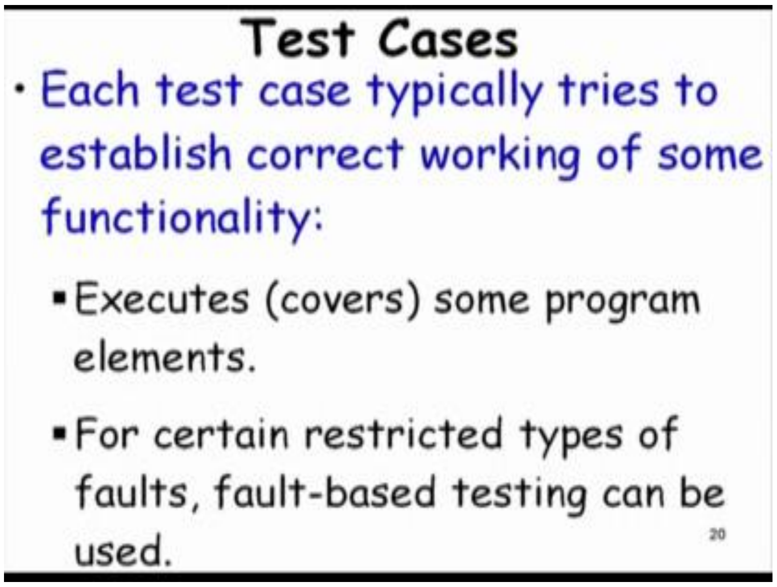
- Essentially only four types:
  - Stuck-at 0
  - Stuck-at 1
  - Open circuit
  - Short circuit
- Testing is therefore simple:
  - Devise ways to test the presence of each
- Hardware testing is usually fault-based testing.

19

But, one thing to notice is that in contrast to software where the number of types of bugs is very large tens up thousands, it is very small category of faults, very small number of types of faults. Essentially, there are only 4 types; Stuck-at 0, Stuck-at 1, Open circuit, Short circuit. Therefore hardware testing is much more simpler task then software testing, because in software we are trying to eliminate bugs that can be of various types, large number of times. Whereas, in hardware we know that there can be 4 or 5 categories of bugs. And for each category of bug we can design effective testing.

For example, in hardware there are tests to check whether there is a Stuck-at 0 problem or a Stuck-at 1 problem. So, hardware testing is usually fault based testing, where we check the existence of certain types of problems. On the other hand, in software each test case can detect several types of faults, and we cannot really design test cases which will try to discover a specific type of fault. In that case, will need a large number of test cases; the test cases here and software will see they are designed irrespective of the bug category and they might detect different types of bugs, but then there are few test cases also, few test strategies which are fault based will look those as we proceed.

(Refer Slide Time: 26:44)



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

There are few other basic concepts at test case and a test suite. A test case is basically a set of test data and state at which the test data is to be applied and what result is to be observed. Let me just repeat that, that a test case typically tries to check the correct working of functionality and as we execute it covers some program elements. The program element can be statement, it can be specific condition in for look or may be while look or may be if, so checking for a condition, checking for a statement and so on.

And, we check whether the required types of elements in the program are covered. So that is our testing criterion, we check whether the elements that we are targeting whether it is statements or conditions these are all covered. But then as I was mentioning that we have a few fault based techniques as well. So, we will just stop this session and continue our discussion in the next session.

Thank you.