



SOFTWARE ENGINEERING

**Prof. Rajib MallCivil
Computer Science and
Engineering
IIT Kharagpur**



INDEX

S. No	Topic	Page No.
	<i>Week 1</i>	
1	Introduction-I	1
2	Introduction-II	12
3	Introduction-III	26
4	Introduction-IV	39
5	<i>Week 2</i>	58
6	Life Cycle Model	78
7	Life Cycle Model	96
8	Waterfall Model	110
9	Waterfall Derivatives	130
10	Incremental Model	142
	<i>Week 3</i>	
11	Evolutionary Model	158
12	Agile Model	181
13	Extreme Programming and Scrum	200
14	Scrum	220
15	Introduction to requirement specification	240
	<i>Week 4</i>	
16	Requirement gathering and analysis	256
17	Functional requirements	274
18	Representation of complex programming logic	293
19	Design Fundamentals	309
20	Modular Design	325
	<i>Week 5</i>	
21	Classification of Cohesion	343
22	Classification of Coupling	360
23	Introduction to structured analysis and structured design	378
24	Basics of Data Flow Diagrams (DFD)	393
25	Developing DFD Model	412

Week 6

26	Examples of DFD Model development	430
27	DFD Model - More Examples	451
28	Essentials of Structure Chart	466
29	Transform Analysis, Transaction Analysis	481
30	Structured Design Examples	492

Week 7

31	Use Case Modelling	504
32	Factoring Use Cases	516
33	Overview of Class diagram	530
34	Inheritance relationship	545
35	Association relationship	560

Week 8

36	Aggregation/ Composition and dependency relations	573
37	Interation Modelling	591
38	Development of Sequence diagrams	602
39	State-Machine diagram	617
40	An Object-Oriented design process	629

Week 9

41	Domain Analysis	644
42	Examples of object-oriented design	656
43	Basic concepts in Testing-I	673
44	Basic concepts in Testing-II	690
45	Basic concepts in Testing-III	703

Week 10

46	Unit testing strategies-I	719
47	Unit testing strategies-II	735
48	Equivalance Class Testing-I	750
49	Equivalance Class Testing-II	765
50	Special Value Testing	779

Week 11

51	Combinatorial Testing	796
52	Decision Table Testing	811

53	Cause effect graphing	826
54	Pairwise Testing	840
55	White box Testing	852

Week 12

56	Condition Testing	867
57	MC/DC Coverage	877
58	MC/DC Testing	892
59	Path Testing	904
60	Dataflow and Mutation Testing	923

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

Lecture – 01
Introduction- I

Welcome to the course on Software Engineering. We will have a brief introduction to the various issues in Software Engineering, the motivation and the benefits that you will acquire from here.

(Refer Slide Time: 00:41)

The slide has a yellow background and a blue header bar with a toolbar icon. The title 'About Myself' is centered at the top. Below it is a bulleted list of achievements:

- RAJIB MALL
- B.E. , M.E., Ph.D from Indian Institute of Science, Bangalore
- Worked with Motorola (India)
- Shifted to IIT, Kharagpur in 1994
 - Currently Professor at CSE department

On the right side of the slide, there are two images: one of the Indian Institute of Science Bangalore building and another of the IIT Kharagpur building. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video frame showing a person's face.

About myself: My name is Rajib Mall. Completed all my education - Bachelors, Masters, PhD from the Indian Institute of Science Bangalore. Then worked few years with Motorola India and have been with IIT Kharagpur since 1994 and currently a Professor at the CSE department, Indian Institute of Technology, Kharagpur.

(Refer Slide Time: 01:32)

What is Software Engineering?

- Engineering approach to develop software.
 - Building Construction Analogy.
- Systematic collection of past experience:
 - Techniques,
 - Methodologies,
 - Guidelines.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

3

The definition of software engineering is that it's an engineering approach to develop software. But the question arises that what exactly is the engineering approach to develop software and how it's different from the traditional approach to developing software.

The distinction between an engineering approach to develop software and traditional novice approach to develop software, looking at an analogy with building construction, consider that you want to build a small wall that you will very easily be able to do from your basic intuition with the help of bricks, cement, colour and etc and you will succeed in developing a good small wall. But when you are asked to build a large building having 30-40 floors a complex. Your basic intuition about construction will not work. If you actually start building this with your basic intuition then the building may collapse and never complete and it will be of a poor quality.

Same is with respect to the software. If someone is asked to write a small program like 10-20 or 50 lines of code, you will be able to do it based on this basic intuition. But consider developing a software which is 100-1000 lines of code. Now, if you want to extend the intuition and develop a large software, again you will face the same effect as constructing a large building from the basic intuition.

An alternate definition of software engineering is that it's a collection of past experience which has resulted in various techniques for the development of large software, the methodologies for development, some guidelines and tools.

(Refer Slide Time: 05:50)

The slide has a yellow background with a red border around the text area. The title 'IEEE Definition' is at the top. The text area contains a bullet point: • "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." At the bottom left is the IIT Kharagpur logo and text 'IIT KHARAGPUR'. Next to it is the NPTEL logo and text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side, there is a small video frame showing a man speaking.

- "Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

IEEE definition of software engineering is that it's the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is the application of engineering to software”.

The definition also states that software engineering is the engineering approach to develop software and it further elaborates this by saying: application of a systematic, disciplined, quantifiable approach. All this systematic, disciplined and quantifiable approach has been developed from the experience of past programmers over the last several years.

(Refer Slide Time: 06:54)

The slide has a yellow background with a blue header bar. The title 'Software Crisis' is in large blue font. Below it is a bulleted list of symptoms:

- It is often the case that software products:
 - Fail to meet user requirements.
 - Expensive.
 - Difficult to alter, debug, and enhance.
 - Often delivered late.
 - Use resources non-optimally.

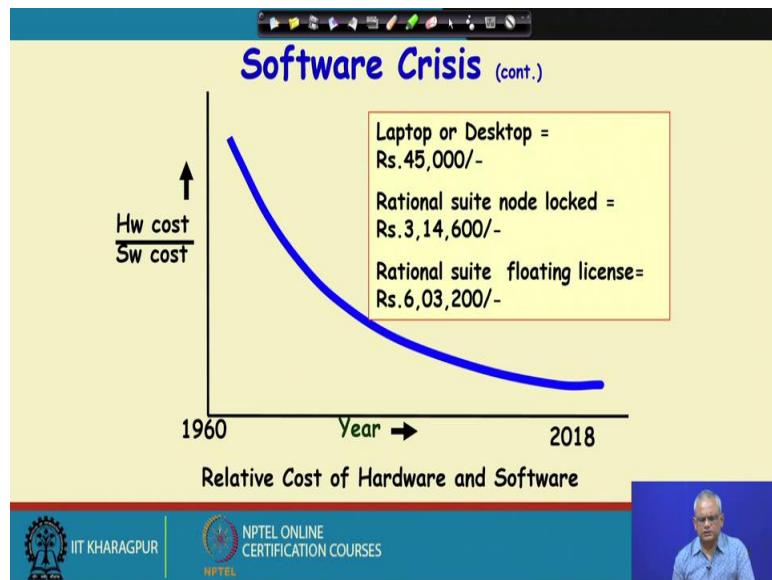
At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

Software crisis: The software crisis is that crisis when the software fails to meet the user requirements due to the complexity of the problems.

Software is more expensive than hardware and every year the difference between software and hardware prices have been increasing dramatically. It's difficult to make any changes to the software and to debug if there is a problem reported and requires a longer time to correct it. Sometimes the bugs cannot be corrected and it's very usual for software to be delivered late and use resources non-optimally.

For most of the software that you order or develop as a part of the project have these symptoms, that they rarely meet the user requirements and turn out to be very expensive. It's quite difficult to alter, debug and enhance, mostly delivered late and use resources non-optimally.

(Refer Slide Time: 08:52)



To compare and contrast the expensiveness of software to that of hardware, if we plot the graph, we will see that 1960s, the hardware was the major aspect, millions of dollars. But then, the hardware costs dropped dramatically and the software costs got risen.

Now, the hardware cost divided by software cost is almost nominal approaching 0. Considering a laptop or desktop which costs 45000 rupees and the software on the other hand such as rational suite costs which costs up to 3 lakh rupees and if its node-locked and floating license it costs around 6 lakh rupees which runs on a desktop, I.e.; the cost of one software is X times the cost of the hardware.

The software is more expensive & companies spend more on software than on hardware.

(Refer Slide Time: 11:03)

The slide has a yellow header with the text "Then why not have entirely hardware systems?..." and a blue footer with the IIT Kharagpur logo and "NPTEL ONLINE CERTIFICATION COURSES".

- A virtue of software:
 - Relatively easy and faster to develop and to change...
 - Consumes no space, weight, or power...
 - Otherwise all might as well be hardware.
- The more is the complexity of software, the harder it is to change--why?
 - Further, the more the changes made to a program, the greater becomes its complexity.

The fundamental question which arises is that if the hardware is so good, delivered in time, extremely reliable, inexpensive then why not have everything in hardware. After all, whatever can be done by software can also be done by hardware.

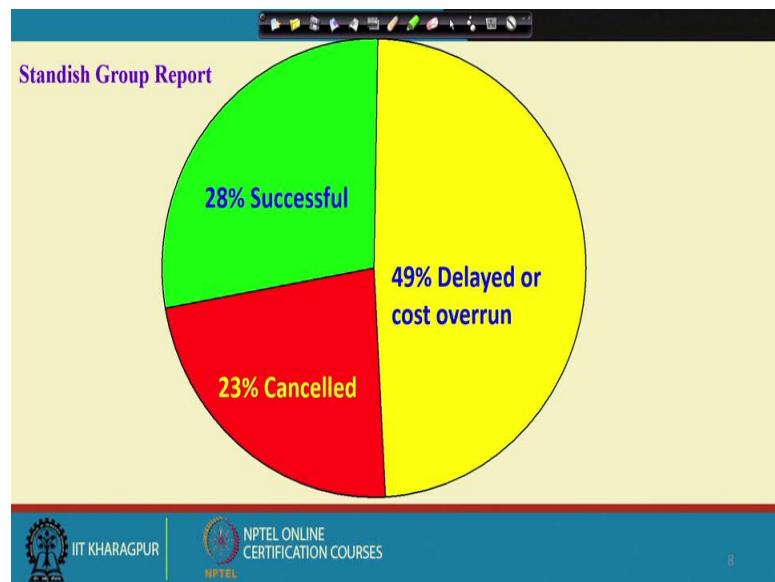
There are two or three major virtues of software's which actually make it a preferred solution compared to hardware. One of the important reasons is that developing software is easy, faster and also if you find anything to be changed you can easily change it. Hardware development is a long run process. If it's a VLSI chip that you are considering, then it requires multiyear, starting from the requirements to fabrication and if you want to make a small change again you have to go on that long process; change requirements, change design, mass and fabrication test which requires a long duration of several years.

Compared to software's, making a fix or a small change may be done just in a few days as software consumes no space, weight or power. If you want to develop something that works in software consider Microsoft Word. It's possible to have this developed in hardware, but then the size of the hardware will be enormous. It will be extremely expensive will consume huge power. But if you add this into the software it will just reside in the hard disk and get executed feature by feature. That's the reason why software is the preferred mode of solution for applications as compared to hardware.

Software is becoming more complex and as we change the software it's complexity increases. To change the software, initially you must understand it, without understanding how it works, you cannot change it. If the software is very complex, it will require a lot of time to make the change. Also, if we make a small change it will change the basic underline, modularity and etc.

Every change degrades the structure of the software and makes it much more complex for someone to understand it. As changes keep on happening to a software due to additional features to be supported or to correct the bugs and etc.

(Refer Slide Time: 16:59)



Looking into the large number of projects. What are the statistics about the success rate of the projects? Just 28 percent is successful. A quarter never see the light of the day and they are cancelled. Meanwhile, half of the projects are delayed, delivered late and also the cost overrun price increases.

Only a quarter of the software is successful while half of the software is challenged and another quarter is a failure and are to be cancelled and the reasons is as follows:

(Refer Slide Time: 18:19)

The slide has a dark blue header bar with standard window controls. The main title 'Which Factors are Contributing to the Software Crisis?' is centered in a large, bold, black font. Below the title is a bulleted list of factors in a smaller black font. One item in the list is highlighted in blue. At the bottom of the slide, there is a footer bar with three sections: a logo for IIT Kharagpur, a logo for NPTEL Online Certification Courses, and a video feed of a man speaking.

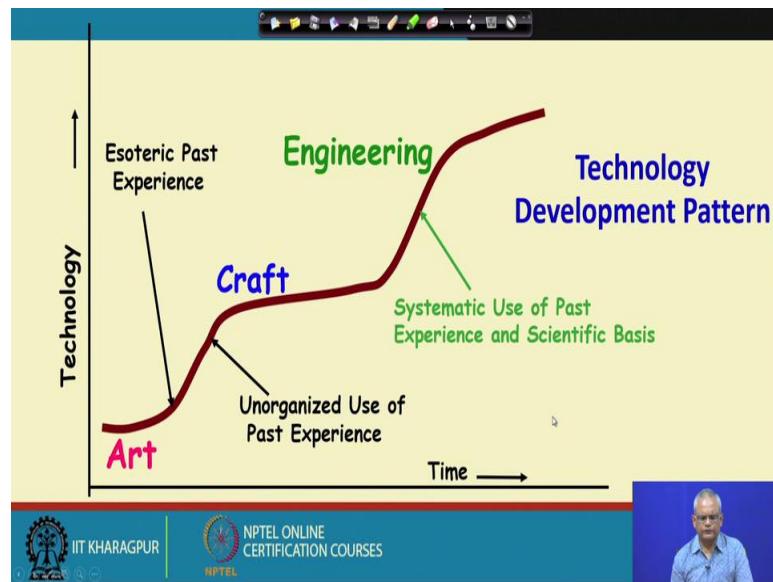
- Larger problems,
- Poor project management
- **Lack of adequate training in software engineering,**
- Increasing skill shortage,
- Low productivity improvements.

As there is a demand for more software's, the hardware is simpler and the software is more complex. Due to obvious reason, any complexity in the application has to be done in the software. Otherwise, the hardware will be very difficult to develop as it will be large, consume lots of power volume and etc.

Poor project management because these are entirely manual efforts, the programmers have to develop, write the code test and etc. But, surprisingly one of the major reasons that have been identified for the software crisis is the lack of adequate training in software engineering. The developers are not very conversant with the latest software engineering techniques.

Even with software engineering the skill improvements are not really very drastic, it improves moderately, slowly over the years. Compared to the way the problem sizes are increasing the productivity development has not really kept pace.

(Refer Slide Time: 20:29)



Is software an art or engineering?

Looking at how the technology develops for any specific domain, whether it is steel making or paper making and also look at the technology development pattern. As stated, paper making was known to some people in China and only, they could develop good quality paper. The others tried but they do not know how to make paper, possibly came up with some very bad quality paper and thought paper making is an art.

Consider an artist draws a nice drawing and if you ask him how did you come up with such a nice drawing. He will say I just draw it. Same with technology, initially they say that we just know how to make it. But then slowly the art form in technology development graduates to a craft form. In a craft form, there are some techniques which can be identified and they share it with only their apprentice.

It's basically a hidden secret. Be it paper making or steel making, there is a craftsman who knows how to do it. Basically, they are the apprentices and they share their skillset with a very smaller number of people. Slowly it transits to an engineering approach. In engineering, the techniques are all investigated. The past experience is analyzed and a scientific basis is given to these techniques. These are then well documented and can be studied by anyone in their open knowledge. But what about program writing; yes, it's the same pattern.

In the early 1960s, there were good programmers and bad programmers. The good programmers just wrote good programs and other programmers struggled and could not come up with good programs. Slowly, the techniques that they were using in writing the good programs were identified and shared with few and that was the craft form. A good programmer was to train other programmers, who can become good programmers.

Later, the techniques that they were using for writing good programs were all systematically investigated, scientific basis was given to them and published in the literature and in the book form.

(Refer Slide Time: 25:35)

The slide has a yellow background with a blue header bar at the top containing various icons. On the right side, there is a blue rectangular box with the text "Programming an Art or Engineering?" in white. The main content area contains the following bullet points:

- Heavy use of past experience:
 - Past experience is systematically arranged.
- Theoretical basis and quantitative techniques provided.
- Many are just thumb rules.
- Tradeoff between alternatives.
- Pragmatic approach to cost-effectiveness.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo and the text "IIT KHARAGPUR". To its right is the NPTEL logo with the text "NPTEL ONLINE CERTIFICATION COURSES". On the far right of the footer bar is a small video frame showing a man speaking.

If you look at the engineering approach to develop software, there is heavy use of past experience, the past experience is systematically arranged. For this, the experience investigation has been made a theoretical basis and quantifiable technique's i.e.; quantitative techniques have been provided. There is a tradeoff between alternatives because when you design software, you have to look at various kinds of tradeoff, the complexity of the software and the cost. Because of that, the tradeoff between alternatives, you will have a pragmatic approach to cost effectiveness.

Summary: Software engineering is an engineering approach to develop software's and this engineering approach has developed from past experience of a large number of programmers. Over the years the programmers have innovated new ways of looking at how to write good programs. All these techniques have been systematically organized so that anyone can go through these topics and be a good programmer and write large programs in a team and also as an individual.

Thank You.

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

Lecture – 02
Introduction- II

In the last lecture we discussed about very basic issues and debated whether program writing is an art or engineering. We stated that, initially every technology starts in an art form slowly becomes a craft and then graduates into an engineering approach.

Software is no different and in the engineering approach, the past experience of large number of program developers is systematically investigated, documented and scientific basis techniques is provided.

(Refer Slide Time: 01:20)



• Early programmers used **exploratory** (also called **build and fix**) style.

- A 'dirty' program is quickly developed.
- The bugs are fixed as and when they are noticed.
- Similar to how a junior student develops programs...

What is
Exploratory
Software
Development?

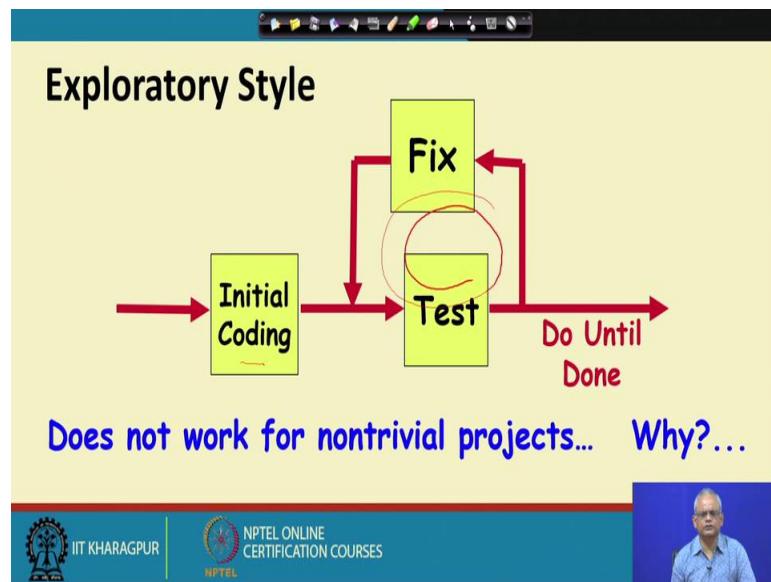
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

A small video player window in the bottom right corner shows a man speaking.

Exploratory software development is the software development practice of a Novice. Consider someone who is starting to program, just learning programming and doing small assignments; how will he be able to develop the program: that you call as the exploratory software development or the build and fix style of development. This is the same style which the early programmers used, because there were no other techniques, they had to use their intuition and develop and that is called as a build and fix style.

In this style as soon as the problem statement is given based on the understanding the programmer starts writing the program. Normally, it's a dirty program in the sense that it will have a lot of bugs. As you document this development style of the build and fix, in the form of a schematic diagram, you will curate the diagram as follows :

(Refer Slide Time: 02:58)



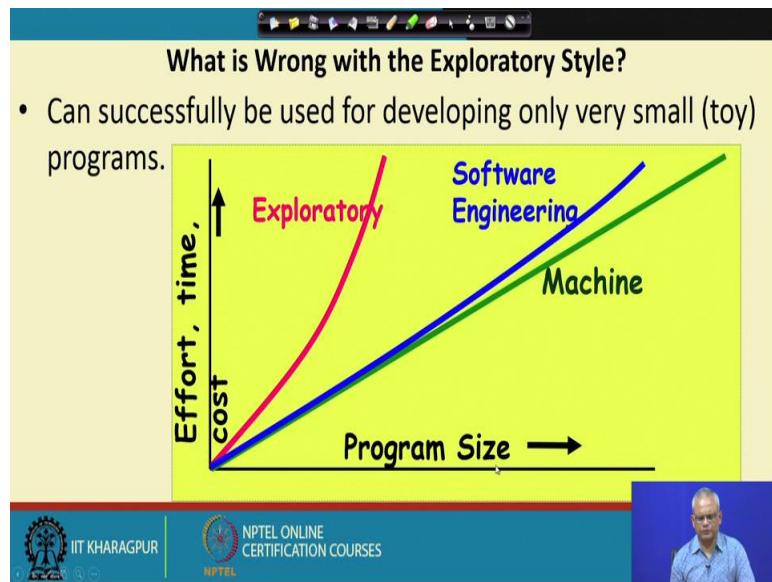
As soon as the problem is provided, you do the initial coding and after the initial coding is complete, start testing and find the large number of bugs getting reported. Maybe some of them are crash or just large number of bugs & compilation errors. Again, compile test and examine that the errors keep on this cycle: fixing the program and testing. Keep on doing it until you find the requirement for the program.

This is the model of an exploratory style of development and as you proceed, you will contrast this basic style of development with the engineering style of development.

You will use the exploratory style as most of the new programmers use to develop small programs. But if the project is nontrivial, large and requires hundreds or thousands lines of code, it will be very difficult to debug and never complete. Even if somebody really puts lots of effort, he will come up with a very poor quality of software.

Let's try to investigate, why the exploratory style does not work for the larger projects but works easily for the trivial programs.

(Refer Slide Time: 05:45)



What is wrong with the exploratory style. Understanding the plot, as the programs size increases you develop large and larger programs and still keep using the exploratory style or the build and fix style. You will see that for small programs we are successful but as the program size increases, the effort, time, cost all increase exponentially. Maybe, to develop a 100-line program it took couple of hours but to develop a 1000-line program it took several months.

But after some size, the exploratory style just breaks down. Even if you try your best using the exploratory style still you cannot develop after a certain size whereas the software engineering approaches comes into action. There is almost a linear increase in the effort, time & cost compared to the program size. So, if a 1000 lines program took couple of hours then a 10000 lines program will take only about 10 times of that not 100 times or 1000 times.

As if there exists a machine to which we can give the problem statement and which will come up with the code.

One very basic thing you need to remember is that in the exploratory style, the cost-effort time increases almost exponentially whereas using software engineering principles it's down to almost a linear increase.

(Refer Slide Time: 08:51)

What is Wrong with the Exploratory Style?

Cont...

- Besides the exponential growth of effort, cost, and time with problem size:
 - Exploratory style usually results in unmaintainable code.
 - **It becomes very difficult to use the exploratory style in team development environments...**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



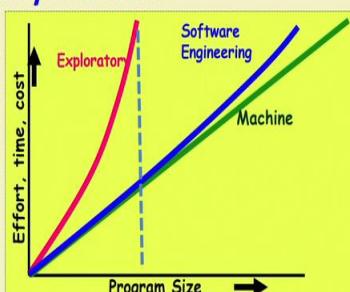
The question naturally arises what is wrong with the exploratory style, why the cost effort time to develop increase exponentially and also break down after certain size?

The other symptoms are that: the programs are poor quality, unmaintainable and it's quite difficult to use the exploratory style in the team development environment whereas in using the software engineering principles we are able to overcome this.

(Refer Slide Time: 09:49)

What is Wrong with the Exploratory Style? Cont...

- **Why does the effort required to develop a software grow exponentially with size?**
- Why does the approach completely breaks down when the size of software becomes large?



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

16

That is a basic question. Why in the exploratory style, there is an exponential increase in the effort, time, cost and you try to keep that down to a linear using software engineering principles. Is it really working here for software engineering.

(Refer Slide Time: 10:25)

An Interpretation Based on Human Cognition Mechanism

- Human memory can be thought to be made up of two distinct parts [Miller 56]:
 - Short term memory and
 - Long term memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

17

It's a very fundamental principle. As, the software engineering works whereas the exploratory style does not.

To understand this, you need to little bit understand the human cognitive mechanism. The long back results from Miller 1956 said that human memory can be made of two distinct parts: one which is short term memory and the other is a long-term memory. The short-term memory is one where you remember something for only a small duration of time, that is may be for several seconds or just minutes whereas in long term memory remembers you remember it for months or years.

(Refer Slide Time: 11:26)

• Suppose I ask: **"It is 10:10AM now, how many hours are remaining today?"**

– 10AM would be stored in the short-term memory.

– "A day is 24 hours long." would be fetched from the long term memory into short term memory.

– The mental manipulation unit would compute the difference (24-10).

Human Cognition Mechanism

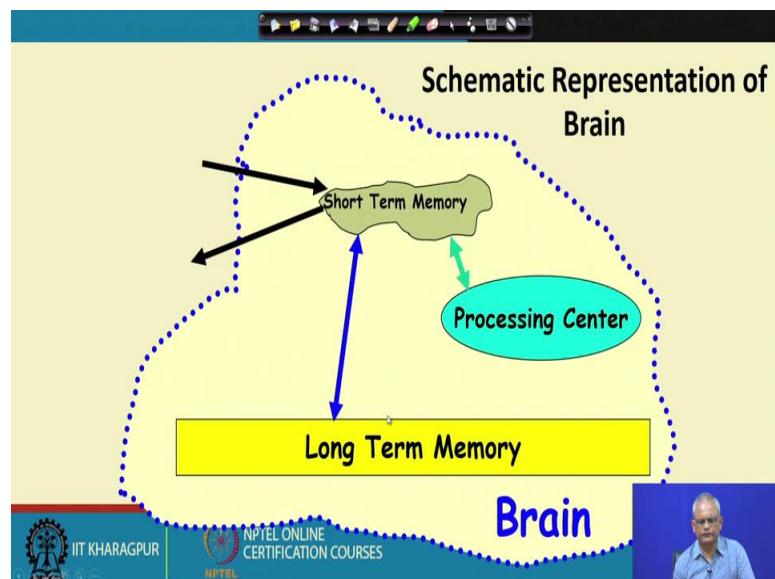


IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



The working of the human cognition mechanism. Let look at the watch it is 10:10, how many hours remaining today? The way the human mind will work is that its 10 AM and that's in the short-term memory and from your long term memory you will fix that one day is of 24 hours then there will be a mental manipulation unit which will compute the difference (24-10). That its 14 hours or 13 hour 50 minutes.

(Refer Slide Time: 12:37)



If you put that down in the form of a diagram, you will see that the long-term memory holds larger number of items. Depending on the person it can hold billions, trillions of

items whereas a short-term memory can hold only few. These are then fixed from the long-term memory, short term memory and from the observation looking at the clock. It then gets into the processing center where some computation can be done and the answer is given.

(Refer Slide Time: 13:26)

The slide is titled "Short Term Memory". It contains the following text:

- An item stored in the short term memory can get lost:
 - Either due to decay with time or
 - Displacement by newer information.
- This restricts the time for which an item is stored in short term memory:
 - Typically few tens of seconds.
 - However, an item can be retained longer in the short term memory by recycling.

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with the text "NPTEL ONLINE CERTIFICATION COURSES". A small video frame shows a man speaking.

What exactly is the short-term memory, what it can store and for how long. An item stored in a short-term memory gets lost with the time may be within several seconds, minutes or hour. It gets lost from the short-term memory because there are too many new information that has come into the short term memory or it just plays and decays with time. The main thing is that it stays for very short time and depending on the new information coming, the time duration can be even very short, typically for tens of seconds.

But someone can remember longer by recycling the memory. For example, you went to the market and wanted to buy few things, just keep on recollecting again and again that you need to buy. Hence, we are were recycling in the short term memory.

(Refer Slide Time: 14:45)

• An item is any set of related information.

- A character such as 'a' or a digit such as '5'.
- A word, a sentence, a story, or even a picture.

• Each item normally occupies one place in memory.

• When you are able to relate several different items together (**chunking**):

- The information that should normally occupy several places, takes only one place in memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

What is an Item?

Software memory stores items. An item can be a character, it can be a digit (like 5,6), character (like a, b, etcetera). It can be a word, a sentence, a story and even a picture. If somebody remember some numbers 5 9 7 etc and each one is considered one item. But if there is a relation between them then it can be considered as a one item. Consider 1 to 9 (i.e.; 1 2 3 4 5 6 7 8 9). So, that is one item, because you recognized that there is a relation (i.e;1 to 9).

Similarly, a word, an individual letter if there is no relation they will be separate items but if they are related, they mean something then that is just one item. The sentence is one item, story is one item and even a picture is one item as they occupy one place in the short-term memory.

The conclusion which we can made here is that if you are remembering something and you are able to build the relation between the things that you want to remember then it becomes easier. That process is called as Chunking.

Example of chunking:

(Refer Slide Time: 16:27)

Chunking

- If I ask you to remember the number **110010101001**
 - It may prove very hard for you to understand and remember.
 - But, the octal form of **6251 (110)(010)(101)(001)** would be easier.
 - You have managed to create chunks of three items each.



22

Let's consider someone asks you to remember the number 110010101001. You will struggle to remember this, because there are too many digits and then the relationship among them is not so obvious.

But, then group these into three in the octal form, 6251 it becomes easier to remember. That is because of the limitation of the human cognition mechanism that the short-term memory can hold only 7 items.

And as long as you are able to build relations it becomes easier for remembering till it is less than 7. So, this is the principle of chunking.

(Refer Slide Time: 18:09)

• In many of our day-to-day experiences:
– **Short term memory is evident.**

• Suppose, you look up a number from the telephone directory and start dialling it.
– If you find the number is busy, you can dial the number again after a few seconds without having to look up the number from directory.

• But, after several days:
– You may not remember the number at all
– Would need to consult the directory again.

Evidence of Short Term Memory

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

23

There are many day to day experience where the short term memory is evident. For example: you looked up to a telephone number and found that you not able to dial it as it's busy. After few minutes you are almost able to remember the number. But, after several days you will hardly remember anything.

(Refer Slide Time: 18:42)

• If a person deals with seven or less number of items:
– These would be accommodated in the short term memory.
– So, he can easily understand it.

• As the number of new information increases beyond seven:
– It becomes exceedingly difficult to understand it.

The Magical Number 7

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

As the number of items that someone can store in short term memory is about 7 and that is known as the magical number 7. Any set of information which has 7 items in it for a ordinary person it becomes easy to understand. But if the number of items are large

considering tens of thousands, let us just look at very trivial example: Consider, you met 5 people somewhere, you will almost remember each of them. But you met 10000 people for 10 minutes you will hardly remember anybody. That is because of the number 7 as the short-term memory capability is restricted to 7.

So, if the information has more than 7 items it becomes exceedingly difficult to understand and to remember.

(Refer Slide Time: 20:18)

What is the Implication in Program Development?

- A small program having just a few variables:
 - Is within easy grasp of an individual.
- As the number of independent variables in the program increases:
 - It quickly exceeds the grasping power of an individual...
 - Requires an unduly large effort to master the problem.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

To understand a program, you must look at the variables as how do they interact. Considering a small program having only a couple of variables, you can easily look through the code and understand what is happening. But when you are looking at a program which has thousands of variables and each variable is set by some programming constructs being used. For the human mind it becomes extremely difficult to understand it.

As, the number of independent variables in a program is small it becomes easy to understand but as the number of variables in the code increases it becomes very difficult to grasp and to understand & requires and unduly large effort to master the problem. If the number of independent variables is more than 7 then it's a large problem dealing with many things. Then the question arises that how does the software engineering principles contain the complexity of this problem.

(Refer Slide Time: 22:40)

Implication in Program Development

- Instead of a human, if a machine could be writing (generating) a program,
 - The slope of the curve would be linear.
- But, how does use of software engineering principles help hold down the effort-size curve to be almost linear?
 - Software engineering principles extensively use techniques specifically targeted to overcome the human cognitive limitations.

For machines there is no such problem as short term memory, of course as they have RAM, etc and the restrictions are not so severe. So, if the machine would like write the code then the slope would be almost linear.

But how the software engineering principles contain the complexity. Because, as the complexity increases the human mind by itself would take exponential time, effort to understand to solve the problem.

(Refer Slide Time: 23:35)

Which Principles are Deployed by Software Engineering Techniques to Overcome Human Cognitive Limitations?

- Two important principles are profusely used:
 - Abstraction
 - Decomposition

There are two major techniques that are used to overcome the cognitive limitation of 7. These two important principles are used in almost every technique: one is called as abstraction and the other is called as decomposition.

Let's try to investigate these two fundamental techniques to handle the complexity. As, software engineering is after all developing programs efficiently. As, the size increases you want to develop only with linear time, cost, effort and etc. For that you need to effectively handle the increase in complexity. These two techniques to handle the complexity are used in almost every software engineering principle.

(Refer Slide Time: 25:08)

The slide has a yellow background and a black header bar with various icons. The title 'What is Abstraction?' is centered in a large, bold, black font. Below the title is a bulleted list:

- Simplify a problem by omitting unnecessary details.
 - Focus attention on only one aspect of the problem and ignore other aspects and irrelevant details.
 - Also called model building.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo. To the right of the footer bar is a small video window showing a man speaking.

Abstraction: In abstraction we focus our attention only on some aspect of the problem and ignore the rest aspects & irrelevant details. This is also known as model building.

Consider you want to develop a large building and see how does it looks like. You will ignore all other aspect like the strength, wall thickness, the internal plan and etc. You will concentrate only on its frontal appearance and ignore everything to construct a model of the frontal view of the building. Similarly, for the floor plan for a building, you will just concentrate on that and don't bother about how does it appear externally, the thickness of the wall and etc.

Concentrating on the hitting requirement of a room; you will just look where is it located, how much heat it is getting, etcetera and you will omit other unimportant

aspects. For, every problem we can construct and abstraction. The abstraction is also called as a model building if you create a model, you concentrate on some aspect and ignore the rest. Hence, Model building is the same as an abstraction. This is an important technique in software engineering and you need to build models of everything that is the requirements, design, code, etcetera and that is one way to tackle complexity.

Let's continue with this abstraction decomposition in the next lecture and we will also look at other issues in software engineering.

Thank you.

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

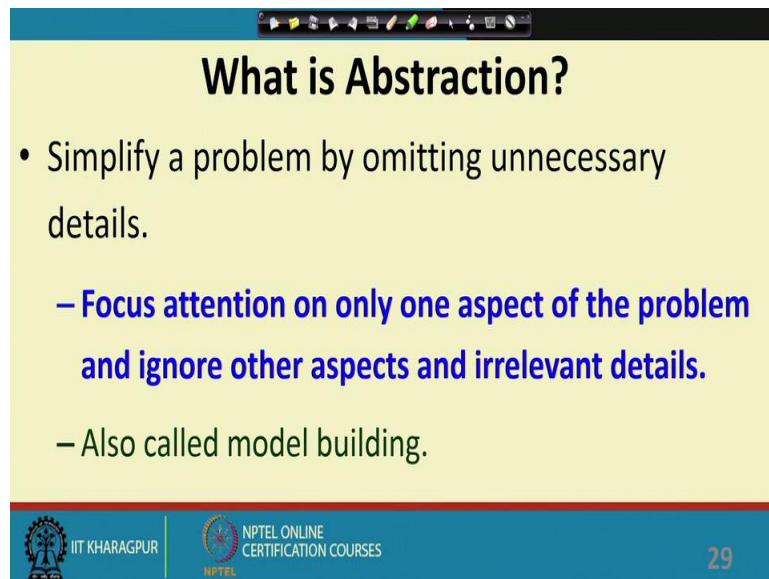
Lecture – 03
Introduction- III

In the last lecture we discussed about the human cognitive limitations & we stated that while developing large software this is the main problem that is being faced.

Software engineering is essential for developing large software as it tries to restrict its complexity so that the effort and time required to develop software is linear with the size of the software. Otherwise, if the software engineering principles are not applied and an intuitive approach like a build and fix module is used then the complexity growth, that is basically the cost, effort and the time for development increases exponentially.

As there are two major principles involved: one is abstraction and the other is decomposition.

(Refer Slide Time: 02:12)



What is Abstraction?

- Simplify a problem by omitting unnecessary details.
 - Focus attention on only one aspect of the problem and ignore other aspects and irrelevant details.
 - Also called model building.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

29

Abstraction is basically leaving out unnecessary part and focusing on some parts that you require. Given a problem if you look at the problem entirely, it appears very complex. We need to focus only on one aspect of the problem and ignore the rest. This process is basically called as Abstraction. As you focus, the attention is only on one aspect of the

problem and ignore the other aspects that are irrelevant to the point that you are focusing, this process is also known as Model Building.

Every abstraction requires construction of a model and a model focuses on one aspect and ignores the rest. For example: you have a very large and complex building to be developed and you focus on how will it appear. You will construct a frontal model of a building & ignore the rest including its material it is build off, the thickness, floor plan, internal design and etc as you are just focused on the frontal view of the building.

Every abstraction require some model building and a model basically focuses on some aspect and ignores the rest.

(Refer Slide Time: 04:23)

• Suppose you are asked to develop an overall understanding of some country.

– Would you:

- Meet all the citizens of the country, visit every house, and examine every tree of the country?
- You would possibly refer to various types of maps for that country only.

Abstraction Example

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES



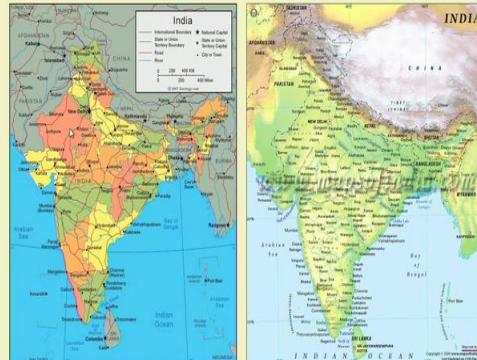
Considering a hypothetical case that you have been asked to develop an overall understanding of some country and given the responsibility to head the marketing & department of a company for some country.

Would you like go to that country move around try to meet people, look at where are the mountains, the rivers and etc. It will be extremely complex task as it will take tens or hundreds of years still you will not be able to develop a proper understanding. What can be really done is to refer to various types of maps of the country.

(Refer Slide Time: 05:51)

You would study an Abstraction...

- A map is:
 - An abstract representation of a country.
 - Various types of maps (abstractions) possible.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



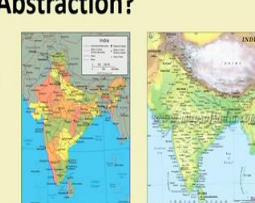
The maps are basically model or an abstraction of a country. We can study the political map which focuses on the different provinces, capital, major cities, railway road connectivity and etc. You will look into the physical map and try to find out the vegetation, the elevation of the different places, rivers, sea shore and etc.

An abstraction can help solve the complex problem very easily. One thing to notice is that there are various types of abstraction which can be made for the same problem.

(Refer Slide Time: 06:48)

Does every Problem have a single Abstraction?

- Several abstractions of the same problem can be created:
 - Focus on some specific aspect and ignore the rest.
 - Different types of models help understand different aspects of the problem.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Several abstractions to the same problem can be created. These focus on some specific aspect and ignore the rest. The different types of models help us to understand different aspects of the problem. For example, in the case of a building you would like a frontal model, a prototype, a floor plan and a thermal model of the building and etc. For the same problem various models can be created each focusing on some aspect of the problem.

(Refer Slide Time: 07:54)

The slide has a yellow header bar with the title 'Abstractions of Complex Problems'. The main content is divided into two columns:

- For complex problems:**
 - A single level of abstraction is inadequate.
 - A hierarchy of abstractions may have to be constructed.
- Hierarchy of models:**
 - A model in one layer is an abstraction of the lower layer model.
 - An implementation of the model at the higher layer.

To the right of the text columns is a white box containing a red dot-and-line network diagram representing a complex system or abstraction.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

If the problem is extremely complex, a single level of abstraction may itself be complicated. If these are the problems, we will create a first level abstraction then a second level abstraction and then abstraction of those second level abstraction and etc. For very complex problems you could create a hierarchy of abstractions. As you proceed with software design, you will look at hierarchy of abstractions that are used and in this hierarchical model, a model at any level will actually be the details on which the next level of model is built. Hence, a model is an abstraction of a lower level model and it's also an implementation of the higher-level model.

(Refer Slide Time: 09:32)

Abstraction of Complex Problems -- An Example

- Suppose you are asked to understand all life forms that inhabit the earth.
- Would you start examining each living organism?
 - You will almost never complete it.
 - Also, get thoroughly confused.
- **Solution: Try to build an abstraction hierarchy.**



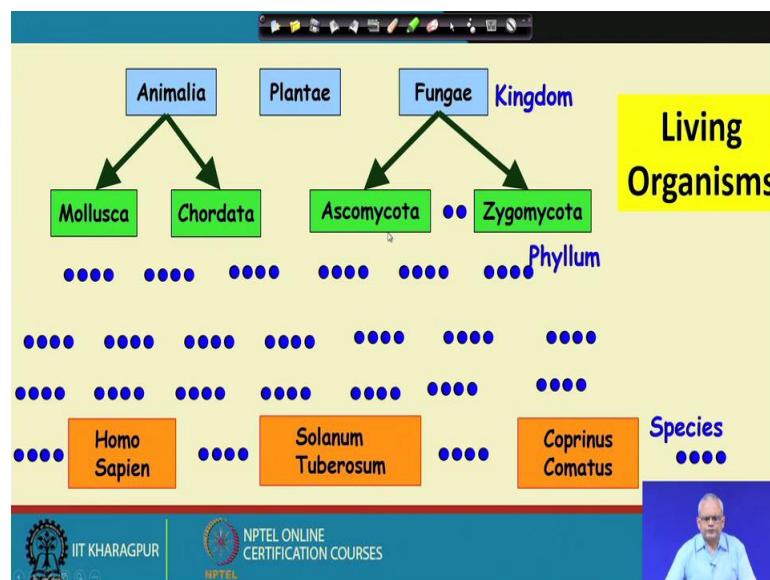
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Let's look at the abstraction of a complex problem that is to understand all the life forms that inhabit the earth containing billions or trillions of species.

In order to understand this life form, you can go on examining various species but nobody can complete the study in his lifetime. On the contrary you can create a hierarchy of abstraction.

(Refer Slide Time: 10:27)



Looking at a biology book you will be able to find such an abstraction, as at the topmost level there are only three types of living organisms these are: the animals, plant, and the

fungae. Then there are the mollusca, chordate & etc and the further hierarchies until you reach the bottom. Thus, for a complex problem the number of layers in the hierarchy can be large and each of these layers is basically a model.

(Refer Slide Time: 11:29)

The image shows a computer screen displaying a presentation slide. The title of the slide is "Quiz". Below the title, there are three bullet points listed:

- What is a model?
- Why develop a model? That is, how does constructing a model help?
- Give some examples of models.

At the bottom of the slide, there is a footer bar containing the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". To the right of the footer, there is a small video window showing a man speaking.

What exactly is a model. A model is an abstraction where we focus on some issues and ignore the rest. Several types of models can be there for a problem and if it is a complex problem, we would create a hierarchy of models.

Is it necessary to develop a model and does its construction help. As understanding a complex problem is extremely difficult due to human cognitive limitations. We would first understand the top of the hierarchy and then look at the other models to develop an incremental understanding of the problem.

Examples of models: Consider the content page of a book and etc.

You may come up with hundreds of different models that we use in the real life.

(Refer Slide Time: 13:47)

Decomposition

- Decompose a problem into many small independent parts.
 - The small parts are then taken up one by one and solved separately.
 - **The idea is that each small part would be easy to grasp and therefore can be easily solved.**
 - **The full problem is solved when all the parts are solved.**





IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another important principle is decomposition. Decomposition means to decompose a complex problem into small independent parts because the problem in its entirety is extremely complex, hard to understand & it takes exponentially large time. We would break the entire problem into small parts and examine the small problems. Once you put them together, you will get the understanding of the entire problem.

Example situations in daily life: There were many places where you can use the principle of decomposition. We need to divided a large problem into small parts and as you put them together, you will get the solution to the entire full problem.

(Refer Slide Time: 15:06)

Decomposition

- A popular example of decomposition principle:
 - Try to break a bunch of sticks tied together versus breaking them individually.
- Any arbitrary decomposition of a problem may not help.
 - The decomposed parts must be more or less independent of each other.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Popular example of the decomposition principle is that if you try to break a bunch of sticks tied together it would be extremely complex but if you decompose it then you can break the sticks individually.

You need to decompose them such that each small problem can be solved separately and this is also a corner stone in all software engineering principles.

(Refer Slide Time: 16:38)

Decomposition: Another Example

- Example use of decomposition principle:
 - You understand a book better when the contents are organized into independent chapters.
 - Compared to when everything is mixed up.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Consider a book is written half hazard, all mixed up. It would become extremely complex for someone trying to read the book. So, a chapter organization of a book is

required and it's actually a decomposition of a complex large thought into small manageable parts.

(Refer Slide Time: 17:31)

The slide has a blue header bar with standard window controls. The main title is "Why Study Software Engineering? (1)" in bold blue font. Below the title is a bulleted list:

- To acquire skills to develop large programs.
 - Handling exponential growth in complexity with size.
 - Systematic techniques based on abstraction (modelling) and decomposition.

A graphic of four interlocking puzzle pieces in green, yellow, blue, and pink is positioned to the right of the list. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video feed of a speaker.

Software engineering helps when you have large problems to solve that are the industry standard problems. But there are two main techniques: abstraction and decomposition. The benefits of the software engineering principles are as follows:

Firstly, you will get the skills to develop the large programs because the intuitive technique of build and fix, try to program and debug does not work for developing large programs. We should be able to create models that is abstraction and decompose the large programs, that would be the way to handle the exponential growth in complexity.

Hence, these are the systematic techniques which will help us in applying the principles of abstraction and decomposition and develop large programs.

(Refer Slide Time: 19:08)

The slide has a yellow header with the title 'Why Study Software Engineering? (2)' in blue. Below the title is a bulleted list of reasons:

- Learn systematic techniques of:
 - Specification, design, user interface development, testing, project management, maintenance, etc.
 - Appreciate issues that arise in team development.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

The second reason is that you will learn systematic techniques. To specify a problem, designing, do user interphase development, testing, project management, maintenance, etc and look into the issues that arise in team-based development.

(Refer Slide Time: 19:50)

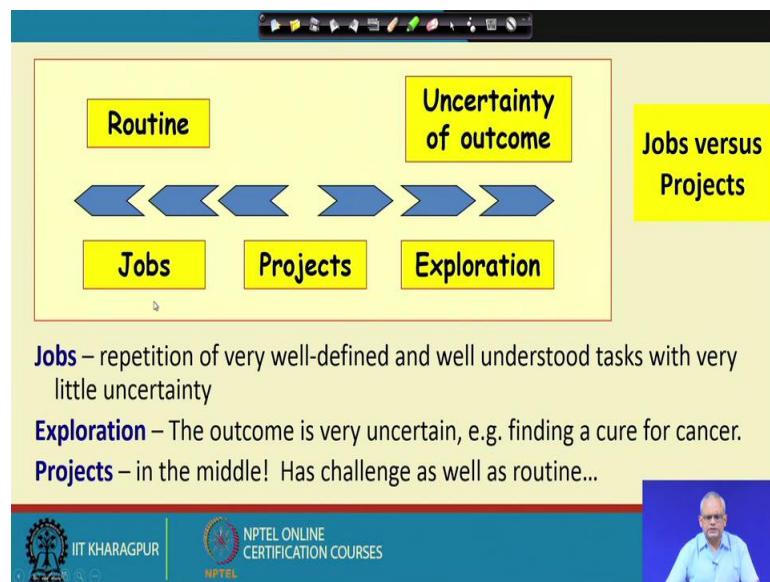
The slide has a yellow header with the title 'Why Study Software Engineering? (3)' in blue. Below the title is a bulleted list of reasons:

- To acquire skills to be a better programmer:
 - Higher Productivity
 - Better Quality Programs

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

The software engineering principles will be also useful to develop small programs in a better way after acquiring the required skillset. Even for small programs after studying the software engineering principles we will have a higher productivity and be able to write better quality programs.

(Refer Slide Time: 20:18)



The difference between the job and project. A job is a repetition of some well-defined and well understood tasks with little uncertainty whereas Project involves some challenges and exploration is where there is an uncertainty of outcome. For example: Consider to go to the market and fetch a chocolate, yes it can be done. It's a well-defined, well understood task can be easily accomplished, there is no uncertainty you can definitely do it.

An exploration is a work whose outcome is uncertain. For example: Consider to find a cure for cancer. You will go about doing the work but never know whether there will be any success at all. Project is somewhat in-between a job and exploration which is completely uncertain and also completely deterministic at the projects where there are many challenges but then some of the work is also routine.

(Refer Slide Time: 22:08)

The slide has a yellow header bar with the title 'Types of Software Projects'. Below it, there are two main bullet points:

- Two types of software projects:
 - Products (Generic software)
 - Services (custom software)
- Total business – Several Trillions of US \$
 - Half in products and half services
 - Services segment is growing fast!**

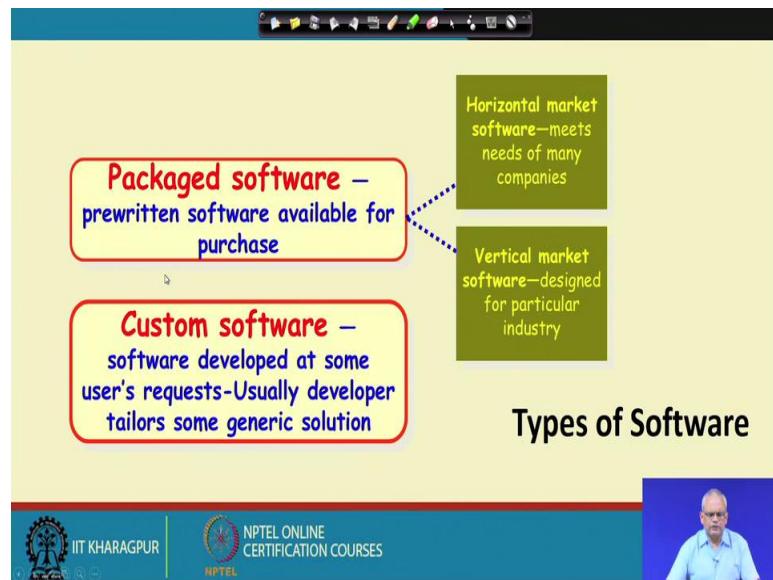
At the bottom, there are logos for IIT Kharagpur and NPTEL, along with a small video window showing a speaker.

A software development project consists of some routine work as well as there are challenges in it.

There are mainly two types of software projects: one is product development projects or generic software development projects and services projects or custom software development. The total size of the industry for software development is extremely large, several trillions of US dollar and half of this is about developing and selling products. On the other hand, the rest are services.

The growth in services is much more than that of products and soon we will have the services market more than the product market. Nearly, 50 ago, almost everything was product & there was hardly any services.

(Refer Slide Time: 23:58)



The issues in the software product development project and the services projects. The product development projects are basically packaged software. These are available for every individual. Anybody can order and buy this software. But if you look at the packaged software, they are of two types: one is the horizontal product which meets the need of mini companies. For example: Consider a word processing software or an operating system and etc. These are horizontal because a large number of companies and individuals are interested.

Whereas, a vertical market software or a vertical product focuses only a small group of companies. For example, considering the telecom domain or chemical plant simulation, that is another vertical market or the banking software where only the banks are interested.

There are of two types, one is the horizontal products which is developed for almost everyone whereas for the vertical products only a specific type of industry would be interested.

Every custom software will basically have some software generic version or might be done for some customer and get tailored or custom made for the other customers.

We will continue in the next lecture. Thank you.

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

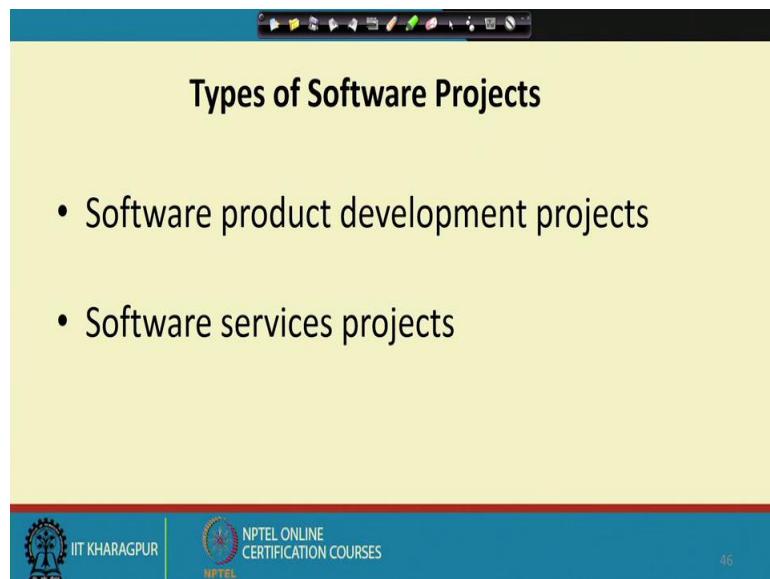
Lecture – 04
Introduction- IV

In the last lecture we discussed about the different types of software projects.

Looking into the different types of software projects that are been undertaken in the industry so that you will be able to appreciate the different principles which are used for the software development.

The software projects are basically of two types:

(Refer Slide Time: 00:58)



Types of Software Projects

- Software product development projects
- Software services projects

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES
NPTEL

One is the product development projects and the other are services projects.

(Refer Slide Time: 01:08)

The slide has a yellow background. At the top center is the title **Software Services**. Below the title is a bulleted list:

- Software service is an umbrella term, includes:
 - Software customization
 - Software maintenance
 - Software testing
 - Also contract programmers (CP) carrying out coding or any other assigned activities.

To the right of the list is a cartoon illustration of two people at a desk. One person is labeled "CP". At the bottom of the slide is a blue footer bar containing the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a man speaking.

As the software services are growing very fast, the market size for software services and the products are evenly balanced but on the contrary the software services projects are growing at a much faster rate. The software services project is basically an umbrella term in the sense that there are various types of software services projects. For example, Software customization, software maintenance and software testing. Consider some organization developed a software but the testing part has outsourced. Here, the software testing is software service. In another extreme form you may consider a company which supplies the contract programmers and a development company which will need some of these for carrying out some work and this is also considered a service.

(Refer Slide Time: 03:07)

Factors responsible for accelerated growth of services...

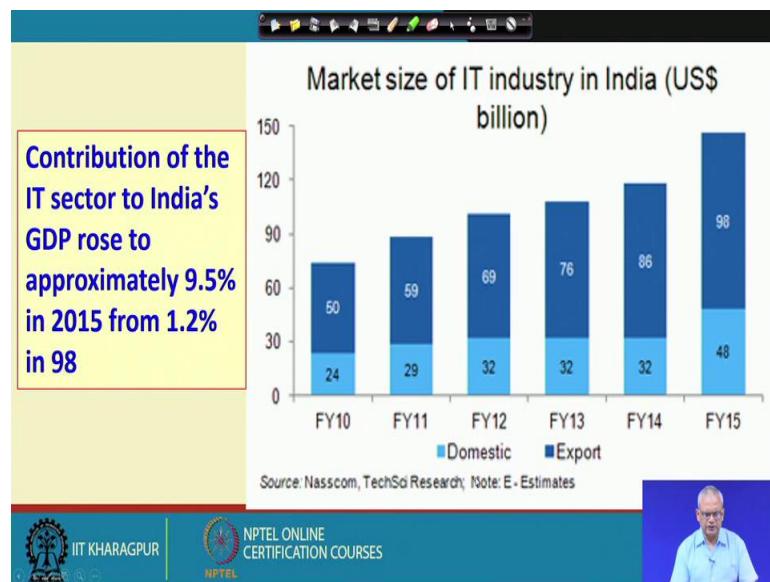
- Now lots of code is available in a company:
 - New software can be developed by modifying the closest.
- Speed of Conducting Business has increased tremendously:
 - Requires shortening of project duration

The number of services projects is large as compared to product development projects. 50-60 years back we had very little code written. Meeting the requirements of any customer needed to develop software from scratch. But, now a days to develop a new software, a lot of reuse is being made from the existing software.

Consider a example that an education institute decides to automate various book keeping activities. If it was in the 1950s or 60s our project would take a year or two to develop the software and install at the educational institute. But right now, a month is an average time in which such a work can be completed or even within 15 days. So, the project duration has shortened and somebody can really develop a million-line code by basically tailoring out some existing software.

As the speed of conducting business has increased tremendously not only in the education institute but all the companies, they want software to be quickly developed and installed and that is one of another reason that the services projects have really increased.

(Refer Slide Time: 05:38)



Looking at the market size of the IT industry in India it is growing rapidly over the last few years as per the report from the NASSCOM. The IT industry in India has made rapid progress, large number of projects are existing and majority of them have been completed. There is almost a growth of 10 percent of the GDP which comes from the IT industry.

(Refer Slide Time: 06:49)

A Few Changes in Software Project Characteristics over Last 40 Years

- 40 years back, very few software existed
 - Every project started from scratch
 - Projects were multi year long
- The programming languages that were used earlier hardly provided any scope for reuse:
 - FORTRAN, PASCAL, COBOL, BASIC
- No application was GUI-based:
 - Mostly command selection from displayed text menu items.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Indian Software Companies had excelled in the services segment, almost every project is a services type of project.

As you know that worldwide 50 percent of the development projects are product development and other 50 percent is services. But in India its entirely services project.

The reasons are that: in product development system there is a large gestation and risk to develop a product. It may do well in the market or just flop. Lots of investment may vanish. But, if a product development project succeeds then it keeps on giving return year after a year whereas the services project is typically paid only once. But the Indian companies concentrate only on the services projects. There are only few product developments projects and some success stories about products in the Indian context.

In the initial years of software development very few software existed. Almost every project started from scratch and projects were multiyear like 2,3,4,5-year long projects existed and the programming languages used were FORTRAN, PASCAL, COBOL, BASIC, etc. These programming languages didn't support the reuse of code and almost all application was NON- GUI based. Now, every application has a GUI and heavy reuse is made. The project duration has really minimized from multiyear to just couple of months and the programming languages supports reuse of the code.

(Refer Slide Time: 10:23)

Traditional versus Modern Projects

- Projects are increasingly becoming services:
 - Either tailor some existing software or reuse pre-built libraries.
- Facilitate and accommodate client feedbacks
- Facilitate customer participation in project development work
- Incremental software delivery with evolving functionalities.
- **No software is being developed from scratch --- Significant reuse is being made...**

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES
NPTEL



The traditional versus modern projects: Services segment is having a large growth rate, we tailor some existing software or try to reuse prebuilt library. In the modern projects the client is a part of the project and the client gives feedback according to the

requirement. There are product development projects largely and once the development starts no feedback can be taken, it freezes the requirements and start according to that.

Right now, the customer is part of the project development. Even the large software is developed incrementally. That means that every 2 week the software is installed at the client and it then evolves new functionalities which gets added.

No software is being developed from scratch and significant reuse of code is made.

(Refer Slide Time: 11:57)

The slide has a yellow background and a blue header bar. The title 'Computer Systems Engineering' is in the center of the yellow area. Below the title are two bullet points in blue:

- Many products require development of software as well as specific hardware to run it:
 - a coffee vending machine,
 - a robotic toy,
 - A new health band product, etc.
- Computer systems engineering:
 - encompasses software engineering.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '53'.

What is computer systems engineering.

In many developments work the hardware and software are developed together.

There are many systems where the hardware also has to be designed from scratch and the software will run on that hardware. For example: Consider a coffee vending machine or a robotic toy. Here, the robotic toy has a special processor containing its mechanical parts and etc and the software has to run on that specific processor which are not general type of processor.

Let us also consider a new health band product. This product has a processor and health band which monitors the health and displays various suggestions and feedback. For this type of system where there is a hardware that needs to be developed and also the

software which has to work on that hardware, it is known as System Engineering. It's a superset of the software engineering.

(Refer Slide Time: 13:54)

The slide has a blue header bar with standard window controls. The main title 'Computer Systems Engineering' is centered in large blue font. Below the title is a bulleted list:

- The high-level problem:
 - Deciding which tasks are to be solved by software.
 - Which ones by hardware.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

How is the computer systems engineering problem solved.

Initially, a high-level understanding is made and then decide which part has to be solved by software and which part by hardware and its development start parallelly. The software will be tested by using a simulator and get developed.

(Refer Slide Time: 14:35)

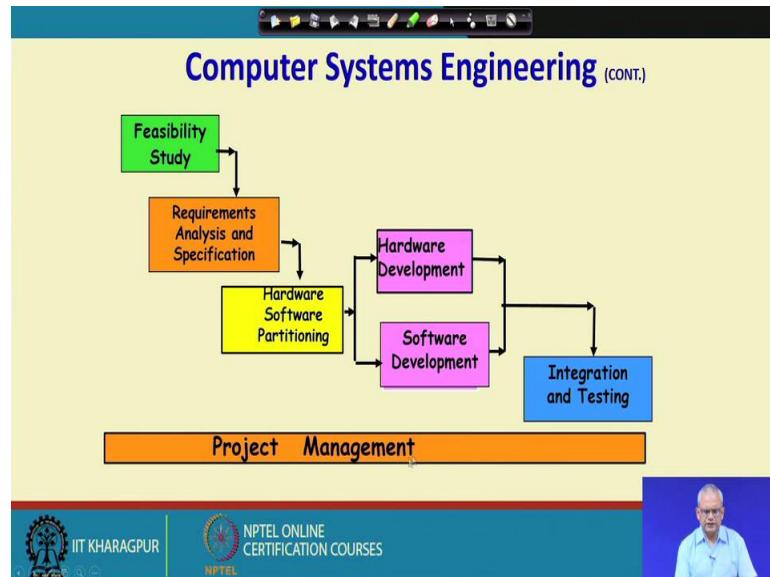
The slide continues from the previous one, with the same blue header and footer. The title is now '(CONT.)'. The main content is a bulleted list:

- Typically, hardware and software are developed together:
 - Hardware simulator is used during software development.
- Integration of hardware and software.
- Final system testing

The video player at the bottom shows the same man speaking.

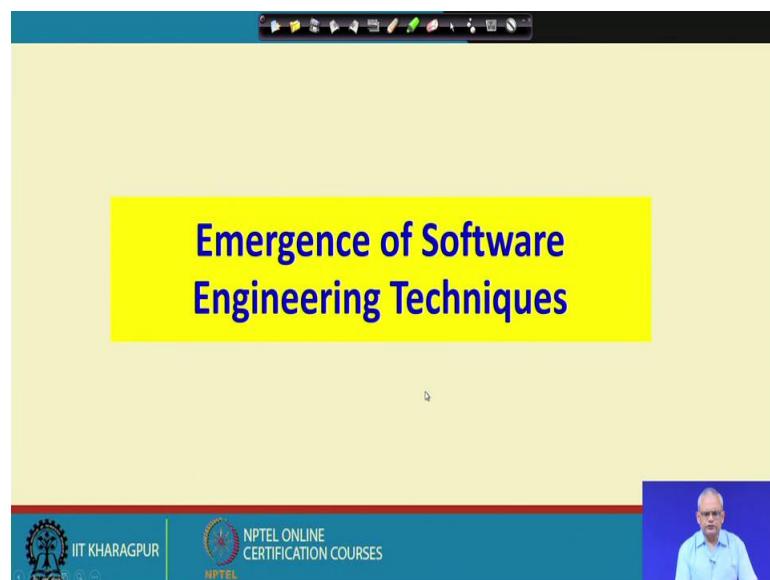
The hardware simulator is used during software development and once the hardware and software development is complete, they are integrated and the entire full system is tested.

(Refer Slide Time: 14:56)



This is a model or a schematic of the computer systems engineering. Throughout the entire project duration, the project management activity is carried out to manage the various activities in the system engineering project.

(Refer Slide Time: 15:49)



Different software engineering techniques: Starting with simple techniques, the software techniques have become more and more sophisticated.

(Refer Slide Time: 16:25)

Emergence of Software Engineering Techniques

- Early Computer Programming (1950s):

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | 58

In early 1950s in the computer programming, the programs were small.

(Refer Slide Time: 16:37)

Early Computer Programming (50s)

- Every programmer developed his/her own style of writing programs:
 - According to his intuition (called **exploratory** or **build-and-fix programming**) .

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

It was written using language like FORTRAN. Every programmer had his own style that is basically the build and fix programming and every program was developed according to the intuition of the programmer.

(Refer Slide Time: 17:00)

High-Level Language Programming (Early 60s)

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:
 - This reduced software development efforts greatly.
 - Why reduces?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Languages like FORTRAN, ALGOL, COBOL were used later in 1960s and the assembly language used were restricted. But the productivity improved significantly with high level language programming.

The productivity of high-level programming languages was higher as compared to assembly language programming.

It's because of two reasons: Firstly, for one single line of high-level program code you need to write many assembly codes and the high-level code is easily understood as the machine language is much more involved. As you will deal with abstraction that would deal only with the variables.

So, high level language programming is much simpler. It abstracts out the details of hardware organization architecture.

(Refer Slide Time: 18:51)

High-Level Language Programming (Early 60s)

- Software development style was still exploratory.
 - Typical program sizes were limited to a few thousands of lines of source code.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Even though the high level programming languages were used the development was largely exploratory and the typical program size was only about thousand lines of code.

(Refer Slide Time: 19:10)

Control Flow-Based Design (late 60s)

- Size and complexity of programs increased further:
 - Exploratory programming style proved to be insufficient.
- Programmers found:
 - Very difficult to write cost-effective and correct programs.

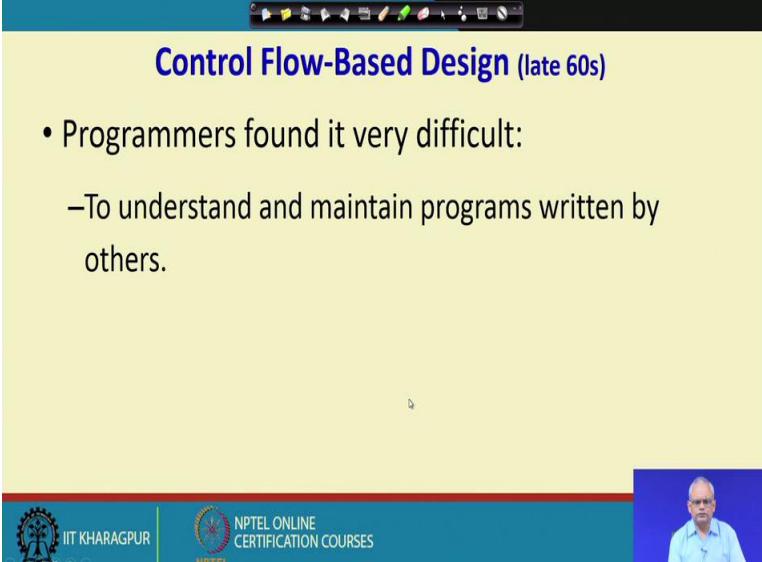
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The size of the program increased and the exploratory style became insufficient. It was taking long to develop code as there were many bugs.

(Refer Slide Time: 19:41)

Control Flow-Based Design (late 60s)

- Programmers found it very difficult:
 - To understand and maintain programs written by others.

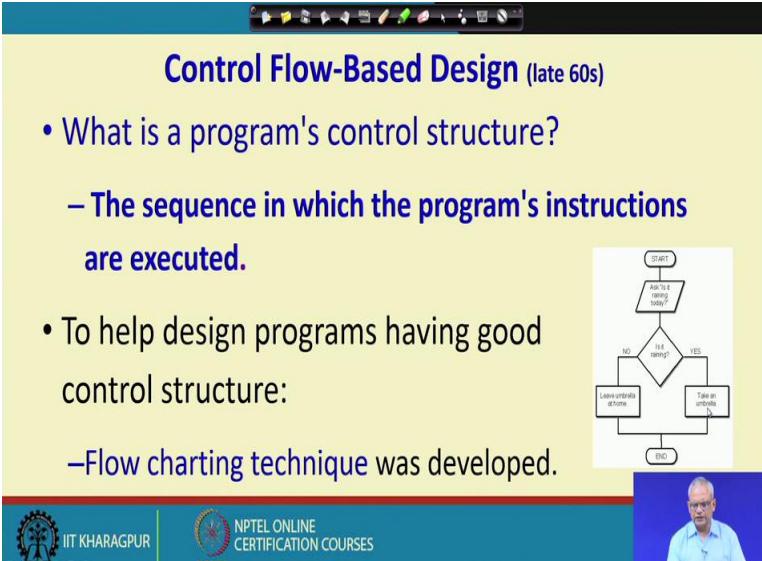


To develop a good program you need to pay attention to the control flow.

(Refer Slide Time: 19:49)

Control Flow-Based Design (late 60s)

- What is a program's control structure?
 - The sequence in which the program's instructions are executed.
- To help design programs having good control structure:
 - Flow charting technique was developed.



The control structure of a program is the sequence in which the programs instructions are executed. It may get altered when there are decisions, loops and etc.

By looking at the control structure you can determine in what sequence the program will get executed.

The programmers got advised from the good programmers to pay attention to the control structure. Thus, the flow-charting technique was developed. The flow charts were used to represent the programs control structure and based on these the code was written.

(Refer Slide Time: 21:36)

Control Flow-Based Design (late 60s)

- Using flow charting technique:
 - One can represent and design a program's control structure.
 - When asked to understand a program:
 - One would mentally trace the program's execution sequence.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



To understand a code having this control structure, you have to understand the different paths through it starting from the first statement which are the statements that is executed and where the output is generated.

(Refer Slide Time: 22:07)

Control Flow-Based Design

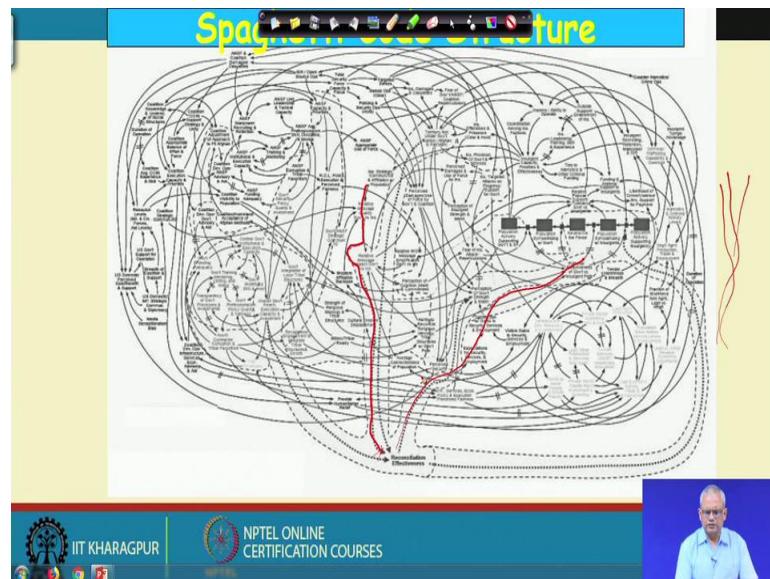
- A program having a messy flow chart representation:
 - Difficult to understand and debug.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



A program having a bad control flow representation will be very difficult to understand and debug as it will have lot of errors.

(Refer Slide Time: 22:43)



To understand this program, you have to trace the different paths through it. Imagine that there are thousands - millions of paths.

To be able to trace the execution through all the paths will be extremely tedious as it will require a long of time. But if it had a good control structure and only few paths (just 3-4) then you can easily understand it compared to a very complex program control structure.

(Refer Slide Time: 24:21)

- What causes program complexity?
 - GO TO statements makes control structure of a program messy.
 - GO TO statements alter the flow of control arbitrarily.
 - The need to restrict use of GO TO statements was recognized.

It gave rise to the control flow base design. The control flow base design consider to have a good control structure for the program.

The GO TO statements were the culprits and they make the control structure bad. Earlier programming languages like FORTRAN,etc had the GO TO statements and they were used heavily.

(Refer Slide Time: 25:09)

Control Flow-Based Design (Late 60s)

- Many programmers had extensively used assembly languages.
 - JUMP instructions are frequently used for program branching in assembly languages.
 - Programmers considered use of GO TO statements inevitable.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video player window in the bottom right corner shows a man speaking.

The assembly programmers said that without jump instruction you cannot write a program. As GO TO statements were inevitable and without using GO TO statement you couldn't write sophisticated programs.

(Refer Slide Time: 25:38)

Control-flow Based Design (Late 60s)

- At that time, Dijkstra published his article:
 - “Goto Statement Considered Harmful” Comm. of ACM, 1969.
- Many programmers were unhappy to read his article.

Dijkstra published an article in the communication of ACM 1969, a landmark article known as ‘Goto Statement Considered Harmful’. He mentioned about the problems that a GO TO statement creates and obviously, many programmers who were basically having assembly programming background were unhappy.

(Refer Slide Time: 26:04)

Control Flow-Based Design (Late 60s)

- Some programmers published several counter articles:
 - Highlighted the advantages and inevitability of GO TO statements.

On the contrary, they wrote counter articles and said that without GO TO statement you cannot really write a large program.

(Refer Slide Time: 26:14)

Control Flow-Based Design (Late 60s)

- It soon was conclusively proved:
 - Only three programming constructs are sufficient to express any programming logic:

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

To solve any programming problem that includes the large complex problems you need only three types of constructs: the sequence, selection and the alteration constructs.

(Refer Slide Time: 26:36)

- It is possible to solve any programming problem without using GO TO statements.
- This formed the basis of Structured Programming methodology.
'."/>

Control-flow Based Design (Late 60s)

- Everyone accepted:
 - It is possible to solve any programming problem without using GO TO statements.
 - This formed the basis of **Structured Programming methodology.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Slowly everybody accepted that it is possible to write large programs without using GO TO statements and this earned the basis of the structured programming methodology.

(Refer Slide Time: 26:48)

The slide has a yellow background. At the top, the title 'Structured Programming' is displayed in blue. Below the title, there is a bulleted list of points. The first point is '• A program is called structured:'. Underneath this, there are two sub-points: '–When it uses only the following types of constructs:' and '–Consists of modules.' To the right of the text area, there is a small video window showing a man speaking.

• A program is called structured:
–When it uses only the following types of constructs:
•sequence,
•selection,
•iteration
–Consists of modules.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Structured programming. A program is called structured when it uses only three types of constructs: the sequence, selection and iteration. Sequence is one statement after the other statement, like one arithmetic statement followed by another arithmetic statement. Selection like if, then, else, switch and etc. Iteration like for loop, while loop, do loop and etc.

This basic feature of a structured program is that it only uses the sequence, selection and iteration type of statements and it also consists of modules.

(Refer Slide Time: 27:58)

The slide has a yellow background. At the top, the title 'Structured Programs' is displayed in blue. Below the title, there is a bulleted list of points. The first point is '• Sometimes, violations to structured programming are permitted:'. Underneath this, there are two sub-points: '◦ Due to practical considerations such as:' and '◦ Premature loop exit (break) to support exception handling.' To the right of the text area, there is a small video window showing a man speaking.

• Sometimes, violations to structured programming are permitted:
◦ Due to practical considerations such as:
◦ Premature loop exit (break) to support exception handling.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Sometimes, you have to use GO TO statement and in the modern languages you do not require the use of GO TO but for the statements like break, premature loop exit, exceptions and etc. These are basically not structured constructs but occasionally it is allowed due to practical considerations.

(Refer Slide Time: 28:32)

Advantages of Structured programming

- Structured programs are:
 - Easier to read and understand,
 - Easier to maintain,
 - Require less effort and time for development.
 - Less buggy

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video player window shows a man in a blue shirt speaking.

Advantages of structured program. A structured program has a good control structure because it does not use GO TO statements and has only few paths. It is simple and therefore it is easy to read and understand and maintain.

It requires less effort and time for development and is less buggy whereas in non-structured program the bug may exist in one of the paths which you never thought and ever existed.

The structured programming principle has been accepted. It is a very popular technique. The modern programming languages facilitate writing structured programs because they do not have constructs.

We will continue in the next lecture.

Thank you.

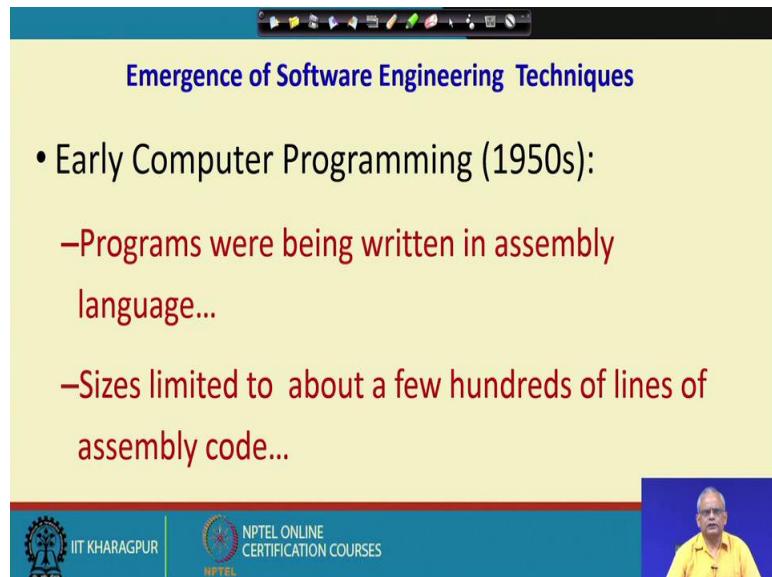
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 05
Introduction- V

In the last lecture we looked at some introductory issues on software engineering.

Now, we will discuss how briefly these techniques have evolved over time which will provide a better understanding about how the techniques have come into action.

(Refer Slide Time: 01:16)



Emergence of Software Engineering Techniques

- Early Computer Programming (1950s):
 - Programs were being written in assembly language...
 - Sizes limited to about a few hundreds of lines of assembly code...

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In the early years of computer programming(1950s) all the programs were written in assembly language and the size of program were small comprising of few hundreds of lines of assembly code.

(Refer Slide Time: 01:40)

The slide has a yellow background. At the top center, the title 'Early Computer Programming (50s)' is displayed in blue. Below the title, there is a bulleted list in blue text:

- Every programmer developed his/her own style of writing programs:
 - According to his intuition (called exploratory or build-and-fix programming) .

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there are two logos: IIT Kharagpur and NPTEL. To the right of the logos, the text 'NPTEL ONLINE CERTIFICATION COURSES' is written. On the far right of the footer, there is a small video window showing a man in a yellow shirt speaking.

The programmers used the build and fix or the exploratory style as per their ease of convince.

(Refer Slide Time: 02:02)

The slide has a yellow background. At the top center, the title 'High-Level Language Programming (Early 60s)' is displayed in blue. Below the title, there is a bulleted list in blue text:

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:
 - This reduced software development efforts greatly.
 - Why reduces?

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there are two logos: IIT Kharagpur and NPTEL. To the right of the logos, the text 'NPTEL ONLINE CERTIFICATION COURSES' is written. On the far right of the footer, there is a small video window showing a man in a yellow shirt speaking.

In, 1960s higher level languages came into action like FORTRAN, ALGOL, COBOL, etc and this increased the productivity greatly.

Writing in a high-level language is much more productive than in assembly language.

It's because of the three main reasons: First, in assembly language one needs to write the program considering the register contents, machines architecture and etc. Using a high-level language, one writes the program in terms of the variables which corresponds to the real problem or the high-level language abstracts the machine details. Secondly, each high-level construct is equivalent to writing 3-4 assembly instructions. And, the third reason is that the high-level language programs are much nearer to the human's language than the assembly & hence it becomes easy to write programs.

(Refer Slide Time: 03:57)

High-Level Language Programming (Early 60s)

- Software development style was still exploratory.
 - Typical program sizes were limited to a few thousands of lines of source code.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In 1960s, even though programs were written in high level language still the development was exploratory. Everyone use their own intuition to write the program and it worked because the program size was small, just about a few thousand lines of code.

(Refer Slide Time: 04:24)

The slide has a yellow background. At the top center, the title 'Control Flow-Based Design (late 60s)' is displayed in blue. Below the title, there are two bullet points:

- Size and complexity of programs increased further:
 - Exploratory programming style proved to be insufficient.
- Programmers found:
 - Very difficult to write cost-effective and correct programs.

At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man in a yellow shirt.

Towards the end of that decade, the program sizes started to increase and it became extremely difficult for programmers to write using the exploratory style as there were lot of bugs involved.

(Refer Slide Time: 05:00)

The slide has a yellow background. At the top center, the title 'Control Flow-Based Design (late 60s)' is displayed in blue. Below the title, there are two bullet points:

- Programmers found it very difficult:
 - To understand and maintain programs written by others.
- To cope up with this problem, experienced programmers advised--"Pay particular attention to the design of the program's control structure."

At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man in a yellow shirt.

The good programmers advised to “pay attention to the design of the program’s control structure”.

(Refer Slide Time: 05:31)

Control Flow-Based Design (late 60s)

- What is a program's control structure?
 - The sequence in which the program's instructions are executed.
- To help design programs having good control structure:
 - Flow charting technique was developed.

A video player window showing a man speaking, likely the professor.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Control structure of a program: A control structure of a program is the sequence in which the programs statements can be executed.

The flow-charting technique was developed to design good control structure. Before writing the program, represent the logic or the algorithm of the program in a flow chart and then translate the flow chart into code.

(Refer Slide Time: 06:38)

Control Flow-Based Design (late 60s)

- Using flow charting technique:
 - One can represent and design a program's control structure.
- When asked to understand a program:
 - One would mentally trace the program's execution sequence.

A video player window showing a man speaking, likely the professor.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Why the flow-charting technique or a good control structure helps in writing quality programs.

(Refer Slide Time: 07:22)

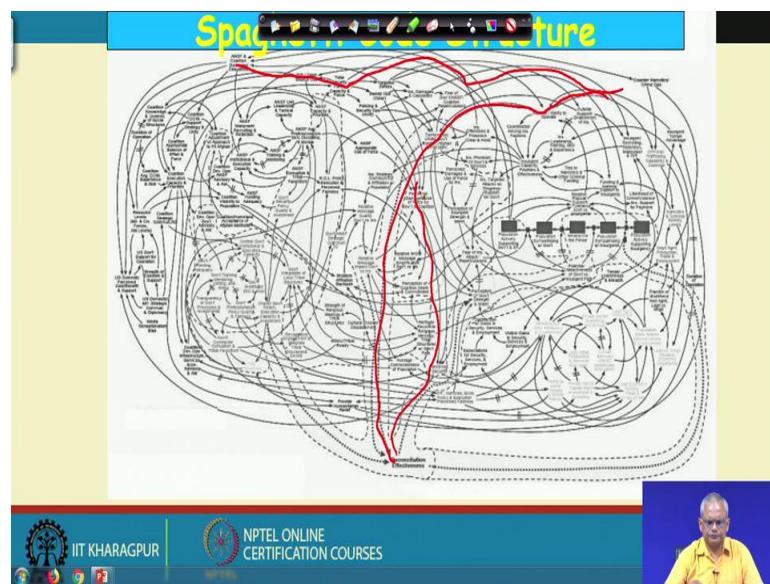
The slide has a yellow background. At the top, the title 'Control Flow-Based Design' is displayed in blue. Below the title is a bulleted list:

- A program having a messy flow chart representation:
- Difficult to understand and debug.

At the bottom of the slide, there is a footer bar containing the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video window showing a man in a yellow shirt speaking.

If you do not use a control structure design and write the program as it comes to mind then the structure will be poor and it will become very difficult for programmer to understand and debug.

(Refer Slide Time: 07:44)



Looking at a program having a bad control structure. To understand it one needs to find every output that is produced, the instructions that are executed and etc. It's extremely difficult to trace from where the input might have come.

To understand this piece of code even if you try for a year or two, you would still not have a full understanding of the program.

But, if we design the control structure using a flow chart: the number of paths will be limited and in reasonable small time, you could understand the program & debug it as the program development becomes fast and the productivity also increases.

(Refer Slide Time: 09:37)

Control Flow-Based Design (Late 60s)

- What causes program complexity?
 - GO TO statements makes control structure of a program messy.
 - GO TO statements alter the flow of control arbitrarily.
 - The need to restrict use of GO TO statements was recognized.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

As the control flow base design reduces the complexity. One thing that was observed by the experienced programmers was that the use of GO TO statements make the control structure bad. As long as a programmer writes program full of GO TO statements, even if he had designed the control structure & etc, he will end up writing bad code.

Hence, the good programmers advised not to use GO TO statements as they will ultimately make the program structure bad.

(Refer Slide Time: 10:29)

Control Flow-Based Design (Late 60s)

- Many programmers had extensively used assembly languages.
 - JUMP instructions are frequently used for program branching in assembly languages.
 - Programmers considered use of GO TO statements inevitable.

```
addi $a0, $0, 1
j next
next:
j skip1
add $a0, $a0, $a0
skip1:
j skip2
add $a0, $a0, $a0
add $a0, $a0, $a0
skip2:
j skip3
loop:
add $a0, $a0, $a0
add $a0, $a0, $a0
add $a0, $a0, $a0
skip3:
j loop
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In the late 1960s, there were many assembly programmers. Every assembly program typically had many jump statements and they thought that it would be impossible to write a program without using jump statements as the GO TO statements were inevitable.

(Refer Slide Time: 11:41)

Control-flow Based Design (Late 60s)

- At that time, Dijkstra published his article:
 - “Goto Statement Considered Harmful” Comm. of ACM, 1969.
- Many programmers were unhappy to read his article.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Dijkstra published a article in the communications of ACM 1969 “Goto Statement Considered Harmful” and he argued that the GO TO statements will not be used unless it is absolutely necessary.

(Refer Slide Time: 12:19)

Control Flow-Based Design (Late 60s)

- Some programmers published several counter articles:
 - Highlighted the advantages and inevitability of GO TO statements.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

They said that GO TO statements are actually advantageous as they help write efficient programs and it's inevitable. (Refer Slide Time: 12:32)

Control Flow-Based Design (Late 60s)

- It soon was conclusively proved:
 - Only three programming constructs are sufficient to express any programming logic:
 - sequence (a=0;b=5;)**
 - selection (if(c==true) k=5 else m=5;)**
 - iteration (while(k>0) k=j-k;)**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

To express any programming logic only three types of programming constructs are needed. These are sequence, selection and the iteration type of constructs. Naturally every programming language provides these three categories of constructs.

(Refer Slide Time: 13:43)

The slide has a blue header bar with standard window controls. The main title 'Control-flow Based Design' is in bold blue text, with '(Late 60s)' in smaller blue text to its right. Below the title is a bulleted list in blue text:

- Everyone accepted:
 - It is possible to solve any programming problem without using GO TO statements.
 - This formed the basis of **Structured Programming methodology**.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a yellow shirt speaking.

A program written without using GO TO statements forms the basis of structured programming and that's one of the main requirements of a structured programming.

(Refer Slide Time: 14:05)

The slide has a blue header bar with standard window controls. The main title 'Structured Programming' is in bold blue text. Below the title is a bulleted list in blue text:

- A program is called **structured**:
 - When it uses only the following types of constructs:
 - sequence,
 - selection,
 - iteration
 - Consists of **modules**.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a yellow shirt speaking.

In the concept of structured programming no GO TOs are used, only three types of construct these are the sequence, selection and iteration.

(Refer Slide Time: 14:33)

The slide has a yellow background. At the top center, the title 'Structured Programs' is displayed in blue. Below the title, there is a bulleted list:

- Sometimes, violations to structured programming are permitted:
 - Due to practical considerations such as:
 - Premature loop exit (break) or for exception handling.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a yellow shirt speaking.

Sometimes they are necessary for practical consideration. For example, premature loop exist which is a form of GO TO statement or exception handling.

(Refer Slide Time: 15:16)

The slide has a yellow background. At the top center, the title 'Advantages of Structured programming' is displayed in blue. Below the title, there is a bulleted list:

- Structured programs are:
 - Easier to read and understand,
 - Easier to maintain,
 - Require less effort and time for development.
 - Less buggy

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a yellow shirt speaking.

If someone does not write structured programs then he will face a lot of problems comprising the control structure will become bad and will be difficult to understand and hard to maintain. The program itself will require too much effort & time and it will carry a lot of bugs.

The structured program has lot of advantages and it's advised to write structured programs using the sequence, selection, iteration constructs and modular programs. As, a result the structured program has become inbuilt into the programming languages.

(Refer Slide Time: 17:16)

Structured Programming

- Research experience shows:
 - Programmers commit less number of errors:
 - While using structured **if-then-else** and **do-while** statements.
 - Compared to **test-and-branch (GOTO)** constructs.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And, the advantages of structured programming was proved through large number of experimental observations.

(Refer Slide Time: 17:32)

Data Structure-Oriented Design (Early 70s)

- As program sizes increased further, soon it was discovered:
 - It is important to pay more attention to the design of data structures of a program**
 - Than to the design of its control structure.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Paying attention to control structure is not sufficient. As the programming is expensive, developing large programs usually takes long time and consists of many bugs.

For developing large programs paying attention to the kind of data structure is required and the type of data structures should be designed very carefully.

(Refer Slide Time: 18:30)

Data Structure-Oriented Design (Early 70s)

- Techniques which emphasize designing the data structure:
 - Derive program structure from it:
 - Are called **data structure-oriented design techniques**.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The data structure which oriented the design in the early 1970s, the main objective was to design the programs data structure and the respective program structure can be derived from it.

(Refer Slide Time: 19:01)

Data Structure Oriented Design (Early 70s)

- An example of data structure-oriented design technique:
 - Jackson's Structured Programming(JSP) methodology
 - Developed by Michael Jackson in 1970s.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Many data structure-oriented design techniques became popular in 1970s; the Jackson's Structured Programming or the JSP methodology.

(Refer Slide Time: 19:17)

Data Structure Oriented Design (Early 70s)

- **JSP technique:**
 - Program code structure should correspond to the data structure.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

They provided constructs to design the data structure, to represent the data structure in diagrammatic form and then to write a program based on the data structure.

(Refer Slide Time: 19:39)

A Data Structure Oriented Design (Early 70s)

- **JSP methodology:**
 - A program's data structures are first designed using notations for
 - sequence, selection, and iteration.
 - The data structure design is then used :
 - To derive the program structure.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

They had notations for representing sequence, selection, and iteration and then these are translated into program structure.

(Refer Slide Time: 20:00)

Data Structure Oriented Design (Early 70s)

- Several other data structure-oriented Methodologies also exist:
 - e.g., Warnier-Orr Methodology.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Other data structure oriented design technique was also very popular; namely the Warnier-Orr Methodology.

(Refer Slide Time: 20:11)

Data Flow-Oriented Design (Late 70s)

- Data flow-oriented techniques advocate:
 - The data items input to a system must first be identified,
 - Processing required on the data items to produce the required outputs should be determined.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

To get really good program we must pay particular attention to the data flows through the program. That is starting from the input what kind of processing is done until the output is done. It had a DFD notation, the data flow diagram notation where you can first represent the data flow from the input, until the output comes out. Once, we have the data flow representation for a program it can be transformed into a good design.

(Refer Slide Time: 21:53)

Data Flow-Oriented Design (Late 70s)

- Data flow technique identifies:
 - Different processing stations (functions) in a system.
 - The items (data) that flow between processing stations.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

85

Starting with the data input, the processing that is done are called as the processing stations or functions and the data which flows between these processing stations is represented and the final design is obtained.

(Refer Slide Time: 22:16)

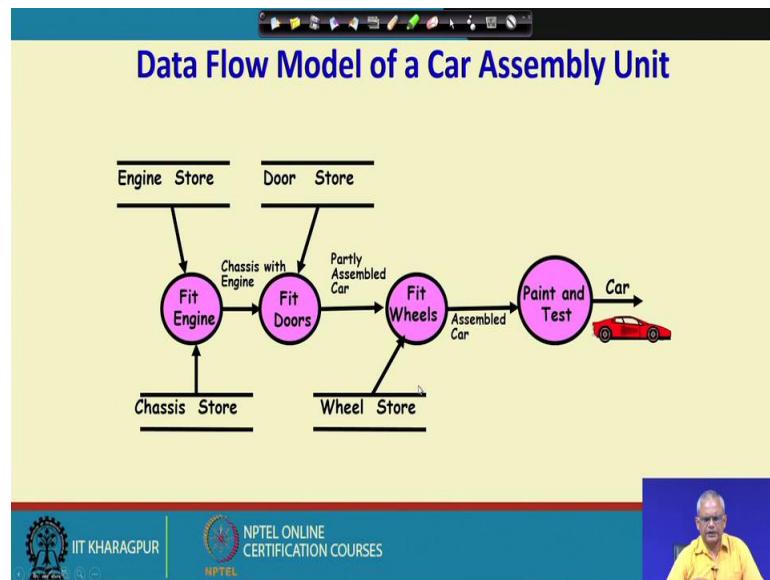
Data Flow-Oriented Design (Late 70s)

- Data flow technique is a generic technique:
 - Can be used to model the working of any system.
 - not just software systems.
- A major advantage of the data flow technique is its simplicity.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

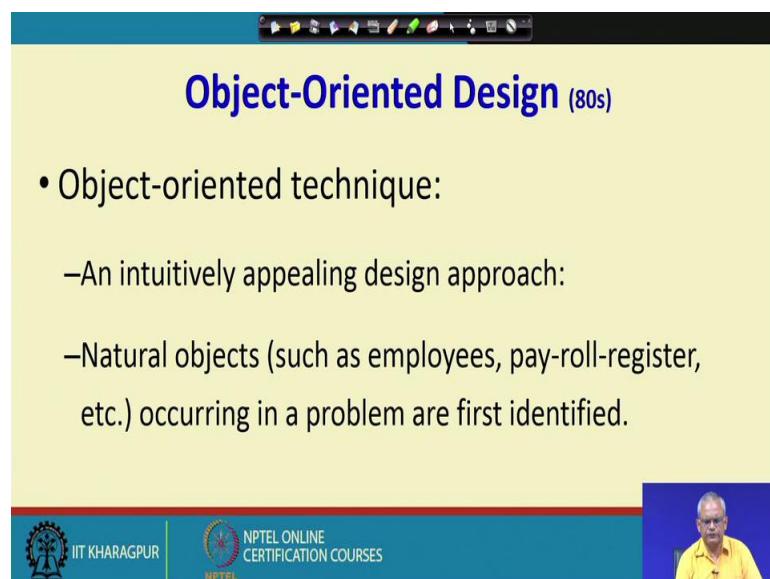
As the data flow diagram is a very simple technique and hardly requires an hour to learn this technique. It can be used to model the working of any system.

(Refer Slide Time: 22:56)



Consider a car assembly unit, where there are processing stations. One processing station takes engines & chassis and it fix the engine to the chassis. Then the chassis with the engine goes to another processing station where it takes out the doors and fits the door to the chassis and it proceeds to the next processing station. The representation of two parallel lines which store, gets it fits the wheels and then the assembled car comes out which is painted and ready to dispatch.

(Refer Slide Time: 23:54)



Subsequently the object-oriented technique in 1980s came into picture. This had the appeal of representing the natural objects in the problem and writing using this technique become quite sophisticated.

(Refer Slide Time: 24:50)

The slide has a yellow background. At the top center, it says "Object-Oriented Design (80s)". Below that is a bulleted list:

- Relationships among objects:
 - Such as composition, reference, and inheritance are determined.
- Each object essentially acts as:
 - A data hiding (or data abstraction) entity.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a person speaking.

The main advantage of the object oriented design is that it leads to a very good modular design because the objects are essentially similar to a good modules which have data hiding or data abstraction entity.

(Refer Slide Time: 25:33)

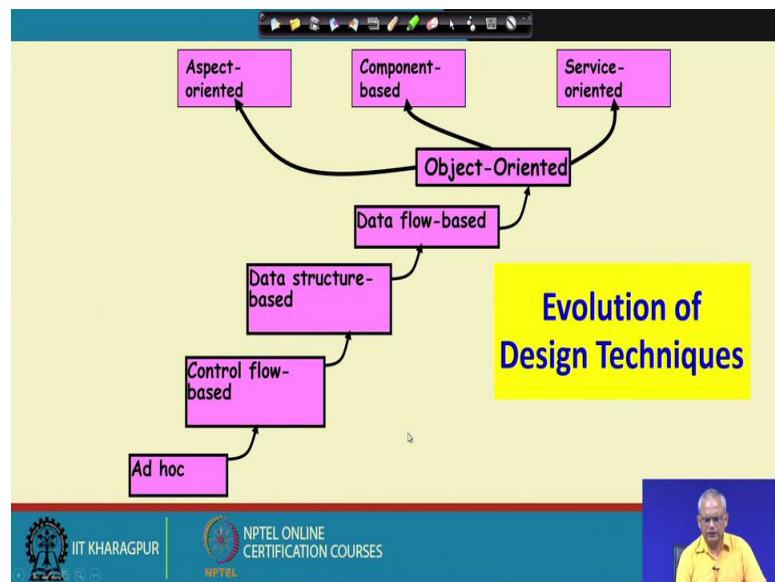
The slide has a yellow background. At the top center, it says "Object-Oriented Design (80s)". Below that is a bulleted list of advantages:

- Object-Oriented Techniques have gained wide acceptance:
 - Simplicity
 - Increased Reuse possibilities
 - Lower development time and cost
 - More robust code
 - Easy maintenance

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a person speaking.

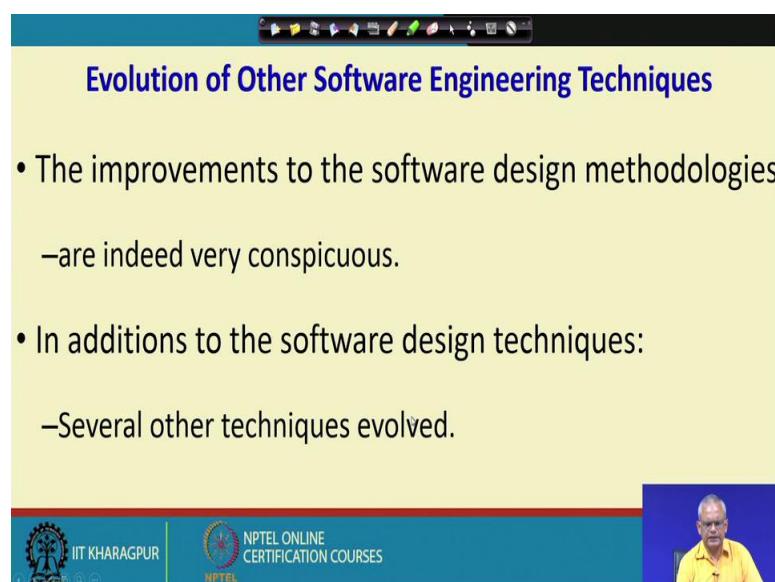
There are many advantages of the object-oriented design technique. The design is simple just by looking at the design of a large and complex problem we will be able have a fair idea about the program. It helps to reuse, lowers the development time, development cost and produces code which is less buggy and easy to maintain.

(Refer Slide Time: 26:22)



The development in this area has been rapid compared to the programs that was written in 1950s and right now we can use the sophisticated techniques to write the program.

(Refer Slide Time: 27:08)



The other software engineering principles are also getting rapidly evolved.

(Refer Slide Time: 27:22)

The slide has a yellow header bar with the title 'Evolution of Other Software Engineering Techniques' in blue text. Below the title is a list of bullet points:

- Life cycle models,
- Specification techniques,
- Project management techniques,
- Testing techniques,
- Debugging techniques,
- Quality assurance techniques,
- Metrics,
- CASE tools, etc.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a man speaking.

For example: Consider the life cycle models. Having a good specification technique will examine project management techniques, testing techniques, debug techniques, quality assurance, matrix, case stools and etc.

Compared to 1950, various software engineering techniques have evolved year after year and now, we have a reasonably sophisticated set of techniques which a developer must know to write good programs.

That's all, we will continue in the next lecture.

Thank you.

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

Lecture – 06
Life Cycle Model

Welcome, we will continue what we were discussing last time.

(Refer Slide Time: 00:25)

The slide content is as follows:

- Use of Life Cycle Models
- Software is developed through several well-defined stages:
 - Requirements analysis and specification,
 - Design,
 - Coding,
 - Testing, etc.

A yellow callout box contains the text: **Differences between the exploratory style and modern software development practices**.

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 94.

The software engineering principles themselves have evolved over the time resulting in the software development.

The software writing problem is not the same. The problems have become much more complex but at the same time these are only incrementally different from the problem that has been solved.

Compared to 1950s or 60s when the exploratory style was used, now the life cycle model have come into action and every program is being developed according to an adopted life cycle model and the development occurs through the stages.

The stages typically consist of the requirements for analysis, specification, design, coding, testing, and etc.

(Refer Slide Time: 01:58)

Differences between the exploratory style and modern software development practices

- Emphasis has shifted
 - from error correction to error prevention.
- Modern practices emphasize:
 - detection of errors as close to their point of introduction as possible.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 95

The major difference between the way program was written between 1950s and 60s to now is that the emphasis has shifted from error correction to error prevention. In the earlier technique, the exploratory style was to first write the program and complete it somehow and then start correcting the errors. As there will be hundreds of errors, keep on eliminating them one by one until all errors are eliminated.

But, now the emphasis is on error prevention. As the program is written, we track the errors and remove it from there itself and as soon as we have a mistake committed by the programmer in the program development, we try to detect it as close to the time of commitment of the error as possible. The main advantage of this is that it's very cost effective.

If you want to correct the error after testing then you will have to first find out where exactly the error has occurred, make changes to the code and again test it. Here, before testing we just identify the place where the error is written and we do not have to look or review the code, the specification & the design and then identify the error and correct it there itself. You don't need to wait for error to be discussed, detected and determined during the testing phase.

(Refer Slide Time: 04:05)

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style,
 - errors are detected only during testing,
- Now:
 - Focus is on detecting as many errors as possible in each phase of development.

The software is developed through number of phases and in each phase wherever a program commits a mistake, it gets detected and corrected in that same phase as far as possible.

(Refer Slide Time: 04:26)

Differences between the exploratory style and modern software development practices (CONT.)

- In exploratory style:
 - coding is synonymous with program development.
- Now:
 - coding is considered only a small part of program development effort.

In the early exploratory style of program development, it was basically coding, you have to start writing the code as soon as the problem is given but now in the modern development practice, coding is actually a small part of the development activities. The major activities are specification, design, coding and testing.

(Refer Slide Time: 05:01)

Differences between the exploratory style and modern software development practices (CONT.)

- A lot of effort and attention is now being paid to:
 - Requirements specification.
- Also, now there is a distinct design phase:
 - Standard design techniques are being used.

A lot of attention is now given to requirement specification then the design which has to be done, (i.e. the standard design techniques which are available).

(Refer Slide Time: 05:16)

Differences between the exploratory style and modern software development practices (CONT.)

- During all stages of development process:
 - Periodic reviews are being carried out
- Software testing has become systematic:
 - Standard testing techniques are available.

At the end of every phase reviews are conducted. The main idea behind review is to detect errors as soon as they are committed by the programmer. You don't need to wait for the error to be detected during testing. The reviews help us to detect the errors much earlier and therefore reduces the cost of development as well as the time of development.

(Refer Slide Time: 06:03)

Differences between the exploratory style and modern software development practices (CONT.)

- There is better visibility of design and code:
 - Visibility means production of good quality, consistent and standard documents.
 - In the past, very little attention was being given to producing good quality and consistent documents.
 - We will see later that increased visibility makes software project management easier.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Earlier the programmer just wrote the code and then never know when it will get completed.

The increased visibility made software project management much easier. Earlier, the projects were going hewer, one-year project taking 5 years was not uncommon but now due to the increased visibility, the project manager can very accurately determine when the project & at what stage it is and how long it will take to complete it.

(Refer Slide Time: 07:49)

Differences between the exploratory style and modern software development practices (CONT.)

- Because of good documentation:
 - fault diagnosis and maintenance are smoother now.
- Several metrics are being used:
 - help in software project management, quality assurance, etc.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Good documents are produced so that the later maintenance becomes much easier. Also, several metrics are used to determine the quality of the software, software project management and etc.

(Refer Slide Time: 08:09)

Differences between the exploratory style and modern software development practices (CONT.)

- Projects are being properly planned:
 - estimation,
 - scheduling,
 - monitoring mechanisms.
- Use of CASE tools.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

There are many project management techniques that are used like estimation, scheduling, monitoring, mechanisms and also case tools which are being used extensively.

(Refer Slide Time: 08:24)

Review Questions

- What is structured programming?
- What problems may appear if a large program is developed without using structured programming techniques?

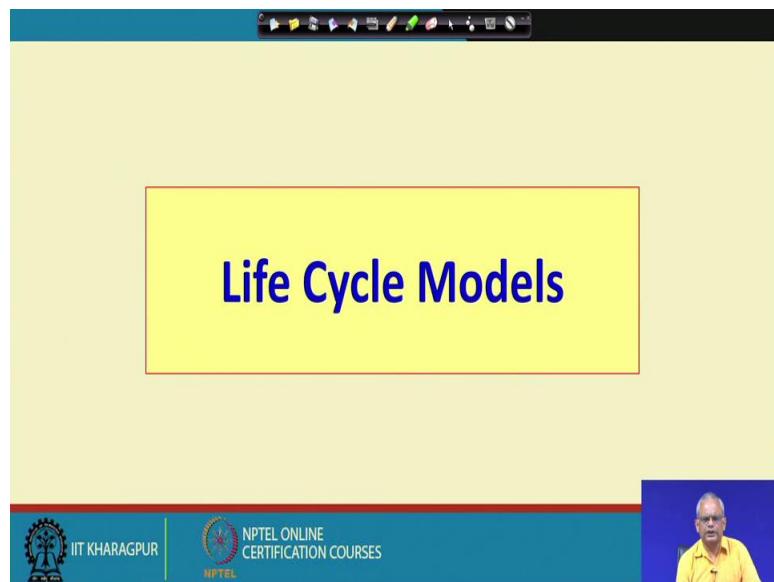
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Review questions: What do we mean by structured programming. Structured programming is writing programs that don't use Go To's, but only use sequence selection and iteration type of constructs and their modular.

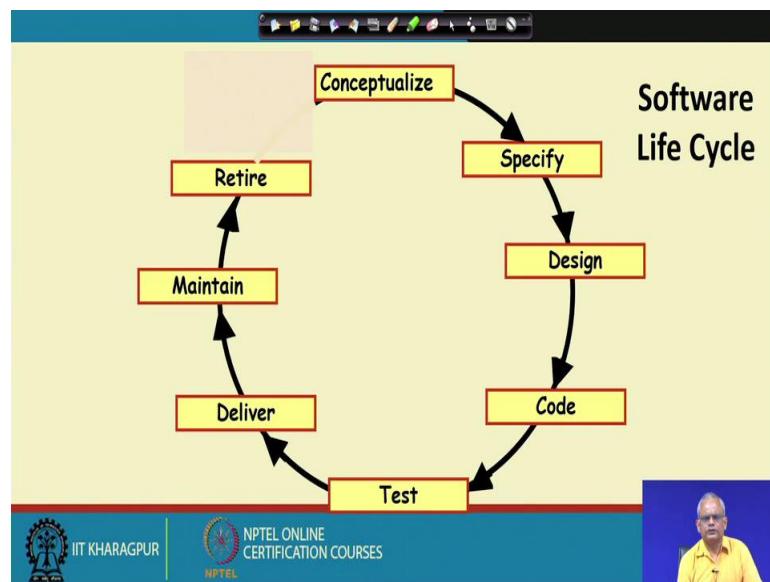
What problems may appear if a large program is developed without using structured programming techniques? The program will take longer time and it would cost much more as it will have lot of bugs & difficult to maintain as well as difficult to understand the code.

(Refer Slide Time: 09:29)



We will spend a couple of lectures on the lifecycle models because these are one of the basic principles that we must know and if a project is using certain lifecycle model, we must know why it's being used and what is involved in that lifecycle model.

(Refer Slide Time: 10:19)



A life cycle is the set of stages through which something evolves. Considering an example: A human being life cycle involves from child to adult to old age and finally retired.

Similar is with the software life cycle, conceptually you can think that somebody thought of the software – i.e. conceptualization, then it found out what exactly is needed - specification, then designing the software, test it, install it and finally delivery. The software's may not be needed anymore because different software have become available which are much better or the hardware has changed and slowly with the passage of time it is no more used and eventually retired.

(Refer Slide Time: 11:48)

Life Cycle Model

- A software life cycle model (also process model or SDLC):
 - A descriptive and diagrammatic model of software life cycle:
 - Identifies all the activities undertaken during product development,
 - Establishes a precedence ordering among the different activities,
 - Divides life cycle into phases.

What exactly is a software lifecycle model [also known as process model, the development process model or the software development life cycle (SDLC)]. Normally, we represent software lifecycle model (the process model) in the form of a diagrammatic representation but just a diagrammatic representation is not enough as it lacks details.

Thus, with every lifecycle model we will not only have a diagrammatic model but also a descriptive description of what exactly is involved. As, every software is developed through different types of activities and the lifecycle model represents all these activities.

It also establishes a precedence ordering and divides all these activities into phases.

(Refer Slide Time: 13:59)

Life Cycle Model (CONT.)

- Each life cycle phase consists of several activities.
 - For example, the design stage might consist of:
 - structured analysis
 - structured design
 - Design review

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

In other words, every phase of development consists of many activities. Considering an example, the design stage activity may be structured analysis, structure design, design review and etc.

(Refer Slide Time: 14:18)

Why Model Life Cycle?

- A graphical and written description:
 - Helps common understanding of activities among the software developers.
 - Helps to identify inconsistencies, redundancies, and omissions in the development process.
 - Helps in tailoring a process model for specific projects.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Why do we need a lifecycle model?

As now a days every development organization have their own life cycle model which is a written description - graphical and it's also available in the form of books which can be referred by existing programmers' as well as new programmers.

The main advantage of having such a graphical and written description, (i.e.; having a documented life cycle model) is that all developers will have a common understanding of activities.

Therefore, the development becomes much more disciplined and it helps to identify the inconsistencies in the activities, redundancies & omissions.

As long as we have a written lifecycle model for a specific project, we can look at the lifecycle model and tailor it for the activities which are not required and add new activities that will be required for the specific projects.

Most of the quality standard requires organization's which would qualify for the quality standard & they need to have a written lifecycle model or the process model. Quality standards like ISO, ACI are required for a documented lifecycle model.

(Refer Slide Time: 16:50)

Life Cycle Model (CONT.)

- The development team must identify a suitable life cycle model:
 - and then adhere to it.
 - Primary advantage of adhering to a life cycle model:
 - Helps development of software in a systematic and disciplined manner.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video window in the bottom right corner shows a man speaking.

Once the written life cycle model is available, the organization identifies the suitable lifecycle model for the type of the projects and the programmers & developers are asked to understand it fully and use it rigorously.

(Refer Slide Time: 17:11)

Life Cycle Model (CONT.)

- When a program is developed by a single programmer ---
 - The problem is within the grasp of an individual.
 - He has the freedom to decide his exact steps and still succeed --- called Exploratory model--- One can use it in many ways
 - Code → Test → Design
 - Code → Design → Test → Change Code →
 - Specify → code → Design → Test → etc.

Do Until Done

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** | **NPTEL**

The main advantage of the lifecycle model is that it helps to solve a big problem. As long as the problem is small it is done by one programmer or developer but when there is a team of developers working to solve a large problem then the lifecycle model has to be used because every developer needs to have a precise understanding of the problem.

The earlier informal development technique that was used was the build and fix style. This can be used only when the problem is very small and within the grasp of an individual.

The developer's have flexibility in using whatever style that suited them but when there is a team development, this kind of informal style of development would create real problem and the project will fail.

(Refer Slide Time: 19:19)

Life Cycle Model (CONT.)

- When software is being developed by a team:
 - There must be a precise understanding among team members as to when to do what,
 - Otherwise, it would lead to chaos and project failure.

In a team development there must be precise understanding of the team members that who will do what, when & etc. Without such a disciplined life cycle model, there can be chaos and the project may fail.

(Refer Slide Time: 19:46)

Life Cycle Model (CONT.)

- A software project will never succeed if:
 - one engineer starts writing code,
 - another concentrates on writing the test document first,
 - yet another engineer first defines the file structure
 - another defines the I/O for his portion first.

Considering an example where life cycle is not followed, let a programmer writes the code for some part & another do the test document design & another doing the file structure & another one writing the specification and etc.

This type of project is destined to failure because all the programmers are doing different activities and when the specification is done by one programmer, the other programmers have already written the code.

(Refer Slide Time: 20:31)

The slide has a title 'Phase Entry and Exit Criteria'. Below it is a bulleted list:

- A life cycle model:
 - defines entry and exit criteria for every phase.
 - A phase is considered to be complete:
 - only when all its exit criteria are satisfied.

At the bottom left, there are logos for IIT Kharagpur and NPTEL. At the bottom right, there is a small video frame showing a person speaking.

Every lifecycle model consists of a set of phases but before a phase starts certain conditions must be satisfied. Similarly, for a requirement phase to complete, you must know the conditions that must be satisfied for the stage to complete.

For example, the required document is completely written and has been reviewed by the team members internally and also by the customer. Without having a phase entry and exit criterion clearly mentioned in the life cycle model, it will create a lot of confusion and different teams may interpret it differently and some might just write and then proceed to the next phase and etc.

So, for the unambiguous interpretation of the life cycle model, every phase must have entry and exit criteria.

(Refer Slide Time: 22:22)

The slide has a yellow header bar with the title "Life Cycle Model (CONT.)". Below the title is a bulleted list:

- What is the phase exit criteria for the software requirements specification phase?
 - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.

A callout box highlights a point from the list:

- A phase can start:
 - Only if its phase-entry criteria have been satisfied.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and a small video window showing a speaker.

The exit criteria for the requirement specification phase is that requirements should have been gathered, analyzed, documented, internally reviewed by the team members and the customer must approve the requirements. The phase can start only after the phase entry criteria is satisfied.

(Refer Slide Time: 23:03)

The slide has a yellow header bar with the title "Life Cycle Model: Milestones". Below the title is a bulleted list:

- Milestones help software project managers:
 - Track the progress of the project.
 - Phase entry and exit are important milestones.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and a small video window showing a speaker. There is also a graphic of a highway sign indicating "18 K.M" and "HIGHWAY".

The main advantage of having the phase entry and exit criteria is that it helps us define milestones. A milestone is an important event during the development of software.

If the phase exit criteria are satisfied then an important milestone is reached. The other advantage of having milestones is that the project manager knows how far the project has progressed and it becomes easy to plan & monitor it.

(Refer Slide Time: 24:26)

The slide has a yellow background. At the top center, the title 'Life Cycle and Project Management' is displayed in blue. Below the title, there is a bulleted list:

- When a life cycle model is followed:
 - The project manager can at any time fairly accurately tell,
 - At which stage (e.g., design, code, test, etc.) the project is.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

If we do not have such milestones then the project manager is at a loss and does not know how far the project has progressed as he can just ask the team members that how long do you think it will take and etc.

(Refer Slide Time: 24:49)

The slide has a yellow background. At the top center, the title 'Project Management Without Life Cycle Model' is displayed in blue. Below the title, there is a bulleted list:

- It becomes very difficult to track the progress of the project.
 - The project manager would have to depend on the guesses of the team members.
- This usually leads to a problem:
 - known as the **99% complete syndrome**.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

As the project manager has to rely on the guess of the team members, typically the programmers tend to be optimistic and say that its almost done.

This used to be a big problem when the lifecycle model was not being followed and this is known as the 99 percent complete syndrome. In this syndrome, whenever the project manager asked to the team member, they say its almost 99 percent done because they were highly optimistic but then it just keeps on taking more and more time and the project manager becomes impatient.

Hence, the project management without a lifecycle model is very difficult and suffers from the 99 percent complete syndrome.

(Refer Slide Time: 25:58)

The slide has a yellow background. At the top, the title 'Project Deliverables: Myth and Reality' is centered. Below the title, under the heading 'Myth:', the text reads: 'The only deliverable for a successful project is the working program.' Under the heading 'Reality:', the text reads: 'Documentation of **all aspects** of software development are needed to help in operation and maintenance.' At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man speaking.

With the development precedes what are the documents that are produced? Is code, the only important active artifact and is to be given to the customer.

As all the documentation needs to be developed. Considering the specification document, design document, test document, maintenance document, user's manual and etc. All these documents have to be developed during the development process as code is just one part of the deliverables.

(Refer Slide Time: 27:08)

• Many life cycle models have been proposed.

• We confine our attention to only a few commonly used models.

–Waterfall
–V model,
–Evolutionary,
–Prototyping
–Spiral model,

Traditional models

Life Cycle Model (CONT.)

So, far we have seen some very basic concepts on the life cycle, the phases, phase entry and exit criteria, milestones and etc.

In the next lecture we will look at the important life cycle models, the traditional models like waterfall, V model, evolutionary, prototyping and spiral model and even the modern agile models.

Thank you.

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

Lecture – 07
Life Cycle Model (Contd.)

Welcome to this lecture. In the last lecture we had discussed about some basic concepts about the Life Cycle Models. Now, we look at some of the well accepted life cycle models.

(Refer Slide Time: 00:37)

The slide has a yellow header bar with a toolbar icon. Below it is a yellow main area containing text and a red bracket. A red bracket groups the first five items under the heading 'Traditional models'. The text in the yellow area is:

- Many life cycle models have been proposed.
- We confine our attention to only a few commonly used models.
 - Waterfall
 - V model,
 - Evolutionary,
 - Prototyping
 - Spiral model,
 - Agile models

Life Cycle Model (CONT.)

Traditional models

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 119.

Many life cycles models have been proposed, the popular ones are waterfall, V model, evolutionary model, prototyping model and spiral model these have existed for quite some time and we will call this as the traditional models.

In the recent years, the agile models have come into being. We will see the difference between the traditional model and the agile model and how this has come into being. Why these have become much more suitable for modern projects and etc.

(Refer Slide Time: 01:31)

• Software life cycle (or software process):
–Series of identifiable stages that a software product undergoes during its life time:
• Feasibility study
• Requirements analysis and specification,
• Design,
• Coding,
• Testing
• Maintenance.

Software Life Cycle

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A software lifecycle is also known as a software process. It's a series of identifiable stages that a software product undergoes during its lifetime.

What are stages? The stages typically are the feasibility study, requirements analysis, specification, design, coding, testing and maintenance.

(Refer Slide Time: 02:58)

Classical Waterfall Model

• Classical waterfall model divides life cycle into following phases:
–Feasibility study,
–Requirements analysis and specification,
–Design,
–Coding and unit testing,
–Integration and system testing,
–Maintenance.

```
graph TD; Conceptualize --> Specify; Specify --> Design; Design --> Code; Code --> Test; Test --> Deliver; Deliver --> Maintain; Maintain --> Retire; Retire --> Conceptualize;
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The most intuitive lifecycle model is called as the classical waterfall model. It matches very closely to the model that we discussed in the last lecture.

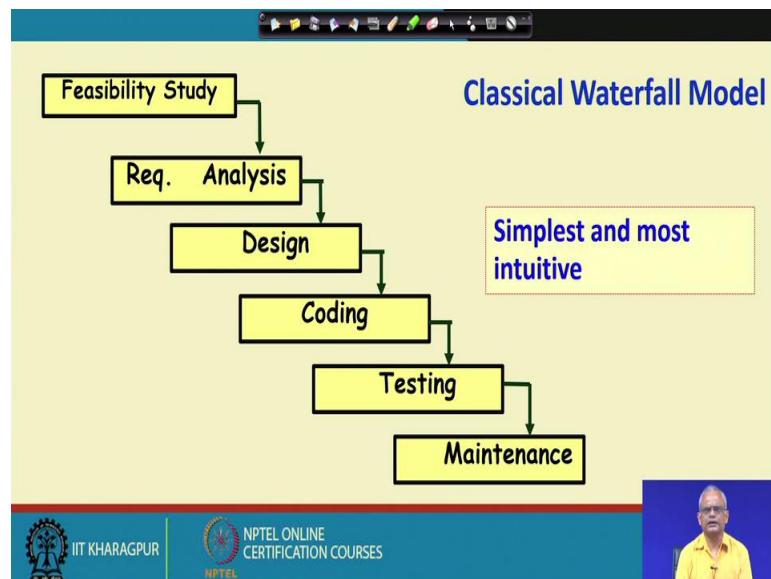
But, how exactly software's have developed.

For the development, we need to design based on the specification, code based on the design then test it and after testing is complete, install it and if there are any difficulties, problems or enhancements required then we should change and finally retire.

The classical waterfall model matches closely to this intuitive development style. The phases consists of the feasibility study, requirements analysis & specification, design, coding & unit testing, integration and system testing & maintenance.

But how the development should proceed? Initially, you do the feasibility study then the requirements analysis and specification. Based on the requirements, you do the design then code and unit test and then move to next stage that is integration and system testing and finally it's delivered. This matches very closely to the intuitive model that we had discussed.

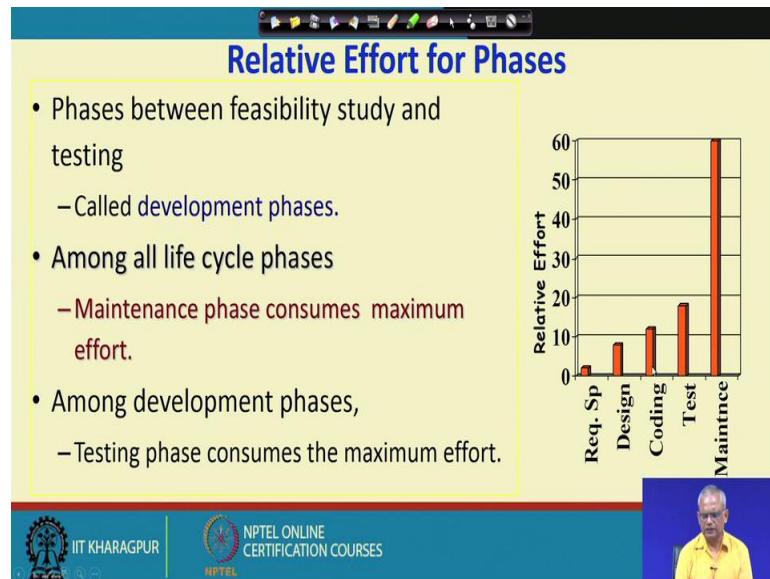
(Refer Slide Time: 05:06)



If we represent the classical waterfall model in a diagrammatical form, we will have feasibility study followed by requirements analysis followed by design, coding and once coding is complete you will carry out the testing and maintenance. It is the simplest and the most intuitive development style.

Even though this model is very easy to understand and if you understand this model, it becomes easy to understand the other models. Still, this model is more of an idealistic model as it is not really used in real projects & we will investigate the reason later.

(Refer Slide Time: 06:04)



In the classical waterfall model, there are many phases.

Among all lifecycle phases the maintenance phase actually requires the maximum effort because the development might complete within couple of months but the maintenance goes on for years together and for almost every software the maintenance requires much more effort than the development effort.

Similarly, among the development phases the testing phase typically consumes the maximum effort. If you plot the effort graph for a typical project you can see that the maintenance requires much more effort.

Thus, the development phases testing requires the largest effort. Coding & design requires less effort on the requirements specification.

(Refer Slide Time: 08:01)

- Most organizations usually define:
 - Standards on the outputs (deliverables) produced at the end of every phase
 - Entry and exit criteria for every phase.
- They also prescribe methodologies for:
 - Specification,
 - Design,
 - Testing,
 - Project management, etc.

Process Model

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The classical waterfall model or any other model used to document such a model must not only have the diagrammatic representation but also have the entry and exit criteria. What are the methodologies that are used in different phases and the outputs that is produced in every phase and is termed as the process model.

(Refer Slide Time: 08:45)

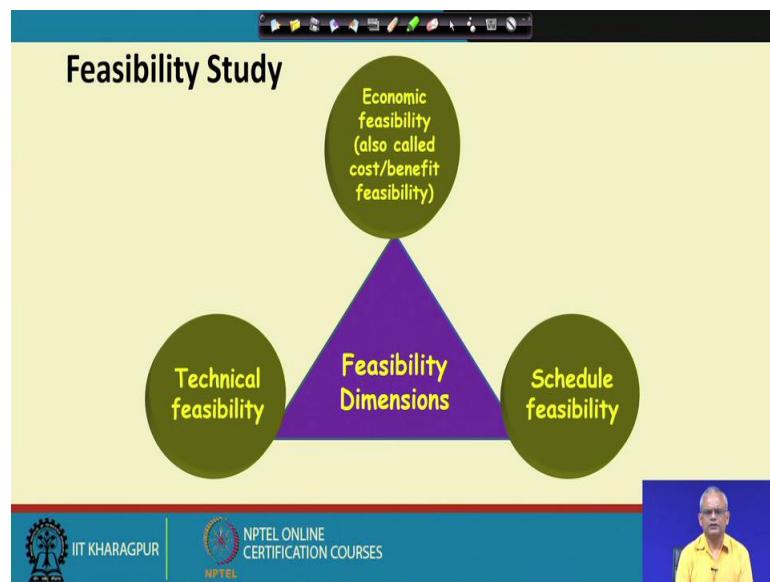
- The guidelines and methodologies of an organization:
 - Called the organization's **software development methodology**.
- Software development organizations:
 - Expect fresh engineers to master the organization's software development methodology.

Software development methodology

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Every good organization documents the process model and gives it to the developers. The new developers are asked to master the development process before they start the development of the software.

(Refer Slide Time: 09:11)



Looking into the different stages briefly. The first activity in the classical waterfall model is the feasibility study. In the feasibility study there are 3 main aspects that are determined, i.e.; whether the software that to be developed is economically feasible, whether the development effort and the cost that will be spent on developing software is it worth and lastly whether the developing organization has the technical competence required to develop the software or not.

This is also termed as the cost benefit analysis.

Considering the example for the development of some satellite communication. As the developers do not know how to use satellite communication and even write programs for satellite communication, they will find it technically infeasible.

The third feasibility that needs to be determined during the feasibility study stage is scheduled feasibility. The time by which the customer requires the product to be delivered and the development involved to complete the work as per the required time.

During the feasibility study, the project manager needs to determine 3 types of feasibility. Whether it is cost wise feasible, whether it can be done and the scheduled feasibility (i.e. whether it can be done in time or not).

(Refer Slide Time: 11:38)

The slide has a yellow background with a black header bar at the top containing standard window controls. The main content area is divided into two columns by a vertical line. The left column contains bulleted text, and the right column contains a yellow box with the title 'Feasibility Study'. Below the main content is a yellow box labeled 'First Step'. At the bottom of the slide is a blue footer bar featuring the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man in a yellow shirt speaking.

- Main aim of feasibility study: determine whether developing the software is:
 - Financially worthwhile
 - Technically feasible.
- Roughly understand what customer wants:
 - Data which would be input to the system,
 - Processing needed on these data,
 - Output data to be produced by the system,
 - Various constraints on the behavior of the system.

Let's look how exactly the feasibility study can be carried out. The first thing is to roughly understand the requirements of the software and the customer requirements. What are the features of the software? (I.e. Comprising of different sort of data, the volume, the processing that needs to be done & finally the code to be written.)

What is the output to be produced by the system and the various constraints & behavior of the system? Considering an example, whether the backup needs to be kept or whether the downtime requirement is very high or whether the data to be input from different locations of the company. These are some of the different constraints.

(Refer Slide Time: 13:20)

The slide has a yellow header bar with the title 'Case Study'. Below it is a white content area containing a bulleted list of requirements for the SPF Scheme for CFL. The list includes:

- SPF Scheme for CFL
- CFL has a large number of employees, exceeding 50,000.
- Majority of these are casual labourers
- Mining being a risky profession:
 - Casualties are high
- Though there is a PF:
 - But settlement time is high
- There is a need of SPF:
 - For faster disbursement of benefits

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a yellow shirt speaking.

Once the requirements are understood then the project manager likes to execute it. As per the proposed alternate solution you need to find out the cost for its solution, do the cost benefit analysis, check whether schedule wise it is technically feasible and then find out the best solution and discuss with the customer and determine whether to execute or abandon the project.

Looking at a case study that will help us appreciate what are the work or the activities that needs to be undertaken during the feasibility study phase. Let's take a project of the special provident scheme for the coalfields limited, which itself is a large company having large number of employees exceeding 50,000. Now, the company has many casual laborers and mining is a risky profession, casualties are high, people get injured, killed even though there is a provident fund but the settlement time is high. Therefore, the company feels that it needs a special provident fund which can fasten the disbursement of the benefits at the earliest.

The company invites software vendors to develop software for the special provident fund scheme. Many companies show interest and the project manager visits the main office of the company & try to understand what are the functionalities that is required.

(Refer Slide Time: 15:30)

The slide is titled "Feasibility: Case Study". It contains a bulleted list of steps:

- Manager visits main office, finds out the main functionalities required
- Visits mine site, finds out the data to be input
- Suggests alternate solutions
- Determines the best solution
- Presents to the CFL Officials
- Go/No-Go Decision

On the right side of the slide, there is a hand-drawn style diagram of a network. It shows a central computer icon connected to three circular nodes, each representing a mine site. A red double-headed arrow connects the central computer to one of the mine sites. To the right of the diagram, there is a small video window showing a man in a yellow shirt speaking.

At the bottom left, there is a logo for IIT Kharagpur. At the bottom center, there is a logo for NPTEL ONLINE CERTIFICATION COURSES.

There is a formula for how much compensation is to be paid. What are the restrictions on the compensation under which conditions compensation will not be paid and etc.

The manager might visit some mine sites and finds out how the data will be input, that is what are the details of the employees, what is their contribution on a daily weekly or monthly basis and after understanding the complete situation he tries to propose an alternate solution.

The alternate solutions proposes that the data needs to be maintained locally at the mine sites and periodically the data will be transferred to the main office (I.e. there will be computers at the mine site which will maintain the data in each mine and there will be a main computer & main server at the head office and depending on the request it will provide the answers).

The other solution might be of transmitting the data as soon as it gets entered or at the end of the day to the mine site. All these data will be maintained at the head office. Instant results can be provided as here the communication costs are less but there is a delay and data are not up to date at the main office.

To do a cost benefit analysis for each alternate solution, initially find out what is suitable to the coalfield officials and then find the cost for its solution and determine the best solution & check whether that is acceptable to the organization & whether there is

technical competence to develop the solution or whether it can be developed within the timeline and based on this the decision will be taken (no go decision).

Just to summarize, the first thing during feasibility study is to understand the work and the features required. What are the data and how they will be input? What is the processing time and the constraints and etc. Once the rough understanding of the requirements is done, find out what are the best ways to solve it as there can be many alternate solutions.

Examine the cost implications of different solutions & find out the best solution, present it to the customer & take their feedback.

If everything goes fine and is acceptable to the officials, the project manager will take a go decision and the high-level solution that has been defined will be used by the developers and the development will proceed in this style.

But there might be a situation where the technical competence is not there with the company or maybe the timeline by which the company needs it would be difficult to meet and hence the project manager might take a no-go decision and abandon the project.

(Refer Slide Time: 21:12)

Activities During Feasibility Study

- Work out an overall understanding of the problem.
- Formulate different solution strategies.
- Examine alternate solution strategies in terms of:
 - resources required,
 - cost of development, and
 - development time.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



The activities during the feasibility study, we can summarize that the work out on an overall understanding of the problem. Formulate the different solution strategies, propose

alternate solutions and determine the cost in terms of the resource required per cost of development and the development time.

(Refer Slide Time: 21:38)

The slide has a yellow background. At the top, there is a navigation bar with icons. Below it, a bulleted list starts with '• Perform a cost/benefit analysis:'. Under this, there are two bullet points: '-Determine which solution is the best.' and '-May also find that none of the solutions is feasible due to:'. To the right of this text is a yellow rectangular box containing the text 'Activities during Feasibility Study' in blue. Below the list is a horizontal bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', and the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the slide, there is a small video window showing a man in a yellow shirt speaking.

- Perform a cost/benefit analysis:
 - Determine which solution is the best.
 - May also find that none of the solutions is feasible due to:

Activities during Feasibility Study

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Form a cost benefit analysis, find out the best solution and then decide whether to go ahead with the project or abandon because of high cost resources, constraints or technical reasons.

(Refer Slide Time: 21:57)

The slide has a yellow background. At the top, there is a navigation bar with icons. Below it, the title 'Cost benefit analysis (CBA)' is displayed in a large, bold, black font. A bulleted list follows, starting with '• Need to identify all costs --- these could be:'. This is followed by three sub-points in blue: '- Development costs', '- Set-up', and '- Operational costs'. The list continues with '• Identify the value of benefits' and '• Check benefits are greater than costs'. Below the list is a horizontal bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', and the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the slide, there is a small video window showing a man in a yellow shirt speaking.

Cost benefit analysis (CBA)

- Need to identify all costs --- these could be:
 - Development costs
 - Set-up
 - Operational costs
- Identify the value of benefits
- Check benefits are greater than costs

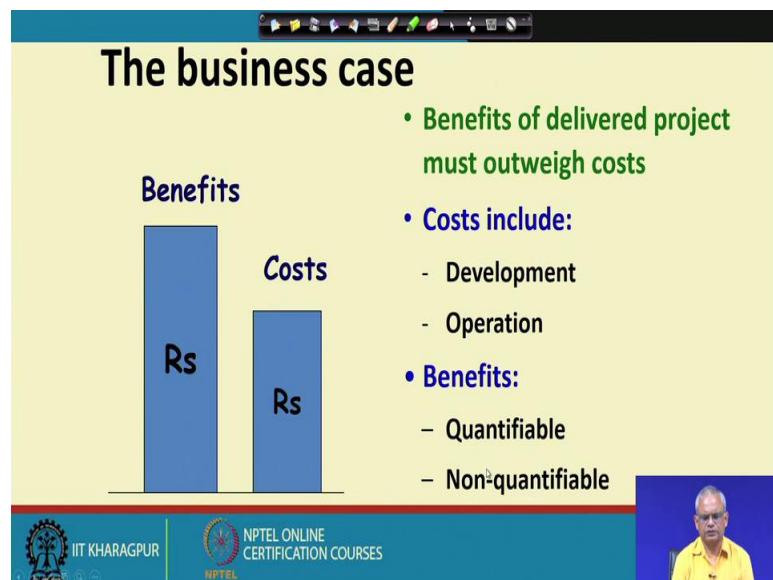
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The cost of a project is not only the development cost but also consists of the operational costs.

Consider an educational institute wants to have an automated solution. It is not just developing the software but also setting up the required infrastructure. Also, the operational cost like who will enter the data, we will take the backup, maintenance and will be taken into account.

An organization trying to deploy an IT solution would identify all these costs and the cost of the solution would be the sum of the development cost setup and operational cost. Finally, you have to identify the benefits that you will accrue from this automation and then check whether the benefits are greater than the cost.

(Refer Slide Time: 23:08)



For a project to be taken up by an organization, the benefits must outweigh the costs. The costs are the development operation setup costs & the benefits. Some of them are quantifiable and some of them may be non-quantifiable. For example, let's say we want to automate the book-keeping activities in an educational institute.

Some of the benefits are quantifiable that how much manpower is saved by automating it but some are non-quantifiable. For example, the results may come out faster, the students may benefit. But we cannot really put a cost on through this & it's difficult to quantify.

(Refer Slide Time: 24:18)

The business case

- Feasibility studies should help write a ‘business case’
- Should provide a justification for starting the project
- Should show that the benefits of the project will exceed:
 - Various costs
- Needs to take account of business risks

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

31

After the feasibility study is over, typically the project manager prepares a ‘business case’. The business case basically mentions the costs and the benefits & then presents it to the top management about the cost implications and the benefits.

(Refer Slide Time: 25:03)

Writing an Effective Business Case

1. Executive summary
2. Project background:
 - The focus must be on what, exactly, the project is undertaking, and should not be confused with what might be a bigger picture.
3. Business opportunity
 - What difference will it make?
 - What if we don't do it?
4. Costs
 - Should include the cost of development, implementation, training, change management, and operations.
5. Benefits
 - Benefits usually presented in terms of revenue generation and cost reductions.
6. Risks
 - Identify risks.
 - Explain how these will be managed.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

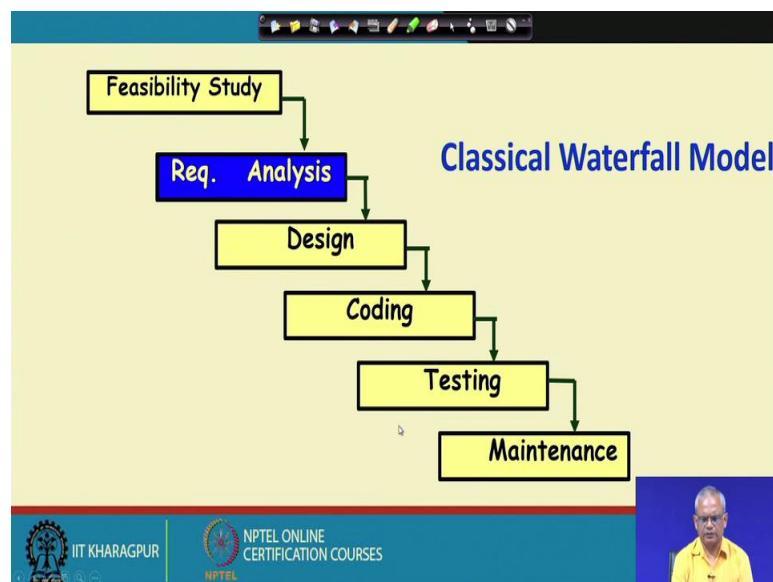


Typically, a good business case should touch upon the following points out of which one is the executive summary that what exactly is done is to be taken up to the top management to understand.

The project background that under what circumstances the need for the project arose and then the business opportunity that what exactly will be the benefit? What difference it will make and how much it will cost for the development, implementation, set of trainings, operations and the benefits in terms of revenue generation, cost reduction, non-quantifiable benefits and even the risks.

The risks can be that the development costs will be much higher than estimated. Maybe, it will take more time than what is estimated. After it is developed, it may not be liked by the stakeholders and they may not use it. But there must be some arguments or plan regarding how these different threats or risks will be contained or handled and these are the different sections for the effective business cases.

(Refer Slide Time: 28:03)



We have so far looked at the feasibility study phase, as we look at the requirement analysis that the design, coding, testing and maintenance. But here it looks like a cascade of waterfall, the water is falling from one level to the next level and that is known as the waterfall model.

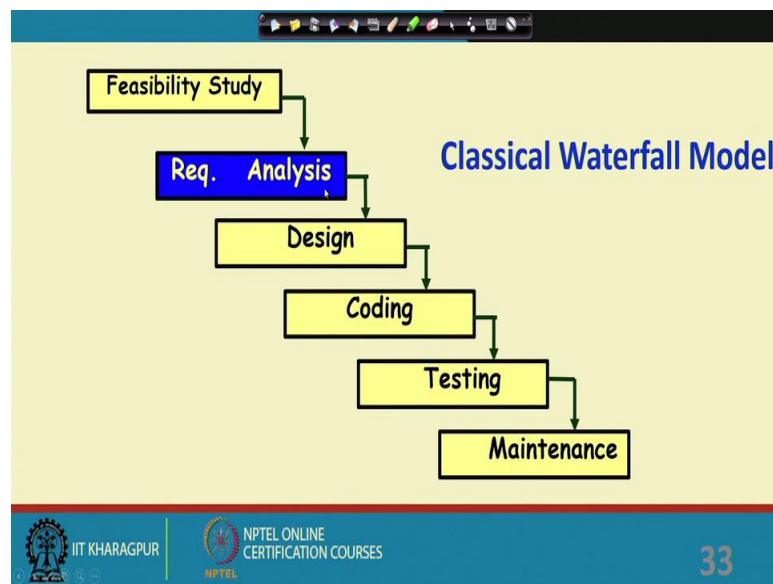
The first stage is the feasibility study and the second stage is the requirements analysis and specification. We will start discussing very briefly about the activities done in requirement analysis as we will continue the discussion in the next lecture. Thank you.

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

Lecture – 08
Waterfall Model

Welcome to this lecture we will continue from where we ended last time and we will look at the second phase of the classical Waterfall Model which is requirement analysis.

(Refer Slide Time: 00:38)



Looking at requirement analysis and specification.

(Refer Slide Time: 00:41)

Requirements Analysis and Specification

- Aim of this phase:
 - Understand the exact requirements of the customer,
 - Document them properly.
- Consists of two distinct activities:
 - Requirements gathering and analysis
 - Requirements specification.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

NPTEL

The main aim of this page is to understand the exact requirements of the customer and then analyze the requirements. If there are any difficulties in the requirements, eliminate all the problems and then document it properly.

Every developer will have to do this sometime or the other, it's an important skill because it will determine whether the project will succeed or fail. So, this is a very important skill to understand how to gather the requirements, analyze the requirements and then document this properly. We will look at some standard ways of documenting. There are basically two main activities: one is the requirement analysis and the other is requirement specification.

The first activity is requirements gathering and analysis and the second activity is requirement specification.

Let us look into the requirements gathering and analysis and what is involved in requirement specification.

(Refer Slide Time: 02:39)

Requirements Analysis and Specification

- Gather requirements data from the customer:
 - Analyze the collected data to understand what customer wants
- Remove requirements problems:
 - Inconsistencies
 - Anomalies
 - Incompleteness
- Organize into a Software Requirements Specification (SRS) document.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES



In requirements gathering, you need to gather the requirements, collect the data & understand the customer requirements and during the analysis we analyze the collected requirement, identify what sort of problems can be there in a requirement? Basically, there are 3 main types of problems that normally exist in a gathered requirement first is called as inconsistency, second-anomalies and third-incompleteness.

Inconsistencies means that one part of the requirement contradicts with some other part of the requirement. Anomaly is ambiguity, some requirements are anomalous or not clear. In incompleteness somehow the requirement has been missed. So, these 3 problems need to be eliminated and then documented in the form of a requirement specification document known as the SRS document or the software requirement specification document.

(Refer Slide Time: 04:10)

The slide has a yellow header with the title 'Requirements Gathering'. Below the title is a bulleted list:

- Gathering relevant data:
 - Usually collected from the end-users through interviews and discussions.
 - Example: for a business accounting software:
 - Interview all the accountants of the organization to find out their requirements.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a yellow shirt.

During the requirements gathering, the relevant data is gathered from the end users. It's normally done through interviews and discussions. For a business accounting software you might have to interview, meet all the accountants in the organization who are doing in the manual mode, find out what exactly what they are doing and the requirement from the software.

(Refer Slide Time: 04:45)

The slide has a yellow header with the title 'Requirements Analysis' and a small '(Cont...)' next to it. Below the title is a bulleted list:

- The data you initially collect from the users:
 - Usually contain several contradictions and ambiguities.
 - Why?
 - Each user typically has only a partial and incomplete view of the system.**

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a yellow shirt.

Once the data is gathered then there will be several contradictions and ambiguities. The main reason for the contradictions and ambiguities is that you are gathering requirement

from a set of end user and each user has a different view of the software. Sometimes the views may contradict each other and also they might miss out some of their requirements or they may give a ambiguous statement. Each user has partial and incomplete view of the system. Thus, this is the main reason why there are difficulties in the gathered requirements.

(Refer Slide Time: 05:51)

Requirements Analysis (Cont...)

- Ambiguities and contradictions:
 - must be identified
 - resolved by discussions with the customers.
- Next, requirements are organized:
 - into a Software Requirements Specification (SRS) document.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

All these problems in the requirement have to be identified but how will this be resolved? One of the ways is by discussing with the customers that how exactly that problem will be addressed and once all the difficulties are eliminated from the gathered requirements, these are documented in the SRS document.

After few lectures we will look at the exact techniques by which requirement gathering analysis is done and then the format it in which the SRS document will be written.

Right after the requirements analysis phase is the design phase, the design phase is based on the requirements. Let us briefly look at what exactly is involved in the design activity.

(Refer Slide Time: 07:06)

Design

- During design phase requirements specification is transformed into :
 - A form suitable for implementation in some programming language.
- Two commonly used design approaches:
 - Traditional approach,
 - Object oriented approach

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

The design activity typically uses the requirement specification document. The designers consult the SRS document and then come up with a design which will be easily implemented in some programming language. The two major approaches which are used are 1) traditional approach and 2) object-oriented approach.

(Refer Slide Time: 07:35)

Traditional Design Approach

- Consists of two activities:
 - Structured analysis (typically carried out by using DFD)
 - Structured design

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

In the traditional approach, there are two main activities, one is to develop the DFD representation known as the structured analysis and once the DFD representation or the dataflow diagram representation is complete its translated into a structured design.

(Refer Slide Time: 08:07)

Structured Design

- High-level design:
 - decompose the system into **modules**,
 - represent invocation relationships among the modules.
- Detailed design:
 - different modules designed in greater detail:
 - data structures and algorithms for each module are designed

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video player interface is visible on the right side of the slide.

At the end of the structure design, we will have a high-level design in terms of the module structure. Once the module structure is ready then the detailed design is done where the data structure for each module and also the functions algorithms are designed.

(Refer Slide Time: 08:47)

- First identify various objects (real world entities) occurring in the problem:
 - Identify the relationships among the objects.
 - For example, the objects in a pay-roll software may be:
 - employees,
 - managers,
 - pay-roll register,
 - Departments, etc.

Object-Oriented Design

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video player interface is visible on the right side of the slide.

On the other hand, in the object-oriented design we identify the relations among objects and then based on that we develop the design. For example, the objects in a payroll software may be the employees, managers, payroll register, departments etc.

(Refer Slide Time: 09:22)

The slide has a yellow background. At the top center, it says "Object Oriented Design (CONT.)". Below that is a bulleted list:

- Object structure:
 - Refined to obtain the detailed design.
- OOD has several advantages:
 - Lower development effort,
 - Lower development time,
 - Better maintainability.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a photo of a man in a yellow shirt.

The object structures are refined into the detailed design.

The object-oriented design technique has become very popular as it offers several advantages comprising of lower development effort, lower development time and better maintainability.

Once the design phase is complete, coding is taken into account. Based on the design document the coding & testing activities are done.

(Refer Slide Time: 10:15)

The slide has a yellow background. At the top center, it says "Coding and Unit Testing". Below that is a bulleted list:

- During this phase:
 - Each module of the design is coded,
 - Each module is unit tested
 - That is, tested independently as a stand alone unit, and debugged.
 - Each module is documented.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a photo of a man in a yellow shirt.

During the coding phase, each module of the design is coded typically that its developer is given couple of modules to code and once this passes the unit testing the modules are documented and the phase completes.

The next stage is testing where integration and system testing is done.

(Refer Slide Time: 11:03)

Integration and System Testing

- Different modules are integrated in a planned manner:
 - Modules are usually integrated through a number of steps.
- During each integration step,
 - the partially integrated system is tested.

M1 M2 M5 M7
M3 M4 M6 M8

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video player window shows a man speaking.

During integration testing the different modules are integrated in a planned manner that is there are number of steps through which the modules are integrated. To integrate these modules the main idea is to check whether they are interacting properly or not.

The interface bugs are the main focus in the integration testing. During unit testing, the bugs in the different modules are identified. So, the system is integrated over a number of steps and each time it is tested to check if there are any bugs and the typical focus of the integration testing is to identify the interface bugs.

(Refer Slide Time: 12:29)

System Testing

- After all the modules have been successfully integrated and tested:
 - System testing is carried out.
- Goal of system testing:
 - Ensure that the developed system functions according to its requirements as specified in the SRS document.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Once all the modules have been integrated and the system testing is carried out, the fully integrated system is checked to see if it meets the requirements that are experienced by the customer.

The goal of the system testing is to check if the developed software satisfies all the requirements that have been expressed in the SRS document and once the system testing is done, the software is delivered to the customer and then the maintenance phase begins. (I.e. if there are any bugs reported by the customer then it may get fixed enhancements and etc) as the maintenance phase continues for quite long time.

(Refer Slide Time: 13:38)

The slide has a yellow background. At the top center, the word "Maintenance" is written in blue. Below it, there is a bulleted list in black text:

- Maintenance of any software:
 - Requires much more effort than the effort to develop the product itself.
 - Development effort to maintenance effort is typically 40:60.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video window showing a man in a yellow shirt speaking.

Earlier we said that the maintenance takes the maximum effort much more than the effort required to develop the product itself. As maintenance occurs over a long time, the lifetime of the software is large. The development time is only a small fraction of the lifetime and typically the effort for maintenance is much more and the development effort to maintenance effort is typically 40 to 60 (i.e. 40 percent development effort and 60 percent maintenance effort).

(Refer Slide Time: 14:20)

The slide has a yellow background. At the top center, the words "Types of Maintenance?" are written in blue. Below it, there is a bulleted list in black text:

- **Corrective maintenance:**
 - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance:**
 - Improve implementation of the system
 - enhance functionalities of the system.
- **Adaptive maintenance:**
 - Port software to a new environment,
 - e.g. to a new computer or to a new operating system.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video window showing a man in a yellow shirt speaking.

Why maintenance is needed? Well there are 3 main reasons; One is that the bugs may be reported and need to correct those that is called as corrective maintenance.

It may be necessary to enhance the functionalities that is hard new functionalities which were not visualized earlier or improve the implementation in some way maybe the response time is not satisfactory & try to improve the response time and etc that is called as perfective maintenance and the third type of maintenance is known as the adaptive maintenance.

In adaptive maintenance the software may be required to work with a new hardware. May be that the company procured a new server and it has to be installed and needs to work on this or maybe there is new computer and it needs to work with a new operating system and thus the program needs to be changed a little bit. Hence, this kind of maintenance is called as adaptive maintenance.

(Refer Slide Time: 16:34)

The slide is titled "Iterative Waterfall Model". It contains the following text:

- Classical waterfall model is idealistic:
 - Assumes that no defect is introduced during any development activity.
 - In practice:
 - Defects do get introduced in almost every phase of the life cycle.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a video player showing a man in a yellow shirt speaking.

The classical waterfall model is very simple as it matches with the intuition but it cannot be used or it's very difficult to use in a real-life project. The main difficulty that will be observed when we try to use the classical model in a real project is that it has a waterfall where it can only go from one stage to the other stage.

Once a stage is complete no further activity on that stage will occur as it will transit to the next stage. But what if there was a design error which was discovered during testing

(Let's assume). As every programmer makes mistakes and maybe a mistake was discussed much later. Hence, the classical waterfall model is idealistic, assuming that no mistake is done or no mistake has escaped a phase. In practice there are many mistakes which are committed and escaped in that phase.

(Refer Slide Time: 18:01)

The slide is titled "Iterative Waterfall Model (CONT.)". It contains the following text:

- Defects usually get detected much later in the life cycle:
 - For example, a design defect might go unnoticed till the coding or testing phase.
 - The later the phase in which the defect gets detected, the more expensive is its removal --- why?**

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES".

As soon as there is a problem reported, you have to repeat some activities in the phase in which it was done. Correct those and also the subsequent document.

For example, if a requirement is faulty as there is a mistake in doing the requirements and consider we discovered the mistake in the requirement during the design phase then we need to rework the requirements and that will have some cost.

But consider we had a mistake in the requirements in the design phase and we did not discover nor in the coding phase but discovered the requirements problem in the testing phase, then it will be much more expensive.

Now a question arises why the latter the phase the defect gets detected, the more expensive is the removal. The answer is that we need to rework the results of many phases, if we detect the requirement problem during testing then we not only have to take up the requirements document and change all the requirements that are in problem but also need to redesign, change the code and again do the testing. Obviously, that is much

more than if we had discovered the problem during the design stage, we might have to just change the requirements and some part of the design.

(Refer Slide Time: 20:34)

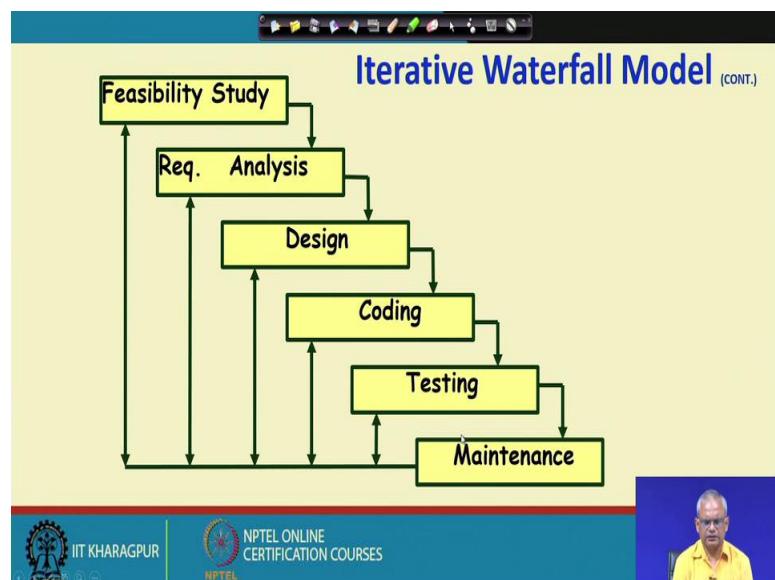
Iterative Waterfall Model (CONT.)

- Once a defect is detected:
 - The phase in which it occurred needs to be reworked.
 - Redo some of the work done during that and all subsequent phases.
- Therefore need feedback paths in the classical waterfall model.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Thus, once a defect is detected in a latter phase, we need feedback paths in the classical waterfall model.

(Refer Slide Time: 20:51)



A defect of one phase may get detected in any of the later phase and as it is detected, we need to rework the results produced at that phase and thus we need the feedback paths.

We should be able to revisit that phase that is basically the representation of the iterative waterfall model where we have added feedback paths to the classical waterfall model.

(Refer Slide Time: 21:28)

Phase Containment of Errors (Cont...)

- **Errors should be detected:**
 - In the same phase in which they are introduced.
- For example:
 - If a design problem is detected in the design phase itself,
 - The problem can be taken care of much more easily
 - Than say if it is identified at the end of the integration and system testing phase.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



The later the defect is detected the more expensive it will become. That gives us an idea is that if we detect the defect in the same phase itself then it will cost the least. For example, if the design problem is detected in the design phase, then we can just correct it there itself. But if we detect this during coding or unit testing, we not only have to change the design but also have to change the code.

As the programmers commit mistakes that is universal but then the main idea here is that we must be able to detect those mistakes as quickly as possible and within the same phase for the cost of the correction to be low. This is called as the phase containment of errors, the errors must be detected within the same phase in which they are introduced and the major reason behind the phased containment of errors is that it will cost the least and this one of the most important principle in the phase of containment of errors.

(Refer Slide Time: 23:09)

The slide has a blue header bar with standard window controls. The main title 'Phase Containment of Errors' is centered in a blue box. Below the title is a bulleted list of points. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video camera icon.

- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible:
 - is known as **phase containment of errors**.
- Iterative waterfall model is by far the most widely used model.
 - Almost every other model is derived from the waterfall model.

The iterative waterfall model is used very heavily and it is one of the most fundamental models in the sense that all other models are derived from this model.

(Refer Slide Time: 23:43)

The slide has a blue header bar with standard window controls. The main title 'Waterfall Strengths' is centered in a blue box. Below the title is a bulleted list of points. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video camera icon.

- Easy to understand, easy to use
- Provides a reference to inexperienced staff
- Milestones are well understood by the team
- Provides requirements stability
- Facilitates strong management control (plan, staff, track)

The waterfall models are easy to understand and easy to use. Even the inexperienced staff who never had any project experience if they work in a project using the waterfall model they can easily relate to the model and feel comfortable. As far as the project manager is concerned the milestones here are well understood. The requirements have been collected in the beginning and based on that the development starts.

Thus, the requirements and management control do not change during the development in the sense that based on the requirements it can plan how long will the design & coding take and even have a track on the project.

(Refer Slide Time: 24:49)

Waterfall Deficiencies

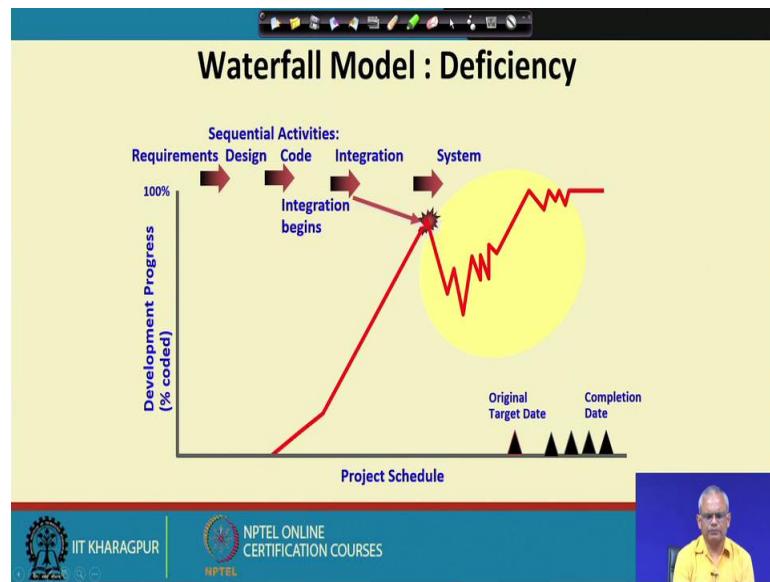
- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – **inhibits flexibility**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But what are the deficiencies of the waterfall model? One of the major deficiencies of the waterfall model is that it requires all the requirements to be gathered and documented and there is no scope to change this later. But it is often the case that all requirements cannot be given, someone might miss out on some requirements, requirements may change and that cannot be accommodated in a pure waterfall model of development.

It lacks flexibility because requirement changes & can't be accommodated. It can give a false impression of progress because we find that requirements complete, design complete, coding complete but during integration problems may occur.

(Refer Slide Time: 25:54)



Typically, the scenario is such that the project is on time, everything is under control, the project manager is happy, the requirements, design everything is progressing and then the integration begins. From here, the project starts getting delayed. Till now it was under the grasp of one developer who has done the unit testing but then different developers they have made different assumptions and they do not integrate well.

The problem starts during integration testing and the project starts getting delayed. The project manager announces the new delivery and keeps on changing it. This is a typical problem of the waterfall model (i.e.; late integration).

(Refer Slide Time: 27:46)

The slide has a yellow header with the title 'Waterfall Deficiencies'. Below the title is a bulleted list of six items. At the bottom of the slide is a footer bar containing the IIT Kharagpur logo, the NPTEL logo, and a video camera icon.

Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – **inhibits flexibility**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

The waterfall model often observed that it gives a false impression of progress.

One of the main reasons is that integration is only at the end and many a times, the problem starts once the integration activity start. Also, during the time of development, the customer is entirely under dark. The customer has no clue regarding what has happened to the software, when it will be delivered, what will it be like and etc.

(Refer Slide Time: 29:06)

The slide has a yellow header with the title 'When to use the Waterfall Model?'. Below the title is a bulleted list of three items. At the bottom of the slide is a footer bar containing the IIT Kharagpur logo, the NPTEL logo, and a video camera icon.

When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Experienced Development team

The waterfall model is used for projects which are well known and the development team are very familiar with these kinds of projects. The development by the developers are experienced in developing these kinds of projects.

Consider an accounting package, let's say a company develops the accounting solution for different customers. The developers are experienced on accounting software, the technology, requirements and are almost stable. In this case a waterfall model will be a suitable model.

We will stop at this point and we will look at the other models, life cycle models and the type of projects for which they are suitable & how they differ from the waterfall model in the next lecture.

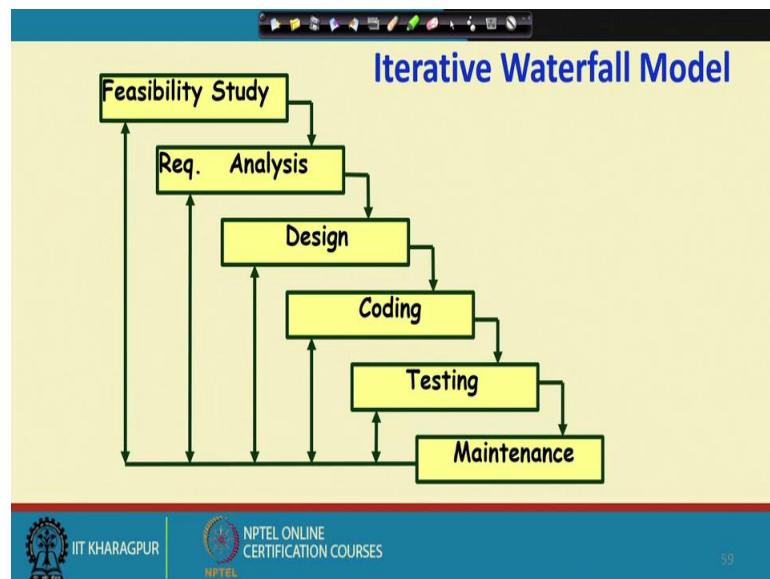
Thank you.

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

Lecture - 09
Waterfall Derivatives

Welcome to this lecture. In the last few lectures we looked at the life cycle models. We even looked at the classical waterfall model which is the basis for all other models but the classical waterfall model is hard to use in a project. The main problem is that its idealistic and it has no way to accommodate corrections to the work products. It is just a pure waterfall model. But 1000s of mistakes occur during the development and the iterative waterfall model overcomes this issue with the classical model and provides feedback paths.

(Refer Slide Time: 01:28)



The iterative waterfall model provides feedback paths and if a mistake is detected in any phase then there is a way to correct those mistakes and also redo the subsequent phases. The iterative waterfall model was hugely popular in 1970s and 80s. It was used in almost every development work but it had some difficulties for which the newer life cycle models came into account. Slowly the characteristics of the projects changed and the projects became sought.

Let's look into the strong points of the waterfall model and why it was so popular?

(Refer Slide Time: 02:56)

Waterfall Strengths

- Easy to understand, easy to use, especially by inexperienced staff
- Milestones are well understood by the team
- Provides requirements stability during development
- Facilitates strong management control (plan, staff, track)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video feed of a man speaking is visible on the right side of the slide.

The strengths of the waterfall model include conceptual ease to understand and use for the consumers. It matches with the conceptual understanding of how software is developed. Even if the development staffs are inexperienced, they can easily understand the lifecycle model and start developing. In the waterfall model, the milestones are well understood by every member of the team. The milestones are basically the phase entry and exit & it also provides requirement stability during development. After the requirements are gathered and documented there is no change in the requirements.

The developers are not interrupted and they start developing the software. But if there is a change, they would have to redo their entire work, do the entire plan & overall design and etc.

Thus, the waterfall model provides requirement stability. Once the requirement document is prepared it can't be changed. Also, the project manager can plan all the phases & track whether the project is proceeding as per the plan and if not then take corrective action to put it back on track. The waterfall model has lot of strengths but then there are several deficiencies of this model.

(Refer Slide Time: 05:35)

Waterfall Deficiencies

- All requirements must be known upfront – in most projects requirement change occurs after project start
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Looking at the deficiencies of the waterfall model. Possibly the most problematic deficiency is that the requirements must be known upfront. The customer has to give all the requirements before the project starts and that is usually not feasible because the software is not there and the client has to imagine what is required. As it is very easy to miss requirements and to give ambiguous requirements & wrong requirements.

In real projects there is a large number of requirement changes. The customer in the beginning of the project cannot visualize the exact requirements and as the project progresses the customer might require changes as per the requirements.

The second problem with the waterfall model is that it gives a false impression of progress. Basically, the problem starts with the integration testing and once the integration starts then the problems appear.

This is one of the major problem areas in waterfall model and the schedule delays starts from the integration and the project gets delayed further. One of the other major problem in the waterfall model is that it rarely meets the customer requirement and the reason is that the customer is kept out of the development work.

(Refer Slide Time: 09:01)

When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Development team have experience with similar projects

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The projects for which the waterfall model is suitable. If the requirements are well known and stable then it is well understood that what is required of the software and can be frozen upfront.

The technology is well understood and the development team is familiar with the software to be developed. If you look at the characteristics these are basically some software which exists and we just want to have a small change version. But if we want to develop customized software then the waterfall model will not be suitable.

(Refer Slide Time: 10:29)

Classical Waterfall Model (cont.)

- Irrespective of the life cycle model actually followed:
 - The documents should reflect a classical waterfall model of development.
 - Facilitates comprehension of the documents.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

What about the classical waterfall model? As it is not useful for the real projects does it have any use at all. The documentation of a project is done as per the classical waterfall model. The development might have occurred using the iterative model or any other model but the documents are written as if the classical waterfall model was used.

Let's look into the reason.

(Refer Slide Time: 11:06)

Classical Waterfall Model (CONT.)

- Metaphor of mathematical theorem proving:
 - A mathematician presents a proof as a single chain of deductions,
 - Even though the proof might have come from a convoluted set of partial attempts, blind alleys and backtracks.

The slide footer includes logos for IIT Kharagpur and NPTEL, and a small video frame showing a person speaking.

Thinking of the way the mathematician proves theorem. Given a problem as he wants to prove, he would work in many directions, he will backtrack, he will change in between cross it and out & start again, etc.

Finally, when he writes the theorem it appears as a single chain of thought, all the mistakes that had been made & the different alternatives tried are not shown because it will help anyone to understand it completely. If the mistakes were included there than it would appear extremely confusing for somebody and that is precisely the reason why the documentation in software projects were written as if classical waterfall model is used because that way the documents can be easily understood by somebody trying to understand the documents.

(Refer Slide Time: 12:40)

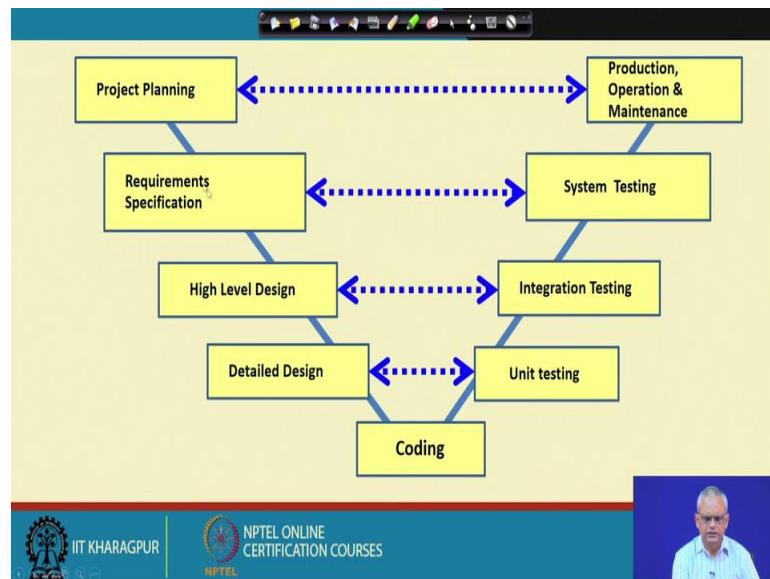
V Model

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

Looking at a derivative of the waterfall model named as V model. It is a variant of the waterfall model and it emphasizes verification and validation. It is useful for software that are used for safety applications where reliability & safety are important. The verification and validation activities are spread throughout the entire lifecycle.

As you are starting with the requirements and design, all phases are the verification and validation activities and the testing activities are planned in parallel with development as different aspects of the project progressed, more accurate test case designs are made.

(Refer Slide Time: 13:59)



A visual representation of the model looks like a V shape and hence this model is known as a V model. On the left side it's the development phase and on the right side it's the testing and maintenance. The system test cases are designed during the requirements specification because the final software has to conform the requirements as far as the final software is concerned.

In the V model it is advocated that during the requirement specification phase itself, the system test cases should be written. The advantages is that the testability of the requirements is kept in mind. Another issue is that in the iterative waterfall model, what do the testers do during the development phases, do they join only during testing phase?

As the V model provides a solution, the testers are actually busy writing the system test cases. During the high-level design, they are busy writing the test cases for integration testing and during the detailed design they are busy developing the unit test cases. Thus, during every development phase, testing is kept in mind and test cases are developed.

As, it becomes easier for the implementers to know what is expected, they become conscious of the work that they do & it has to finally satisfy those test cases.

(Refer Slide Time: 17:19)

V Model Steps

- Planning
- Requirements Analysis and Specification
- High-level Design
- Detailed Design
- System test design
- Integration Test design
- Unit test design

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

If we look at the different steps of the V model, we will see that during the requirements analysis and specification some test activity is done. During high-level design the

integration test case design and integration planning is done. During detail design the unit test design is done.

(Refer Slide Time: 17:58)

V Model: Strengths

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

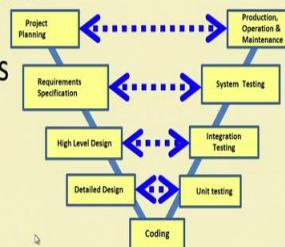


The strength of the V model is obvious as it emphasizes verification and validation. Every deliverable is made testable. As the model is easy to use and similar to the waterfall model accepting that the V and V activities that is the verification & validation are spread throughout the lifecycle.

(Refer Slide Time: 18:38)

V Model Weaknesses

- Does not support overlapping of phases
- Does not handle iterations or phases
- Does not easily accommodate later changes to requirements
- Does not provide support for effective risk handling



Weaknesses of the V model? It suffers from the same weakness as the iterative waterfall model, that is the requirements are frozen before the project starts and there is no scope of changing the requirements later just as waterfall model. Another issue with the waterfall model and the V model is that it does not support overlapping of phases.

(Refer Slide Time: 20:34)

The slide has a title 'When to use V Model' and a bulleted list of reasons:

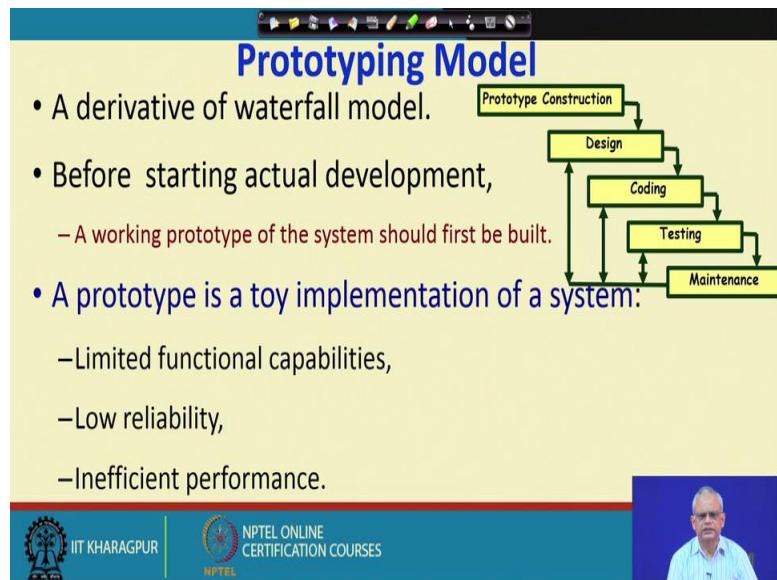
- Natural choice for systems requiring high reliability:
 - Embedded control applications, safety-critical software
- All requirements are known up-front
- Solution and technology are known

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a person speaking.

It must be clear that the V model is used when there is a high emphasis on safety reliability embedded such as the embedded control applications. The requirements should be known upfront and if these are the characteristics of the project then V is a good model.

Now let us look at another variant of the waterfall model which is known as the prototyping model.

(Refer Slide Time: 21:13)



The prototyping model is very similar to the waterfall model and the small difference is that there is a prototype construction and the requirements phase is not there, it's actually a prototype construction. It is a small change of the waterfall model.

Before the development starts a prototype needs to be built. A prototype is a toy implementation of the system. Imagine that we are trying to implement a function, for a toy implement of the function we might have a table stored inside the function, that just looks up the values that is required for the function.

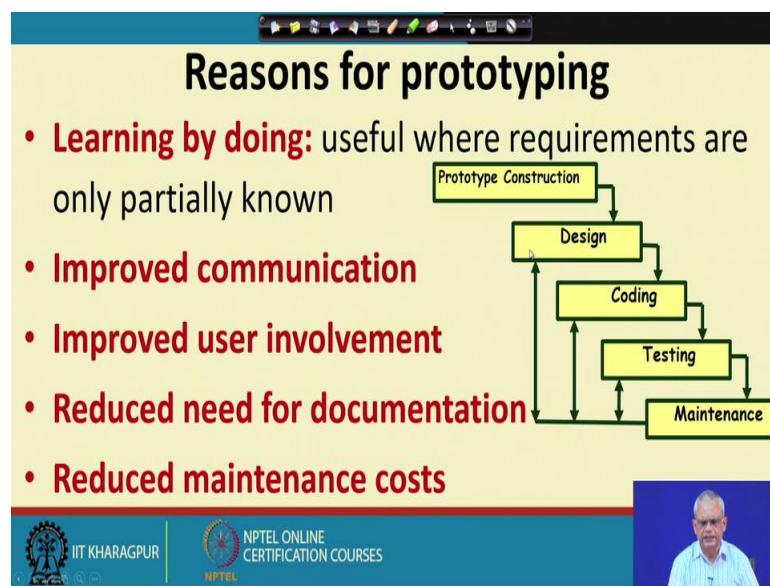
For restricted input values it will work as we have stored that in a table & it would have a limited functional capability and may not work for many inputs. The functions are simplified as much as possible. We need to develop a prototype because we can show it to the customer.

It's very hard for the customer to visualize the entire software and that is one of the reasons why the waterfall model is unsuccessful in many projects. Since in the prototyping model before starting the development, we are creating a prototype of the software. The customer will get a better feeling of how the software will finally appear and he might suggest changes, that can be incorporated into the prototype and until the customer exactly meets his requirement, the prototype refinement will continue.

Another reason why a prototype needs to be constructed is that sometimes the developers do not know that whether some technical issues can be met.

There are two advantages for developing the prototype; one is that the customer gets a feel of the system and can modify any requirements. The second is technical that the developers might want to experiment with some aspect on the technicalities and then decide which way to start the construction.

(Refer Slide Time: 25:46)



There is another reason why prototyping is useful is that the developers learn by doing the work. So, the second once the prototype is complete they start the development and they can do a better job. And the user looking at the prototype can finalize the requirements. That is improved communication with the customer. In the pure waterfall model the customer feels isolated for the entire development and the customer has to just wait but here they could see the prototype and look how the system will look like finally.

It also reduces the need for documentation and also typically a prototyping model. The maintenance costs are low because the quality of the development is good and thus it incur less maintenance costs.

(Refer Slide Time: 27:20)

Reasons for Developing a Prototype

- **Illustrate to the customer:**
 - input data formats, messages, reports, or interactive dialogs.
- **Examine technical issues associated with product development:**
 - Often major design decisions depend on issues like:
 - Response time of a hardware controller,

The customer is illustrated about the software and can make up his mind & specially examine the user interface issues, the format of the display, interactive dialogues and etc. Even the developers themselves can examine technical issues like the response time of a hardware controller. Hence, these two are the major advantages of prototype model that the customer can get a view of the software and also the developers can examine technical issues.

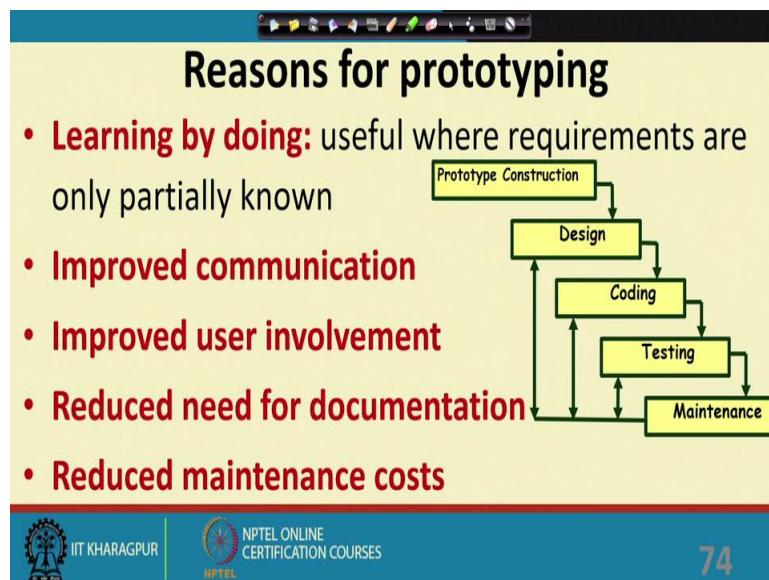
We will stop at this point and we will continue in the next lecture. Thank you.

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

Lecture - 10
Incremental Model

Welcome. In the last lecture, we discussed about the Prototyping model. The prototyping model is a variant of the waterfall model.

(Refer Slide Time: 00:32)



In the prototyping model, in place of the requirements analysis and specification we have the prototype construction. First a prototype is constructed and then it is demonstrated to the user and any changes that are requested by the customer are accommodated. Once the customer agrees to the prototype then the development starts using the waterfall model. There are many advantages of the prototype model. First is that when the requirements are written it's very difficult to identify what is missing and what needs to be changed which are ambiguous and etc.

But if the developer start developing the prototype then it become clear that what are the difficulties with the requirements. It improves the communication with the customer rather than just exchanging documents or just talking in different meetings. Here they can see something in real life and they can provide their feedback on that.

The user feels involved in the development because the changes are made as per his comments. There is a reduced need for documentation because the prototype serves as the purpose of the requirement. A service document is not there because the prototype serves as an animated requirement specification. If necessary, depending on the project an SRS document can be written based on the prototype.

Normally an SRS document is not needed because the prototype serves as the requirement specification. It results in reduced maintenance costs because the development quality is good. The developers first develop the prototype and gain expertise by doing and are better prepared to carry out the actual development and normally the software has better quality. Therefore, the maintenance costs are lower.

(Refer Slide Time: 03:28)

Reasons for Developing a Prototype

- **Illustrate to the customer:**
 - input data formats, messages, reports, or interactive dialogs.
- **Examine technical issues associated with product development:**
 - Often major design decisions depend on issues like:
 - Response time of a hardware controller,
 - Efficiency of a sorting algorithm, etc.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

To summarize the main reasons why the prototype model is being used; There are basically 2 reasons. First is that the customer can be illustrated about the software. Especially they can look at the input data formats, the messages being displayed, reports being produced, interactive dialogues and etc because this user interface part is to a large extent is a personal choice.

That's the reason why for almost every project in the GUI part is developed using a prototype model and even for some software, the entire development is according to a prototype model. The GUI for every project typically is through a prototype model

where a quick prototype is made for the GUI and the client feedback is obtained. It also allows the developers to examine the technical issues for which they are not sure.

For example, the design decisions might depend on what is the response time of a hardware controller and if they can experiment that using a prototype, they can design it better.

(Refer Slide Time: 05:29)

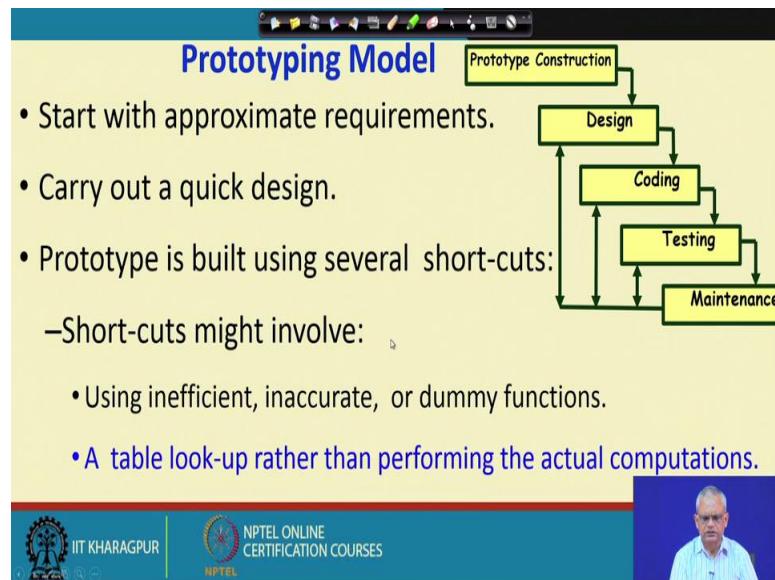
The slide has a yellow background and a blue header bar. The title 'Prototyping Model' is in blue, with '(CONT.)' in smaller parentheses. Below the title is a bulleted list:

- Another reason for developing a prototype:
 - It is impossible to “get it right” the first time,
 - We must plan to throw away the first version:
 - If we want to develop a good software.

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a man speaking.

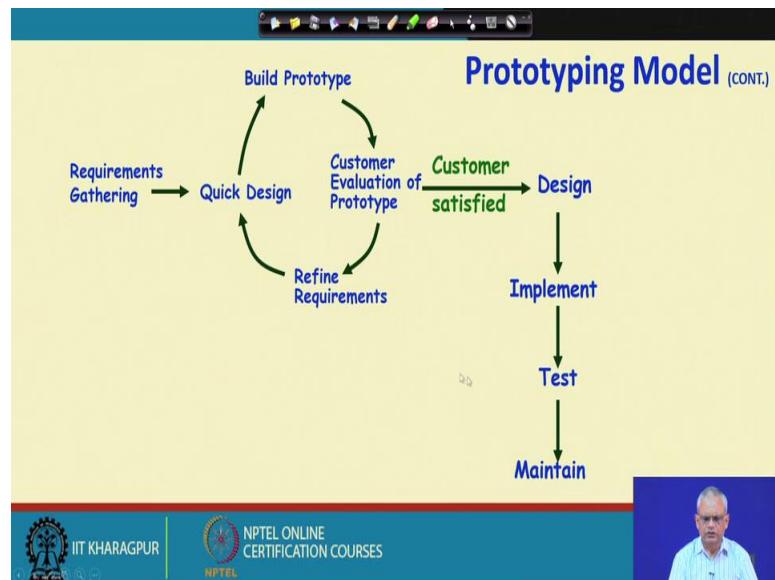
Another reason that leads to a good quality software is that it is impossible to get it right the first time. If the actual development starts then it becomes easy and good quality software's can be obtained and that's the reason why many experienced developers say that they must plan to throw away the first version, if we want to develop a good software.

(Refer Slide Time: 06:15)



It is demonstrated to the customer and the prototype is built using several shortcuts maybe dummy functions. Table look up rather than the actual computation and etc. The client feedback is obtained on the prototype and as long as the client agrees the actual development continues.

(Refer Slide Time: 07:02)



We can represent the prototyping model with the above schematic. Initial requirement gathering is a quick design & build prototype. Give the prototype to the customer for evaluation and based on the feedback refine the requirements. Again, do a quick design

based on the changed requirements, refine the prototype and etc. It goes on this cycle until the customer is satisfied and then the traditional waterfall model of development starts.

(Refer Slide Time: 07:38)

Prototyping Model (CONT.)

- The developed prototype is submitted to the customer for his evaluation:
 - Based on the user feedback, the prototype is refined.
 - This cycle continues until the user approves the prototype.
- The actual system is developed using the waterfall model.

The diagram illustrates the Prototyping Model as an iterative process. It starts with 'Requirements Gathering' leading to 'Quick Design'. This leads to 'Customer Evaluation of Prototype'. If the customer is satisfied, it moves to 'Design', then 'Implement', 'Test', and finally 'Maintain'. If not satisfied, it loops back to 'Refine Requirements' and re-enters the cycle at 'Customer Evaluation of Prototype'.

Thus, the actual development here is the waterfall model and the main difference with the waterfall model is the prototype construction.

(Refer Slide Time: 07:59)

Prototyping Model

- Requirements analysis and specification phase becomes redundant:
 - Final working prototype (incorporating all user feedbacks) serves as an **animated requirements specification**.
- Design and code for the prototype is usually thrown away:**
 - However, experience gathered from developing the prototype helps a great deal while developing the actual software.

The diagram shows a linear waterfall-like process for prototyping. It starts with 'Prototype Construction', followed by 'Design', 'Coding', 'Testing', and 'Maintenance'. There is a feedback loop from 'Testing' back to 'Design'.

The prototype serves as animated requirements specification. The prototype serves for the SRS document. But it's important to remember that once the prototype works, it's

not really refined into the actual software. The software is written again from scratch. The initial prototype is thrown away and new software is written. But the experience gathered from developing the prototype helps the developers.

(Refer Slide Time: 08:44)

The slide has a yellow background and a blue header bar. The title 'Prototyping Model (CONT.)' is centered in the header. Below the title, there are two bulleted lists. The first list discusses cost efficiency, mentioning 'overall development cost usually lower for:' and specific cases like 'Systems with unclear user requirements' and 'Systems with unresolved technical issues.' The second list discusses the resolution of requirements and technical issues, noting that problems 'would have appeared later as change requests and resulted in incurring massive redesign costs.' At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Even though construction of a working prototype model involves additional cost --- **overall development cost usually lower for:**
 - Systems with unclear user requirements,
 - Systems with unresolved technical issues.
- Many user requirements get properly defined and technical issues get resolved:
 - These would have appeared later as change requests and resulted in incurring massive redesign costs.

Starting the development would finally cost more rather than experimenting upfront by developing a prototype that would be a better idea.

For many types of projects, prototype construction may be a small overhead but lot of saving can be there in actual development. Many user requirements get defined, technical issues get resolved and therefore change requests are reduced to a large extent and that results in a good quality software. Thus, the prototype model is often preferred and especially for the GUI the prototyping model is normally used.

(Refer Slide Time: 10:16)

Prototyping: advantages

- The resulting software is usually more usable
- User needs are better accommodated
- The design is of higher quality
- The resulting software is easier to maintain
- Overall, the development incurs less cost

The resulting software is more usable. User needs are better accommodated and the software is of good quality. The resulting software is easier to maintain and the overall development cost is very low.

(Refer Slide Time: 10:39)

Prototyping: disadvantages

- For some projects, it is expensive
- Susceptible to over-engineering:
 - Designers start to incorporate sophistications that they could not incorporate in the prototype.

What are the disadvantages? For some projects prototyping may be expensive.

For example, if all the issues are understood, the requirements are clear then the prototype construction will just add to an extra overhead. Another difficulty with the prototype model is over-engineering.

Once the developers develop the prototype then they start to incorporate the sophistications that did not incorporate into prototype & even if some features are not required, they might be tempted to do it because they had the experience in writing simple prototype. They would like to make it much more sophisticated. Hence, when the prototyping model is used, it is often the case that the developers feel tempted to over engineer the actual software.

(Refer Slide Time: 11:47)

The slide has a blue header bar with various icons. The main title is 'Major difficulties of Waterfall-Based Models'. Below it is a list of three items:

1. Difficulty in accommodating change requests during development.
■ **40% of the requirements change during development**
2. High cost incurred in developing custom applications.
3. **"Heavy weight processes."**

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a man speaking.

Let's look into the major difficulties with the waterfall-based model and why these models were not used and newer models came up. One of the big problems is that the requirements need to be frozen before the project development starts, that is the characteristic of every waterfall-based model.

In reality, it is observed that about 40 percent of the requirements change as the development starts and in this situation, if a waterfall model is used then the final developed software will not meet the user requirements. The waterfall model is good for software that is written from scratch as it was being done in 1970's and 80's.

But now in the customization of applications using waterfall model becomes more problematic because the waterfall model needs to define all the requirements. The customization maybe such that we need to change some small aspect of an already working software. Hence, the work here may involve understanding what is required to

be changed & changing those are will be difficult as to accommodate within a waterfall model.

The last important difficulty with a waterfall-based model is that there are “Heavy weight processes” the name is as such because a lot of documentation is produced. At the end of every phase documentation is produced which are reviewed and a symptom of all waterfall-based models is that at the end of the software development there is a huge mountain of documents and that is the reason why the waterfall model is called heavy weight processes. It is observed that in waterfall model nearly half the time the developers are documenting and it is not real development. So, if we reduce the documentation will the development can be done faster. We will see that in the later development models namely the Agile models as the documentation is kept to minimum.

(Refer Slide Time: 15:48)

Major difficulties of Waterfall-Based Life Cycle Models

- Requirements for the system are determined at the start:
 - Are assumed to be fixed from that point on.
 - Long term planning is made based on this.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

NPTEL

But one of the major difficulties which is acutely felt is that the requirements need to be defined at the start and from that point it becomes fixed. As the project manager makes plans based on the requirements & if any change is incorporated it will increases the cost to a larger extent.

(Refer Slide Time: 16:35)

“... the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built, and install it. ...this assumption is fundamentally wrong and many software acquisition problems spring from this...”

Frederick Brooks



IIT KHARAGPUR

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Let's look what Frederick Brooks, a celebrated author in this area has to say “the assumption that one can specify a satisfactory system in advance, get bids for its construction, have it built and install it. This assumption is fundamentally wrong and many software accusation problems spring from this. Brooks has experienced in many projects and observed that for any customer it becomes very difficult to give all the requirements in advance and then get somebody to construct it. He says that this is fundamentally wrong and most project failures can be attributed to this.

(Refer Slide Time: 17:32)

- “The basic idea... take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system. Learning comes from both the development and use of the system... Start with a simple implementation of a subset of the software requirements and iteratively enhance the evolving sequence of versions. At each version design modifications are made along with adding new functional capabilities.”

Victor Basili



IIT KHARAGPUR

NPTEL

NPTEL ONLINE CERTIFICATION COURSES

Looking at the incremental model. In this model, we develop a increment and then we learn by developing one increment and then do the next increment and make the installment at the client place and then take the feedback and in the next increment we accommodate it and this is the iterative enhancement of the versions.

Briefly, “The basic idea, take advantage of what is being learned during development of a increment. Learning comes from both development and use of the system. Start with a simple implementation of a subset of the requirement; iteratively enhance the evolving sequence of versions. At each version design modifications are made along with adding new functional capabilities.”

(Refer Slide Time: 18:36)

The slide has a yellow background with a black border. At the top right, it says 'Incremental and Iterative Development (IID)'. On the left, there's a bulleted list:

- Key characteristics
 - Builds system incrementally
 - Consists of a planned number of iterations
 - Each iteration produces a working program
- Benefits
 - Facilitates and manages changes
- Foundation of agile techniques and the basis for
 - Rational Unified Process (RUP)
 - Extreme Programming (XP)

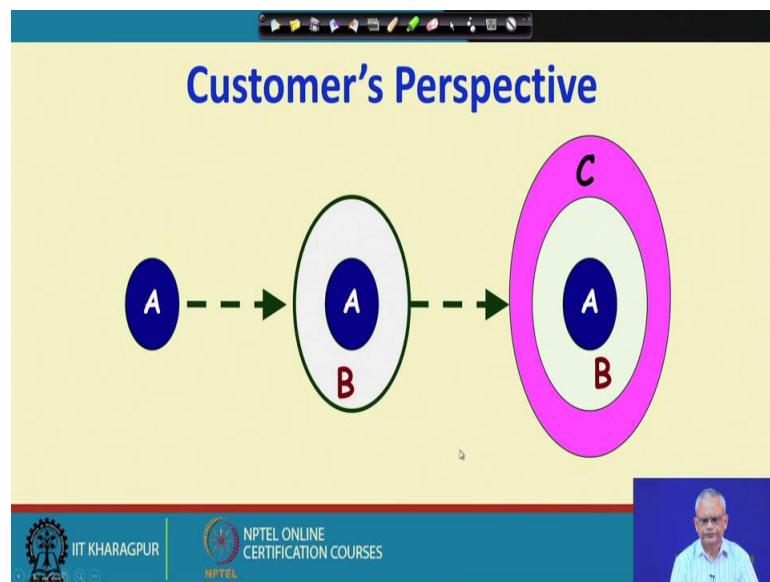
At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a man speaking.

The key characteristic of the incremental and iterative development is that it builds the system incrementally. There is a planned number of iterations and the initial requirements are split into versions and those are the increments. Each iteration produces a working program which is installed at the client’s place.

The main benefits are that the client can give feedback which can be incorporated in the next increment and these techniques are actually the foundation for the agile techniques (i.e. the rational unified process and the extreme programming).

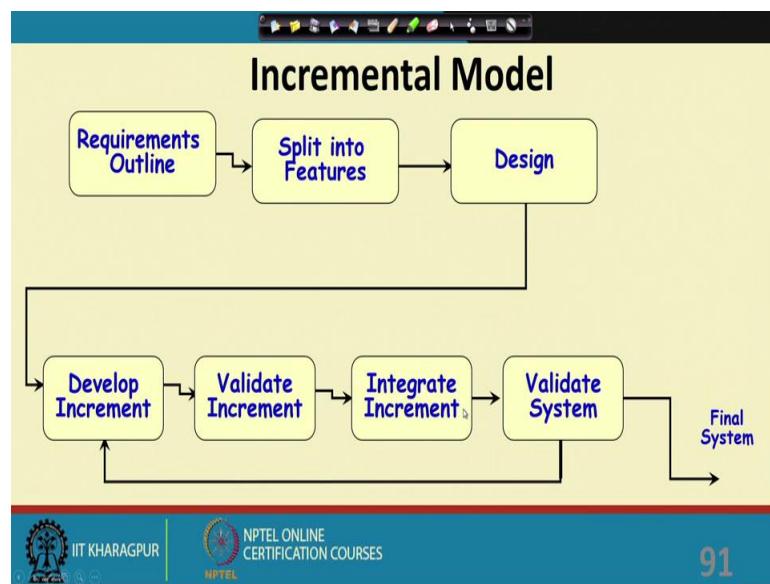
The incremental development has many advantages.

(Refer Slide Time: 19:38)



If we look from a customer's perspective of an incremental development. Initially they get a core part of the software, experiments and provides feedback. Then they receive a slightly larger software until the full software is made.

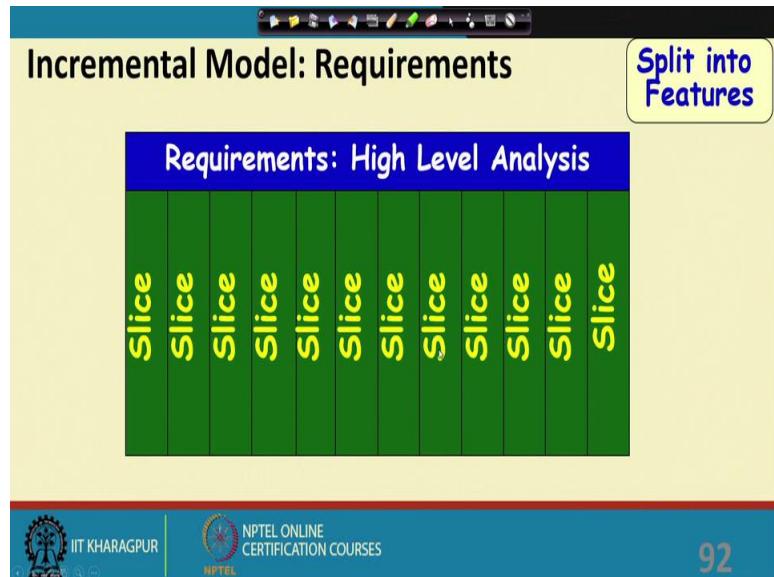
(Refer Slide Time: 19:59)



But as far as the designers are concerned, the developers get the requirements and they split this into deliverable features. An overall design is made and they keep on developing the increments, test the increment, integrate it with the previous software,

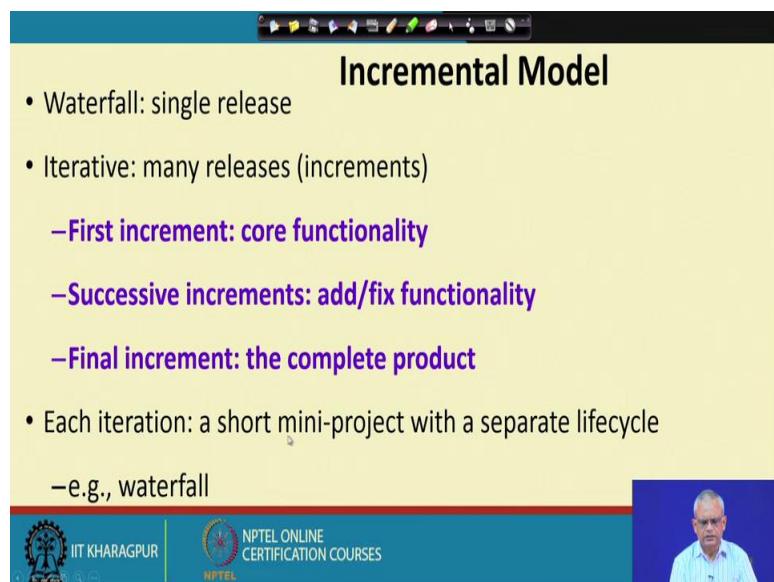
validate the system and etc until all the features are accommodated and finally deliver the system.

(Refer Slide Time: 20:46)



The initial requirements are split into features and they can go on developing one feature at a time.

(Refer Slide Time: 21:01)

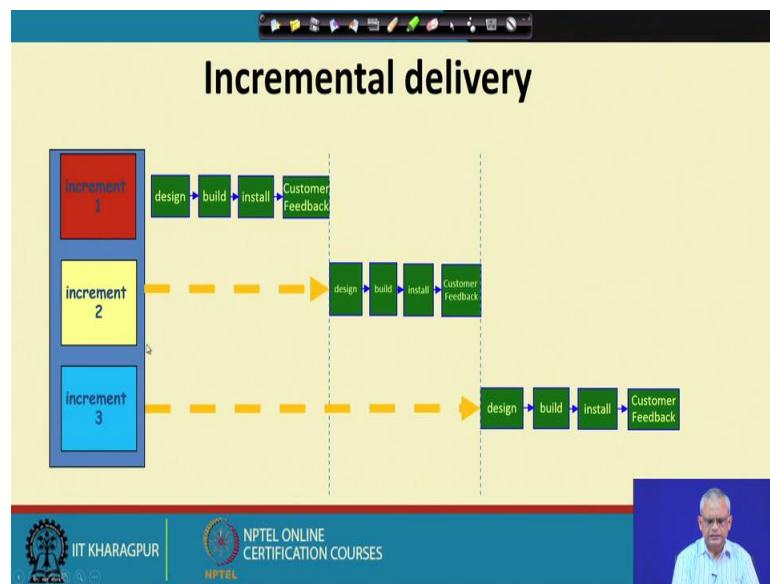


If we compare the incremental model with the waterfall model, the waterfall model has a single release but in the iterative model there are many releases to the customer. The first increment is typically the core functionality and on each increment new

functionalities are added and as per the customer's feedback, the modification are made and the final increment of complete product is done.

But each iteration is a short mini project which may be used in a life cycle like waterfall.

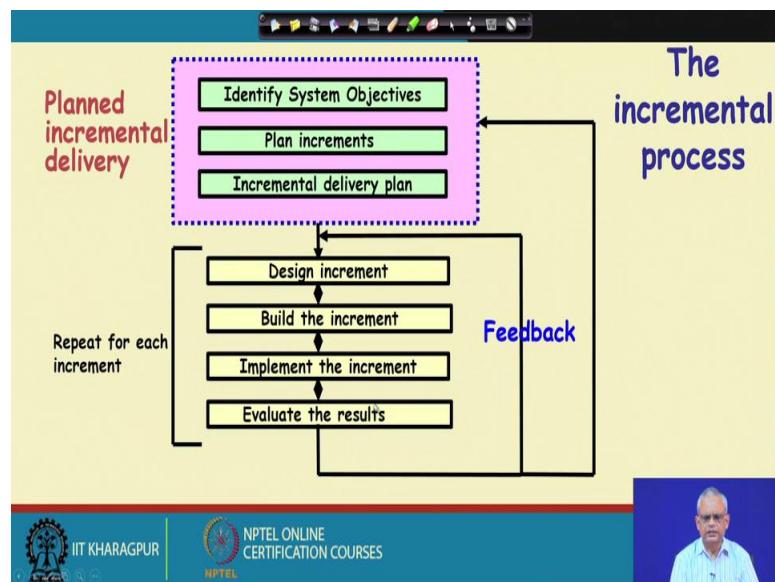
(Refer Slide Time: 21:54)



The entire requirement is split into increments; the first increment is the design, build, install and customer feedback. Once the first increment customer feedback is obtained, the second increment gets started and then the third and rest as follows.

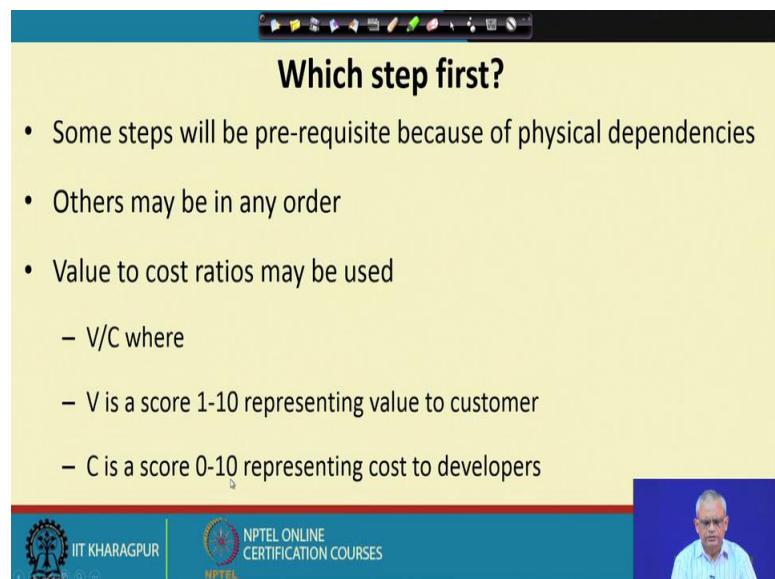
The time to develop one increment is typically known as a time box.

(Refer Slide Time: 22:34)



We can represent the incremental model. The increments are planned such that as the development starts each time one increment is designed. Build the increment and implement it and even evaluate the results and this goes on building one increment after the other until the full software is developed.

(Refer Slide Time: 23:17)



But which increment will be done first and how will the developers decide?

Initially we have to do some increments first because the others might have dependency on that. But if the requirements are done in any order then we can compute a value to cost ratio which is known as the V by C ratio; where, V is the value to the customer.(Give a value between 1 to 10) and C is the cost is the cost of development (Between 1 to 10).

(Refer Slide Time: 24:40)

V/C ratios: an example				
step	value	cost	ratio	
profit reports	9	2	4.5	2nd
online database	1	9	0.11	5th
ad hoc enquiry	5	5	1	4th
purchasing plans	9	4	2.25	3rd
profit-based pay for managers	9	1	9	1st

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

97

This above one just an example and we will discuss about the evolutionary model with the iterations in the next lecture.

We saw that incremental model is a very important model as it has lot of advantages and next, we look at the evolutionary model and based on these two models we look at the agile models.

Thank you.

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

Lecture – 11
Evolutionary Model

Welcome to this lecture.

In the last lecture, we had discussed about the Incremental Development Model. The Incremental model overcomes many shortcomings of the waterfall model. This lecture, we will look at the Evolutionary model with iterations.

(Refer Slide Time: 00:43)



An Evolutionary and Iterative Development Process...

- Recognizes the reality of changing requirements
 - Capers Jones's research on 8000 projects: **40% of final requirements arrived after development had already begun**
- Promotes early risk mitigation:
 - Breaks down the system into mini-projects and focuses on the riskier issues first.
 - **"plan a little, design a little, and code a little"**
- Encourages all development participants to be involved earlier on,:
 - End users, Testers, integrators, and technical writers

In any real project, the requirements change during development. There are many reasons behind the requirements change. One of the reasons is that the customer is not able to view the software in entirety before the software is built, and therefore, many requirements are missed, many requirements are given ambiguously at the time of software built. Capers Jones's an established researcher, studied 8000 projects and found that 40 percent of the requirement change during the development.

This is a very significant issue because neither waterfall model nor the incremental model handled this satisfactory. Waterfall model has no provision for requirement change during development. In the incremental model we know, the model involves

identifying all the requirements upfront and then slicing the requirements into incremental features to be deployed at the customer side. Of course, while a feature is being installed, the customer's feedback is obtained, which may necessitate additional requirements or change of requirements. But still, the fact remains that the incremental model requires all the requirements to be identified upfront as far as possible.

Let's look at the Evolutionary model, which tries to overcome the necessity of having to identify all the requirements upfront. There are some similarities between the evolutionary model and the incremental model in the sense that increments are deployed at the client-side. But here, in the evolutionary model, no upfront requirement specification is required. Initially, some features are implemented which have been identified. And then, as the customer experiments with those more and more features get developed and deployed. So, the software actually evolves, starting with something very simple. But the main difference between the incremental model, and the purely evolutionary model is respect to upfront identification of the requirements.

The evolutionary model is also called as "plan a little, design a little, code a little." Because in the evolutionary model, each time only a few features are identified. Plan for those feature developments which are only designed, coded, and deployed at the customer side. This model, just like the incremental model, involves the end-users. End-users can give their experience on that. The testers, incrementors, technical writers all get involved from the start of the project, unlike the iterative waterfall model.

(Refer Slide Time: 04:48)

Evolutionary Model with Iteration

- “A complex system will be most successful if implemented in small steps... “retreat” to a previous successful step on failure... opportunity to receive some feedback from the real world before throwing in all resources... and you can correct possible errors...” [Tom Glib in Software Metrics](#)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Tom Glib, an eminent personality in the software engineering area, has said. “A complex system will be most successful if implemented in small steps... “retreat” to a previous successful step on failure... opportunity to receive some feedback from real world before throwing in all resources... and you can correct possible errors...”

So, he has identified the big advantage of the revolutionary model that the client feedback is obtained and the features are determined as the client keeps on using the system evolves. And if some feature the customer does not like, then that feature is simply discarded and new features replacing that are built.

(Refer Slide Time: 05:53)

Evolutionary model with iteration

- Evolutionary iterative development implies that the requirements, plan, estimates, and solution evolve or are refined over the course of the iterations, rather than fully defined and “frozen” in a major up-front specification effort before the development iterations begin. Evolutionary methods are consistent with the pattern of unpredictable discovery and change in new product development.” **Craig Larman**



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Craig Larman, another celebrated personality in the area software engineering, has said “Evolutionary iterative development implies that the requirements, plan, estimate, and solution evolve or a refined over the course of the iterations, rather than fully defined and “frozen” in a major up-front specification effort before the development iterations begin. Evolutionary methods are consistent with the pattern of unpredictable discovery and the change in new product development.”

So from this, we can say, Craig Larman also agrees that upfront specification is very difficult, and the evolutionary model is bound to be much more useful in real-life projects.

(Refer Slide Time: 06:54)

Evolutionary Model

- First develop the core modules of the software.
- The initial skeletal software is refined into increasing levels of capability: (Iterations)
 - By adding new functionalities in successive versions.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Let's look at what is involved in the Evolutionary model. Based on an overall understanding of the software, first the core modules of the software are developed and this core module are refined into increasing capability levels which are called as the iterations.

(Refer Slide Time: 07:28)

The slide has a yellow header bar with the title 'Activities in an Iteration'. Below the title is a bulleted list of activities:

- Software developed over several “mini waterfalls”.
- The result of a single iteration:
 - Ends with delivery of some tangible code
 - An incremental improvement to the software --- leads to evolutionary development

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

Each iteration adds new functionalities and these are successively deployed at the client site. Each iteration is actually a mini waterfall. After a single iteration some code is deployed and the client expresses his next requirement and this leads to the evolutionary development.

(Refer Slide Time: 07:56)

The slide has a yellow header bar with the title 'Evolutionary Model with Iteration'. Below the title is a bulleted list of characteristics of the evolutionary model:

- Outcome of each iteration: tested, integrated, executable system
- Iteration length is short and fixed
 - Usually between 2 and 6 weeks
 - Development takes many iterations (for example: 10-15)
- Does not “freeze” requirements and then conservatively design :
 - Opportunity exists to modify requirements as well as the design...

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

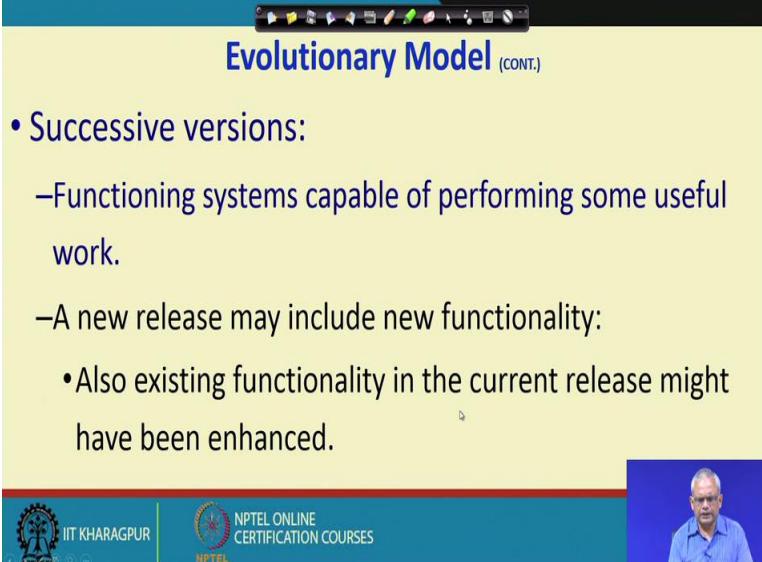
In each iteration, some code which is tested, integrated, executables system is deployed at the client side. The iteration length is short typically 2 to 6 weeks. A software to be completely developed may take somewhere between 10 to 15 iterations.

We have emphasized several times that requirements are not collected upfront and frozen and then, conservatively designed to accommodate any future changes. But here, the requirements are allowed to change and the design also changes.

(Refer Slide Time: 08:52)

Evolutionary Model (CONT.)

- Successive versions:
 - Functioning systems capable of performing some useful work.
 - A new release may include new functionality:
 - Also existing functionality in the current release might have been enhanced.

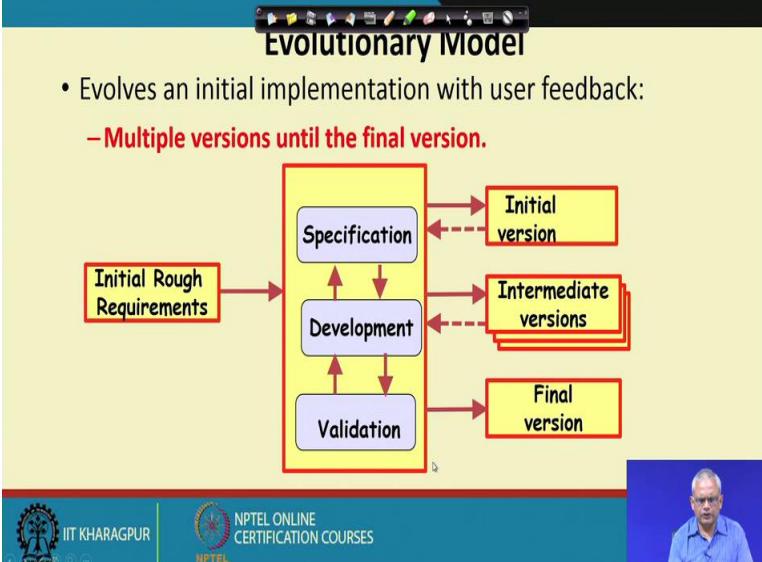


All the successive versions are fully functional systems. The customer can make use of them in real work. When a new release made new functionality are added. But some of the existing functionality based on the customer feedback might have been changed or enhanced.

(Refer Slide Time: 09:24)

Evolutionary Model

- Evolves an initial implementation with user feedback:
 - **Multiple versions until the final version.**



We can represent the model in this schematic: initially, some rough understanding of the system is done, which is not the full specification of the system, unlike the incremental model. Here just the overall feature required and then the core, and each iteration carried out. In every iteration, there is a specification development validation, and then finally, the deployment.

The core is deployed, and the dotted arrow in the above figure shows feedback obtained. And again, the iteration occurs, and all the successive versions are deployed at the customer site; feedback obtained goes on as the iterations go on until the final version is deployed at the customer site.

(Refer Slide Time: 10:30)

Advantages of Evolutionary Model

- Users get a chance to experiment with a partially developed system:
 - Much before the full working version is released,
- **Helps finding exact user requirements:**
 - Software more likely to meet exact user requirements.
- **Core modules get tested thoroughly:**
 - Reduces chances of errors in final delivered software.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

There are many advantages of this evolutionary model. One of the advantages is the users can experiment, can give their feedback; they can use the software also. So, that they get a real feel of how it performs, and therefore, it helps find exact user requirements, and finally, when it is deployed at the customer site very likely to meet the exact user requirements.

And also, since the modules are deployed and used at the customer site, any defects are noticed and reported. So, the core modules are used for a long time, and all defects must have been detected. And therefore, the final delivered software is much more reliable than would be the case in the waterfall model.

(Refer Slide Time: 11:39)

Advantages of evolutionary model

- Better management of complexity by developing one increment at a time.
- Better management of changing requirements.
- Can get customer feedback and incorporate them much more efficiently:
 - As compared when customer feedbacks come only after the development work is complete.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Since, in the evolutionary model the developers focused only one or two features. So, the complexity is easily manageable. It welcomes changes from the customer after the use and is incorporated. No long-term plans are made and that is the region where the changing requirements can be easily accommodated unlike the waterfall model.

(Refer Slide Time: 12:16)

Advantages of Evolutionary Model with Iteration

- Training can start on an earlier release
 - customer feedback taken into account
- Frequent releases allow developers to fix unanticipated problems quicker.

The slide includes logos for IIT Kharagpur and NPTEL, and a video player interface showing a speaker's face.

The customer deploys and can incremental learn the software and also find any part that is difficult can give feedback. Customer feedback is obtained and made much more usable. Frequent releases allow the developers to fix any problems quicker because they are into the development and any problem that occurs they can developed something to fix the problems quickly.

(Refer Slide Time: 12:56)

Evolutionary Model: Problems

- **The process is intangible:**
 - No regular, well-defined deliverables.
- **The process is unpredictable:**
 - Hard to manage, e.g., scheduling, workforce allocation, etc.
- **Systems are rather poorly structured:**
 - Continual, unpredictable changes tend to degrade the software structure.
- **Systems may not even converge to a final version.**

The slide includes logos for IIT Kharagpur and NPTEL, and a video player interface showing a speaker's face.

But then, the evolutionary model has also some problems or drawbacks. One is that the process as we described our edge is described in the literature is rather vague. And

therefore, the process is intangible. The process is not regular, not well defined; the process is unpredictable.

Because no long-term plans are made, we don't know how long a software will take to fully developed once we start the iterations. We don't know how many iterations it will take to complete the process. It may complete in 3 months; it may complete in 6 months. So, the control is very less. From the beginning it is hard to predict how many iterations it will take, it is hard to manage the workforce, it is hard to know how many persons will work, for how long they will work, it is hard to calculate the total cost etc.

Another major difficulty here is that the software keeps on changing. And therefore, Continuous changes to the software degrade the design of the software, and also it may so happen that the clients keep on giving features, modifying features, and so on. And in the bad case, the system may not even converge to a final solution, and the project may not complete.

(Refer Slide Time: 14:57)

The slide has a yellow background. At the top, the title 'Rapid Application Development (RAD) Model' is centered. Below the title is a bulleted list:

- Sometimes referred to as the **rapid prototyping model**.
- Major aims:
 - Decrease the time taken and the cost incurred to develop software systems.
 - Facilitate accommodating change requests as early as possible:
 - Before large investments have been made in development and testing.

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there are logos for IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man speaking.

Now, we will discuss about the RAD model. The RAD model stands for rapid application development model. This is also a popular model; it is called as the Rapid prototyping model. As the name rapid here indicates that the emphasis here is to reduce the development time and the cost and also it facilitates accommodating changes as early as possible.

(Refer Slide Time: 15:28)

The slide has a yellow background. At the top center, it says "Important Underlying Principle". Below that is a bulleted list: "• A way to reduce development time and cost, and yet have flexibility to incorporate changes:" followed by a sub-point in a blue box: "• Make only short term plans and make heavy reuse of existing code." At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, "NPTEL ONLINE CERTIFICATION COURSES", and a video window showing a man speaking.

Here only short-term plans are made and another feature is heavy reuse of existing code that is customization projects.

(Refer Slide Time: 15:47)

The slide has a yellow background. At the top center, it says "Methodology". Below that is a bulleted list: "• Plans are made for one increment at a time." followed by a sub-point in a blue box: "• The time planned for each iteration is called a **time box**." Then it says "• Each iteration (increment):" followed by a sub-point in a blue box: "• Enhances the implemented functionality of the application a little." At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, "NPTEL ONLINE CERTIFICATION COURSES", and a video window showing a man speaking.

RAD model also obeys the incremental development methodology that at anytime only one increment is planned, and over one iteration it is completed, and deployed. And the iteration is called as a time box here. Each iteration or increment enhances the functionality of the application little.

(Refer Slide Time: 16:17)

The slide has a yellow header with the title 'Methodology'. Below the title is a bulleted list of four items. At the bottom of the slide, there is a footer bar with three sections: 'IIT KHARAGPUR' (with a logo), 'NPTEL ONLINE CERTIFICATION COURSES' (with a logo), and a video player showing a man speaking.

- During each iteration,
- A quick-and-dirty prototype-style software for some selected functionality is developed.
- The customer evaluates the prototype and gives his feedback.
- The prototype is refined based on the customer feedback.

Unlike the incremental and the evolutionary model, in the rapid development model since the focus is to develop something very rapidly. Therefore, a quick and dirty prototype is developed and deployed at the customer site to obtain the feedback, and as the customer gives the feedback, this prototype itself is refined based on the customer feedback.

So, we can this is a very different style advocated by the RAD development. In rapid development, the prototype is refined into the actual software. We know that in the prototyping model, the prototype was a throwaway software. And of course, the incremental and the evolutionary model, each increment was not a prototype, but a properly planned developed according to some lifecycle may be the waterfall model.

(Refer Slide Time: 17:37)

How Does RAD Facilitate Faster Development?

- RAD achieves fast creation of working prototypes.
 - Through use of specialized tools.
- These specialized tools usually support the following features:
 - **Visual style of development.**
 - **Use of reusable components.**
 - **Use of standard APIs (Application Program Interfaces)**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video thumbnail of a man speaking is visible in the bottom right corner.

As the emphasis of the RAD is to facilitate faster development, it creates prototype using specialized tools and the development can be fast by using visual style of development, drag and drop style, use of reusable components and also standard APIs. Standard APIs need to be used because that helps in reuse.

(Refer Slide Time: 18:16)

For which Applications is RAD Suitable?

- Customized product developed for one or two customers only
- Performance and reliability are not critical.
- The system can be split into several independent modules.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video thumbnail of a man speaking is visible in the bottom right corner.

The RAD model is suitable for software that is developed for one or two customers. And here, since the design is not so structured, the code quality is not so great. The

performance and reliability of the project are not at a premium level and also not suitable for very, very small software.

Software should be able to split into several independent modules.

(Refer Slide Time: 18:58)

The slide has a yellow background and a blue header bar with various icons. The title 'For Which Applications RAD is Unsuitable?' is centered in bold black font. Below it is a bulleted list of reasons:

- Few plug-in components are available
- High performance or reliability required
- No precedence for similar products exists
- The system cannot be modularized.

At the bottom left, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. On the right side, there is a small video window showing a man speaking.

And also, if we want to RAD model to really work, we should have many reusable components. For new product where reusable software is not available, there run RAD model will be unsuitable. If we need high performance or reliability, RAD model is unsuitable. RAD model will be useful when the software should be reasonably large and can be modularized into several modules.

(Refer Slide Time: 19:48)

The slide has a yellow header bar with the title 'Prototyping versus RAD'. Below the title, there are two main bullet points:

- In prototyping model:
 - The developed prototype is primarily used to gain insights into the solution
 - Choose between alternatives
 - Elicit customer feedback.
- The developed prototype:
 - Usually thrown away.

At the bottom of the slide, there is a footer bar with three logos: IIT Kharagpur, NPTEL Online Certification Courses, and a video camera icon.

It's easy to compare prototyping with the RAD model. In the prototyping model the developed prototype is a throwaway prototype. The main reason the prototype is developed is to get the customers suggestions and also insight into the solution.

(Refer Slide Time: 20:03)

The slide has a yellow header bar with the title 'Prototyping versus RAD'. Below the title, there are two main bullet points:

- In contrast:
 - In RAD the developed prototype evolves into deliverable software.
 - RAD leads to faster development compared to traditional models:
 - However, the quality and reliability would possibly be poorer.

At the bottom of the slide, there is a footer bar with three logos: IIT Kharagpur, NPTEL Online Certification Courses, and a video camera icon.

The developers can choose between alternatives by developing the prototype. Once the prototype has been used to evaluate alternatives, client feedback is obtained and then the prototype is thrown away and new completely planned development starts in a iterative waterfall model.

In contrast, in a RAD model the prototype itself is modified into deliverable software. Of course, that matches with the requirement or the objective of the RAD which is faster development, but quality and reliability become a question.

(Refer Slide Time: 21:00)

The slide has a yellow background and a blue header bar. The title 'RAD versus Iterative Waterfall Model' is at the top. Below it is a bulleted list comparing the two models:

- In the iterative waterfall model,
 - All product functionalities are developed together.
- In the RAD model on the other hand,
 - Product functionalities are developed incrementally through heavy code and design reuse.
 - Customer feedback is obtained on the developed prototype after each iteration:
 - Based on this the prototype is refined.

At the bottom left, there are logos for IIT Kharagpur and NPTEL. At the bottom right, there is a small video frame showing a man speaking.

But how does RAD compared with iterative waterfall model?

Iterative waterfall model, initially all requirements are captured and all functionalities are developed together, whereas in the RAD model these are incrementally developed. So, it has similarity with the incremental development model and on each iteration the customers feedback is obtained. And therefore in RAD model, the client interactions or customer interactions are high compare to iterative waterfall model.

(Refer Slide Time: 21:29)

RAD versus Iterative Waterfall Model

- The iterative waterfall model:
 - Does not facilitate accommodating requirement change requests.
- Iterative waterfall model does have some important advantages:
 - Use of the iterative waterfall model leads to production of good documentation.
 - Also, the developed software usually has better quality and reliability than that developed using RAD.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

It is easy to accommodate requirement changes based on customer feedback. But then, compared to RAD model, iterative waterfall model usually results in good quality design, higher reliability and also production of good documentation.

(Refer Slide Time: 21:57)

RAD versus Evolutionary Model

- Incremental development:
 - Occurs in both evolutionary and RAD models.
- However, in RAD:
 - Each increment is a quick and dirty prototype,
 - Whereas in the evolutionary model each increment is systematically developed using the iterative waterfall model.
- Also, RAD develops software in shorter increments:
 - The incremental functionalities are fairly large in the evolutionary model.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

How does RAD model compare with the evolutionary model? In both RAD and evolutionary increments are deployed the development occurs over several increments. In RAD the increment is a quick and dirty prototype. On the other hand, in the evolutionary model each increment is developed using iterative waterfall model.

And if we look at the size of the increment, the RAD increment is shorter and called as time box and on the other hand, in the evolutionary model each increment takes longer time and more functionality is completed in iteration.

Now, let's look at another very popular process the Unified Process.

(Refer Slide Time: 23:03)

The slide has a yellow background and a blue header bar with various icons. The title 'Unified Process' is centered above a bulleted list. The footer features the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a video player window showing a man speaking.

- Developed Ivar Jacobson, Grady Booch and James Rumbaugh
 - Incremental and iterative
- Rational Unified Process (RUP) is version tailored by Rational Software:
 - Acquired by IBM in February 2003.

The Unified Process was developed by Jacobson, Booch and Rumbaugh. This is extensively used for object-oriented software development and it is incremental and iterative. So, by the word incremental and iterative means here that the features had to be identified beforehand. And then, over several iterative development the software is deployed at the client-side and the client feedback is obtained.

The unified process tailored by the rational corporation is called as a rational unified process and the rational corporation was acquired by IBM in 2003.

(Refer Slide Time: 24:01)

Four Phases --- and iterative Development at Every phase

- Inception Phase
- Elaboration Phase
- Construction Phase
- Transition Phase

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

The development here in the unified process occurs over 4 phases: Inception, Elaboration, Construction and Transition. In each of these phases, consists of many increments.

(Refer Slide Time: 24:18)

Unified Process Iterations in Phases

The duration of an iteration may vary from two weeks or less.

Iterations

Iterations

Iterations

Iterations

Inception

Elaboration

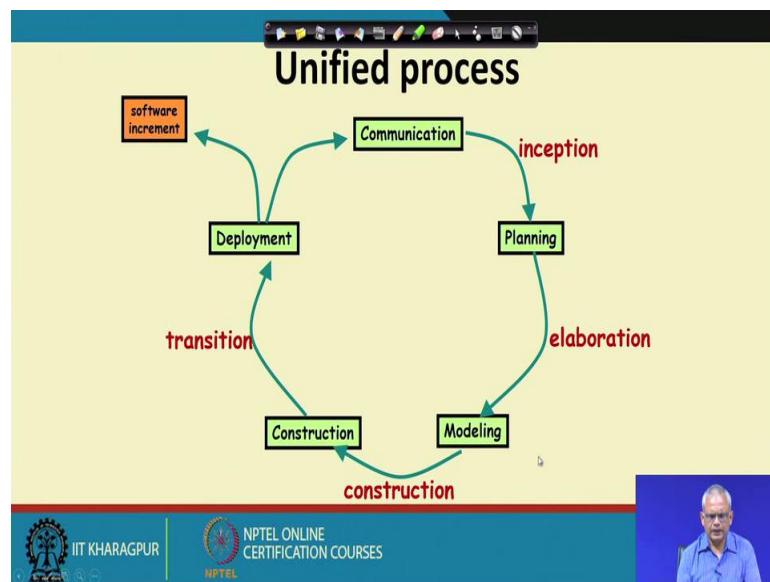
Construction

Transition

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

The inception phase occurs over many increments or iterations. In the Elaboration also many iterations can be there; In construction several iterations are there and finally, the transition. Now we will see, what all the phases implies.

(Refer Slide Time: 24:44)



In the Inception phase, communication with the customer is made and all the features are identified and also plan about the development process. During the Elaboration, identified features are modeled. And finally, in the Construction phase the software is constructed and developed. And during the Transition, the software is deployed in the client side.

May be at this point its looking simple but all these are not that much simple.

(Refer Slide Time: 25:26)

Unified process work products			
Inception phase	Elaboration phase	Construction phase	Transition phase
vision document initial use-case model initial business case initial risk list project plan prototype(s) ...	use-case model requirements analysis model preliminary model revised risk list preliminary manual ...	design model SW components test plan test procedure test cases user manual installation manual ...	SW increment beta test reports user feedback ...

We will see what exactly occurs that was a much simpler model to understand. Now, let see what are the output of every phase? In the inception phase, the initial use case model that is the features, the business case, risk list, project plan, any prototype etc. come as output.

In the elaboration phase, analysis model, preliminary model, manual use case model etc. come as output. The construction phase, the software is developed, the test plan is made, user manual and installation manual are written and finally, these are deployed at the customer site. In the construction phase the beta test reports and the user feedback is also obtained.

(Refer Slide Time: 26:29)

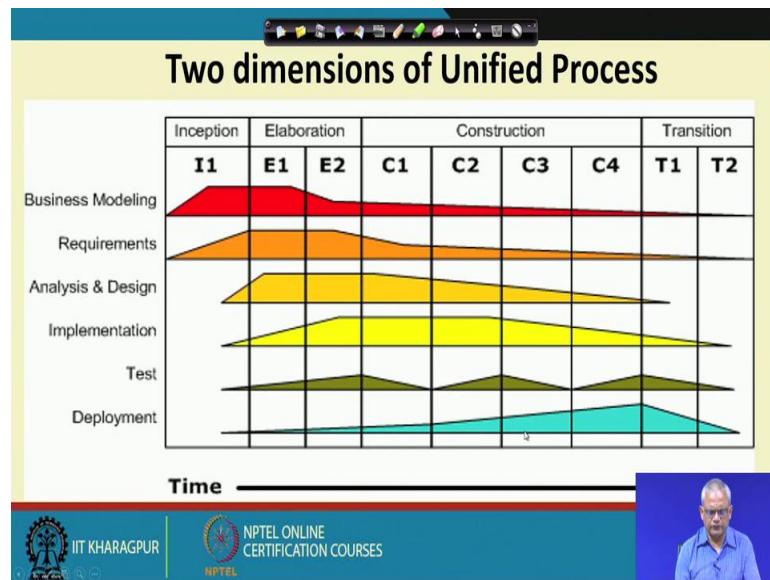
The slide has a yellow background and a dark blue header bar. The title 'Structure of RUP Process' is centered in the header. Below the title is a bulleted list:

- Two dimensions.
- Horizontal axis:
 - Represents time and shows the lifecycle aspects of the process.
- Vertical axis:
 - Represents core process workflows.

At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man speaking.

But then if we want to visualize this graphically which will show a more realistic picture of the rational unified process. The horizontal axis will show the lifecycle aspects and the vertical axis will represent the core process flow.

(Refer Slide Time: 26:47)



From the graphical representation we can see that the four phases are marked here and each one occurs over several iterations. The requirements and the planning peak during the inception phase. And even they continue during the elaboration and construction phase. The analysis and design, they peak during the elaboration and construction and slowly taper off. Implementation peaks during the construction phase and tapers off; but then, there are some construction during the elaboration phase also and we can see that the test is present throughout and deployment peaks towards the transition.

We are nearing the time for this lecture. We will stop here and continue from this point in the next lecture.

Thank you.

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

Lecture - 12
Agile Model

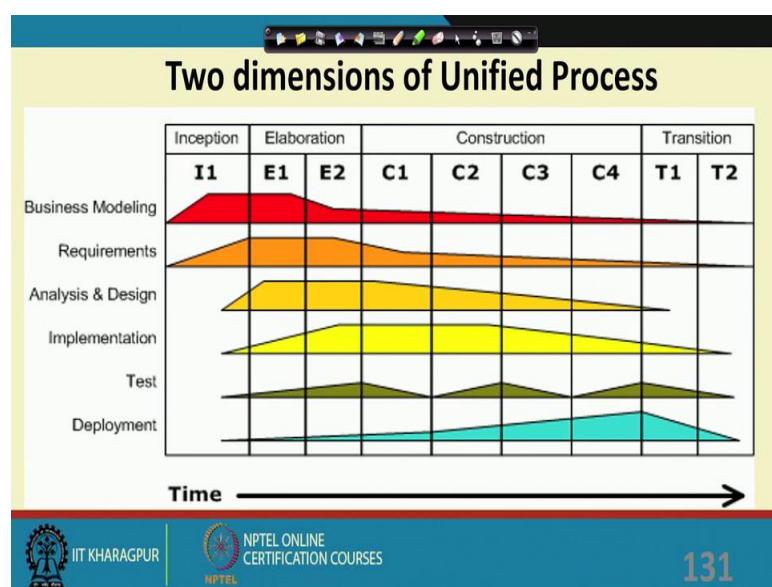
Welcome to this lecture.

In the last lecture, we had discussed two very conceptually important lifecycle models, namely, evolutionary model and the incremental model. And, many successful models are based on these two principles. We looked at the RAD model and also the unified process.

The unified process is extensively used for object-oriented software development. It's an incremental model in the sense that much of the requirement is identified upfront. Unified process also has features of evolutionary because the features continue to be discovered as the development proceeds.

The development occurs over 4 phases.

(Refer Slide Time: 01:19)



The inception, elaboration, construction, and transition, and each phase can consist of several iterations. The life cycle activities that overlap across different phases, but then

they peak during certain phases. For example, in the business modeling, the requirement identification like how the business operates peaks during inception and elaboration and slowly tapers off. Analysis and design start during elaboration and slowly start to taper off. Implementation peaks during construction and during the elaboration also and then tapers off. The testing done all through phases. And then these are deployed at the customer site, but the deployment peaks during the transition.

(Refer Slide Time: 02:50)

The slide has a yellow background. At the top, the title 'Inception activities' is centered in bold black font. Below the title is a bulleted list of three items, each preceded by a blue circular bullet point:

- Formulate scope of project
- Risk management, staffing, project plan
- Synthesize a candidate architecture.

At the bottom of the slide, there is a footer bar with three sections: 1) IIT KHARAGPUR logo and text 'IIT KHARAGPUR'. 2) NPTEL logo and text 'NPTEL ONLINE CERTIFICATION COURSES'. 3) A small video window showing a man in a blue shirt speaking.

During the inception, the scope of the project is determined, some plan documents like risk management plan, staffing project plan are made and, also an overall architecture is determined based on the project requirements.

(Refer Slide Time: 03:22)

The slide has a yellow header bar containing the text 'Outcome of Inception Phase'. The main content area lists several bullet points under the heading:

- Initial requirements capture
- Cost Benefit Analysis
- Initial Risk Analysis
- Project scope definition
- Defining a candidate architecture
- Development of a disposable prototype
- Initial Use Case Model (10% - 20% complete)
- First pass Domain Model

At the bottom left, there are logos for IIT Kharagpur and NPTEL. At the bottom right, there is a video frame showing a man speaking.

Now, we will just quickly look at the outcome of the inception phase. We can say the outcome of inception phase are capturing the initial requirements, cost benefit analysis, risk analysis, project scope definition, defining the candidate architecture, development of the prototype, development of the initial use case model and the first pass domain model etc.

(Refer Slide Time: 03:54)

The slide has a yellow header bar containing the text 'Spiral Model'. The main content area lists several bullet points under the heading:

- Proposed by Boehm in 1988.
- Each loop of the spiral represents a phase of the software process:
 - the innermost loop might be concerned with system feasibility,
 - the next loop with system requirements definition,
 - the next one with system design, and so on.
- There are no fixed phases in this model, the phases shown in the figure are just examples.

At the bottom left, there are logos for IIT Kharagpur and NPTEL. At the bottom right, there is a video frame showing a man speaking.

Now, we will discuss about the spiral model. Spiral model is a rather old model compared to the unified process. Spiral model has been proposed by Boehm in 1988.

This model has features of incremental development and, also evolutionary development that means it is developed over several iterations or loops which are similar to increments. The innermost loop concerned with system feasibility, the next loop with requirement definition, then the next one with system design and so on. So, one thing that we can say is that in the spiral model each loop may not result in a deliverable software. Whereas, in the incremental model every increment actually leads to a deployable increment at the customer site.

Each loop is called as a phase here and there are no fixed phases in this model. The number of phases are determined by the project manager and the team members as the development proceeds.

(Refer Slide Time: 05:37)

The slide has a yellow background and a blue header bar. The title 'Spiral Model (CONT.)' is centered in the header. Below the title, there is a bulleted list of instructions:

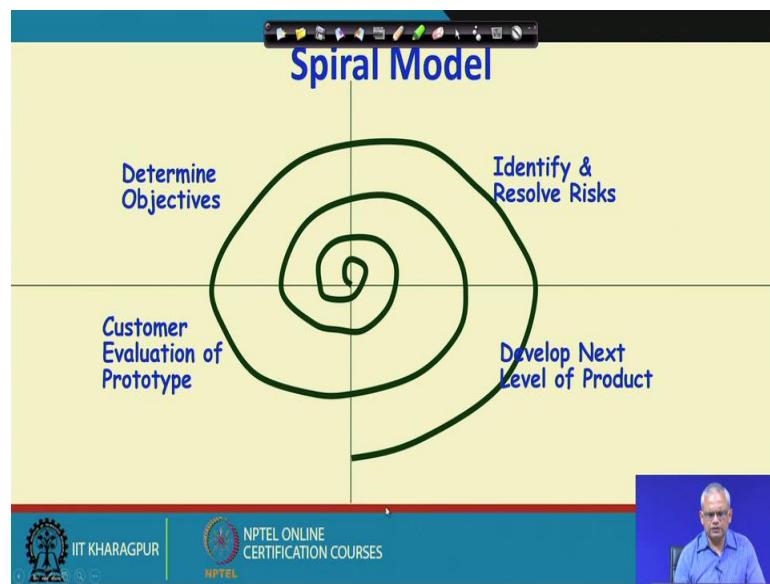
- The team must decide:
 - how to structure the project into phases.
- Start work using some generic model:
 - add extra phases
 - for specific projects or when problems are identified during a project.
- Each loop in the spiral is split into four sectors (quadrants).

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

The phases are decided by the team members and each phase some risk is identified. And, these are resolved using prototype. So, one of the very important issue with the spiral model is risk handling. In every phase the most important risk that is being faced by the project is identified and these are resolved by developing a prototype; just contrast this with the prototyping model where only one prototype is made before the start of the project. So, the risks that could be identified the beginning of the project can be resolved in the prototyping model whereas, in the spiral model as the project continues more and more risks are identified and these are resolved.

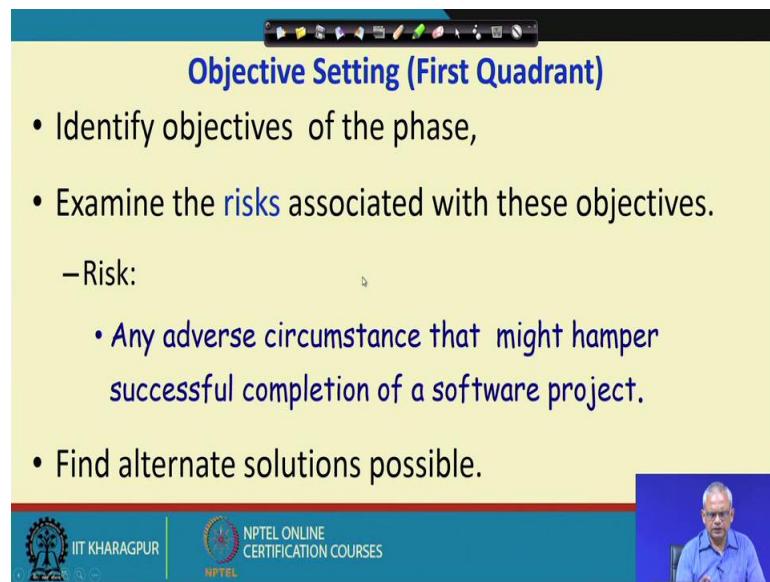
From the below figure, we can see each loop is actually split into 4 sectors.

(Refer Slide Time: 06:56)



In the spiral model, there are four quadrants. Each loop of the spiral model is called as a phase and over each loop some risk is identified. And, those risk are resolved through developing prototype.

(Refer Slide Time: 07:22)



After resolving all the risks the development occurs and finally, the customer site the customer evaluate the prototype. As we are mentioning that one of the important characteristics of the spiral model is risk handling. Risk is basically any adverse circumstance that might hamper the successful completion of the software project. For

example, whether the throughput will be sufficient to meet the customer requirement? in this case, the prototype is built for resolving the issue and alternate solutions may be tried out and the best one may be used.

(Refer Slide Time: 08:25)

The slide has a blue header bar with standard window controls. The main title 'Risk Assessment and Reduction (Second Quadrant)' is centered in a large blue font. Below the title is a bulleted list of points:

- For each identified project risk,
 - a detailed analysis is carried out.
- Steps are taken to reduce the risk.
- For example, if there is a risk that requirements are inappropriate:
 - A prototype system may be developed.

At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL Online Certification Courses. On the right side, there is a small video window showing a man speaking.

In the second quadrant, detailed analysis of the identified feature is carried out. and After that the risk is identified through building a prototype.

(Refer Slide Time: 08:45)

The slide has a blue header bar with standard window controls. The title 'Spiral Model (CONT.)' is centered in a large blue font. Below the title is a bulleted list of points:

- Development and Validation (**Third quadrant**):
 - develop and validate the next level of the product.
- Review and Planning (**Fourth quadrant**):
 - review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral:
 - progressively more complete version of the software gets built.

At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL Online Certification Courses. On the right side, there is a small video window showing a man speaking.

In the third quadrant, after the risk is resolved the development occurs and in the fourth quadrant review and planning occurs. So, customer feedback is obtained based on the

way the risk was handled and development was done. And, with each iteration around the spiral more and more complete version of the software gets built but one thing is that every spiral or every phase may not lead to a deployable software at the client site.

(Refer Slide Time: 09:21)

The slide has a yellow background with black text. A blue box on the right contains the title 'Spiral Model as a Meta Model'. The text lists features of the spiral model, including its relationship to other models and its evolutionary nature. It also mentions its uses, such as prototyping and retaining the step-wise approach of the waterfall model. The bottom of the slide features logos for IIT Kharagpur and NPTEL, along with a small video window showing a speaker.

- Subsumes all discussed models:
 - a single loop spiral represents waterfall model.
 - uses an evolutionary approach --
 - iterations over the spiral are evolutionary levels.
 - enables understanding and reacting to risks during each iteration along the spiral.
- Uses:
 - prototyping as a risk reduction mechanism
 - retains the step-wise approach of the waterfall model.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A spiral model is called as a Meta model, because it has features of the waterfall model, incremental model, evolutionary model, and so on. We can see that if we have a single loop of the spiral, then it's actually a waterfall model. And, then it has the features of the prototyping model, it has the features of the incremental and the evolutionary models.

Now, we will see agile development models which have come into the picture for last 2 decades or so.

(Refer Slide Time: 10:07)

What is Agile Software Development?

- **Agile:** Easily moved, light, nimble, active software processes
- **How agility achieved?**
 - Fitting the process to the project
 - Avoidance of things that waste time

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



The dictionary meaning of agile is fast development. But then question arise how fast development is achieved? We will see in the agile model is that anything that wastes time are avoided, and also any activities that are not required are eliminated. So, for a specific project any activities that are not required are eliminated. And, also any things that waste time is eliminated. But what things waste time? In the waterfall model we had seen that about 50 percent of the effort is spent on developing the documentation. And, some of this documentation are rarely used by anybody.

(Refer Slide Time: 11:38)

Agile Model

- To overcome the shortcomings of the waterfall model of development.
 - Proposed in mid-1990s
- The agile model was primarily designed:
 - To help projects to adapt to change requests
- In the agile model:
 - The requirements are decomposed into many small incremental parts that can be developed over one to four weeks each.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



In agile model one thing is that it produces very little documentation. One of the major focus here, is to facilitate the change requests from the customer and incorporate them efficiently. And, here it has features of the incremental and evolutionary model, just like the incremental model here these are the software is developed over increments and deployed at the client site. The time duration for each increment is about 1 to 4 weeks.

(Refer Slide Time: 12:35)

Ideology: Agile Manifesto

- Individuals and interactions over
 - process and tools
- Working Software over
 - comprehensive documentation
- Customer collaboration over
 - contract negotiation
- Responding to change over <http://www.agilemanifesto.org>
 - following a plan

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The Agile Manifesto was released towards the year 2000 and the Agile Manifesto is available at this website www.agilemanifesto.org. If, you look at the Agile Manifesto, it says that there are certain things that are important for the agile development. One is individual interactions are very important, these are much more important than the process itself or any tools that are used.

So, one of the things that is recognized here is that producing working code is much more important, than writing extensive documentation. Here, customer has to be involved in the project possibly by making some customer representatives part of the team. And, a contract negotiation made with the customer to make changes over following a plan.

(Refer Slide Time: 14:16)

Agile Methodologies

- XP
- Scrum
- Unified process
- Crystal
- DSDM
- Lean

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The agile is actually an umbrella term. Many development methodologies actually qualify as agile. The features of the Agile Manifesto are extreme programming or XP, scrum, even the unified process, crystal, DSDM, lean etc.

(Refer Slide Time: 14:47)

Agile Model: Principal Techniques

- **User stories:**
 - Simpler than use cases.
- **Metaphors:**
 - Based on user stories, developers propose a common vision of what is required.
- **Spike:**
 - Simple program to explore potential solutions.
- **Refactor:**
 - Restructure code without affecting behavior, improve efficiency, structure, etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The agile model has 4 important manifestos. Which emphasizes many techniques. One is that the requirement should be in the form of a user story, user story as the name implies its bit informal.

So, understand the requirement informally, these are simpler than use cases and the development can start based on the stories. And, then based on the customer feedback this can be refined.

It uses metaphors; metaphors are actually an overall design of the software; Based on the various user stories requirements an overall design is obtained which is the metaphor.

A spike is like a prototype; it is a program to explore the potential solutions and evaluate the alternatives. And, once the software works and the client agrees to that, after that design is put into it. It is refined made structured and some design is put into it without affecting the behavior of the code. It may improve efficiency, design structure et cetera.

(Refer Slide Time: 16:48)

- At a time, only one increment is planned, developed,
deployed at the customer site.
– **No long-term plans are made.**
- An iteration may not add significant functionality,
– But still a new release is invariably made at the end of each
iteration
– Delivered to the customer for regular use.

Agile Model: Nitty Gritty

This incremental model at a time only one increment is planned and focus on one increment at a time and that is deployed at the customer site. No long-term plans are made. Each iteration may add even a small amount of code but still after a time box an increment is made and that is deployed at the customer site. Sometimes the incremental feature may be very small, but still these are deployed.

(Refer Slide Time: 17:43)

Methodology

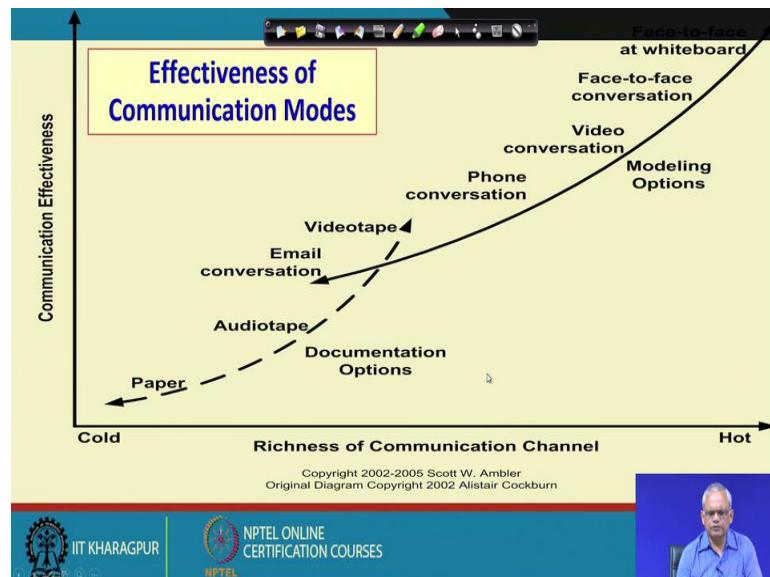
- Face-to-face communication favoured over written documents.
- To facilitate face-to-face communication,
 - Development team to share a single office space.
 - Team size is deliberately kept small (5-9 people)
 - This makes the agile model most suited to the development of small projects.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A video player window showing a man speaking.

One of the very important aspect of the agile methodology is face to face communication. The team members should communicate to each other rather than passing documents to each other. And, to facilitate this, the development team shares the same office space. Small team size (best 5 to 9 people) prefer for communication..

(Refer Slide Time: 18:34)



The evaluation of various communication modes reported by Scott Ambler, Alistair Cockburn. They evaluated various communication modes and found that the worst way to communicate is to pass around the paper. Something that may be still little better

maybe through audio tape, maybe email conversation is better, video tape may be slightly better, phone conversation is still better, video conversation like Skype may be better, but the best form of communication is face to face communication using a whiteboard.

So, for explaining anything using of whiteboard and face to face communication is the best medium. And, that's the reason why the agile model is recommends the face to face communication for successful development of software.

(Refer Slide Time: 19:57)

The slide has a yellow background and a blue header bar. The title 'Agile Model: Principles' is centered in the header. Below the title, there are two main bullet points:

- The primary measure of progress:
 - Incremental release of working software
- Important principles behind agile model:
 - Frequent delivery of versions --- once every few weeks.
 - Requirements change requests are easily accommodated.
 - Close cooperation between customers and developers.
 - Face-to-face communication among team members.

At the bottom of the slide, there is a footer bar with three logos: IIT Kharagpur, NPTEL Online Certification Courses, and a video thumbnail of a man speaking.

The software is developed in increments and deployed at the customer site. Once every few weeks customer feedback is obtained just like the incremental and evolutionary model and face to face communication is emphasized among team members.

(Refer Slide Time: 20:31)

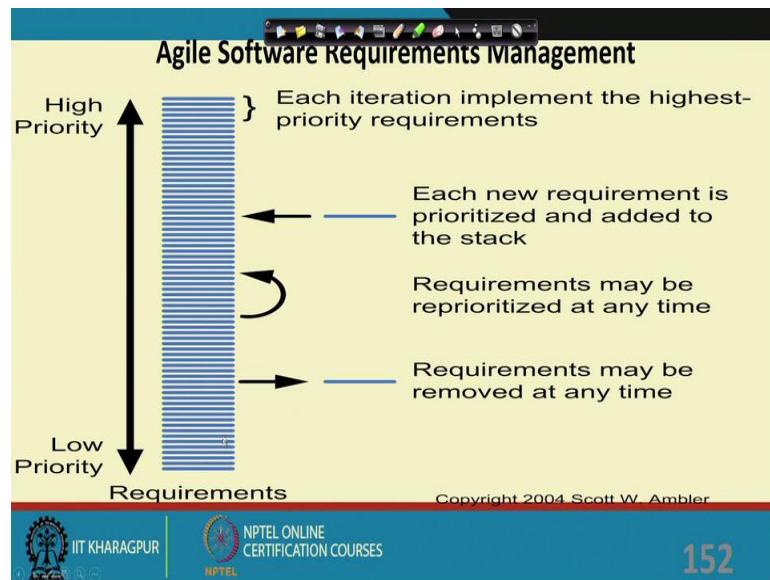
The slide has a yellow background and a blue header bar. The title 'Agile Documentation' is in the center of the header. Below the title is a bulleted list of characteristics:

- Travel light:
 - You need far less documentation than you think.
- Agile documents:
 - Are concise
 - Describe information that is less likely to change
 - Describe “good things to know”
 - Are sufficiently accurate, consistent, and detailed
- Valid reasons to document:
 - Project stakeholders require it
 - To define a contract model
 - To support communication with an external group
 - To think something through

At the bottom of the slide, there is a blue footer bar with three logos: IIT Kharagpur, NPTEL, and NPTEL Online Certification Courses. On the right side of the footer bar, there is a small video window showing a man speaking.

But, does agile development model require documentation to be produced? Yes, documentation can be produced which is very minimal, that is far less documentation than you think less than that is needed. All documents are to the point very small and the things that are not likely to change are very less likely to change are documented. Only those aspects will be documented which are somebody may be interested to know. The document should be accurate consistent and detailed. Before we document we must identify the important things which are going to be documented. Valid regional document can be that, it may be useful by the customer. For example, user manual may be to define a contact model. Maybe there is external group we want to get their suggestion and for them we want to prepare a document. And of course, we might document something just to think over it.

(Refer Slide Time: 22:09)



In agile model also the requirements they keep on coming as is any evolutionary model. Scott ambler, he describes, how the requirements are maintained? All the requirements are maintained in terms of their priority. Each time a requirement comes these are put in the appropriate place, maybe in the excel sheet as a user story. Each requirement is actually a user story. User stories at the top of excel are the high priority requirements. All these high priority requirements are the ones to be implemented first. Later some inserted requirements may be modified, some requirement may be moved on or load on the list, and some requirements may be deleted based on the customer feedback.

(Refer Slide Time: 23:33)



The agile model has many advantages as you had seen that it helps produce software. In least time, at least cost good quality software is produced and its able to accommodates customer requirements also. But then there are some problems that we have to keep in mind. The definition of the agile model is rather sketchy, several interpretations may be possible and therefore, high quality people skills are required. Long term plan, long term design et cetera are not made here only focus on increment. And, also since we keep on getting the customer feedback in this model, so it's harder to manage feature creep and customer expectations. And, also upfront it would be difficult to quantify the cost time and quality, because we do not know how many increments, what features exactly will be there.

(Refer Slide Time: 24:55)

Agile Model Shortcomings

- Derives agility through developing tacit knowledge within the team, rather than any formal document:
 - Can be misinterpreted...
 - External review difficult to get...
 - When project is complete, and team disperses, maintenance becomes difficult...

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL



The other difficulties are that lot of the knowledge is shared among the developers through verbal communication rather than any formal document. And therefore, after the developers disperse those knowledges may vanish.

And, also verbal communication can be misinterpreted sometime, external review can be difficult to get. When the project is complete and the team disperses that time maintenance may become difficult but towards the end of the project all the important aspects of the project is documented before the project team disperses.

(Refer Slide Time: 25:49)

The slide has a yellow header bar with the title 'Agile Model versus Iterative Waterfall Model'. The main content area contains a bulleted list comparing the two models:

- The waterfall model steps through in a planned sequence:
 - Requirements-capture, analysis, design, coding, and testing .
- Progress is measured in terms of delivered artefacts:
 - Requirement specifications, design documents, test plans, code reviews, etc.
- In contrast agile model sequences:
 - Delivery of working versions of a product in several increments.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video thumbnail of a speaker.

Now, let's quickly compare the agile model with various other models. How does the agile model compare with the iterative waterfall model? Waterfall model is a heavy weight process and also the requirements are needed to be identified upfront, and the development proceeds for a planned sequence of activities. The manager plans about when requirements gathering, analysis, design, coding and testing will complete. And, after each phase documents are produced and the progress is made measured in terms of the delivered artifacts. In contrast the agile model, the progress is measured in terms of the delivered software. And also, initial requirement capture, requirements specification et cetera is not there. No long term planner made only short term plan for one increment is made.

(Refer Slide Time: 27:10)

Agile Model versus Iterative Waterfall Model

- As regards to similarity:
 - We can say that Agile teams use the waterfall model on a small scale.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



But, do they have any similarity at all? we can say that the agile teams for every increment, they may use a waterfall model, in a small scale. So, over a two week or one-week period they might develop one increment and that they use waterfall model.

(Refer Slide Time: 27:37)

Agile versus RAD Model

- Agile model does not recommend developing prototypes:
 - Systematic development of each incremental feature is emphasized.
- In contrast:
 - RAD is based on designing quick-and-dirty prototypes, which are then refined into production quality code.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



How does the agile model compare with the RAD model?

The agile model as we discussed does not recommend developing prototypes and here every iteration or the increment is developed using a systematic technique. In the contrast, in the RAD model quick and dirty prototypes are produced and these are refined

into production quality code. But how does the agile model compared with exploratory program? because exploratory programming and agile have many similarities.

(Refer Slide Time: 28:17)

The slide has a yellow background and a blue header bar. The title 'Agile versus exploratory programming' is at the top. Below it, there are two main bullet points:

- **Similarity:**
 - Frequent re-evaluation of plans,
 - Emphasis on face-to-face communication,
 - Relatively sparse use of documents.
- Agile teams, however, do follow defined and disciplined processes and carry out rigorous designs:
 - This is in contrast to chaotic coding in exploratory programming

At the bottom left, there are logos for IIT Kharagpur and NPTEL. At the bottom right, there is a small video window showing a man speaking.

Both models have face to face communication, documents are discouraged, frequent revaluation of the plans and so on. But then in contrast to the exploratory model here has some systematic approach. It's developed every increment according to a plan. And, design, coding, all are according to increment we use waterfall model and these all are contrast to the chaotic coding in the exploratory style.

So, we are nearing the end of this lecture. We will stop here. And, in the next lecture we look at one of the agile development techniques, it's extreme program.

Thank you.

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

Lecture - 13
Extreme Programming and Scrum

Welcome to this lecture.

In the last lecture we looked at the agile development. In the changing software projects, we had seen that over the years a lot of changes have occurred to the software projects. In multiyear projects also lot of reuse and customization is being made and from product development, many projects are now service oriented projects.

In this context, the agile model has become extremely popular. In any software development organization, large number of projects carried out by the agile development practices. And, we had said that agile is actually an umbrella term, there are certain characteristics that all agile development projects have to follow. There are several methodologies which come under the agile umbrella. And two prominent methodologies here are the extreme programming and the scrum.

In this lecture we look at these 2 agile methodologies: the extreme programming and a scrub.

We will first look at the extreme programming also called as XP.

(Refer Slide Time: 02:08)

Extreme Programming Model

- Extreme programming (XP) was proposed by Kent Beck in 1999.
- The methodology got its name from the fact that:
 - Recommends taking the best practices to extreme levels.
 - If something is good, why not do it all the time.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 160

XP was proposed by Kent Beck in 1999. The name extreme implies that the best practices are taken to the extreme level. So, the main focus of this model is that whatever good works in projects why not put it to the maximum use.

So, the principle here is that if something works and it is good then why not maximize its use.

(Refer Slide Time: 03:08)

- If code review is good:
 - Always review --- pair programming
- If testing is good:
 - Continually write and execute test cases --- test-driven development
- If incremental development is good:
 - Come up with new increments every few days
- If simplicity is good:
 - Create the simplest design that will support only the currently required functionality.

Taking Good Practices to Extreme

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we will see some of the good practices and how these are taken to the extreme level. Research results suggests that code review is a very good practice. Code review is

a much better than testing, code review can eliminate bugs in the software which maintain cost effectively. And even code review can eliminate bugs which cannot be detected by testing or is very difficult to detect by testing.

So, it has been accepted by all developers, that code review is a good practice. The extreme programming proposes to take this code review to extreme level by pair programming. But then what is pair programming? In pair programming, the code is written by 2 programmers on one desk. So, on one computer there are 2 programmers, one programmer writes the code while the other programmer reviews that. And, then the interchange every half an hour or so one programmer writes and the other reviews. So, we can understand that they switch their place. So, two programmers have understanding of the code and they put their mind together and also review any mistakes.

The other good practice is testing. Testing is good, it makes software more reliable. To take testing to the extreme level the extreme programming suggests to write the test cases continuously. So, that is called as test-driven development. In test-driven development, even before the code is written the test cases are written. And, after the test cases are written, the code is written. And each time the code is written to some extent the test cases are run to see whether the code passes or not. If the code failed then the code is modified.

Incremental development is good and it eliminates lot of problems of the waterfall model. And therefore, extreme programming come up with a new increment every few days.

Simplicity is good because it produces less bugs, easy to maintain and therefore, the extreme programming says take the simplicity to the extreme level. And for that it says that only focus on what is needed now and do the simplest design for that. It says that focus on present and don't need to about future requirement. So, the extreme programming says do the best design under the present circumstance.

So, pair programming, test driven development, incremental development, simplicity of the design these are the important characteristics of the extreme program.

(Refer Slide Time: 07:28)

Taking to Extreme

- **If design is good,**
 - everybody will design daily (refactoring)
- **If architecture is important,**
 - everybody will work at defining and refining the architecture (metaphor)
- **If integration testing is important,**
 - build and integrate test several times a day (continuous integration)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The another thing that is taken to the extreme is designing. To take the design extreme, everybody will design daily and it called refactoring. Architecture overall design is also taken into extreme. The agile principle say that everybody will define and refine the metaphor. Integration testing is also important we had seen that in the waterfall model, the delay starts from integration testing. The schedule slippage initially starts at the integration testing and becomes worse after the integration testing. But here in the extreme programming, we must build every day and whatever we build in a day we must integrate all together. Integration done several times a day so that later surprises are not there.

(Refer Slide Time: 08:47)

4 Values

- **Communication:**
 - Enhance communication among team members and with the customers.
- **Simplicity:**
 - Build something simple that will work today rather than something that takes time and yet never used
 - May not pay attention for tomorrow
- **Feedback:**
 - System staying out of users is trouble waiting to happen
- **Courage:**
 - Don't hesitate to discard code

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now we will see four values of extreme programming. Just like any other agile model, enhanced communication among team members and also with the customer is required in XP. Another value is simplicity. Builds something simple that will work rather than making it extremely complicated. Feedback is another important value in XP. Take continuous customer feedback and the customer can be part of the team. Another value required is courage. It says don't hesitate to discard a bad code. It says if the code structure has become bad then discard it and write new code.

(Refer Slide Time: 09:29)

Best Practices

- **Coding:**
 - without code it is not possible to have a working system.
 - Utmost attention needs to be placed on coding.
- **Testing:**
 - Testing is the primary means for developing a fault-free product.
- **Listening:**
 - Careful listening to the customers is essential to develop a good quality product.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now we will discuss about the best practices of extreme programming. Coding is an important place in extreme programming and require most attention. Testing is important for developing fault-free product and also listening to the customer is essential to develop good quality product.

So, listen very carefully to the customer feedback.

(Refer Slide Time: 09:55)

The slide has a yellow header with the title 'Best Practices'. Below the title, there are two main bullet points: '• Designing:' and '• Feedback:'. Under 'Designing:', there are two sub-points: '– Without proper design, a system implementation becomes too complex' and '– The dependencies within the system become too numerous to comprehend.'. Under 'Feedback:', there is one point: '– Feedback is important in learning customer requirements.' At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video thumbnail of a person speaking.

Designing is also important practice. After the code works put design into it called as a refactoring. Feedback is important learning the exact customer requirement.

(Refer Slide Time: 10:07)

The slide has a yellow header with the title 'Extreme Programming Activities'. Below the title, there are two main bullet points: '• XP Planning' and '• XP Design'. Under 'XP Planning', there are four sub-points: '– Begins with the creation of “user stories”', '– Agile team assesses each story and assigns a cost', '– Stories are grouped to for a deliverable increment', and '– A commitment is made on delivery date'. Under 'XP Design', there are five sub-points: '– Follows the KIS principle', '– Encourage the use of CRC cards', '– For difficult design problems, suggests the creation of “spike solutions”—a design prototype', and '– Encourages “refactoring”—an iterative refinement of the internal program design'. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '166'.

The XP planning are only a short term. XP planning begins with the creation of user stories and each story or requirement is assigned a cost. These stories are grouped into deliverable increments.

XP Design is simple. XP design follows KIS principle and encourage use of CRC cards. And, whenever there is confusion about which is the best way to go, it suggests the creation of spike solution. And after a code works refactor and put design into it.

(Refer Slide Time: 11:10)

The slide has a yellow header bar with various icons. The main content area is titled 'Extreme Programming Activities' in a yellow box. It contains two sections: 'XP Coding' and 'XP Testing', each with a bulleted list of activities. The footer features the IIT Kharagpur logo, the NPTEL logo, and a photo of a speaker.

- **XP Coding**
 - Recommends the construction of unit test cases *before* coding commences (test-driven development)
 - Encourages “pair programming”
- **XP Testing**
 - All unit tests are executed daily
 - “Acceptance tests” are defined by the customer and executed to assess customer visible functionalities

Extreme Programming Activities

16

Extreme programming activities: coding and testing. The coding is by test driven development where first the test cases developed before any code is written. And, then write the code until it passes the test cases written. Pair programming is encouraged in XP coding activity. In XP testing, all software that is developed is tested daily and acceptance tests are defined and are executed to assess customer visible functionalities.

(Refer Slide Time: 11:46)

The slide has a yellow header bar with the title 'Full List of XP Practices'. Below it is a list of five numbered items:

1. **Planning** – determine scope of the next release by combining business priorities and technical estimates
2. **Small releases** – put a simple system into production, then release new versions in very short cycles
3. **Metaphor** – all development is guided by a simple shared story of how the whole system works
4. **Simple design** – system is to be designed as simple as possible
5. **Testing** – programmers continuously write and execute unit tests

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video frame showing a speaker.

Now, we will see all of the extreme programming practices. First is planning where only short-term plans are made. Then small releases which say release new versions in very short cycles. Then metaphor which depicts how the whole system works. Next is simple design and then testing which say continuously write and execute test cases.

(Refer Slide Time: 12:05)

The slide has a yellow header bar with the title 'Full List of XP Practices'. Below it is a list of four numbered items:

7. **Refactoring** – programmers continuously restructure the system without changing its behavior to remove duplication and simplify
8. **Pair-programming** – all production code is written with two programmers at one machine
9. **Collective ownership** – anyone can change any code anywhere in the system at any time.
10. **Continuous integration** – integrate and build the system many times a day – every time a task is completed.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video frame showing a speaker.

Refactoring, Pair-programming, collective ownership also apart of good XP practices. Pair programming say that all code needs to be written by two programmers at one machine. On the other hand, collective ownership says not only that two programmers,

but any programmer should be able to modify any other programmer's code. The XP practice Continuous integration says every day whatever code is written has to be integrated with the existing code.

(Refer Slide Time: 12:29)

Full List of XP Practices

- 11. 40-hour week** – work no more than 40 hours a week as a rule
- 12. On-site customer** – a user is a part of the team and available full-time to answer questions
- 13. Coding standards** – programmers write all code in accordance with rules emphasizing communication through the code

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

40-hours week practice says that work no more than 40 hours a week. On-site customer says customer must be part of the team and coding standards says that a standard to be followed for writing code.

As you can see that many of these are actually philosophical and little bit sketchy. And therefore, good quality manpower is needed for extreme programming or any other agile projects.

(Refer Slide Time: 13:07)

Emphasizes Test-Driven Development (TDD)

- Based on user story develop test cases
- Implement a quick and dirty feature every couple of days:
 - Get customer feedback
 - Alter if necessary
 - Refactor
- Take up next feature

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

In test driven development any increment starts with a user story, the first thing is to develop test cases based on the user story. And, then develop the software and get the customer feedback. Based on the customer feedback alter necessary and then put design into the code. Make the code refine it into good quality code and then take up the next feature.

(Refer Slide Time: 13:44)

Project Characteristics that Suggest Suitability of Extreme Programming

- Projects involving new technology or research projects.
 - In this case, the requirements change rapidly and unforeseen technical problems need to be resolved.
- Small projects:
 - These are easily developed using extreme programming.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Extreme programming is successful in small projects and also the projects which are of challenging nature.

(Refer Slide Time: 14:00)

Practice Questions

- What are the stages of iterative waterfall model?
- What are the disadvantages of the iterative waterfall model?
- Why has agile model become so popular?
- What difficulties might be faced if no life cycle model is followed for a certain large project?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Now, having looked at several development models including some of the agile models let's try to check our understanding with few questions.

Can you identify what are the stages of the waterfall model? It is a very, very basic question. The stages are: feasibility study, requirement analysis and specification, design, coding, unit testing, testing and maintenance.

Now, next question is what are the disadvantages of the iterative waterfall model? If, you recollect one major disadvantage is with respect to accommodation of change requests the customer is required to identify all the requirements upfront. The customer has to visualize what the software will be and it is extremely tough job. It's a hard to identify all the requirements that are required at the beginning and invariably 40 percent or 50 percent of the requirements changed during the development time itself. And therefore, the iterative waterfall model is very inflexible. The second problem with the iterative waterfall model is that it's a heavy weight model emphasizes production of documents, rather than production of code increments. The third disadvantage may be that it doesn't allow overlapping of the phases and so on. We have discussed several disadvantages of iterative waterfall model please review them.

Now, the next question is why has agile model become so popular? We discussed that the software projects have changed now a days. The projects are now a short duration, lot of code reuse been made and now the customer satisfaction is very important. Now, need

to deploy incremental software at the customer site and so on. So, all these can be achievable in agile model. So that's why agile model become popular.

What difficulties might be faced if no lifecycle model is followed for a certain large project?

The answer would be that the project may not complete there will be too much of cost overrun, it will take too much time to develop or it may not complete at all, the quality of the software will be poor and so on.

(Refer Slide Time: 17:46)

Suggest Suitable Life Cycle Model

- A software for an academic institution to automate its:
 - Course registration and grading
 - Fee collection
 - Staff salary
 - Purchase and store inventory
- The software would be developed by tailoring a similar software that was developed for another educational institution:
 - 70% reuse
 - 10% new code and 20% modification

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

NPTEL

Now, suggest a suitable lifecycle model for a software, which is to be developed for an academic institution to automate activities like course registration, grading, fee collection, staff salary, purchase and in store inventory. The software would be developed by tailoring a similar software that was developed for another educational institution. It is expected that there will be 70 percent reuse of the code, 10 percent of new code to be written and 20 percent modification will be done.

Waterfall model is clearly unsuitable for this, because it will involve tailoring a similar software. So, an agile model can be suitable answer for this.

(Refer Slide Time: 18:43)

Practice Questions

- Which types of risks can be better handled using the spiral model compared to the prototyping model?
- Which type of process model is suitable for the following projects:
 - A customization software
 - A payroll software for contract employees that would be add on to an existing payroll software

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Next practice question is which types of risk can be better handled using the spiral model compared to prototyping model?

In prototyping model, the risks which can be identified upfront those can be handled, but in the spiral model the risks which appear after the development starts, they can be handled better.

which type of process model is suitable for the following projects?

One project is customization software. For this project agile model is suitable.

Next project is payroll software for contract employees that would be an add on to an existing payroll software. So, there is basically a small increment to a existing software so, this can be also an agile model.

(Refer Slide Time: 19:36)

Practice Questions

- Which lifecycle model would you select for the following project which has been awarded to us by a mobile phone vendor:
 - A new mobile operating system by upgrading the existing operating system
 - Needs to work well efficiently with 4G systems
 - Power usage minimization
 - Directly upload backup data on a cloud infrastructure maintained by the mobile phone vendor

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Another practice question: Which lifecycle model would you select for the following project which has been awarded to us by mobile phone vendor?

In mobile project, new mobile operating system, upgrading the existing operating system need to done and all these needs to work well efficiently with 4G systems. In this type of project, needs to do power uses minimization, directly upload backup data on cloud infrastructure maintained by a mobile phone vendor etc. So, there is lot of challenge in this project and it is new technology and therefore, agile model will be suitable.

Now, let's look at one more agile model which is also very popular: the scrum.

(Refer Slide Time: 20:20)

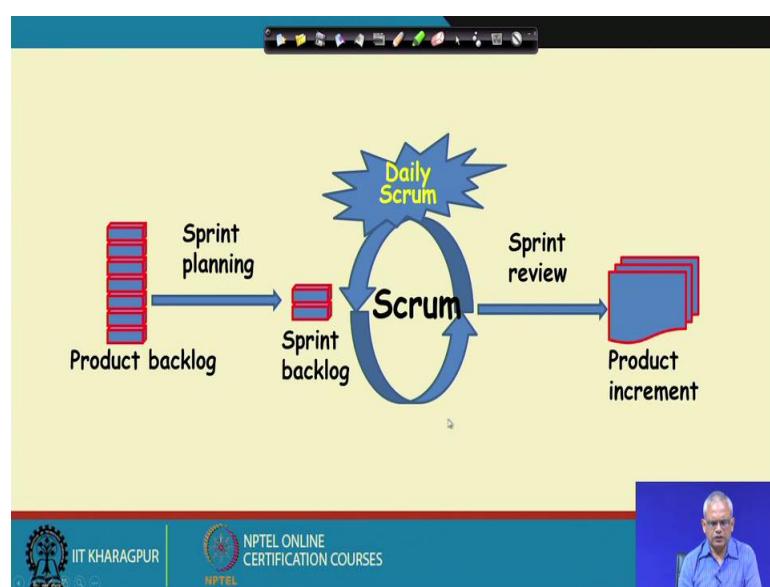
Scrum: Characteristics

- Self-organizing teams
- Product progresses in a series of month-long **sprints**
- Requirements are captured as items in a list of **product backlog**
- One of the agile processes

The slide has a yellow header and a blue footer. The footer contains the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. A small video window in the bottom right corner shows a man speaking.

The scrum model has all the characteristics of an agile model but it has its own characteristics over the agile model. One of its own characteristics is self-organizing teams. The self-organizing teams means that the small team they decide who will do which work. Here, the terminologies are also bit different. Here, the increments are about a month and these increments are called as sprints. And, the requirements are captured as product backlog. Product backlog is another terminology with this the software requirements are captured continuously. This product backlog is the crux of the scrum model.

(Refer Slide Time: 21:26)



Product backlog actually store user stories. The product backlog keeps on arising and in the sprint planning meeting, one of the product backlog that is user story is selected. Some of the user stories are chosen for the next sprint. A sprint is about a month long.

Every day the developers meet that call daily scrum. They identify if any problem is being faced. And, then they provide any solution and then after the daily 10-15-minute meeting they go to do their work. Once the sprint backlog completed then there is a sprint review meeting. After the sprint review meeting is successful over, the product increment is deployed at the customer site.

(Refer Slide Time: 23:04)

Sprint

- Scrum projects progress in a series of “sprints”
 - Analogous to XP iterations or time boxes
 - Target duration is one month
- **Software increment is designed, coded, and tested during the sprint**
- **No changes entertained during a sprint**

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES
NPTEL



Scrum project progresses in a series of “sprints”. These are similar to XP iterations or time boxes and typically about a month is the duration. In sprint, software increment is designed, coded, and tested. One thing to remember is that once the sprint backlog is formed then during the sprint (one month) no changes entertained. The idea here is that, when the short-term plan is made and the team is working on it, we should not disturb it otherwise it will never converge.

(Refer Slide Time: 24:01)

Scrum Framework

- Roles : Product Owner, ScrumMaster, Team
- Ceremonies : Sprint Planning, Sprint Review, Sprint Retrospective, and Daily Scrum Meeting
- Artifacts : Product Backlog, Sprint Backlog, and Burndown Chart

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

There are some terminologies which constitute the scrum framework. There are 3 roles in the team, one is the product owner. The product owner is either a customer representative or one of the development team members. Another role is scrum master who is actually the project manager and then the third number role is team members. There are some meetings, these are called here as ceremonies: the sprint planning meeting, sprint review meeting, sprint retrospective meeting, and daily scrum meeting.

There are some documents which are used: the product backlog, which has all the requirements; The sprint backlog, which is basically the requirements per one sprint; and the Burndown charts, which capture how far the project has progressed.

(Refer Slide Time: 25:14)

The slide has a yellow header section containing the title and a blue footer section with logos and course information.

Key Roles and Responsibilities in Scrum Process

- **Product Owner**
 - Acts on behalf of customers to represent their interests.
- **Development Team**
 - Team of five-nine people with cross-functional skill sets.
- **Scrum Master (aka Project Manager)**
 - Facilitates scrum process and resolves impediments at the team and organization level by acting as a buffer between the team and outside interference.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** | 182

Now, we will see key roles and responsibilities in more detail.

The product owner acts on behalf of the customer he may be a customer representative or maybe a team member who acts like the customer. The development team about 5 to 9 people with cross functional skills like testing GUI, database development and so on. The scrum master is the project manager who facilitates the scrum process and resolves any problems that the team may be face. And, also interacts with the customer and also the top management and shields the team from outside influence.

(Refer Slide Time: 26:02)

The slide has a yellow header section containing the title and a blue footer section with logos and course information. A video player window is visible in the bottom right corner.

Product Owner

- Defines the features of the product
- Decide on release date and content
- Prioritize features according to market value
- Adjust features and priority every iteration, as needed
- Accept or reject work results.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

So, the product owner acts as a customer and defines the features of the product, decides on release date, prioritizes the features according to requirement, adjust the features and priority every iteration and then at the end of a sprint product owner either accepts or rejects the result.

(Refer Slide Time: 26:30)

The Scrum Master

- Represents management to the project
- Removes impediments
- Ensure that the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Shield the team from external interferences

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

The scrum master is actually the project manager. Scrum master removes any problems that faces team member and ensures that the team is fully functional. Scrum master also enables cooperation across all roles and shield the team from external interfaces.

(Refer Slide Time: 26:53)

Scrum Team

- Typically 5-10 people
- Cross-functional
 - QA, Programmers, UI Designers, etc.
- Teams are self-organizing
- Membership can change only between sprints

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

The scrum team members are actually 5 to 10 people. They have expertise in various cross functional areas like quality assurance, coding, user interface design, database et cetera. The team members are self-organizing they decide who can do the which work best and then they decide among themselves. And, in between one sprint membership cannot be not changed, new members can be added only at the end of a sprint.

We are towards the end of this lecture; we will stop here. And we will continue from this point in the next lecture.

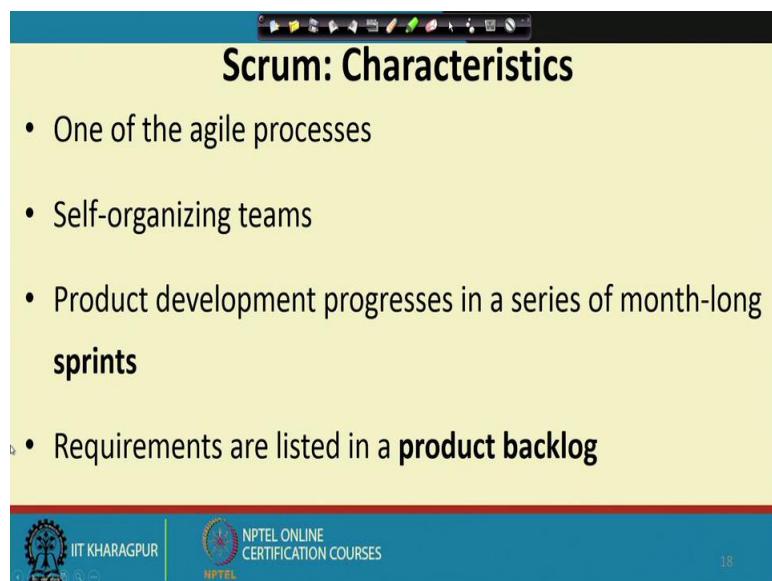
Thank you.

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

Lecture – 14
Scrum

Welcome to this lecture. Till now we had looked at some introductory topics and then we had looked at life cycle models. And, then we said that for last two decades or so, the agile models have become very important. Agile models are used extensively in the industry and the main reason for that is that the project characteristics have changed. In the old times, all programs are developed from scratch and they were multiyear projects. That is the reason the waterfall-based models were very popular, but then slowly the project characteristics have changes and later the service-oriented projects have become prevalent, and that's one of the reasons that the agile modes are being extensively used. Agile is actually an umbrella term which denotes some characteristics of the life cycle. There are also several life cycle models, which satisfies this criterion. In last lecture we are looked at extreme programming and in today's lecture we will discuss about the scrum. Scrum is a very popular agile model which is being used.

(Refer Slide Time: 02:01)



The slide has a yellow background and a blue header bar. The title 'Scrum: Characteristics' is centered in the header. Below the title is a bulleted list of five items:

- One of the agile processes
- Self-organizing teams
- Product development progresses in a series of month-long **sprints**
- Requirements are listed in a **product backlog**

At the bottom of the slide, there is a red footer bar containing the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '18'.

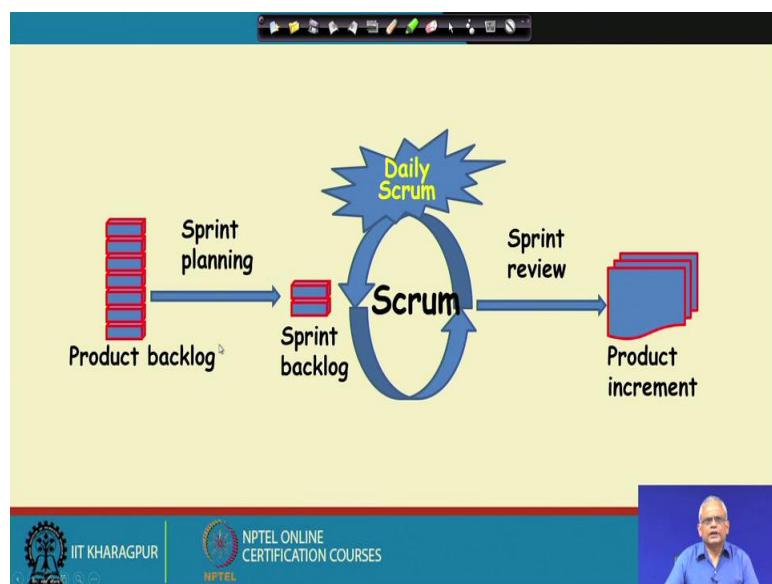
Scrum consists of small teams which are self-organizing; by self-organizing we mean that the team members by themselves choose their own roles activities. There is no chief

programmer or somebody who assigns to others activities, but they self-organized themselves and decide who will do what part.

There are several technical terms and concepts we need to look at here. One of the main principles is the sprint. A sprint is something like an iteration or a time box, used in other agile models like extreme programming. The development work is done over several sprints and each sprint is typically about a month long. As the software gets developed the requirement get build and starting with some initial requirements.

Later, the initial requirements are refined and more requirements are added in a list, called as a product backlog. Typically, in an excel file the requirements are kept track of. Requirements are actually called as user stories but in other agile models, which are simpler form of functional requirements.

(Refer Slide Time: 03:55)



The above diagram characterizes, the main principles in the scrum. The product backlog is the place where all the requirements are maintained in a excel file and each requirement is called as a user story. We know that the user story is an informal description of a requirement. There is a sprint planning meeting to decides which product backlog and which stories to take up next and the selected user stories in planning meeting are taken up in the next sprint.

The selected user stories are kept in the sprint backlog for about three to four weeks. Every day the team members meet and work together to complete the sprint backlog. This type of meet is called as the daily scrum meeting. In the daily scrum meeting they discuss what they achieved previously, and what their next plan and they also discussed about difficulties they faced.

And, at the end of the scrum, the sprint backlog would have got exhausted and an increment of the software would have got built. These all works are reviewed in a sprint review meeting and the product increment that gets approved in the sprint review meeting is get installed at the client side. This cycle continues until all the product backlog gets exhausted. One point to remember is that the product backlog is dynamic in nature. As the work progresses more user stories are put into the product backlog and some user stories may get deleted, because of requirements change.

(Refer Slide Time: 06:21)

Sprint

- Scrum projects progress in a series of “sprints”
 - Analogous to XP iterations or time boxes
 - Target duration is one month
- **Software increment is designed, coded, and tested during a sprint**
- **No changes entertained during a sprint**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL



In an iteration or a time box usually in one month, one sprint software designed, coded and tested and no changes are entertained during a sprint. So, once the product backlog items that are user stories have been selected that development starts and the main idea here is that once the development starts, the developer should not be disturbed, otherwise it will never converge. So, we can say that as a rule once the features that are most accepted or stable have been selected these are developed during a sprint.

(Refer Slide Time: 07:33)

Sprint

- Fundamental process flow of Scrum
- A month-long iteration, during which an incremental product functionality completed
- NO outside influence can interfere with the Scrum team during the Sprint
- Each day begins with the Daily Scrum Meeting

The sprint is possibly the most fundamental process flow. It is a month long. At the end of the sprint some working software comes out, which is the incremental product functionality. During the sprint, no outside influence is allowed; that means, the scrum backlog which contains the user stories to be completed during the sprint, will be frozen. And, during the sprint each day starts with a daily scrum meeting, where a review is done about what was achieved previously and what to do next and are there any obstacles that any team member is facing.

(Refer Slide Time: 08:30)

Scrum Framework

- **Roles :** Product Owner, ScrumMaster, Team
- **Ceremonies :** Sprint Planning, Sprint Review, Sprint Retrospective, and Daily Scrum Meeting
- **Artifacts :** Product Backlog, Sprint Backlog, and Burndown Chart

These are some of the terminologies in the scrum framework. One of the terminology is the roles. All the team members have some roles. One of the team members is the product owner. Another, team member is the scrum master and then there is the team members, who will see what these roles mean and what are their responsibilities?

Another terminology is sprint planning ceremony. In the sprint planning ceremony, the user stories to be taken up for the next sprint from the product backlog is decided. Sprint review ceremony is done after a sprint is complete and to review that the incremental software that has got developed is working fine or not. Another terminology is the daily scrum meeting. The daily scrum meeting as we said that it is done every day in the morning during sprint. In daily scrum meeting, the team members just check what was done yesterday and what will be done today and are there any obstacles.

There are some artifacts that are maintained during sprint. One is the product backlog which contains the user stories. Another is the sprint backlog which contain the selected user stories from the product backlog that will be completed during the sprint. And, then there are various types of burn down charts which are useful in monitoring the progress of the development.

(Refer Slide Time: 10:24)

Key Roles and Responsibilities in Scrum Process

- **Product Owner**
 - Acts on behalf of customers to represent their interests.
- **Development Team**
 - Team of five to nine people with cross-functional skill sets.
- **Scrum Master (aka Project Manager)**
 - Facilitates scrum process and resolves impediments and acts as a buffer between the team and outside interference.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's look at the different roles of the team member; one is the product owner. Product owner has very important role. The product owner has the customer perspective and represents customer interest. He can be a member who is a part of the customer as

well as organization. He understands the customer requirements well, interacts with the customer. And, then there are the development team members which are five to nine people with cross functional skills, like coding, design, testing, quality assurance and so on. And, then one of the team member is the scrum master. The scrum master is also known as the project manager.

The scrum master is the management representative in the team. The responsibilities of the scrum master are facilitating development, analysis obstacles that are faced and talk to the management facilitate. The scrum master also acts as a buffer between the team and outside influence.

(Refer Slide Time: 11:54)

Product Owner

- Defines the features of the product
- Decides on release date and content
- Prioritizes new features
- Adjusts features and priority every iteration, as needed
- Accepts or rejects work results.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's look at the responsibilities of the different roles. The product owner as we said, is the member of customer as well as organization. The product owner understands the software very well. Some of the responsibilities of product owner are defines the features of the product, decides on the release date, prioritizes new features, adjust features and define priority in every iteration, and accepts or rejects the work result.

(Refer Slide Time: 12:39)

The Scrum Master

- Represents management
- Removes impediments
- Ensures that the team is fully functional and productive
- Shield the team from external interferences

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Now we will see the responsibilities of the scrum master. The scrum master as we said is a project manager, is the management representative in the team. The responsibilities of a scrum master are removes any obstacle that the team facing, hardware not working, network issues et cetera et cetera. The scrum master ensures that the team is fully functional and productive, shields the team from external interference, acts as a buffer between the management and the team and also with the customer.

(Refer Slide Time: 13:15)

Scrum Team

- Typically 5-10 people
- Cross-functional
 - QA, Programmers, UI Designers, etc.
- Teams are self-organizing
- Membership can change only between sprints

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



The scrum team typically consist of five to ten people. They have expertise in various areas like quality assurance, programming, user interface, design, testing, and so on. The team members are self-organizing they select their own roles the one that they can do best. In the scrum team, sometimes members may leave or new members may join, but it is ensured that when the sprint is in progress membership change is not allowed, otherwise the development work will get hampered.

(Refer Slide Time: 13:55)

The image shows a presentation slide titled "Ceremonies". The slide content is as follows:

- Sprint Planning Meeting
- Daily Scrum
- Sprint Review Meeting

The slide has a yellow background and a dark blue header bar. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

There are three main ceremonies: one is the sprint planning meeting, where the spring backlog is selected from the product backlog. Another one is the daily scrum; it is a daily meeting. In daily scrum the team members meet to review what each member did previous day? And, what we will be achieved today and any obstacles that they are facing? The last one is the sprint review meeting. Which is conducted at the end of the sprint to check whether the developed increment is alright.

(Refer Slide Time: 14:31)

Sprint Planning

- Goal is to produce Sprint Backlog
- Product owner works with the Team to negotiate what Backlog Items the Team will work on in order to meet Release Goals
- Scrum Master ensures Team agrees to realistic goals

The sprint planning produces the sprint backlog from the product backlog. Here the product owner who understands the customer requirement well and the team member they decide which user story to take up next.

Obviously, the product owner would like to select those stories which are most value adding to the customer. And, the team members will like to take up those stories which will require least cost of development. But the scrum master, ensures during a sprint planning meeting whether the selected sprint backlog to be completed in one month or not. The team agrees to realistic goals otherwise it will put undue pressure in the team or they may not have enough work.

(Refer Slide Time: 15:38)

The slide has a yellow header with the text "Daily Scrum". Below the header is a bulleted list of characteristics:

- Daily
- 15-minutes
- Stand-up meeting
- Not for problem solving
- Three questions:
 1. What did you do yesterday
 2. What will you do today?
 3. What obstacles are in your way?

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video feed of a speaker.

The daily scrum meeting is at morning everyday is a very small meeting about fifteen minutes. It's a stand-up meeting, to indicate that they just need to discuss and they don't allow to become it long meeting. It's not a problem-solving meeting. It is just to take status of what was done yesterday? What they are planning to complete today? And are they facing any obstacles? And, this is the one of specific concern for the project manager, the scrum master, who would try to remove the obstacle that are been faced.

(Refer Slide Time: 16:20)

The slide has a yellow header with the text "Daily Scrum". Below the header is a bulleted list of characteristics:

- Is NOT a problem solving session
- Is NOT a way to collect information about WHO is behind the schedule
- Is a meeting in which team members make informal commitments to each other and to the Scrum Master
- Is a good way for a Scrum Master to track the progress of the Team

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video feed of a speaker.

The daily scrum is not really a problem-solving session as we said it is just for collecting information and removing any obstacles. Here the team members make informal commitments to each other and to the scrum master and this is the way that the scrum master can track the progress of the team.

(Refer Slide Time: 16:44)

The slide has a yellow header bar with a title and a blue footer bar with logos and text. The title is 'Sprint Review Meeting' in a yellow box. The main content is a bulleted list:

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features
- Informal
 - 2-hour prep time rule
- Participants
 - Customers
 - Management
 - Product Owner
 - Other teammates

At the bottom left is the IIT Kharagpur logo and text 'IIT KHARAGPUR'. At the bottom center is the NPTEL logo and text 'NPTEL ONLINE CERTIFICATION COURSES'. At the bottom right is a small video frame showing a man speaking.

The sprint review meeting is done at the end of the sprint. In the sprint review meeting, the team presents the increment that has been completed, typically in the form of a demonstration. It's an informal meeting just two hours preparation should be enough and the participants are typically customer representatives, management representatives, product owner and other team members.

(Refer Slide Time: 17:16)

Product Backlog

- A list of all desired work on the project
 - Usually a combination of
 - **story-based work** (“allow user to search and replace”)
 - **task-based work** (“improve exception handling”)
- List is prioritized by the Product Owner.

The product backlog, contains all the work that needs to be completed towards the development. There are two types of entries here these are typically maintained in a excel file: one is the story based which are basically something like functional requirements simpler form of functional requirements like, allow the user to search and replace or allow the user to create new book entries and so on. Another type of entry in the product backlog are task based, these are not really functional requirement, but small items that might have to be completed which are not yet been done. For example, improve the exception handling facilities or refined the graphical interface and so on. And, once the entries have been populated in the product backlog is prioritized by the product owner in continuous basis.

(Refer Slide Time: 18:35)

Product Backlog

- Requirements for a system, expressed as a prioritized list of Backlog Items
 - Managed and owned by Product Owner
 - Spreadsheet (typically)

The slide has a yellow background and a blue footer bar. The footer bar contains the IIT Kharagpur logo, the NPTEL logo, and a video camera icon. On the right side of the slide, there is a small video window showing a man speaking.

The product backlog is typically maintained by the product owner and it's typically a spread sheet. Different priorities are high priority, very high priority, medium priority and so on.

(Refer Slide Time: 18:47)

Sample Product Backlog

	Item #	Description	Est	By
Very High	1	Finish database versioning	16	KH
	2	Get rid of unneeded shared Java in database	8	KH
	3	- Add licensing	-	-
	4	- Concurrent user licensing	16	TG
	5	- Demo / Eval licensing	16	TG
	6	- Analysis Manager	-	-
	7	- File formats we support are out of date	160	TG
8	- Round-trip Analyses	250	MC	
High	9	- Enforce unique names	-	-
	10	- In analysis application	24	KH
	11	- In imports	24	AM
	12	- Admin Program	-	-
	13	- Delete users	4	JM
	14	- Analysis Manager	-	-
	15	- When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	16	- Orphans	-	-
	17	- Support for wildcards when searching	16	T&A
	18	- Sorting of number attributes to handle negative numbers	16	T&A
	19	- Horizontal scrolling	12	T&A
	20	- Population Genetics	-	-
	21	- Frequency Manager	400	T&M
	22	- Disease Manager	400	T&M
Medium	23	- Additional Editors (which ones)	240	T&M
	24	- Study Variable Manager	240	T&M
	25	- Haplotypes	320	T&M
26	- Add icons for v1.1 or 2.0	-	-	
27	- Pedigree Manager	-	-	
28	- Validate Derived kindred	4	KH	

The slide has a yellow background and a blue footer bar. The footer bar contains the IIT Kharagpur logo, the NPTEL logo, and a video camera icon. On the right side of the slide, there is a small video window showing a man speaking.

(Refer Slide Time: 18:55)

The slide has a yellow header with the title 'Sprint Backlog'. Below the title is a bulleted list of characteristics of a Sprint Backlog:

- A subset of Product Backlog Items, which define the work for a Sprint
 - Created by Team members
 - Each Item has its own status
 - Updated daily

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

The sprint backlog are the subset of the backlog items which are selected for the next sprint. Later, some items can get added in sprint backlog because the team members may find that some items need to be completed for example, the database connectivity has to be completed and so on.

(Refer Slide Time: 19:30)

The slide has a yellow header with the title 'Sprint Backlog during the Sprint'. Below the title is a bulleted list of changes that occur during a sprint:

- Changes occur:
 - Team adds new tasks whenever they need to in order to meet the Sprint Goal
 - Team can remove unnecessary tasks
 - But: Sprint Backlog is only updated by the team
- Estimates are updated whenever there's new information

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

So, they updated it daily. As we mention that the sprint backlog changes during the development. The team members may add new entries, they may also remove some

unnecessary tasks. Unlike the product backlog which is maintained by the product owner, the sprint backlog is maintained by the team.

(Refer Slide Time: 20:04)

Burn down Charts

- Are used to represent “work done”.
- Are remarkably simple but effective Information disseminators
- Three Types:
 - Sprint Burn down Chart (progress of the Sprint)
 - Release Burn down Chart (progress of release)
 - Product Burn down chart (progress of the Product)

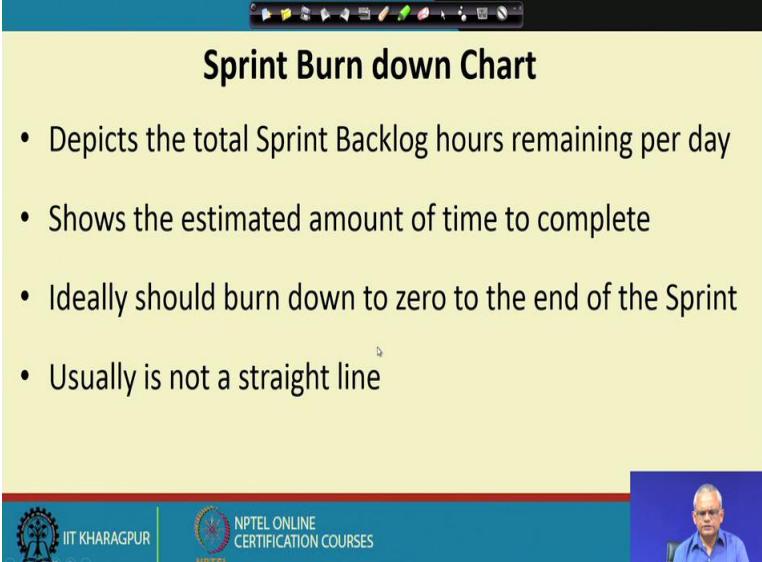
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

There are some charts to keep track of the progress of the work, these are very simple charts, but these very effective. Three main types of charts are used: the sprint Burn down chart, which is how much progress has been achieved during the sprint? Release Burn down chart how much progress has been achieved till the next release? Product Burn down chart this is for the entire work to complete how much progress has been achieved. Now we will look on these charts in detail.

(Refer Slide Time: 20:38)

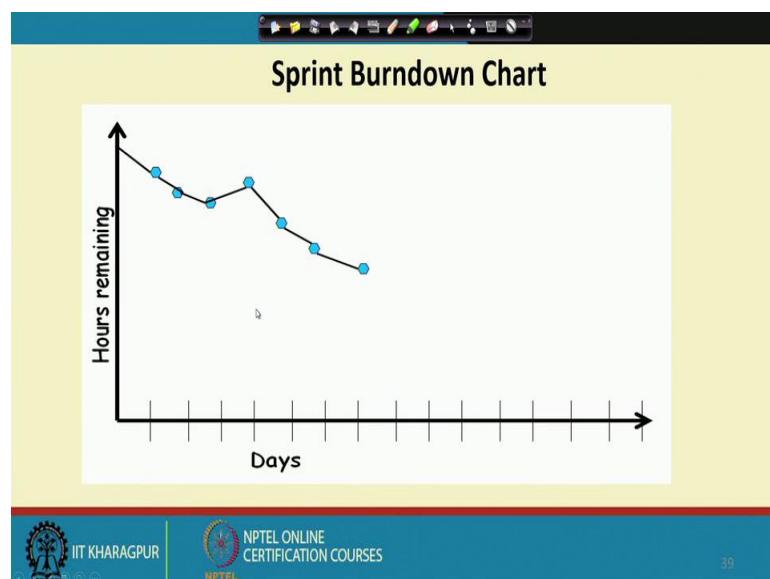
Sprint Burn down Chart

- Depicts the total Sprint Backlog hours remaining per day
- Shows the estimated amount of time to complete
- Ideally should burn down to zero to the end of the Sprint
- Usually is not a straight line



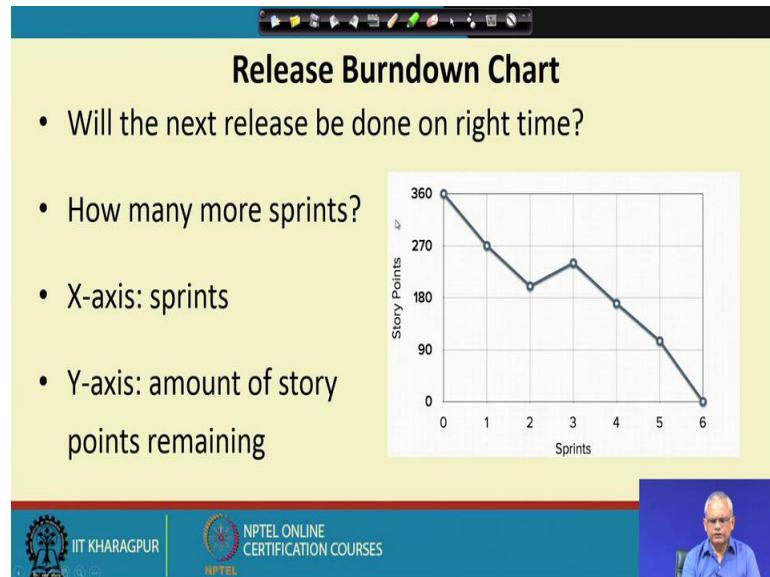
The first one is the sprint Burn down chart. It captures how much progress has been achieved the current sprint or more accurately how many hours are remaining for the sprint to complete? As, you will see the diagram the chart shows the estimated amount of time to complete, and at the end of the sprint the number of hours remaining will burn down to zero. Typically, it's not a straight line, because sometimes the work progress is very fast sometimes there are obstacles not much progress and so on.

(Refer Slide Time: 21:32)



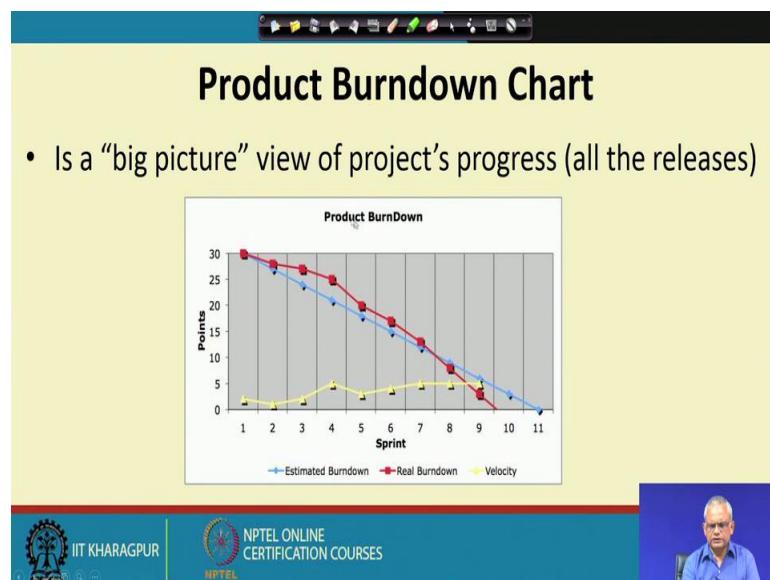
In above diagram, we can see a burndown chart and as the days progress here, the hours remaining comes down. Every day, the chart get updated.

(Refer Slide Time: 21:45)



The next one is the release burndown chart which shows that for the next release how much work is remaining? Each release can consist of many sprints and as each sprint complete this gets updated.

(Refer Slide Time: 22:07)



The last one is the product burndown chart it denotes how much work remaining until the development completes? So, we have the estimation here that the velocity with how quickly these are being burndown?

(Refer Slide Time: 22:26)

The slide has a yellow header with the title "Scalability of Scrum". Below the title is a bulleted list:

- A typical Scrum team is 6-10 people
- Jeff Sutherland proposed and experimented with up to over 800 people
- "Scrum of Scrums" also called "Meta-Scrum"

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with a small video thumbnail of a man speaking.

Scrum is typically design for small teams. But some people tried it for large projects also. For example, Jeff Sutherland reported that he has experimented it with over 800 people, which is called the scrum of scrums. The work is divided among many scrum teams the product owners, the scrum masters meet and they decide on the features that will be done on each team and it is also called as a meta scrum.

We have been looking at various life cycle models starting with very intuitive classical model, classical waterfall model and then various waterfall-based models. Then we discuss about the evolutionary model, the incremental models, and some models based on those principles and then we were focusing on the agile models. Now, let's start our discussion on the requirements analysis and specification.

(Refer Slide Time: 23:58)

What are Requirements?

- A Requirement is:
 - **A capability or condition required from the system.**
- What is involved in requirements analysis and specification?
 - **Determine what is expected by the client from the system. (Gather and Analyze)**
 - **Document those in a form that is clear to the client as well as to the development team members. (Document)**

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

This is an important phase in software development. Before we start discussing requirements analysis and specification. First let's try to answer the question that what is a requirement? A requirement is defined as a capability or condition required from the system which to be developed. A system typically consists of many capabilities and conditions. For example, if you are developing a library software, then the capability is maybe we can create books, we can create members, we can issue books, return books and the conditions may be that to be a web-based software and so on.

Let see in more detail, but then we can define roughly that a requirement is a capability or a condition that is required from a system. A capability is a facility that the software provides to the users. For example, the users can issue book, return book et cetera. Each of that is a capability of the software.

And, condition is not really a facility provided, but then it's more general purpose for example, it runs on, it's a web-based software and so on. As, we proceed we will be able to understand what are the capabilities? How to identify? And, what are the conditions that are required from the system?

Now, let see how to go about doing the requirements analysis and specification? What are the main activities that are involved to carry out requirement analysis and specification? The first work that needs to be done is to understand, what really the customer has in mind. We will have to gather the requirements from the customer.

Hopefully the customer has all the requirements in their minds, we need to gather this by meeting the customer and using few techniques. Those techniques we will discuss later. And, then we need to analyze what we have gathered from the customer. As we gather the requirements it can have several problems in the requirements by analyzing we will identify what are the problems and how to overcome them?

And, once we have got all the requirements and we have overcome all the problems in requirements, then we get down to requirement specification. Mainly, we document all the requirements that have been gathered and all the problems that have been. After all the problems removed from the document, we prepare the SRS document, which captures the requirement. The SRS stands for software requirement specification document. And, as we will see, it's an important skill to write the SRS document.

We have to write in a form that subsequently captures all the requirements. SRS will be well organized according to some accepted standards and the requirements have to be clear to the clients. By looking at the SRS document client should be able to check whether it meets all the requirements or not.

And also, the development team members should need to understand the SRS document by reading the SRS document to know, what is to be done? Writing the SRS document is a very important skill.

In the next lecture, we will see how to do requirement analysis and specification and how to develop the SRS document. We will look at some case studies and based on that, we will see what are the issues, how to go about the requirement analysis and specification and so on.

Thank you.

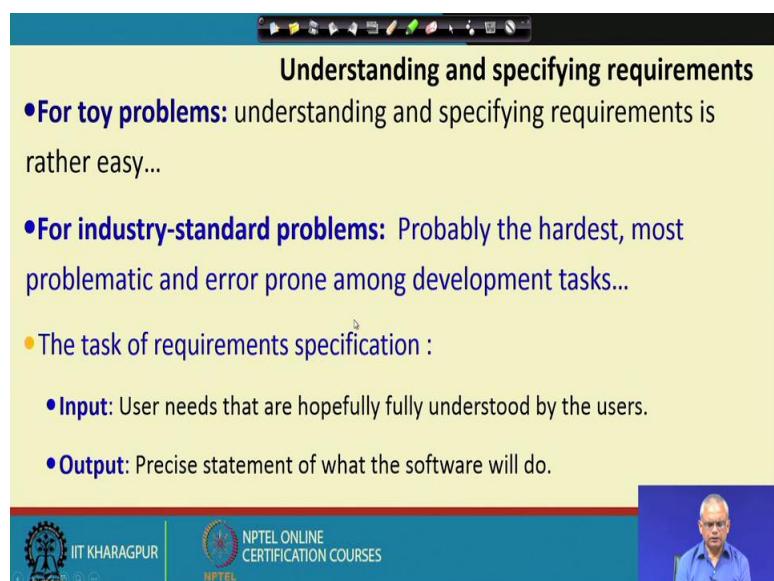
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 15
Introduction to requirement specification

Welcome to this lecture.

In the last lecture we had started to discuss about requirements analysis and specification. Requirements analysis and specification is one of the most important phases in the life cycle of a software development work. It has many skills to learn. This lecture will focus at how to go about carrying out the requirements analysis and specification.

(Refer Slide Time: 00:55)



Understanding and specifying requirements

- **For toy problems:** understanding and specifying requirements is rather easy...
- **For industry-standard problems:** Probably the hardest, most problematic and error prone among development tasks...
- The task of requirements specification :
 - **Input:** User needs that are hopefully fully understood by the users.
 - **Output:** Precise statement of what the software will do.

There are two aspects here: one is understanding the requirements and the second one is about specifying the requirements. If, you are developing a very small toy problem, then requirement specification is extremely easy, because the issues are rather clear and we can specify those. But then for an industry standard problem the requirement specification is typically the hardest among all the phases.

It is the most problematic and error prone among the development tasks and it is important to note that any requirements error has a huge cost overhead, unlike a coding

error or something which can be quickly corrected. A requirement error costs maximum. During requirement specification, the input is the user needs. Hopefully the users understand what are their need to gather, to analyze and remove problems from need and then document that and at the end of the requirements analysis will have a precise treatment of what the software will do and this is documented in the form of a SRS document: software requirement specification document.

(Refer Slide Time: 02:55)

The slide has a blue header bar with standard presentation icons. The main title 'Requirements for Products' is centered in a large black font. Below it, there are two bullet points:

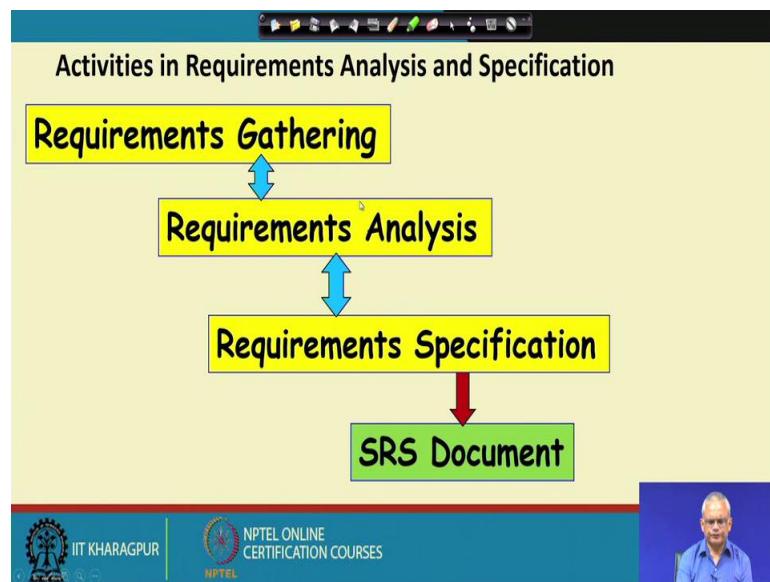
- When a company plans to develop a generic product:
 - Who gives the requirements?
- **The sales personnel!**

At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man speaking.

So, far we have been saying that requirements are there in the clients mind and we need to gather it and document it, but what about the case where a company wants to develop a generic product? That is some product which may be useful to people. In this case, there is no specific client here. The company wants to develop something, hoping that there will be many buyers for it. For example, a company might like to develop a health embedded product which will monitor various health parameters and give feedback to people. Let say, you are the customer who want purchase the product; it can be purchased by anybody who likes it later. But then for development purpose who will be the customer? Who will give the requirements?

Typically, it is the sales person who understand that what will sell well? What features are required typically would be demanded by the customers? And, they act as the clients for the software and they actually need the requirements.

(Refer Slide Time: 04:34)

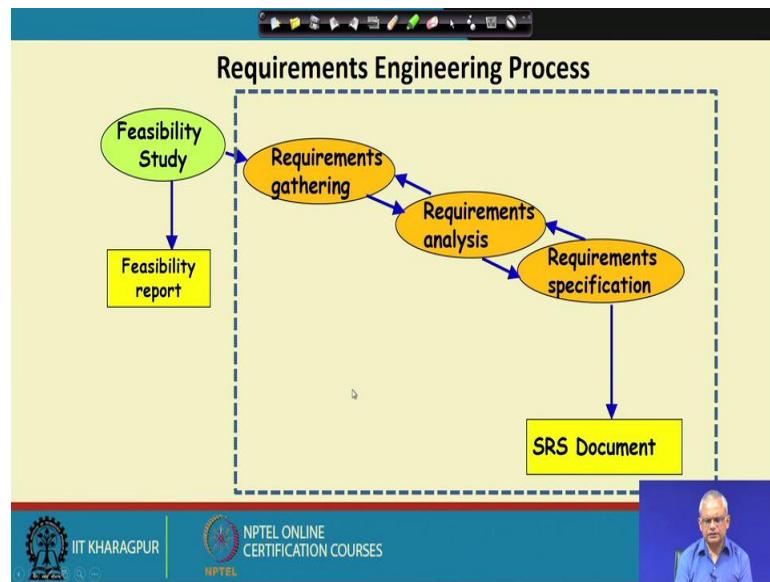


Now, let's try to understand in more detail the activities carried out during requirement analysis and specification. As, we are saying that the first is requirements gathering, because the requirements are there in the minds of the users. There has some techniques to do it. And, once requirements are gathered, start doing the requirements analysis. But during requirement analysis we might find that there are some requirements, which are missing, which are not clear and so on.

We need to again meet the client and do more requirement gathering. And, once our requirements analysis task is complete, we go for specification and then document the requirements and the final outcome is the SRS document.

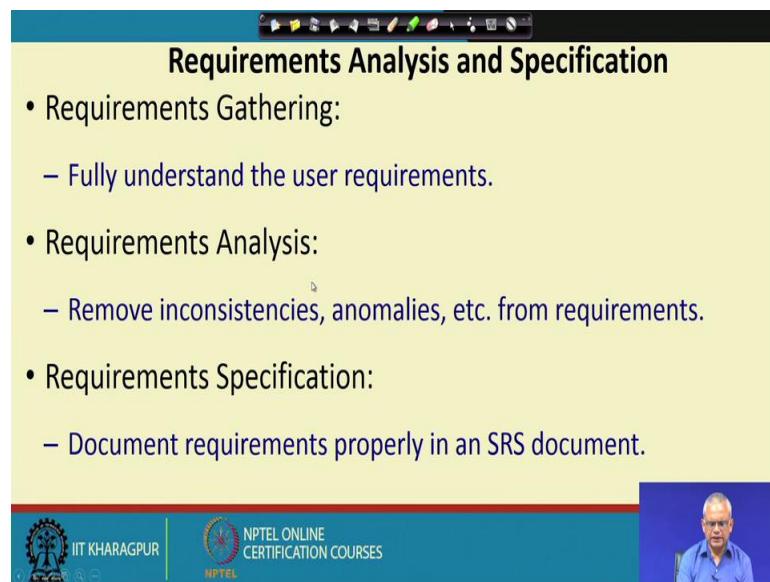
So, there are three main activities here: requirements gathering, requirements analysis and requirement specification. All these are iteratively carried out, because even though one is complete, we need to revisit it if required during the analysis and specification.

(Refer Slide Time: 06:05)



We can also represent it, in more elaborate way in the form showed in the above diagram. After the feasibility study, the project has been found feasible then the requirements gathering started. This is a iterative work, because once we do the requirement gathering we start requirements analysis, but then we might find that need to do more gathering, need to resolve issues and do requirement specification.

(Refer Slide Time: 06:35)



But, then during requirement specifications, we find that there are still requirements problem and might do requirement analysis and again do requirement gathering.

So, these all are the iterative set of activities and at the end of the requirement specification, we have the SRS document produced.

Now, let's look at these three activities in more detail. Requirements gathering, here the requirements are there in the minds of the clients we need to understand the requirements. And, then we need to analysis the requirements and remove problems from the requirement for example, are there any inconsistencies, anomalies and so on. And, then we do the requirement specification, where we document the requirements that we have gathered and analyzed.

(Refer Slide Time: 07:45)

• Good SRS reduces development cost:

- Req. errors are expensive to fix later
- Req. changes cost a lot (typically 40% of requirements change later)
- Good SRS can minimize changes and errors
- Substantial savings --- effort spent during req. saves multiple times that effort

• An Example:

- Cost of fixing errors in req. , design , coding , acceptance testing and operation increases exponentially

Need for SRS...

Let see why SRS document is an important document? a good SRS document actually reduces development cost, because if you do not have proper requirements gathering and development starts there will be many missing requirements, incorrectly understood requirements and so on. And, these will be expensive to fix later even if the requirements are done. It is reported that typically there is a 40 % requirement change later. But, if the requirements are not done well then there will be much more changes and the development will become very expensive. Good SRS can minimize changing errors and as we are mentioning if a error is found during the requirement specification, it leads to substantial saving because otherwise with a wrong requirement, you might go on doing the other life cycle activities. Then again come back and correct the requirements and that will cost multiple times the effort that we would put in the requirements phase to

come up with a good requirement. We rather spent time in the requirements phase try to make it as complete and as truly reflective of the customer requirement as possible, because any problems in the requirement has huge cost implication.

The cost of fixing requirements error during the design, coding, and acceptance testing increase cost exponentially. If during the requirements we fix the bug or the error then it costs very minimal. But, as the development proceeds the design, coding, testing, the cost increases substantially and the main reason is that, if we find a problem let say a requirement is wrong we realize during testing then, we not only need to correct the requirements here but also need to review it, change the design document, review it again and then again change the coding and again carry out the testing. So, the cost implications of errors are huge it increases exponentially. The more delay we have in identifying any error the more cost increase. So, it's really costs effective if we spend some time during the requirement gathering, analysis and specification and we produce a good SRS document.

(Refer Slide Time: 11:37)

The slide has a yellow header bar with various icons. The main content area is yellow and contains a bulleted list of uses for an SRS document. A yellow callout box labeled 'What are the Uses of an SRS Document?' is positioned to the right of the list. The footer is blue and features the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Establishes the basis for agreement between the customers and the suppliers
- Forms the starting point for development.
- Provide a basis for estimating costs and schedules.
- Provide a basis for validation and verification.
- Provide a basis for user manual preparation.
- Serves as a basis for later enhancements.

Now, let see if we produce SRS document in what ways it will be used? One is that it's an agreement between the customer and the developer. As, long as the customer agrees to the requirements those reflect the requirements of the customer and the developer agrees to develop those. And, then the developers take up the SRS document, understand it and start their design and development based on that. The project manager tests the

SRS document and estimates the cost and schedule staffs based on the number of features that are mentioned in the requirements and so on.

The test team uses the SRS document and based on that they design the test cases for the system testing. The, SRS document is used for the user manual preparation because the SRS document contains all the functionalities to be provided by the software to the user. The user manual is typically based on the SRS document and also later when maintenance works start, the SRS document is used for maintenance. Again, when specific features to be enhanced the SRS document required to identifies the features and then the work starts from there.

(Refer Slide Time: 13:31)

Forms A Basis for User Manual

- The SRS serves as the basis for writing User Manual for the software:
 - **User Manual: Describes the functionality from the perspective of a user --- An important document for users.**
 - Typically also describes how to carry out the required tasks with examples.

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES 

So, the SRS document is a very important document. It solves for multiple purposes including the basis for writing the user manual. In the user manual, we describe the way the users can understand what are the functionalities that are provided by the software. The user manual is an important document of the user here not only that we write in the user understandable form, but also illustrate the functionalities with examples. The SRS document as we had seen has various users.

(Refer Slide Time: 14:24)

SRS Document: Stakeholders

- SRS intended for a diverse audience:
 - Customers and users use it for validation, contract, ...
 - Systems (requirements) analysts
 - Developers, programmers to implement the system
 - Testers use it to check whether requirements have been met
 - Project Managers to measure and control the project
- Different levels of detail and formality is needed for each audience
- Different templates for requirements specifications used by companies:
 - Often variations of **IEEE 830**

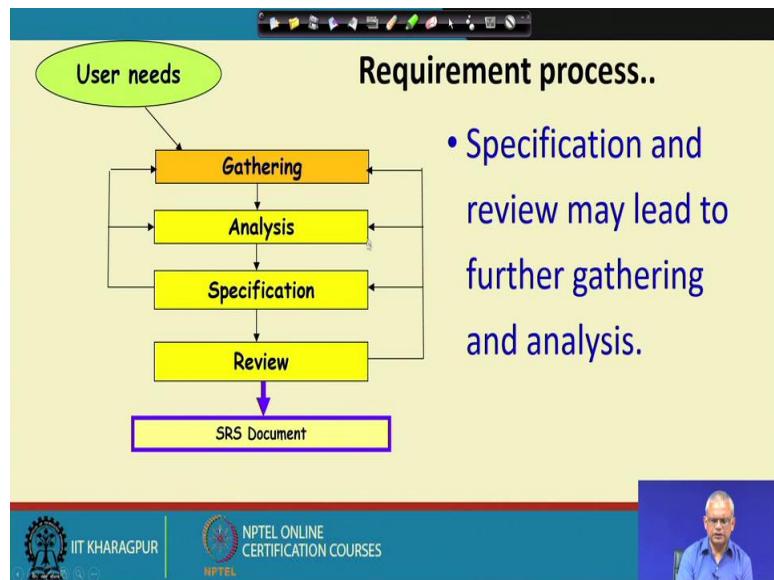
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

And therefore, there are many users of the SRS document. The customers use it for checking whether the developed software is as per their requirement. The requirement analyst they write the SRS document. The developers, programmer et cetera often referred to the SRS document, to carry out their work. The testers use SRS document to design test cases and to ensure that the developed software meets the requirements. The projects managers use SRS to estimate the work that is required and use the estimation for controlling the project.

So, there are many stakeholders for the SRS document. So, the SRS document should be readable and understandable to the customers, to the developers, to the testers, to the project managers, and of course to the requirement analysts they write the document. Each stakeholder, needs different types of information from the SRS document and in a way that they can appreciate.

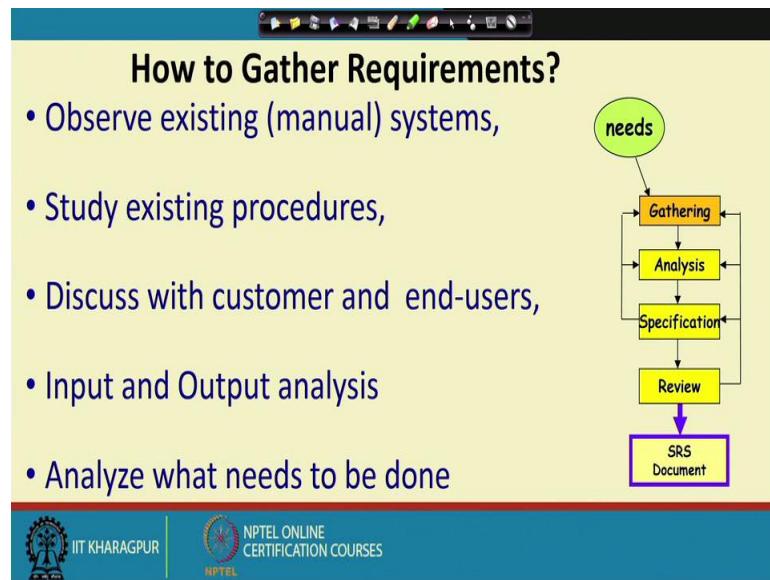
Since, it is a very important document so one standard needs to follow and one of the well accepted standards is the IEEE 830 standard. We will look at the IEEE 830 standard and typically the standard that is used in the industry as small variations of the IEEE 830 standard. We understand IEEE 830 standard, how to document it? We should not only be able to read through requirements developed by others, but also, we should be able to write requirements that are accepted by others.

(Refer Slide Time: 16:42)



Before, that let's try to understand the requirements process. That is, what are the activities that are undertaken? And, how these are accomplished? we had some discussion on that, but let's look at more elaborately. Here, the requirements gathering starting with the user needs. We gather the requirements and then analyzed, but then we might have to go back to gathering. And, then once this is done satisfactorily the gathering is complete and the analysis satisfactorily complete, then we proceed to specification. But during specification also we might have to do some analysis and may need to do gathering if we notice problems during writing down or documenting the requirements. And, once the SRS document draft is produced, it is reviewed by the customer to check whether all requirements have been captured and also reviewed by the development team members whether these have been written in sufficient detail and so on. The testers also review it to check whether the testing or the test cases can be satisfactorily designed based on the requirements that have been written down in the document. And, once the review is done then the SRS document is produced. It's important to understand here that this is not a series of stages rather it consists of lot of iterations. Lot of iterations are required because, during analysis you might realize that some problems need clarification from the customer, we need to do gathering, are some requirements are missing and during specification also we might need to do some analysis and gathering.

(Refer Slide Time: 18:52)



Now, let's try to see how to do the gathering? this is the first task in the requirements process here. There are many techniques that are typically deployed, we need to use multiple techniques for gathering requirements; we will take up a case study, simple case study to see how it can be done?

The first is that if there is an existing manual system or there is an existing software, which is something similar to the one that we are developing we need to study that. Let say an office work needs to be automated. We need to first observe that what work is exactly done in the office. If a software exists which is similar, we need to use it and check what all facilities it provides and where all we want changes. And, once we observe the work, we need to study the procedures how this work carried out for this we need to discuss with the customer and end users. Also, need to do input and output analysis. Now let say there are some forms that are input. Take the example of an office. If the students and the faculty need to fill some forms and submit in the office those are the input, we need to study what are the types of forms that are being used for input? And what output office produce and then what are the forms they use to producing the output? what are exactly the data that is input and the data that is output by the office.

And, based on this gathered requirement we decide what exactly need to be done. We will look at some more detail about the gathering work.

(Refer Slide Time: 21:24)

The slide has a yellow header bar with the title 'Requirements Gathering Activities'. Below the title is a bulleted list of five activities. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- 1. Study existing documentation
- 2. Interview
- 3. Task analysis
- 4. Scenario analysis
- 5. Form analysis

The gathering activity typically we can say that there are five activities: One is study existing documentation where we see study the documentation about the system to be developed. The second is interview. We need to interview the end users and also the customer who is involve in the software development. The third is called as task analysis that is based on the interview, we first identify that they use it for certain tasks, they plan to use it for performing certain tasks. And, then we need to do the task analysis to find out what are the exact procedure based on which the task accomplished. And, then for each task we need to do the scenario analysis, that is each task consists of multiple scenarios. For example, let's just take a simple case of a student in the office of a department. The students can take leave. One is that the students fill up form give it to the office or get it approved by the head of department and give it to the office. This is just one scenario. The taking leave can have multiple scenario for example, they want to take medical leave or they want to take a semester withdrawal and so on. So, these are the different scenarios where they might have to do a slightly different procedure. Even, though the task is applying for leave by a student that can have several scenarios. And the fifth technique to gather requirements is the form analysis. In the form analysis, the input form and the output form are analyzed to find out what are the data that are input and what are the data that output.

So, five well known requirements gathering techniques are: studying existing documentation, interviewing the end users and the customer, task analysis, scenario

analysis, and form analysis. If, you want to do requirements gathering for any project these are the five tasks that typically need to carry out.

(Refer Slide Time: 24:50)

Requirements Gathering (CONT.)

- In the absence of a working system,
 - Lot of imagination and creativity are required.
- Interacting with the customer to gather relevant data:
 - Requires a lot of experience.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



Now, let's take up a simple case study. And, try to understand how these activities are carried out? Of course, many times the customer does not understand the problem well and here it is the duty responsibility of the person, who is gathering the requirements suggest to the customer what features might help and would be useful. The person should have lot of imagination and creativity. Requirements gathering, even though it appears like a simple task but it requires lot of experience. Because, this is a crucial task, need to gather all the requirements from the customer who has some of the requirements in mind, and others the person doing the gathering need to suggest to him and make him appreciate that what features might be useful.

(Refer Slide Time: 25:56)

Requirements Gathering (CONT.)

- Some desirable attributes of a good requirements analyst:
 - Good interaction skills,
 - Imagination and creativity,
 - Experience...

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The person doing the requirements gathering not only should have good communication skill, should have imagination creativity and experience.

(Refer Slide Time: 26:07)

Case Study: Automation of Office Work at CSE Dept.

- The academic, inventory, and financial information at the CSE department:
 - At present carried though manual processing by two office clerks, a store keeper, and two attendants.
- Considering the low budget he had at his disposal:
 - The HoD entrusted the work to a team of student volunteers.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us do a case study. Let say the computer science department has an office which is operating right now manually. There are two office clerks, a store keeper and two attendants. Their main work is to keep track of the academic records, the inventory of different equipments and so on and also the financial information in the department. Now, considering the low budget the head of department just entrusted these to a team of

student volunteers to develop office automation work. Now, let see, how the student members of the development team went about gathering analyzing and how they specifying the requirement? They selected the most experienced person: the student having good communication skill to carry out the requirement gathering.

(Refer Slide Time: 27:27)

Case Study: Automation of Office Work at CSE Dept.

- The team was first briefed by the HoD:
 - Concerning the specific activities to be automated.
- The analysts first discussed with the two office clerks:
 - Regarding their specific responsibilities (tasks) that were to be automated.
- The analyst also interviewed student and faculty representatives who would also use the software.

Interview

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

They met the head of department first and then try to identify what exactly is required. So, the head of the department is the customer here and then they met the office clerks they are the end users because finally, they will be using the software. From the end users they gathered what are the tasks that they do every day and that they would like to automate. Even, they met with other users like students who right now manually submit forms. Later, they would have to submit it through the software to be developed so, their expectations of the software also need to be gathered. They interviewed the students and faculty who are also the stakeholders and the users of the software.

This task that was achieved by meeting the head of the department, office clerk, the students and the faculty. They identify in what way user will use the software? What is their expectation of the software? What features will be required? All these are the interviewed task.

(Refer Slide Time: 28:51)

Case Study: Automation of Office Work at CSE Dept.

- For each task that a user needs the software to perform, they asked:
– The steps through which these are to be performed.
– The various scenarios that might arise for each task.
- Also collected the different types of forms that were being used.

Task and Scenario Analysis

Form Analysis

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

After identifying the tasks, they try to identify the steps through which these will be performed and then identify various scenarios that may arise for each task. Then they do the task and scenario analysis. They collected various types of forms that are used by the office. They collected filled form from the users that is student, faculty etc. and also collected the forms that office users use to produce any output. All these are the form analysis.

(Refer Slide Time: 29:40)

Case Study: Automation of Office Work at CSE Dept.

- The analysts understood the requirements for the system from various user groups:
– Identified inconsistencies, ambiguities, incompleteness.
- Resolved the requirements problems through discussions with users:
– Resolved a few issues which the users were unable to resolve through discussion with the HoD.
- Documented the requirements in the form of an SRS document.

Requirements Gathering

Requirements Analysis

Design

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

And, then the student members who were carrying out the requirements task, they looked at the requirements gathered. Identified inconsistency, ambiguity and incompleteness. All these are the requirement problems.

And, they identify the problems and overcome them by discussing with the head of department. And, then they went about documenting the requirement specification.

We will stop here this lecture and we will continue from this point in the next lecture.

Thank you.

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

Lecture – 16
Requirement gathering and analysis

Welcome to this lecture. In the last lecture, we were discussing about Requirements analysis and specification. We identified the main task that are to be completed during requirement analysis and specification is that to do the requirements gathering and then, need to analyze the gathered requirements. And finally, we have to go about the specification and then get the specification reviewed by the customer and other stakeholders.

Now, we will look at the second activity in the development process which is the analysis. We had looked at the gathering activity, seen the main activities that are to be performed during this activity is requirements gathering and then, we are looking at a case study; simple case study to understand that what are the activities and how they are to be performed. Now, let's look at analysis of the gathered requirements.

(Refer Slide Time: 01:30)

The slide has a title 'Analysis of Gathered Requirements'. Below the title is a bulleted list of the main purpose of requirements analysis:

- Main purpose of req. analysis:
 - Clearly understand user requirements,
 - **Detect inconsistencies, ambiguities, and incompleteness.**
- Incompleteness and inconsistencies:
 - Resolved through further discussions with the end-users and the customers.

On the right side of the slide, there is a vertical flowchart diagram:

```
graph TD; needs((needs)) --> Gathering[Gathering]; Gathering --> Analysis[Analysis]; Analysis --> Specification[Specification]; Specification --> Review[Review]; Review --> SRS[SRS Document]
```

The flowchart shows a sequence of steps: needs → Gathering → Analysis → Specification → Review → SRS Document. Arrows indicate the flow from one step to the next, with a final arrow pointing down to the 'SRS Document' box.

So, this is the second task in requirement process. In the requirements process, the main focus of analysis is to identify the requirements problems. For this, we have to go through all the gathered requirements and clearly understand the customer needs and

then, we analyze to identify and remove requirements problems. There are three main problems that typically exist in a requirement gathered; these are called as inconsistencies, ambiguity, and incompleteness.

Let's try to find out that what are these three problems inconsistency, ambiguity and incompleteness and how to identify these problems and once the problems are identified, these are eliminated by discussing with the customer and end user.

(Refer Slide Time: 02:57)

Ambiguity

"All customers must have the same control field."

Do you notice any problems?

Multiple interpretations possible

1. All control fields have the same format.
2. One control field is issued for all customers.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, first let's look at the ambiguity problem. Ambiguity as the name says is that the requirement is vague; it's not precise enough, it can have multiple meanings. Let's start by taking an example. Let say one of the requirements that was obtained have the same control field for all the end users and customers. This is actually an ambiguous requirement where all customers have the same control field.

What are the problems here? The problem is that different interpretations are possible. All customers have the same control field, does it mean that the one control field is shared with the multiple customers? And if one customer changes it, other customers are able to cheat and so on or is it that similar control fields are present with different customers?

We need to discuss with the customer to check what was really meant? Is it that there is only one control field which is shared among customers, where an action done by one

customer also affects the other customers or is it that they have similar but different control fields?

These two are independent problems.

(Refer Slide Time: 05:08)

Inconsistent Requirement

- Some part of the requirement:
 - contradicts some other requirement.
- **Example:**
 - One customer says turn off heater and open water shower when temperature > 100°C
 - Another customer says turn off heater and turn ON cooler when temperature > 100°C

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Now, let's look at an example of an inconsistent requirement. As the name says, in the inconsistent requirement, some requirement contradicts some other requirement. For example, let say one customer says that turn on the heater and open water shower when the temperature and the chemical reactor exceeds 100 degree centigrade.

But then, another customer has given the requirement saying turn off the heater and turn on the cooler but not the water shower when the temperature exceeds 100 degree centigrade. So, when the temperature exceeds 100 degree centigrade, the action to be taken is contradictory; need to discuss with the customer, which one is the correct one?

(Refer Slide Time: 06:12)

The screenshot shows a presentation slide with a yellow header bar containing icons. The main content area has a light beige background. On the right side, there is a yellow callout box with the text "Incomplete Requirement". The slide contains the following text:

- Some requirements not included:
 - Possibly due to oversight.
- Example:
 - The analyst has not recorded that when temperature falls below 90°C :
 - heater should be turned ON
 - water shower turned OFF.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a portrait of a man.

Incomplete requirement is the third category of problems where we need to identify all incomplete requirements and eliminate them. Incomplete requirements are the requirements that the customer has forgot or does not realize that these will be needed. Just for an example the customer has not recorded or has not expressed what will happen, when the temperature falls below 90 degree centigrade? They says, that when the temperature is greater than 100 degree centigrade do something; but then, on the lower side of the temperature has not recorded any action to be taken. So, that's an incomplete requirement. We should have actually recorded that when the temperature is less than 90 degree, heater should be again turned on and the water shower should be turned off to maintain the temperature between 90 and 100.

(Refer Slide Time: 07:33)

The slide has a yellow header bar with the title 'Analysis of the Gathered Requirements'. Below the title is a bulleted list of requirements analysis tasks:

- Requirements analysis involves:
 - Obtaining a clear, in-depth understanding of the software to be developed
 - Remove all ambiguities and inconsistencies from the initial customer perception of the problem.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

During the analysis of the gathered requirement, must understand the gathered requirements well and then remove the ambiguities, inconsistencies, and incompleteness from the gathered requirements.

(Refer Slide Time: 07:53)

The slide has a yellow header bar with the title 'Analysis of the Gathered Requirements (CONT.)'. Below the title is a bulleted list of difficulties in obtaining requirements analysis:

- It is quite difficult to obtain:
 - A clear, in-depth understanding of the problem:
 - Especially if there is no working model of the problem.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

It's look like to be a simple work, but it's actually a difficult task; without having a depth understanding of the problem based on what the customers have in their mind it is a difficult problem.

(Refer Slide Time: 08:15)

Analysis of the Gathered Requirements (CONT.)

- Experienced analysts take considerable time:
 - Clearly understand the exact requirements the customer has in his mind.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

NPTEL

The person who analysis the gathers requirement and specifies is typically called as analysts; the analysts take time to completely understand the customers need, doing analysis and specify. They devote a lot of time here because this is a difficult task.

(Refer Slide Time: 08:50)

Analysis of the Gathered Requirements (CONT.)

- Experienced systems analysts know - often as a result of painful experiences ---
 - ‘Without a clear understanding of the problem, it is impossible to develop a satisfactory system.’

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

NPTEL

The experienced analyst known that it is worth while spending little bit more time to understand the requirements gathering and analysis. Because unless we have a clear understanding of what is to be done; it is impossible to develop a satisfactory solution.

(Refer Slide Time: 09:18)

The slide has a yellow header and a blue footer. The header contains the title 'Analysis of the Gathered Requirements'. The footer features the IIT Kharagpur logo, the NPTEL logo, and a video player showing a man in a blue shirt.

Analysis of the Gathered Requirements

- Several things about the project should be clearly understood:
 - What is the problem?
 - What are the possible solutions to the problem?
 - What complexities might arise while solving the problem?

During the gathering of requirements, and the analysis, several things must be clearly understood that what are the problems that the user wants to solve? What features are needed and how can this be implemented? Because if the user wants to do something very far fetch, which is difficult to implement. We should not really accept that; we should change modifications to that which can be meaningfully solved and then, must understand what are the complexities while solving the problem.

(Refer Slide Time: 10:07)

The slide has a yellow header and a blue footer. The header contains the title 'Analysis of the Gathered Requirements'. The footer features the IIT Kharagpur logo, the NPTEL logo, and a video player showing a man in a blue shirt.

Analysis of the Gathered Requirements

- Some anomalies and inconsistencies can be very subtle:
 - Escape even most experienced eyes.
 - If a formal specification of the system is constructed,
 - Many of the subtle anomalies and inconsistencies get detected.

Even though, the analysis activities carried out very thoroughly, still some problems may remain. Using formal specification technique, very critical software can be developed. Because formal specification is a very expensive, time consuming task. If some part of the software extremely sensitive (critical parts), we can have a formal specification after we have developed the requirement specification document.

(Refer Slide Time: 10:48)

Analysis of the Gathered Requirements_(CONT.)

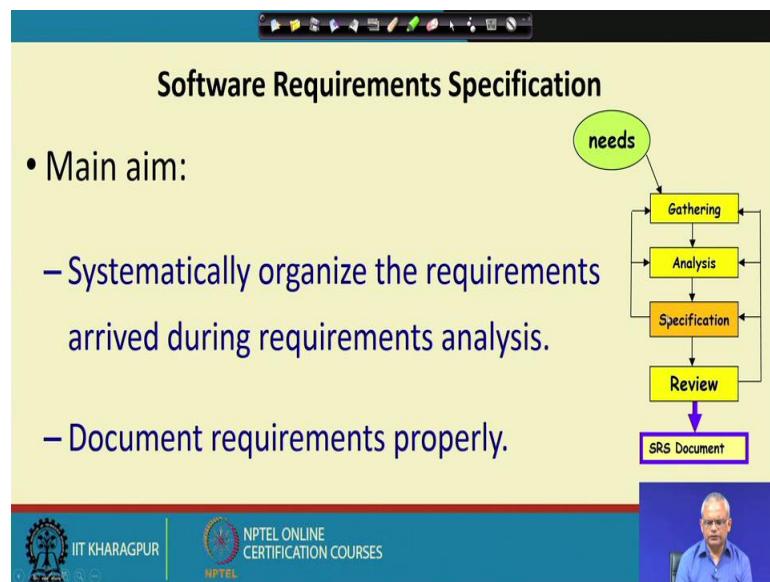
- After collecting all data regarding the system to be developed,
 - Remove all inconsistencies and anomalies from the requirements,
 - Systematically organize requirements into a Software Requirements Specification (SRS) document.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

NPTEL

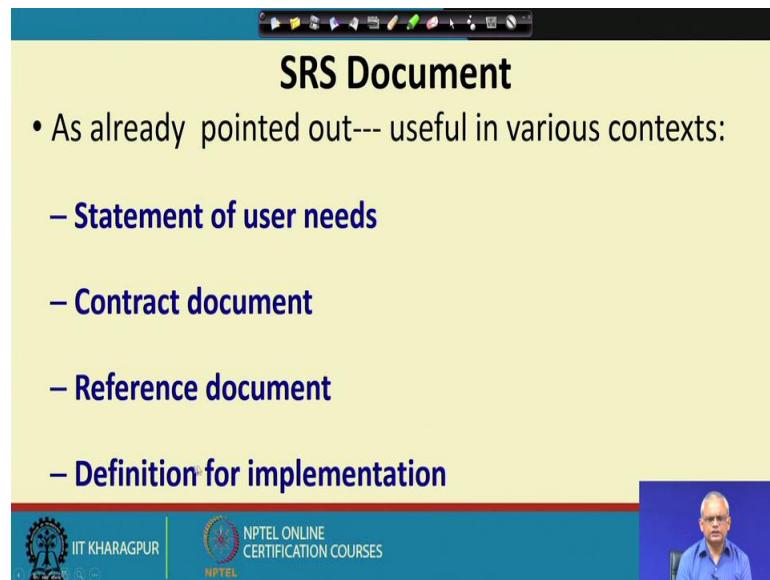
Now, after we have removed all inconsistency, anomalies, incompleteness from the document, we go about documenting the requirements. Now, let see how to write the SRS document. This is a very important skill. Let see how to go about.

(Refer Slide Time: 11:15)



Now, let's look at the specification task that is how to document the requirements. Here, we have the gathered requirements and all the problems from them have been removed through the analysis activity and now, we get down to systematically organize the requirements and then document them properly.

(Refer Slide Time: 11:44)



The SRS document is a very important document, used by various types of users and SRS document should be written such that it can be understood and useful to different users. Because users may need look for different types of information in the SRS

document. It serves as a statement of the user need. It's a contract document. It's a reference document for the development team to start development work; It's an important document for the testers to design the test cases and also it is the definition of the problem based on which the developers start to develop the work.

(Refer Slide Time: 12:40)

The slide has a blue header bar with various icons. The main title is 'SRS Document (CONT.)'. Below the title, there is a bulleted list:

- SRS document is known as **black-box specification**:
 - The system is considered as a black box whose internal details are not known.
 - Only its visible external (i.e. input/output) behavior is documented.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

The SRS document should be written in the form of a black-box specification. What do you mean by a black box specification of the problem? The black-box specification is actually the functionality as can be observed externally that is what is the input to the system and what gets output by the system?

The system appears as a black box where we don't know that based on the input data which function here involved and what it does to get the output. But we know that what will be input and based on this, what output will be generated. That's the input output behavior. The internal details we should not document or try to document, we just document the input output behavior of the system.

(Refer Slide Time: 13:58)

SRS Document (CONT.)

- SRS document concentrates on:
 - What needs to be done in terms of input-output behaviour
 - Carefully avoids the solution ("how to do") aspects.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



In other words, you can say that the SRS document should focus on what needs to be done based on the input data rather than how to do.

(Refer Slide Time: 14:23)

SRS Document (CONT.)

- The requirements at this stage:
 - Written using end-user terminology.
- If necessary:
 - Later a formal requirement specification may be developed from it.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



The requirement document should be written in end user terminology because it is the end users who will see whether all the requirements have been satisfactorily represented. It's written as a document, which can be understood by several types of users that is the end user, the testers, the project manager, the development team members, and so on.

But then, if some parts are very critical then for that separate formal requirement specification document can be written.

(Refer Slide Time: 15:15)

- **It should be concise**
 - and at the same time should not be ambiguous.
- **It should specify what the system must do**
 - and not say how to do it.
- **Easy to change.,**
- **It should be consistent.**
- **It should be complete.**

Properties of a Good SRS Document

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let see before starting to write a document, we have to see what should be the sections, how should we go about writing the subsections and so on. Now, let see overall what exactly is a good SRS document? A good SRS document should not be too verbose. It should be used the minimum writing that is needed, should be concise and at the same time should not leave out details otherwise it will be ambiguous. It should be a black-box specification; should specify what the system must do and not say how to do it. It should be easy to change; that is if some functionality tomorrow changes, we should be able to change very quickly but how do we ensure that it is the document that is easy to change? for that we have to structure it well, like if you want to change one functionality, we should be able to change it in one place, should not have to change in multiple different places. A good SRS document also should be consistent. Different functionalities represented in the SRS document, should be consistent; should be complete.

(Refer Slide Time: 16:50)

Properties of a Good SRS Document (cont...)

- **It should be traceable**
 - You should be able to trace which part of the specification corresponds to which part of the design, code, etc and vice versa.
- **It should be verifiable**
 - e.g. "system should be user friendly" is not verifiable

And also, it should be traceable. Traceable means for a specific functionality, we should be able to identify the functionality is implemented by which part of the design, which part of the code, and so on. To be able to have traceability, the first thing is that we should structure the document properly. For example, the functions have to be precisely written. Each function should be identifiable by a number or by some id and should be written at one place. And also, for a given design element, we should be able to tell it implements which functionality or for a given code segment suppose we found an error in a code segment, then we should be able to tell which functionality is would have got affected. So, traceability is very important concept that given a function requirement number should be able to tell which part of the design implements that function requirement and which part of the code implements that design. Also, in the other way given a code element should be able to tell, it implements which part of the design and also which functionality.

The SRS document should be written in a way such that it is verifiable to easily write test cases and we can check whether the requirements is satisfied or not. If you have a requirement like, let say the developed software should be user friendly. It is very difficult to test. Even if we test for user friendly, then it may not be accepted by some others.

So, this is a very difficult requirement to verify. While writing the requirement, we have to ensure that every requirement can be tested satisfactory.

(Refer Slide Time: 19:46)

SRS should not include...

- **Project development plans**
 - E.g. cost, staffing, schedules, methods, tools, etc
 - Lifetime of SRS is until the software is made obsolete
 - Lifetime of development plans is much shorter
- **Product assurance plans**
 - Configuration Management, Verification & Validation, test plans, Quality Assurance, etc
 - Different audiences
 - Different lifetimes
- **Designs**
 - Requirements and designs have different audiences
 - Analysis and design are different areas of expertise

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

SRS document should not include not only the how-to aspects but other things like let say project development plans like what will be the cost, staffing schedule, tools to be used et cetera. Because the SRS document is used even after the development work complete. So, the lifetime of the development plans is till the development completes but the SRS document is required after the development work also. Similarly, it should not include testing assurance plans, Configuration Management, test plans, Quality Assurance because they have different audience and different lifetime. SRS document has several audiences and we write only those aspects which are interest to all the audience. The designs et cetera, they are also not to be included.

(Refer Slide Time: 21:04)

The screenshot shows a presentation slide titled "SRS Document (CONT.)". The slide content is as follows:

- Four important parts:
 - **Functional requirements,**
 - **External Interfaces**
 - **Non-functional requirements,**
 - **Constraints**

The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video window showing a person speaking.

Now, let see how to go about writing the SRS document? There are four important parts in any good SRS document. Possibly, the most important part is the Functional requirements and then, we must identify the External Interfaces. The external software and hardware with which the software needs to interact and the users and then, the Non-functional requirements. In the non-functional requirements we document some requirements which are cannot be specified as functions.

What is the distinction between a functional and non-functional requirement? SRS document have a section on the non-functional requirements and what are various constraints on the system? For example, one constraint maybe that it has to follow certain standards or maybe that it has to use an open source database.

(Refer Slide Time: 22:27)

The slide has a yellow header with the title 'Functional Requirements'. Below the title is a bulleted list of points. At the bottom of the slide is a footer bar containing the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Specifies all the functionality that the system should support
 - Heart of the SRS document:
 - Forms the bulk of the Document
- Outputs for the given inputs and the relationship between them
- Must specify behavior for invalid inputs too!

First, we will see the details of functional requirements because this is the most important part of any SRS document. Here, we identify what are the functionalities or facilities that the software will offer to the users and we document it in the form of the input; We have to mention the output that will be produced and also we have to mention that what would happen when the user gives invalid input.

(Refer Slide Time: 23:16)

The slide has a yellow header with the title 'Functional Requirement Documentation'. Below the title is a bulleted list of points. At the bottom of the slide is a footer bar containing the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

- Overview
 - describe purpose of the function and the approaches and techniques employed
- Inputs and Outputs
 - sources of inputs and destination of outputs
 - quantities, units of measure, ranges of valid inputs and outputs
 - timing

Identifying the functional requirements is a very important skill just to give an example, let say, we have to develop a library software.

So, what are the functionalities that it should have? or let take another problem that is you are asked to develop a word processing software that is somebody should be able to write a document, something like a Microsoft word or something that will be a good word processing software. So, what will be the functional requirements of that? To identify the functional requirements, we have to see what facilities it provides. For example, let us say library software. We have to look at who are the different users the librarian and then, the library members and possibly the accountant. Now then, we have to look at what are the facilities required by each type of user? A library member might like to issue book, return book, query whether book is available. The librarian might like to create members, it like to delete members, might like to check if there is fine against anybody, might like to create books, delete books et cetera. The accountant might like to check, what is the financial position, might like to enter the grants, might like to enter the fees, might like to register the expenses and so on. So, from each users perspective, we have to look at the software terms of the facilities provided and each facility is basically some meaningful work that the user can accomplish using that and once you have identify the functional requirements, each functional requirement, we need to document. For every function, we need to describe what will be achieved and what are the approaches and techniques that will be deployed and also need to identify the input and output. Typically, every function requires an input. It does some processing and produces some output. For example, a query book function, for a library software, the user gives the name of the book. So, that is an input. The software checks the book database for those books which match the user query and then, it produces the output. So, every functional requirement must be documented in the form of an overview of what the functionality will achieve and then, it possible we need to identify the input and output.

For every functionality, it may not be feasible to produce the input and output. We will look at more details of how to document the functional requirements. So, one thing must be clear from this lecture is that what is a functional requirement? A functional requirement is a facility provided by the software using which user can get some meaningful piece of work done.

Let's now stop here and from this we will proceed. We will look at more details of functional requirements, non-functional requirements, constraints and so on in the next lecture.

Thank you.

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

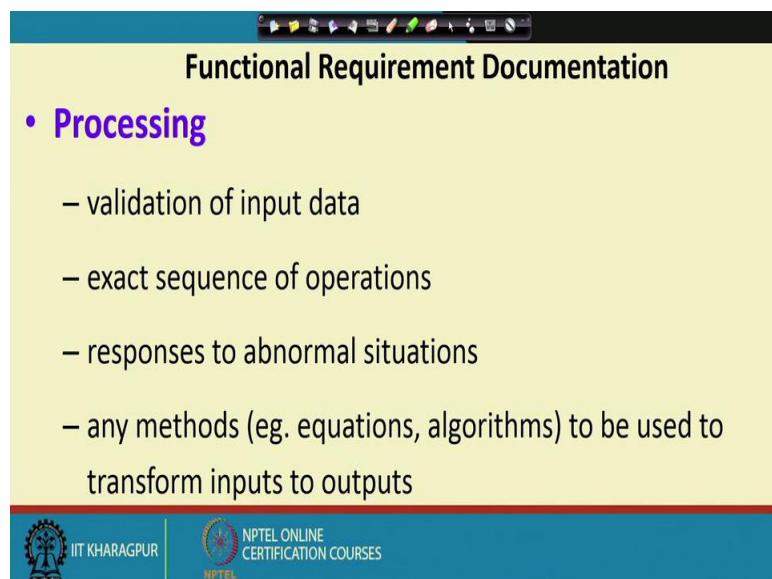
Lecture – 17
Functional requirements

In the last lecture, we were examining how to write SRS document. We identified what are the important information that SRS document must have. One of the most important information that is need to present in the SRS document is the Functional requirement. We are discussing what exactly is a Functional requirement? We said functional requirement is a facility offered by a software to the users using which the user can get some meaningful work done.

And then, we were discussing how to document the functional requirement? We said that for each functional requirement, we must give an overview of the functional requirement and then, the input data that will be given and the output that will be produced and also the procession.

The processing should be in terms of maybe validation of the input data.

(Refer Slide Time: 01:49)



Functional Requirement Documentation

- Processing
 - validation of input data
 - exact sequence of operations
 - responses to abnormal situations
 - any methods (eg. equations, algorithms) to be used to transform inputs to outputs

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

It can be the exact sequence of operations that will be carried out. It can also contain responses to abnormal situations and also any methods or description of algorithms, equations that can be used to transform the input to output.

(Refer Slide Time: 02:13)

The slide has a yellow header with the title 'Nonfunctional Requirements'. Below the title is a bulleted list of characteristics:

- Characteristics of the system which can not be expressed as functions:
 - Maintainability,
 - Portability,
 - Usability,
 - Security,
 - Safety, etc.

At the bottom left, there are logos for IIT Kharagpur and NPTEL. At the bottom right, there is a small video frame showing a person speaking.

Now, let's try to see what exactly is a non-functional requirement? A non-functional requirement is a characteristic which cannot be characterized or expressed as a function. A function is a feature using which a user can get meaningful work done; but there are some requirements, which are not like the functional requirements. For example, we might say that a developed should run on both Windows and the UNIX environment. It should be easily maintainable; it should have a graphical user interface that can be used in a standalone mode or through a web browser. It should be extremely secure et cetera. So, these are all examples of non-functional requirement, these apply to the software as a whole and these are not a functionality or a feature offered to the user to get some meaningful work done.

(Refer Slide Time: 03:54)

Nonfunctional Requirements

- Reliability issues
- Performance issues:
 - **Example:** How fast can the system produce results?
 - At a rate that does not overload another system to which it supplies data, etc.
 - Response time should be less than 1sec 90% of the time
 - Needs to be measurable (verifiability)

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



It can also include reliability issues that it should not fail frequently and if it fails should be able to recover the data quickly. It can also contain performance issues. Non-functional requirement can also contain performance issue. For example, how fast the system should produce result? We can say that 90 percent of the time, the response time should be less than one second or we may say that it should not produce result so fast such that other systems that work with it are not overloaded. But for all non-functional requirement, we have to ensure that somewhere we can test that; that means we should be able to measure and see whether the non-functional requirement working correctly or not.

(Refer Slide Time: 05:09)

The slide has a yellow header with the title 'Constraints'. Below the title is a bulleted list of requirements:

- Hardware to be used,
- Operating system
 - or DBMS to be used
- Capabilities of I/O devices
- Standards compliance
- Data representations by the interfaced system

At the bottom left is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. At the bottom right is a video frame showing a person speaking.

Another important section in the requirement documents are the Constraints. For example, what hardware to be used? What is the other software's with which it needs to work? What operating system to be used? What are the capabilities of the I/O devices? Data representation the interface system, these are all constraints.

(Refer Slide Time: 05:36)

The slide has a yellow header with the title 'External Interface Requirements'. Below the title is a bulleted list of interface types:

- User interfaces
- Hardware interfaces
- Software interfaces
- Communications interfaces with other systems
- File export formats

At the bottom left is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. At the bottom right is a video frame showing a person speaking.

Now we will discuss about the external interface requirement. This is also an important section need to document. Some external interface requirements are user interface, the interfaces with other hardware that is hardware interface, software interfaces which

needs to work with other software. For example, we might say that for other software, it needs to produce an XML file or maybe a comma separated text file. Other two external interface requirements are communication interface and file export formats interface.

(Refer Slide Time: 06:30)

Goals of Implementation

- Goals describe things that are desirable of the system:
 - But, would not be checked for compliance.
 - For example,
 - Reusability issues
 - Functionalities to be developed in future

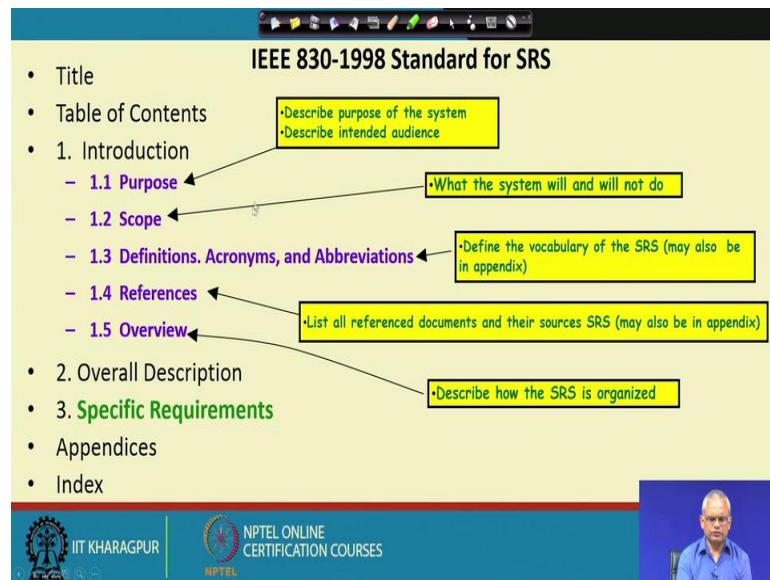
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

There is another section which is the goal of implementation. In both functional and non-functional requirement, after the software is complete, we need to test whether both functional and non-functional requirement constraints are satisfied or not. Where, goals of implementation are the general suggestions, which can help the developer to see what is required. But this will not be tested.

For example, under the goal of implementation, we might say that it should be very reliable because you will use it in a very critical application. But then, as long as it is under the section goal of implementation, this will not be tested; it's only a guideline or it's a suggestion to the developers.

(Refer Slide Time: 07:39)



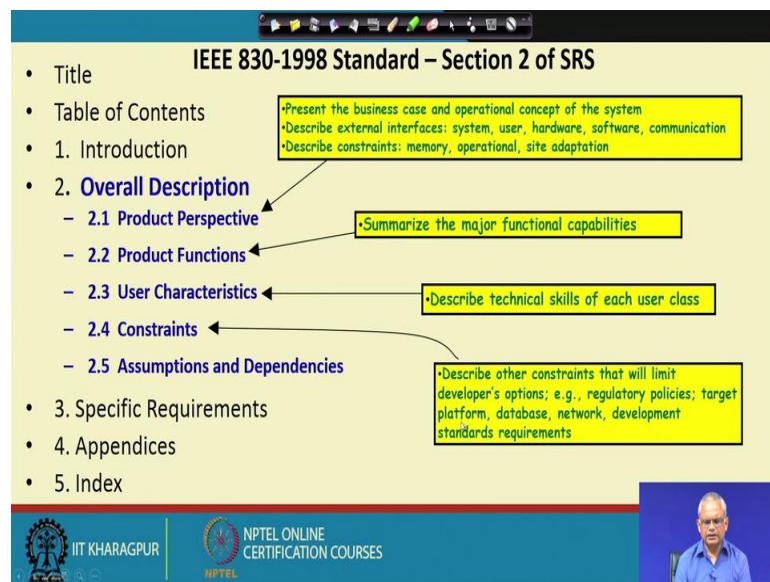
Now, let's look at the IEEE 830 standards. This is a standard format of writing the SRS document. Most developers use this standard and some developers also use small variations to the standard. But if we know the IEEE 830 standard should be able to understand also the small variations of this. Now let's look at sections of the IEEE 830 standard: title that is name of the software, table of contents and then, there are three main sections: one is the Introduction, Overall Description and the Specific Requirements and then, there are the Appendices and Index.

The introduction should have some subsection called as Purpose, Scope, Definition, Acronym, Abbreviations, References and Overview. The purpose should say that what the system will solve. So, this is general introduction and describe who are the users. Scope give very overall idea about what the system will do; what it will not do, very crisply written and overall description of what is expected of the system and what it should not do. Then, there are the various terminologies; the definitions, acronyms, abbreviations that will be used in the SRS document.

But, often variation to the standard, these are put under the appendices. Similarly, a setup documents that are referred in the SRS document. These references can also be put under a separate appendix and then, an overview of the document. Overview describe that how the SRS is organized. So, all these subsections are part of the introduction.

Now, let us look at the overall description. This is the section 2.

(Refer Slide Time: 10:32)



The sub sections under the Overall Description are the Product Perspective, Product Functions, User Characteristics, Constraints, Assumptions and Dependencies.

In the product perspective, we write here the business case that is how it will help the customer; how it will benefit, how it will be used and so on. Here we also write about the external interfaces, briefly the overview of what type of users, hardware, software et cetera will be used and then also write any constraints in the memory, operational constraints, site constraints and so on.

Then the product functions this not really the functional requirement specification but a brief overview or a summary of the functional capabilities. For example, in a library, we might just write that the library software should serve the users for their issuing of books, returning of the books, for the librarian to create members, manage books and so on and for the accountant to manage the financial aspects of the library etc.

The user characteristics are the technical skills of each user class. This is also required because finally, the user interface development we will need the user characteristics. If it is software is to be used by factory workers as well and programmers, we need to have different user interfaces. So, need to identify the user characteristics and how much what are the technical skills, how will their convergent with the computers and so on.

The constraints here, what are the platforms that will be used, the database that will be used, the network development standards and so on.

(Refer Slide Time: 13:05)

IEEE 830-1998 Standard – Section 3 of SRS (1)

- ...
- 1. Introduction
- 2. Overall Description
- 3. Specific Requirements
 - 3.1 External Interfaces
 - 3.2 Functions
 - 3.3 Performance Requirements
 - 3.4 Logical Database Requirement
 - 3.5 Design Constraints
 - 3.6 Software System Quality Attributes
 - 3.7 Object Oriented Models
- 4. Appendices
- 5. Index

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And then, if there are any assumptions or dependencies need to mention that in the Assumptions and Dependencies subsection.

Now we will look at the section 3. This is possibly the most important section. Here we have the external interfaces subsection, functions subsection. Functions subsection is the major section here. This contains the functionalities of the functional requirement.

The performance requirement, Logical Database Requirement, Design Constraints, Quality Attributes and any Object-Oriented models are also the subsections of Specific Requirements. This section has to be written very carefully and in sufficient detail so that the designers can have all the information they need to carry out the development work and the testers are able to write the test cases. It's a very important section.

While writing the functional requirement, we should make sure that we have all the input and output that is produced by the system and the functionalities.

(Refer Slide Time: 14:38)

The slide title is "IEEE 830-1998 Standard – Section 3 of SRS (2)". The main menu includes sections 1. Introduction, 2. Overall Description, 3. Specific Requirements (with sub-sections 3.1 to 3.7), 4. Appendices, and 5. Index. Callout boxes provide detailed descriptions for each sub-section:

- 3.1 External Interfaces: "Detail all inputs and outputs (complement, not duplicate, information presented in section 2). Examples: GUI screens, file formats"
- 3.2 Functions: "Include detailed specifications of each use case, including collaboration and other diagrams useful for this purpose"
- 3.3 Performance Requirements: "Types of Data entities and their relationships"
- 3.4 Logical Database Requirements: "Standards compliance and specific software/hardware to be used"
- 3.5 Design Constraints: "Class Diagram, State and Collaboration Diagram, Activity Diagrams etc."
- 3.6 Software System Quality Attributes
- 3.7 Object Oriented Models

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video feed of a speaker.

Now, first let's look at the external interface. Here, we have to write about all the user interface maybe you can give a GUI screenshots or maybe you can give the different file formats that may be needed. It should actually have some part that were mentioned in the section 2; external interface actually elaborates that and should not duplicate information here.

Functional requirements is the main section under Specific Requirements. This will contain the detailed specification of each functionality or use case including collaboration and other diagrams. We will see these diagrams later.

Then, the Performance Requirements and then the Logical Database Requirement. what are the different data that will be stored in the database, what will be their relationship; all these things are part of Logical Database Requirement. Next, is the Design Constraints. Design Constraints are including of Performance standard complaints. For example, we may say that have to use UML 2.0 or let say MISRA C for writing the code. Next, is the Software System Quality Attributes and then Object-Oriented Models which include class diagram, state collaboration diagram, activity diagram et cetera.

(Refer Slide Time: 16:26)

IEEE 830-1998 Standard – Templates

- Section 3 (Specific Requirements) can be organized in several different ways based on
 - Modes
 - User classes
 - Concepts (object/class)
 - Features
 - Stimuli

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The section 3 that we just saw now is just one example and IEE 830 specifies that there can be different organization of the section 3 based on the modes of the software that is software maybe having an expert mode, a novice mode and so on. If there are modes presents then we might have to organize it such that the SRS document is written in terms of the modes.

Similarly, depending on different user Classes, Concepts, Features, Stimuli there can be slightly different organization of the section 3. We just looked at a very typical organization of the section 3 but there can be variations of the section 3.

(Refer Slide Time: 17:32)

The slide is titled "Example Section 3 of SRS of Academic Administration Software". It lists "SPECIFIC REQUIREMENTS" under section 3.1 Functional Requirements, subsection 3.1.1 Subject Registration. The requirements are:

- **SPECIFIC REQUIREMENTS**
- **3.1 Functional Requirements**
- **3.1.1 Subject Registration**
 - The subject registration requirements are concerned with functions regarding subject registration which includes students selecting, adding, dropping, and changing a subject.
 - **F-001:**
 - The system shall allow a student to register a subject.
 - **F-002:**
 - It shall allow a student to drop a course.
 - **F-003:**
 - It shall support checking how many students have already registered for a course.

Now, let's look at the functional requirements in more details. As we said that this is one of the most important part of the document. Let's take an example of an Academic Administration Software or AAS. So, first we write Functional Requirements at section 3.1 and then, we write various functional requirements and they might have sub functions. For example, we organize the subject registration at subsection 3.1.1 and then there can be several functionalities for subject registration. In the first function, may be the system shall allow a student to register subject. Second function can be allows a student to drop a course. Third one is that it supports checking how many students have already registered for a course.

(Refer Slide Time: 18:40)

The slide has a yellow header bar with the title "Design Constraints (3.2)". Below the title is a bulleted list of constraints:

- **3.2 Design Constraints**
- **C-001:**
 - AAS shall provide user interface through standard web browsers.
- **C-002:**
 - AAS shall use an open source RDBMS such as Postgres SQL.
- **C-003:**
 - AAS shall be developed using the JAVA programming language

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "97".

Similarly, we write Design Constraints at 3.2; Then we give a number to each constraint which under Design Constraints. First constraint is that AAS shall provide user interface through standard web browsers. Second constraint is that it should use an open source RDBMS. Third constraint we wrote that it should be developed using JAVA programming language. So, all these become different constraints.

(Refer Slide Time: 19:09)

The slide has a yellow header bar with the title "Non-functional requirements". Below the title is a bulleted list of requirements:

- **3.3 Non-Functional Requirements**
- **N-001:**
 - AAS shall respond to query in less than 5 seconds.
- **N-002:**
 - AAS shall operate with zero down time.
- **N-003:**
 - AAS shall allow upto 100 users to remotely connect to the system.
- **N-004:**
 - The system will be accompanied by a well-written user manual.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a person speaking.

And then, the Non-Functional Requirements. Here, again we give a number to each requirement under Non-Functional Requirements. The first non-functional requirement

we wrote is that it should respond to query in less than 5 second. Second one is that, it should operate with zero down time. Third one is that it should allow up to 100 users to remotely connect to the system. The third one is that the system should be accompanied by a well written user manual. So, these all are non-functional requirements.

(Refer Slide Time: 19:37)

Functional Requirements

- It is desirable to consider every system as:
 - Performing a set of functions $\{f_i\}$.
- Each function f_i considered as:
 - Transforming a set of input data to corresponding output data.

Input Data f_i **Output Data**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we will discuss more about functional requirements. For writing a good SRS document, we must know what are the granularity to which we must document the functions and what exactly are the functional requirements. Let's look at it in further detail. As we have already mentioned that every software, we can view it as performing a set of functions. Let say a software. Let say the software supports three functions: f_1 , f_2 and f_3 . Using each function, the user should be able to achieve something and each function will typically take some input and the system should process that input and produce some output. You can think of each function as taking some input data and through some processing activity or a function transfer the input data turns to some output data. Let's take an example of a query book, takes the name of the book as the input data. Searches the book database and produces all the books details that match with the given book name.

(Refer Slide Time: 21:52)

Example: Functional Requirement

- F1: Search Book
 - Input:
 - an author's name:
 - Output:
 - details of the author's books and the locations of these books in the library.

The diagram illustrates a functional requirement F1: Search Book. It shows two yellow rounded rectangles representing data entities: 'Author Name' on the left and 'Book Details' on the right. An arrow points from 'Author Name' to 'Book Details', with the label 'f1' written vertically along the arrow. This visualizes how the input (author's name) is processed by the function to produce the output (book details).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video player window in the bottom right corner shows a man in a blue shirt speaking, likely providing further explanation or context about functional requirements.

Now, let takes some other example let say the search book. The search book let say it takes author's name. So, then we can write the input is author name and the output are the details of the authors books and the locations of this book in the library.

(Refer Slide Time: 22:13)

- Functional requirements describe:
 - A set of high-level requirements
 - Each high-level requirement:
 - takes in some data from the user
 - outputs some data to the user
 - Each high-level requirement:
 - might consist of a set of identifiable sub-functions

The diagram illustrates the concept of functional requirements. It shows a main heading 'Functional Requirements' in a yellow box, followed by a list of bullet points describing its components. The first point is 'A set of high-level requirements'. The second point, 'Each high-level requirement:', is expanded into two sub-points: 'takes in some data from the user' and 'outputs some data to the user'. The third point, 'Each high-level requirement:', is further expanded into a single sub-point: 'might consist of a set of identifiable sub-functions'. This visualizes how a general requirement (set of high-level requirements) can be broken down into more specific components (sub-functions).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video player window in the bottom right corner shows a man in a blue shirt speaking, likely providing further explanation or context about functional requirements.

Each functional requirement we called as a high-level requirement. A high-level requirement takes some data and produce some result but then, each high-level requirement may consists of sub requirements or sub functions.

(Refer Slide Time: 22:44)

The slide has a yellow header bar with the title 'Functional Requirements'. The main content area is light yellow and contains the following bullet points:

- For each high-level requirement:
 - A function is described in terms of:
 - Input data set
 - Output data set
 - Processing required to obtain the output data set from the input data set.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man speaking.

Each functional requirement, is described in terms of input data. Function uses the input data to produce the output data.

(Refer Slide Time: 23:03)

The slide has a yellow header bar with the title 'Is it a Functional Requirement?'. The main content area is light yellow and contains the following bullet points:

- A high-level function is one:
 - Using which the user can get some useful piece of work done.
- Can the receipt printing work during withdrawal of money from an ATM:
 - Be called a functional requirement?
- A high-level requirement typically involves:
 - Accepting some data from the user,
 - Transforming it to the required response, and then
 - Outputting the system response to the user.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man speaking.

We had already said that high-level function is one using which the user can get some useful piece of work done. But suppose, we take the example of ATM machine. What are the functionalities or the requirements of this ATM machine need to have? One can be withdrawn cash, one can be query balance, one can be deposit cash etc. But let say during the withdrawal cash, it asks whether to print a receipt or not? Is printing the

receipt, can be considered as a functional requirement? The answer is no because as per our definition of a functional requirement, it takes some input from the user and produces some useful piece of work for the user. The receipt was actually printed as part of the withdrawal. It's not that just printed the receipt as the user requested the print receipt and it printed. It's always executed as part of the cash withdrawal and therefore, it is a sub function of the cash withdrawal requirement. Every high-level requirement, typically accepts some data from user, transforms its response and then output these responses to the user.

(Refer Slide Time: 25:18)

The slide has a title 'Use Cases' in bold black font at the top right. To its left is a bulleted list:

- A use case is a term in UML:
 - Represents a high level functional requirement.
- Use case representation is more well-defined and has agreed documentation:
 - Compared to a high-level functional requirement and its documentation
 - Therefore many organizations document the functional requirements in terms of use cases

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man in a blue shirt.

Little later, we look at the UML, where the requirements are specified using use cases. The use cases have become standard. These have definite notation and these are much more well defined than just loosely writing the requirements.

So, many times the requirements are specified using use cases.

(Refer Slide Time: 25:57)

The slide has a yellow background with a black header bar at the top containing standard window controls. The title 'Example Functional Requirements' is centered in bold black font. Below the title is a bulleted list under the heading '• Req. 1:'. The list details requirements for a search function, including keyword entry, output of book details matching keywords, and specific details like title, author, publisher, year, ISBN, catalog number, and location.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Now, let's look at some few examples of functional requirement. Here in this example we numbered the requirement. In Req. 1 we wrote that once the user selects the search option asked to enter the keywords. The system should output details of all books whose title or author match the keywords and the details that will be output includes the title, author name, publisher name, year of publication et cetera.

(Refer Slide Time: 26:33)

The slide has a yellow background with a black header bar at the top containing standard window controls. The title 'Example Functional Requirements' is centered in bold black font. Below the title is a bulleted list under the heading '• Req. 2:'. The list details requirements for a renew function, including password validation, displaying borrowed books, and renewing books by clicking a box.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Similarly, we numbered the second requirement: Req. 2. Here we wrote that when the renew option is selected the user is asked to enter the membership number and password.

After password validation list of books borrowed by him are displayed and then, the user can renew the book by clicking on the corresponding renew box.

Just see that this is written in rather free form. It is written in the way that the sequence of operations are carried out that initially the renew option is selected, this is done by the user and the computer ask the user to enter the membership number and password and after password validation, the books borrowed by him are displayed and then the user can renew the book by clicking the corresponding renew books. So, we can see all these are sequence of operations.

(Refer Slide Time: 27:35)

High-Level Function: A Closer View

- A high-level function:
 - Usually involves a series of interactions between the system and one or more users.
- Even for the same high-level function,
 - There can be different interaction sequences (or scenarios)
 - Due to users selecting different options or entering different data items.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES
NPTEL

We will see now closer view of high-level function. We can realize that every high-level function actually is a series of interaction between the system and the users. The user input something based on that the system does some processing; produces some output and based on that the user enters certain other thing and so on. So, we can think of every high-level function as a series of interactions between the user and the computer. But for every functional requirement, we must also document the scenarios. The scenarios arise because there are different ways certain things can get accomplished. For example, let say the cash withdrawal this is a functional requirement and then, the normal course of cash withdrawn is that the user specifies the amount to be taken withdrawn and the system dispenses the cash. There can be other scenarios that the user entered the amount to be withdrawn and the system should display that it exceeds the account balance. There

can be another scenario that the user entered some amount in the system responded that please enter in multiples of 100. So, we have looked that every software performs a set of functions for the users and this has to be carefully documented and all the high-level requirement with scenarios also need to be documented.

We look at some examples in the next lecture and we will continue from that point onwards.

Thank you.

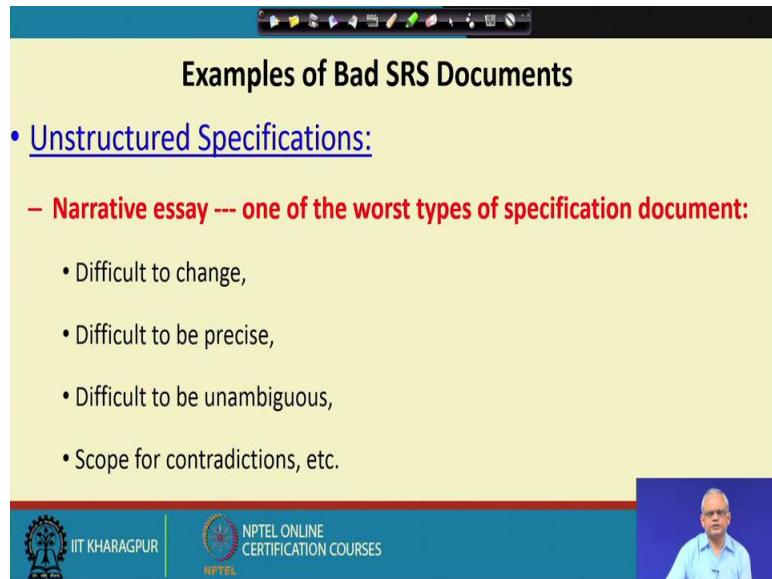
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 18
Representation of complex programming logic

Welcome to this lecture. So, for in the over the last few lectures we had seen that requirement specification is a very important task, during software development. We had looked at how requirements gathering analysis and specification can be done. We also looked at the standard specification template that is IEEE 830.

In this lecture, we will look at few other aspects of the requirement specification before we conclude this topic. Now we will discuss about some of the desirable properties of requirement specification document. We will see what should be or should not be there and how to write it or how not to write it. Let us proceed with that objective.

(Refer Slide Time: 01:24)



The slide has a yellow background. At the top, there is a navigation bar with icons. Below it, the title 'Examples of Bad SRS Documents' is centered. A bullet point leads to a sub-point in red text: '– Narrative essay --- one of the worst types of specification document:'. Below this, there is a list of five bullet points: '• Difficult to change,'; '• Difficult to be precise,'; '• Difficult to be unambiguous,'; and '• Scope for contradictions, etc.' At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' next to the NPTEL logo. On the right side of the footer, there is a small video window showing a man speaking.

Let's look at some examples of bad SRS document. The worst form of document possibly is just an essay. An essay is an unstructured specification. It is just a description of what is required. This is one of the worst types of specification, because it has lot of problems.

Typically, if you want the customer to write about their requirements, they would write in the form of a narrative essay. Here, various types of functional, nonfunctional requirements constraints are all mixed up. Let's look at what would be the problems? If somebody just uses this kind of a document and narrative essay as a specification document, one of the most difficult problems that will be faced is that it is difficult to change, because various issues the functional requirement, non-functional requirements, constraints et cetera are all mixed up.

Let say we want to change the functionality, during the development or maybe after development. It would become very difficult, because this functionality would not be described just at one place. It, would be all mixed up over various phases and to find out where need to change, will be difficult. Maybe we can change few but it may happen that we leave most of those. So, changing is a very tedious task and unless you are careful the changes wouldn't be proper. It's difficult to be precise, because it is all mixed up just a narrative essay and we write the issues as it occurs to the mind. And therefore, we need to refer to various places to find out the description of issues.

So, it is all interspersed and it becomes very descript. It's difficult to become unambiguous, when there is a large set of sentences all mixed up, it can happen that if we read some sentences in isolation for some functionality, it can give an ambiguous meaning. There is also scope for contradiction because as we write a large document forget what was written earlier and it is easy to contradict.

If, such problems exist in a document, then it becomes very difficult for the developers, for the testers to carry out their task. As we know that requirements change a great deal during development and after development. During the development on the average about 40 percent of the requirements are expected to change. So, its become very tedious to make changes if all are mixed up in a document.

Even after the development the requirements continue to change. If it is an unstructured specification, it becomes very cumbersome. It consumes large effort cost to change the document and yet there will be several mistakes and problems. And, that's one of the reasons that hardly there is any commercial organization, who would write their requirement specification as a narrative essay.

(Refer Slide Time: 05:51)

Examples of Bad SRS Documents

- Noise:
 - Presence of text containing information irrelevant to the problem.
- Silence:
 - Aspects important to proper solution of the problem are omitted.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another, very undesirable aspect of a bad SRS document is noise. Noise is something which is unrelated to the problem. Basically, when the document is written in a very verbose style there is a tendency to put information, which is not really part of the problem being solved. And, this makes it difficult for the developers and testers to figure out that which sentences to ignore and which sentences to take. Another, very undesirable aspect of a requirement document is silence. Silence is basically omitting some important issues maybe functional requirement, non-functional requirements, constant interfaces and so on.

(Refer Slide Time: 07:01)

Examples of Bad SRS Documents

- Overspecification:
 - Addressing “how to” aspects
 - For example, “Library member names should be stored in a sorted descending order”
 - Overspecification restricts the solution space for the designer.
- Contradictions:
 - Contradictions might arise
 - if the same thing described at several places to mean different things

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Overspecification is another very undesirable characteristic. By over specification we mainly addressing on “how to” aspects? For example, let say we specified how to store the library member names in a sorted descending order in a file? So, what happened is once we write it in the SRS document it becomes binding and the designers have very little flexibility. If we already mentioned ‘how to do’ then designers have to obey the mentioned way. So, this stored in a sorted descending order in a file then they cannot use a database management system. And, also, they have to keep it in a descending order even if for some other task they may need in ascending order. So, the “how to” aspect should be carefully avoided, while writing the SRS document.

Another, problem with the bad SRS document is contradictions. Contradictions might arise when same thing described in several places. Specifically, this kind of problem occur in narrative essay types specification, where contradictions between various parts of the document exist and this is to be consciously avoided.

(Refer Slide Time: 08:46)

The slide has a yellow header bar with the title 'Examples of Bad SRS Documents'. Below the title are three bulleted sections: '• Ambiguity:' with two sub-points ('– Literary expressions' and '– Unquantifiable aspects, e.g. "good user interface"'); '• Forward References:' with one sub-point ('– References to aspects of problem' followed by a bullet point '• defined only later on in the text.'); and '• Wishful Thinking:' with two sub-points ('– Descriptions of aspects' followed by a bullet point '• for which realistic solutions will be hard to find.'). At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

Another undesirable characteristic of bad SRS document is Ambiguity. Ambiguity occurs when there is anything mention in the document which cannot be quantified, becomes very difficult to test. For example, if we write in the document that it should have a good user interface. Then the tester the developers will all be in confusion what is meant by good user interface? How does somebody test and say that it is a good user interface?

Another problem with the SRS document is forward reference. As we read the document, we only know what has been already written. And, if we write in the document that referred to a later section, then somebody would have to turn pages read that and again come back and if that happens many times, becomes really difficult for somebody to read it.

One more problem is wishful thinking. If the person writing the specification document is not careful, he might agree to some functional requirements, which cannot be implemented, realistic solutions to these, cannot be obtained. And therefore, it's important that the analyst as a person who is writing the document should have an idea that how to solve the functionality that is required or a non-functional requirement. Commitment to unrealistic requirements, becomes very difficult and later would have to again change the requirement after the developers have tried implementing them.

(Refer Slide Time: 10:49)

The screenshot shows a presentation slide with a yellow header bar containing various icons. The main title is "Suggestions for Writing Good Quality Requirements". Below the title is a bulleted list of six items:

- Keep sentences and paragraphs short.
- Use active voice.
- Use proper grammar, spelling, and punctuation.
- Use terms consistently and define them in a glossary.
- To see if a requirement statement is sufficiently well defined,
 - Read it from the developer's perspective

At the bottom of the slide, there is a blue footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the footer bar, there is a small video thumbnail showing a man speaking.

Let's just discuss a few suggestions on how to write good quality requirement document? One thing about the requirement document is that there are many stakeholders who would like to read the document for various purposes and therefore, readability of the document is an important concern. Sentences should be short; paragraph should be short. A sentence which is 20 lines in a long would become very difficult to read the sentences. So it should be simple and typically one line and so on.

Similarly, the paragraph should be small paragraphs. Active voice should be used, use of proper grammar, spelling and punctuation is required. The terms should be consistently used and should be defined in a glossary. So, that somebody who does not understand a term as reading through the document, can refer to the glossary.

And, also before completing writing the document, the reviewer, the developer of the document, analyst, and also the reviewers read it from the perspective of the user and also from the perspective of the developer. From the perspective of the user understandability, readability is very important and from the perspective of the developer sufficient details should be there and from the perspective of the tester, it should be possible to write test cases to test the requirement.

(Refer Slide Time: 12:56)

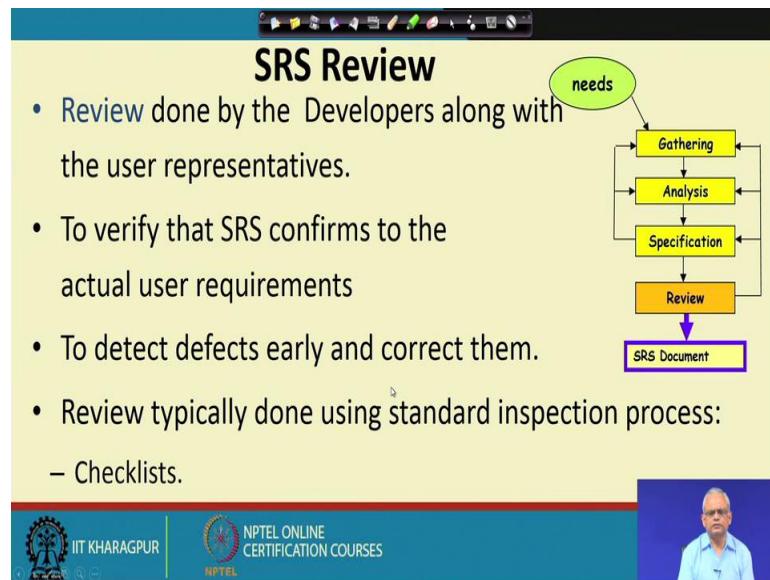
Suggestions for Writing Good Quality Requirements

- Split a requirement into multiple sub-requirements:
 - Because each will require separate test cases and because each should be separately traceable.
 - If several requirements are combine together in a paragraph, it is easy to overlook some during development or testing.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Sometimes the requirements are complex, but in those cases, it is important to split the requirement into sub requirements. One of the major reasons for doing that is that traceability. Since, it is a complex large requirement, its implementation will require many design elements, many modulus or functions in the code, but if we have split it into sub requirements. Then, each sub requirement can be traced into a few design elements and a few functions modules in the code. Also, it becomes easier for the tester to write test cases, because for every sub requirement the tester can write test cases. Otherwise if it a huge complex requirement then tester may overlook some of the aspects to test.

(Refer Slide Time: 14:10)



In the requirement specification process, we said that the first thing is requirement gathering and then analysis of the requirements in an iterative manner. And then finally, trying to write the specification document using a standard template, for that again we might have to do some gathering and analysis.

But, after the requirement document is completely written, it submitted for review. Review team typically consists of the customer, the developer, tester, basically all stakeholders. The customers check whether the requirements confirm to their actual requirements. The developers see if any details are missing. The tester sees if every requirement can be tested meaningfully and the defects that are detected by the reviewers can be corrected. Typically, the review is done by developing a checklist and it's checked whether every element in the checklist is satisfied by the SRS document or not.

(Refer Slide Time: 15:49)

Representation of complex processing logic

- Decision trees
- Decision tables

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let's look at the processing logic. If the processing logic is complex how to meaningfully represent it? The processing logic is that, if certain conditions are satisfied, then do something and check for further conditions and so on.

If, this processing logic is very complex then just text description becomes very difficult to understand and it also become very difficult for the developers to be able to meaningfully develop solutions and also for the testers to write test cases. And, also in a text form if we write a very complex processing logic it is likely that some of the conditions can be overlooked. And, that is the reason why some semi formal techniques are used for representing complex processing logic. Two techniques are very popular decision trees and decision tables.

(Refer Slide Time: 17:04)

Decision Trees

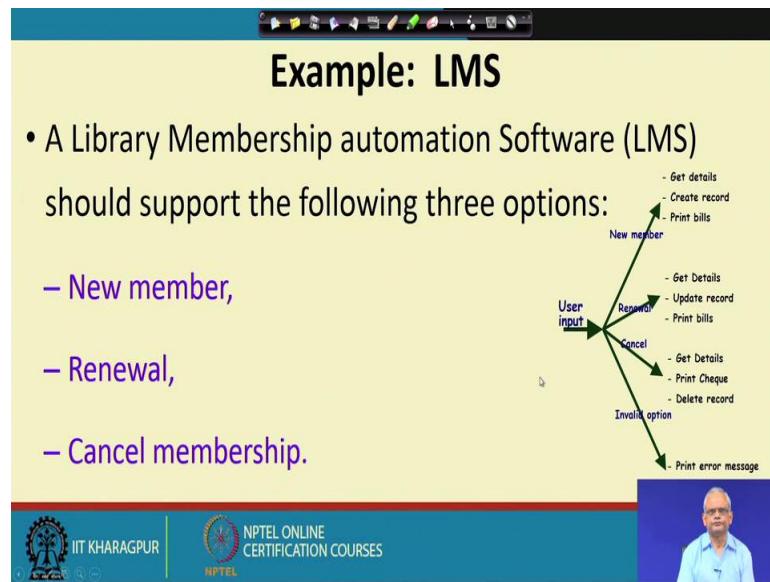
- Decision trees:
 - Edges of a decision tree represent conditions
 - Leaf nodes represent actions to be performed.
- A decision tree gives a graphic view of:
 - Logic involved in decision making
 - Corresponding actions taken.

```
graph LR; UserInput[User input] --> NewMember[New member]; UserInput --> Renewal[Renewal]; UserInput --> Cancel[Cancel]; UserInput --> InvalidOption[Invalid option]; NewMember --> GetDetails1["- Get details  
- Create record  
- Print bills"]; Renewal --> GetDetails2["- Get Details  
- Update record  
- Print bills"]; Cancel --> GetDetails3["- Get Details  
- Print Cheque  
- Delete record"]; InvalidOption --> PrintErrorMessage["- Print error message"];
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Both the techniques, are not specific to this domain there used across various other disciplines. But still let see how it can be used to represent complex processing logic. These are very simple concepts. In decision tree, each decision is represented by an edge of the tree (As can be seen in slide). So, junction here represents some decision to be checked and depending on the decision some actions are taken. And, this gives it a tree like appearance that every node of the tree some condition is check and depending on which condition is satisfied that specific branch is taken. And, the leaf nodes here on the tree represents the actions to be taken. So, if it satisfies certain conditions on the path, then these actions should be taken. One of the advantages of the decision tree is that it gives a graphic view of the logic that is involved and the actions that will be taken, if certain logic conditions hold and from the tree we can easily detect if certain conditions are missed.

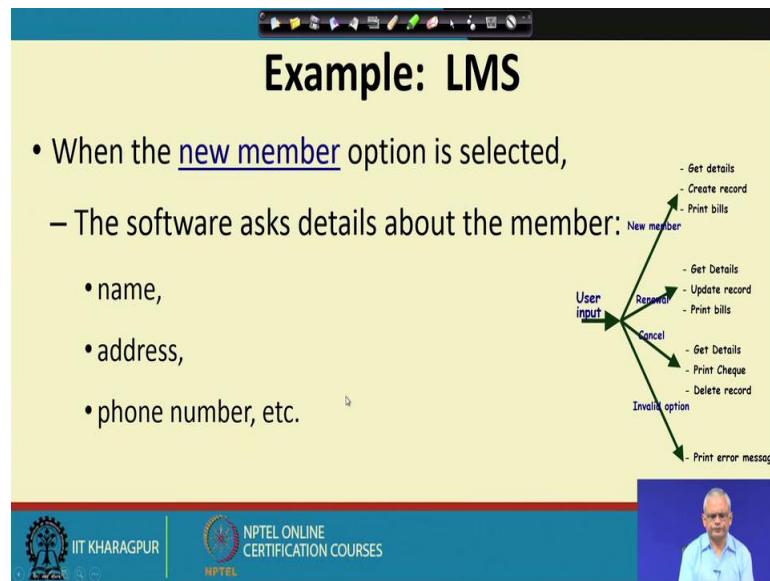
(Refer Slide Time: 18:48)



Let's take an example of a library membership automation software. Here, if a user selects create member then there are three options which come out: New member, membership renewal and cancel membership. Let see first description of the problem then we will see the solution.

So, the problem is that this is a requirement create member and as the create member option is chosen there are three sub options, which are displayed: Create new member, membership renewal, and cancel membership.

(Refer Slide Time: 19:56)



If, the new member is selected then it just asks some details about the member like name, address, phone number etcetera and then creates a record and then prints the bill. So, during in the menu if we select the new member the actions taken is get the details of the member, create the record and print the bill.

(Refer Slide Time: 20:35)

The slide has a blue header bar with standard window controls. The main content area has a light yellow background. At the bottom, there is a dark blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a blue shirt.

Example (cont.)

- If proper information is entered,
 - A membership record for the member is created
 - A bill is printed for the annual membership charge plus the security deposit payable.

So, that's what it says that once the details are obtained membership record is created and the bill is printed for the membership charge.

(Refer Slide Time: 20:46)

The slide has a blue header bar with standard window controls. The main content area has a light yellow background. At the bottom, there is a dark blue footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man in a blue shirt.

Example (cont.)

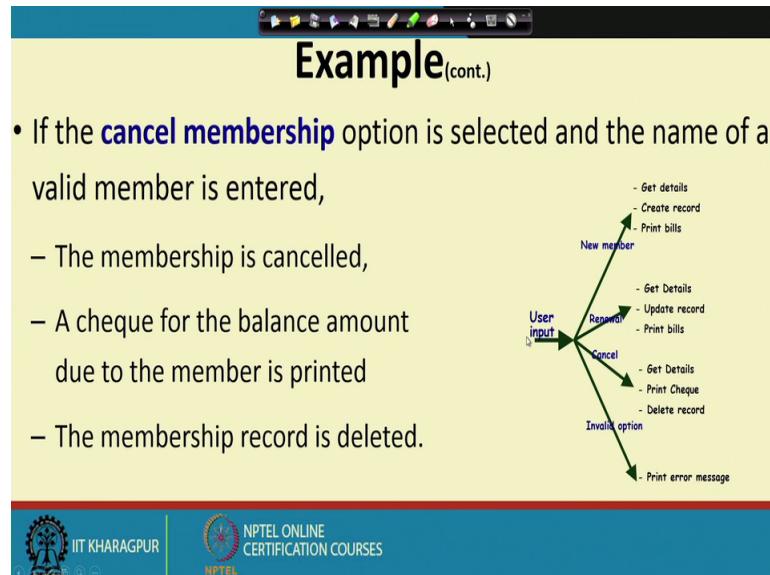
- If the renewal option is chosen,
 - LMS asks the member's name and his membership number
 - checks whether he is a valid member.
 - If the name represents a valid member,
 - the membership expiry date is updated and the annual membership bill is printed,
 - otherwise an error message is displayed.

A flowchart is shown on the right side of the slide:

```
graph LR; UserInput[User input] --> Renewal[Renewal]; UserInput --> Cancel[Cancel]; UserInput --> Invalid[Invalid option]; Renewal --> NewMember[New member]; Renewal --> UpdateRecord[Update record]; Renewal --> PrintBills[Print bills]; Cancel --> GetDetails[Get Details]; Cancel --> PrintCheque[Print Cheque]; Cancel --> DeleteRecord[Delete record]; Invalid --> Error[Error]
```

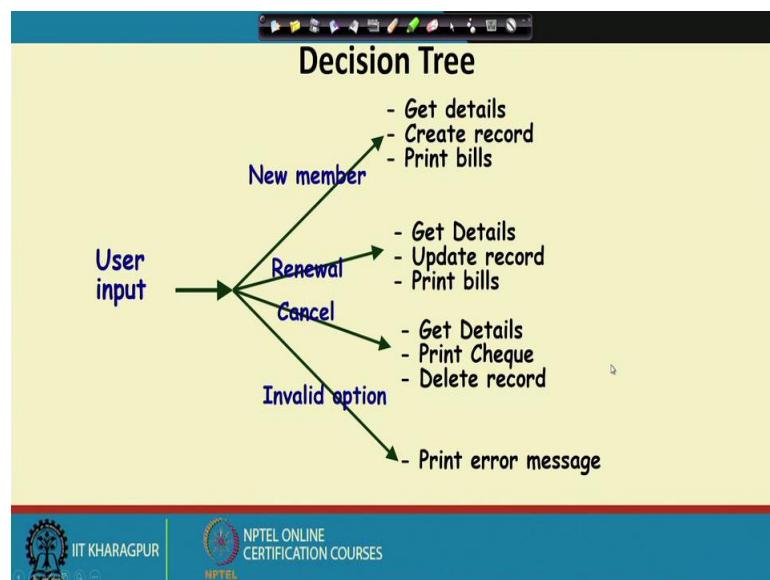
If the renewal option is chosen then the name, expiry date is updated and the bill for the renewal is printed.

(Refer Slide Time: 21:07)



Similarly, if the cancel membership option is chosen, then the details like name, address et cetera are obtained and the balance amount is printed on a cheque and then the membership is deleted.

(Refer Slide Time: 21:31)



If any other option is chosen, then there is a printer error message. So, this is a very simple representation. It may possible for some complex processing logic there can be many decisions at each level and this will become a multilayer tree.

(Refer Slide Time: 22:00)

The slide has a yellow header with the title "Decision Table". Below the title is a bulleted list:

- Decision tables represent:
 - Which variables are to be tested
 - What actions are to be taken if the conditions are true,
 - The order in which decision making is performed.

Handwritten notes are overlaid on the slide, specifically a grid diagram labeled "Action" and "Condition" with arrows indicating flow between them.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video window showing a speaker.

Another alternative is the decision table. The decision table represents an alternative way of complex processing logic. Here, we checked, which variables are to be tested to determine which condition is satisfied and then what actions to be taken if the conditions are true and the order of the decision making. So, it's a table as the name says and the first few rows of the table actually define the conditions. Here, V_1 , V_2 et cetera are variables and each values of the variables define a condition. The lower part of the table defines the actions.

So, actions can be A_1 , A_2 , A_3 et cetera. So, if these set of conditions are satisfied that is some variable has some specific values, then maybe we will say that action A_1 will be taken. And, if the conditions satisfied for some other set of values, then let say action A_3 will be taken and so on.

So, here if you notice that the upper part of the table this represents the conditions. Condition are specified in terms of certain variables having certain values. So, V_1 , V_2 , V_3 et cetera these are some variables and they have some values. These can be Boolean variables, which can be let say on, off et cetera. These can be integer variables also. And,

the lower part of the table represents the actions: if the variables have certain values, then the corresponding action is taken. Each of these column of the table is called a rule.

(Refer Slide Time: 25:26)

Decision Table

- A decision table shows in a tabular form:
 - Processing logic and corresponding actions
- Upper rows of the table specify:
 - The variables or conditions to be evaluated
- Lower rows specify:
 - The actions to be taken when the corresponding conditions are satisfied.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, Decision Table is a tabular representation unlike the tree like representation in decision tree. Typically, these becomes very compact representation than the tree. The tree is tend to become large for very complex processing logic, but here it's a compact representation. In the upper rows of the table, all the variables and their values are mentioned. Typically all possible combinations of the conditions and values should be mentioned. So, for example, let say C_1 and C_2 are two conditions variable, then we have to check for all combination of C_1 and C_2 . Let say, C_1 is true and C_2 is false.

Similarly, C_1 is false C_2 is true; C_1 is true, C_2 is true and C_1 is false, C_2 is false. All possible combinations of the condition should be written and this forms the rule and what is the action to be taken also need to be written. Let say, if C_1 is true and C_2 is false then action A_1 should be taken. So, all those actions are specified below. And, each of these condition and respective action becomes a rule. We discussed here Boolean conditions, but then if there is a integer variable, then we have to consider all possible combinations of the variables.

You can try representing this in a decision tree. One of the difficulties in a decision tree is that it become difficult to check whether all possible combinations of conditions have been taken care or not. And, also the trees are not very compact. Whatever we represent

in the form of a slightly larger decision table, a decision tree may spillover couple of pages. And, also for a given table, it becomes very easy to design test cases for each rule. We write each rule here becomes one test case.

(Refer Slide Time: 28:14)

Decision Table

- In technical terminology,
 - A column of the table is called a **rule**.
 - A **rule implies:**
 - If a condition is true, then execute the corresponding action.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Each column of the table, is called as a rule and each rule implies that if some conditions are true or false then execute the corresponding action.

(Refer Slide Time: 28:24)

Conditions				
	NO	YES	YES	YES
Valid selection	NO	YES	YES	YES
New member	--	YES	NO	NO
Renewal	--	NO	YES	NO
Cancellation	--	NO	NO	YES

Actions				
Display error message	--	--	--	--
Ask member's name etc.	--	--	--	--
Build customer record	--	--	--	--
Generate bill	--	--	--	--
Ask membership details	--	--	--	--
Update expiry date	--	--	--	--
Print cheque	--	--	--	--
Delete record	--	--	--	--

Example

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The same library system we saw in the example of decision tree, we will solve the same problem using decision table representation. So, here the conditions are whether it is a

valid selection, new member, it's a renewal or cancellation. If, it's not a valid selection, then it is immaterial, it's don't care for all other three conditions. And, then the specific action to be taken is to print an error message or display an error message. If, it is a valid selection and then it is a new member then it should print ask for the members name then build customer record and then generate bill. Similarly, for if it is a valid selection and say renewal, then we have to update the expiry date, and then display message. Similarly, if it is a cancellation then update, delete record and print cheque.

As, we can see that decision table, decision tree both are very simple representation of complex processing logic. It has the advantage that if anything missed during specification can be easily checked here. It is easier to understand the decision tree or a decision table and also it helps the developers and the testers.

Now, we are running out of time. We will stop here and continue in the next lecture.

Thank you.

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

Lecture – 19
Design Fundamentals

Welcome to this lecture. In the last lecture, we are discussing about representing complex processing logic. If we write the complex processing logic in the form of a text description, then it becomes difficult for the developers to understand, for the testers to develop test cases. And also, it is very likely that in a text base description of a processing logic, certain combinations of combinations or variables and their corresponding action might get missed.

And therefore, it's a good idea that during requirement specification, if we find that the logic is likely complex; then we have to represent it in the form of a decision or a decision table. We had seen that these are very simple semiformal techniques and we can very easily develop the decision tree and decision table for any problem. Let's take an exercise.

(Refer Slide Time: 01:53)

Exercise

- Develop decision table for the following:
 - If the flight is more than half-full and ticket cost is more than Rs. 3000, free meals are served unless it is a domestic flight. The meals are charged on all domestic flights.

Flight ID	Ticket Cost	Occupancy	Free Meal	Charged
Y	N	N	✓	✓
Y	N	Y	✓	✓
Y	Y	Y	✓	✓
Y	Y	N	✓	✓
Y	Y	Y	✓	✓

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

A small video player window in the bottom right corner shows a man speaking.

We want to develop decision table and decision tree for the problem given in slide. Just see here that the problem statement is very simple; but still the developers and testers might find it bit confusion. If the flight is more than half-full and the ticket cost in the

flight is more than 3000 rupees, then free meals are served. But in domestic flight, this rule does not apply and all meals are charged on all domestic flights.

So, if you can see here that the one of the conditions here is that whether the flight is a domestic flight? So, our first condition is that ‘the flight is a domestic? yes or no’ and then another condition is that ‘ticket cost: ticket cost greater than 3000 or not?’ and then, we have to check whether ‘the flight is half-full occupancy: greater than 50 percent’.

The actions are free meal or charge meal. If it is a domestic flight, then these we will have to charge the meal; but the flight is not domestic and the ticket cost is let us say less than 3000, then also it will be charged meal. If the flight is not a domestic flight and the ticket cost is more than 3000 and occupancy rate is less than 50 percent, then again, we charge the meal. But if it is not a domestic flight and the ticket cost is more than 3000 and the occupancy is also more than 50 percent then we give free meal. So, as you can see here that this gives a conceptually meaningful representation of this conditions.

But we can see here that these all are very simple condition that we are trying to represent here as an example. The conditions can become much more complex and therefore, the true use of a decision table can come out. Exercise to do: develop the decision tree representation for the same problem. Let’s conclude our discussion on requirement specification and let’s proceed to look at software design aspects.

Now, we will see some very basic issues in software design and then, we look at the procedural and object-oriented design principles. We will look at now software design and we will start with some very basic issues in software design. The designed activities are undertaken once the requirement is complete.

(Refer Slide Time: 07:00)

What is Achieved during design phase?

- Transformation of SRS document to Design document:
 - A form easily implementable in some programming language.

The slide title is "What is Achieved during design phase?". Below the title is a bulleted list: "• Transformation of SRS document to Design document:" followed by a dash and "– A form easily implementable in some programming language.". Below the list is a graphic representation of the design phase. It consists of three main components: a blue rounded rectangle on the left labeled "SRS Document", a central blue circle labeled "Design Phase" and "Design Activities", and a purple rounded rectangle on the right labeled "Design Documents". An arrow points from the "SRS Document" to the "Design Phase" circle, and another arrow points from the "Design Phase" circle to the "Design Documents" rectangle. Above the "Design Documents" rectangle is a small icon of a book.

During the design phase, we have the SRS documents as the input and then do some activities during the design and then, we come up with the design document at the end of the design phase. The design document is taken up for coding purpose and our design document is good if the programmers can take it and can start writing the code easily. A graphic representation of the design phase is shown in the slide.

The circle in the slide represents design phase and the input of the design phase is the SRS document. We perform some design activities and then, at the end of the phase, we have the design document. Now let see at the end of the phase what are the documents we should have with us? What we should prepare during the design phase and then will address the question how to prepare those?

(Refer Slide Time: 08:43)

Items Designed During Design Phase

- Module structure,
- Control relationship among the modules
 - call relationship or invocation relationship
- Interface among different modules,
 - data items exchanged among different modules,
- Data structures of individual modules,
- algorithms for individual modules.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we will see items designed during design phase. Typically, we have the module structure in a procedural design or the class structure in an object-oriental design. In a procedural design, we have the control relationship among the modules and similarly in an object-oriental design, we have the invocation sequences among different classes. Call relationship or invocation relationship between the modules are basically control relationship among modules. Then we have the interface that is what data exactly is given as a parameter when there is a function call or a method call that we call as the interface among the modules. So, this is basically the data items that are exchanged between the modules. And also, we need to design the data structure of the individual modules and also the algorithms that would be used. So, these are all the different items that must be designed during the design phase.

The call relationship among the module in a object oriented way we will see it little later that this is a class structure, the invocation relationship among the classes. And then, the interface among the modules that is what parameters, they pass or the data items that are exchanged among the different modules. The data structures of the individual modules and then the algorithms for the individual modules.

(Refer Slide Time: 10:53)

The slide has a yellow background with a black header bar containing icons. The title 'Module' is centered at the top in a large, bold, black font. Below the title, there is a bulleted list:

- A module consists of:
 - several functions
 - associated data structures.

On the right side of the slide, there is a diagram enclosed in a blue border. The diagram is divided into three horizontal sections:

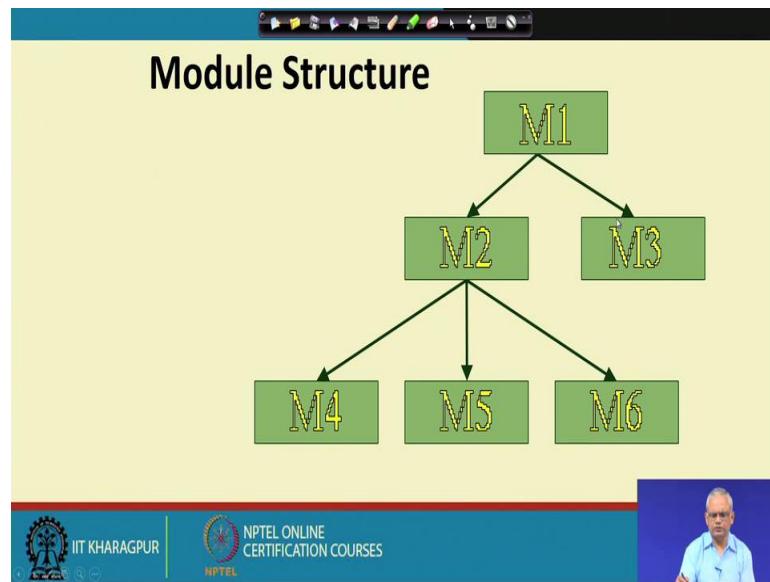
- The top section is green and contains the word 'Data' in yellow.
- The middle section is light blue and contains the words 'Functions' in yellow, followed by a list: F1 .., F2 .., F3 .., F4 .., F5 ..
- The bottom section is light blue and contains the word 'Module' in large, bold, black letters.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a person speaking.

So, far we have been referring modules; but what exactly is a module? In a procedural implementation, a module is basically several functions and then there are associated data structures which are referred by the functions. In the context of object-oriented designs, a module can be classed or a package. So, in a procedural design module consists of several functions and there are some global data may be arrays, link lists or the other structure and these different functions can access those global data.

So, a module consists of certain data structures and also the functions and in a procedural implementation, a module is an independently compliable unit. We can create the object code for a module, we can compile it; then after executing code, we can link the object codes for various modules.

(Refer Slide Time: 12:16)



In the slide, call relationships among the different modules are shown. We will see that it is a good design if our module structure looks like this. Procedural implementation looks like a tree. As we proceed, we will understand the reasons why the module structure or the call relationship among the modules should look like a tree.

(Refer Slide Time: 12:53)



Another thing, we must keep in mind is that when we try to design all the items that we discussed, the modules structure, the call relationships, the parameters, data structures,

the individual modules, algorithms to be used and so on, it's never come as a sequence. Design is a very intellectually stimulating work and need to do several iterations.

(Refer Slide Time: 13:30)

Stages in Design

- Design activities are usually classified into two stages:
 - Preliminary (or high-level) design
 - Detailed design.
- Meaning and scope of the two stages:
 - vary considerably from one methodology to another.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, as we look at the procedural and object-oriented design, we will see that we will have to iterate over several steps. Look at evaluate designs alternatives and then we will come up with a good design. Now let's look at what are the stages in the design? There are two main stages in the design phase: one is called as the high-level or the preliminary design and the other is called as the detailed design.

The high-level design is also called as architectural design and then we have the detailed design. So, first we have preliminary or high-level design or an architectural design and then, we have the detailed design. As we will see, that there is a significant difference in the steps, in the procedural and object-oriented design. And the meaning and scope of these high-level and detailed design vary quite widely between procedural and object-oriented design.

Even in the procedural design, there are several methodologies. We look at one of the methodologies; but there exist dozens of methodologies. Even among the methodologies, there is quite a variation in what should be done the architectural design and what exactly to be done in the detailed design. We look those issues little later.

(Refer Slide Time: 15:36)

High-level design

- Identify:
 - modules
 - control relationships among modules
 - interfaces among modules.

```
graph TD; M1[M1] -- d1 --> M2[M2]; M1 -- d2 --> M3[M3]; M2 -- d3 --> M4[M4]; M2 -- d4 --> M5[M5]; M3 -- d5 --> M6[M6]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Now, let see what is expected in the high-level design? In the high-level design, after we complete this task, we should have come up with the module structure. What are the modules in our implementation and their call relationships and the interfaces between the modules? But then, how do we come up with the modules; how do we determine their call structure and what are the parameters to be exchanged that we will discuss subsequently.

(Refer Slide Time: 16:31)

High-level design

- The outcome of high-level design:
 - program structure, also called software architecture.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

The high-level design is also called as a software architecture. One of the popular notations for representing the high-level design is a structure chart.

(Refer Slide Time: 16:41)

High-level Design

- Several notations are available to represent high-level design:
 - Usually a tree-like diagram called **structure chart** is used.
 - Other notations:
 - Jackson diagram or Warnier-Orr diagram can also be used.

```
graph TD; M1[M1] -- d1 --> M2[M2]; M1 -- d2 --> M3[M3]; M2 -- d3 --> M4[M4]; M2 -- d4 --> M5[M5]; M3 -- d5 --> M6[M6]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Structure chart is very straight forward. It looks like a tree structure with modules as a rectangle. We look at the structure chart representation of the high-level design in the slide and how to come up with the structure chart representation that we will address next. The structure chart is not the only representation that is possible for the high-level design. There are many other notations: for example, a Jackson diagram, Warnier-Orr diagram and so on.

(Refer Slide Time: 17:39)

The slide has a yellow background. At the top center, the title 'Detailed design' is displayed in bold black font. Below the title, there are two main bullet points in black text:

- For each module, design for it:
 - data structure
 - algorithms
- Outcome of detailed design:
 - module specification.

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, the IIT Kharagpur logo and the text 'IIT KHARAGPUR' are visible. In the center, the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' are displayed. On the right side of the bar, there is a small video window showing a man in a light blue shirt speaking.

Once the high-level design is complete or the architectural design is complete, the detailed design is carried out. In detailed design, we look at what are the modules that have been identified during the high-level design and then for each of the module need to design the data structure and also the algorithms to be used.

(Refer Slide Time: 18:00)

The slide has a yellow background. At the top center, the title 'A fundamental question' is displayed in bold black font. Below the title, there is one main bullet point in blue text:

- How to distinguish between good and bad designs?
 - Unless we know what a good software design is:
 - we can not possibly design one.

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, the IIT Kharagpur logo and the text 'IIT KHARAGPUR' are visible. In the center, the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES' are displayed. On the right side of the bar, there is a small video window showing a man in a light blue shirt speaking.

The outcome of the detailed design is called as the module specification. This is a module specification there are various notations for module specification. It can be a

simple text-based notation in formal notation, it can be slightly semi formal notations where represent the data structures and the algorithms.

So far, we just looked at some simple concepts of design phase. We saw that during design, we need to have the high-level or the architectural design that is the first step and once the architectural design is complete, we have the module structure call invocation, call structure. The invocation among the different modules is the data exchange between different modules. And we represent that in a tree like diagram that is a structure chart and after that take up the detailed design where we design the data structure of the individual modules and also the algorithms.

But then, let's try to understand a very fundamental question that let say you have come up with a design, a tree like diagram you have represented, come up with the module specification; but then how do you know that whether your design is good or bad? or what should be the target of any designed methodology to come up with an ideal design? Because unless we know that what we mean by a good software design, even if we have given to learn the methodologies of design and so on, we will not come up with a good design.

Because ourselves before design, we have to be clear about what is meant by a good design; unless we understand that issue that what is a good design, we cannot come up with a good design. Even if we use the most sophisticated tools and design methodologies.

(Refer Slide Time: 20:32)

The slide has a yellow header bar with the title 'Good and bad designs'. Below the title is a bulleted list of points. In the bottom right corner, there is a small video window showing a person speaking.

- There is no unique way to design a software.
- Even while using the same design methodology:
 - different engineers can arrive at very different designs.
- Need to determine which is a better design.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo.

Another reason why we must know about good designs is that we must be able to distinguish between the good and bad designs or in other words, we should be evaluate between design alternatives. During the various design steps, we will see that there are alternate solutions are possible. We should be able to take the better ones because it's not a unique way to design a software. We will see that always evaluating between alternatives and for evaluating between alternatives, we need to distinguish between which is better design and which is a inferior design.

(Refer Slide Time: 21:30)

The slide has a yellow header bar with the title 'What Is a Good Software Design?'. Below the title is a bulleted list of points. In the bottom right corner, there is a small video window showing a person speaking.

- Should implement all functionalities of the system correctly.
- **Should be easily understandable.**
- Should be efficient.
- Should be easily amenable to change,
 - i.e. easily maintainable.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo.

Another reason is that it may possible that different developers, by using the same methodology they can come with very different designs in the end. We should able to tell how good is the design or we can select the better design if we have different designers come up with different designs.

So, let's now address this important question that what is a good software design?

We will answer this very conceptually. The first characteristic of a good software design that it should have addressed all the requirements that were there in the requirements documents. In other words, it should implement all functionalities of the system that are specified in the SRS document.

So, the functionalities et cetera should have been correctly understood and our designs should be as per the requirement. First characteristics is that design need to be correct, unless design is correct, it's no good. The second characteristics of a good design is that it should be easily understandable; it should not look like a messy structure and somebody spends days, months or years trying to understand the design.

It should be an elegant design and somebody looks at the design should be able to easily understand the design. Understandability is a major characteristic of a good software design. Of course, it should result in an efficient solution and another characteristic may be that should be able to change the design easily.

As we know, during development lot of changes happen and even after the development the software continues to change and therefore, our design should be such that we would be able to change any part of the software and that helps in maintaining the software. But we have highlighted one aspect here is that the design should be easily understandable and it turns out that this is one of the most important characteristics of a good software design that it should be easily understandable.

(Refer Slide Time: 24:51)

What Is Good Software Design?

- Understandability of a design is a major issue:
 - Largely determines goodness of a design:
 - a design that is easy to understand:
 - also easy to maintain and change.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The main reason why we consider understandability of a design is a major issue is that unless somebody able to understand the design, he or she cannot write code for that and also becomes very difficult to maintain the design. If the maintainers cannot understand the design then they won't know where to change, what to change et cetera. We also know that maintenance is a major concern because majority of the effort and cost is incurred the maintenance work compared to the development work or that we can say that development work is a fraction of the maintenance work and therefore, while writing any software, whether it is maintainable is a major concern and to facilitate maintenance, we have to develop design which is simple to understand.

Understandability of the design is a major issue, require significant attention by the designers and also every design methodology helps to come up with good design which is easily understandable.

(Refer Slide Time: 26:36)

What Is a Good Software Design?

- Unless a design is easy to understand,
 - Tremendous effort needed to maintain it
 - We already know that about 60% effort is spent in maintenance.
- If the software is not easy to understand:
 - maintenance effort would increase many times.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Majority of the effort spent on maintenance and therefore, unless design is understandable, maintenance effort will become extremely expensive.

(Refer Slide Time: 26:56)

How to Improve Understandability?

- Use consistent and meaningful names
 - for various design components,
- Design solution should consist of:
 - A set of well decomposed modules (**modularity**),
- Different modules should be neatly arranged in a hierarchy:
 - A tree-like diagram.
 - Called Layering

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



But the next question that we will answer is that how to improve understandability? Or how do we come up with a design which has good understandability?

First let us agree that one of the most desirable characteristics of any design solution is that it should be very easy to understand. And then, we will have to address the question that how do we come up with a design that has good understandability?

This is an issue which has some details that we must understand and in the next lecture, we will address this very basic issue that how does one come up with a design that will have good understandability.

We will stop here and continue in the next lecture.

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

Lecture – 20
Modular Design

Welcome to this lecture. In the last lecture, we had looked at some very basic issues in software design. We said that it's important to distinguish a good design from a bad design. And, then we are trying to characterize, what is a good design? And, then we said that there are many factors that determine, whether design is good or bad. But correct implementation of the functionality that possibly the important requirement of a good design, because unless it is a correct design, it's not a good design.

But there are several design alternatives which can be factor for a good design. We can come up with different alternate designs, which are all correct, but then how do we decide which is a better design? And, then we had said that understandability of a design is a major issue. We can rate a design solution to be good or bad based on its understandability.

But there are two questions that arise now. That how do we know, that, which design is more understandable? It's a debatable question, because somebody may say that this design is more understandable, another person will say that no that's not a very understandable design. So, we should be able to tell more formally what do you mean by an understandable design?

And, also, we have to address this question that how do we really come up with an understandable design? So, let's proceed from that. First let's look at the characteristics of a design, that enhance its understandability. So, the first characteristic is the use of consistent and meaningful names in the design.

(Refer Slide Time: 02:44)

The slide has a yellow background and a blue header bar. The title 'How to Improve Understandability?' is centered at the top in a bold black font. Below the title is a bulleted list of six items, each preceded by a black bullet point. The list discusses the use of consistent names, design solution components, modularity, and layering. At the bottom of the slide, there is a blue footer bar containing three logos: IIT Kharagpur, NPTEL Online Certification Courses, and NPTEL.

- Use consistent and meaningful names
 - for various design components,
- Design solution should consist of:
 - A set of well decomposed modules (**modularity**),
- Different modules should be neatly arranged in a hierarchy:
 - A tree-like diagram.
 - Called Layering

Because, unless the names in the design are meaningful in the problem context, anybody trying to understand the design will get confused. All design components should have names, that corresponds to the problem domain as far as possible and they should be meaningful so anybody can understand.

The second characteristic of a design which is understandable is that it should have been decomposed well into modules. This we call it as the modularity of the design. We need to look at this question that what do we mean by modularity? This is an important characteristic that the design should be modular. And, the modules should have been such that they have been well designed. Just arbitrary set of modules is not a good design; the module should have been well decomposed. And, also the call relationship should look like a tree diagram, this is also called as layering. Because, each layer of the tree is an implementation of the previous layer and a specification of the lower layer. This is a desirable characteristic we will see later that it is not only that we should have a well decomposed set of modules, but also their call relation should be representable in the form of a tree like diagram.

(Refer Slide Time: 05:05)

Modularity

- Modularity is a fundamental attributes of any good design.
 - Decomposition of a problem into a clean set of modules:
 - Modules are almost independent of each other
 - Based on **divide and conquer principle**.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's try to understand, what we mean by modularity or well decomposed set of modules? We say that a design is modular, if the modules are almost independent of each other. A set of modules are almost independent of each other, if they either do not call any other module or they call very less. If, a module calls another module, and it requires many parts to be completed by other modules then it becomes a dependent module.

In a well decomposed set of modules, the module should be as independent as possible. It's not possible to have a completely independent set of modules that means no module interacts with other, but the interaction among the module should be kept to a minimum. To understand why it is so, we will argue that this is actually the divide and conquer principle.

A modular design uses the divide and conquer principle to be able to understand the design. We just take up each module and then we understand. And, if after we have understood all the modules, we have understood the design.

But, if the modules have complex relations that is each module calls several other modules then while understanding a single module, we need to also see what it gets done by the other modules and so on. So, as long as the modules don't interact too much, we can understand them well based on the divide and conquer principle.

(Refer Slide Time: 07:34)

Modularity

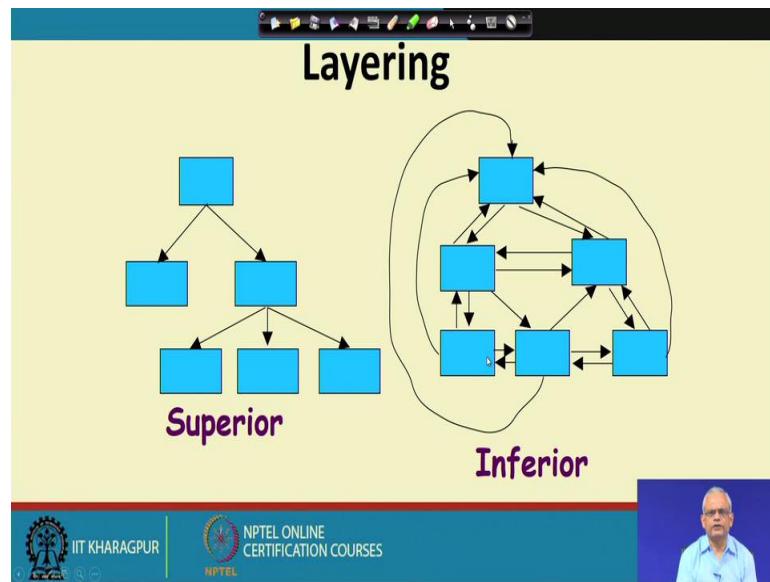
- If modules are independent:
 - Each module can be understood separately,
 - reduces complexity greatly.
 - To understand why this is so,
 - remember that it is very difficult to break a bunch of sticks but very easy to break the sticks individually.

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, if the modules are independent, we can look up the modules one by one and understand this and this is based on the design and divide and conquer principle.

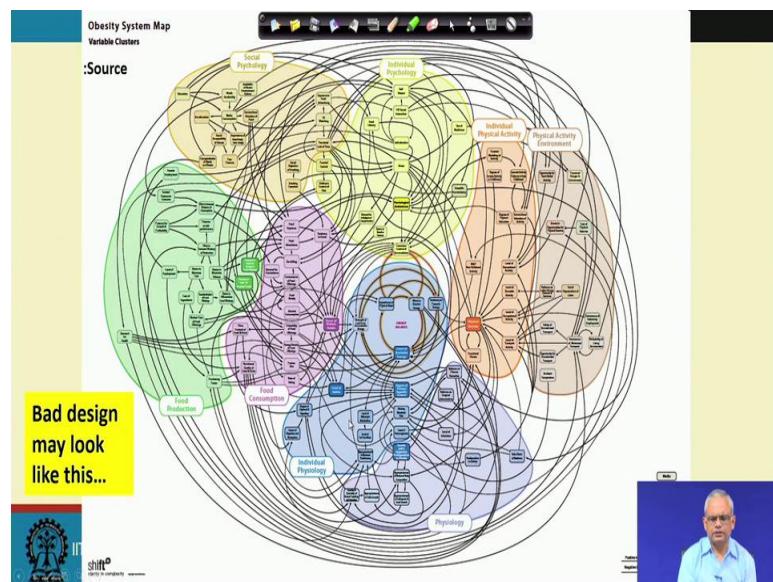
If a bunch of sticks are tied together and we tried to break it, then it becomes very difficult, but if we take one stick at a time and break it, then becomes very easy. It's thus the similar thing here is that if we tried to understand module one by one, we can easily understand, but trying to understand all modules which are interconnected with each other, then if we start with one module we see that we need to understand another module and the third module and so on.

(Refer Slide Time: 08:44)



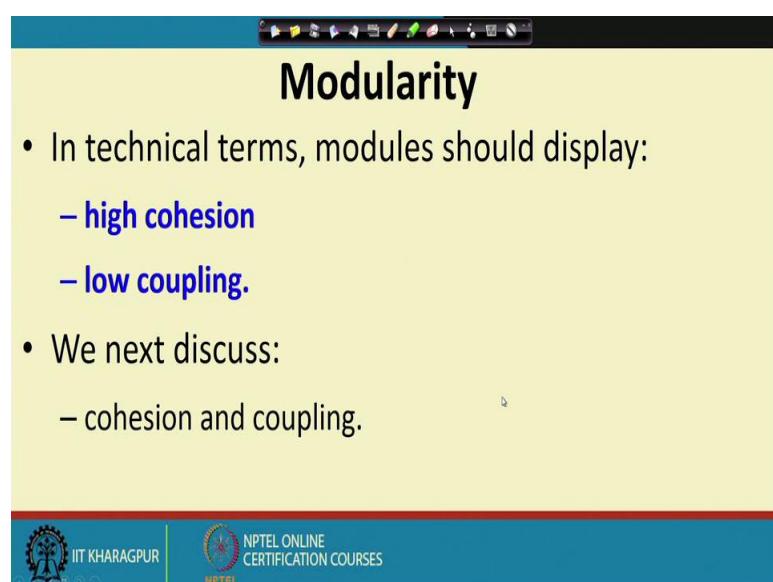
Another requirement of a good design is layering. A tree like call relationship among modules is a superior, where as in an arbitrary call relation is an inferior design. We will see why the layering is a good idea: one is bug localization. In superior layering, if there is a bug in any intermediate node, it does not affect the other modules except its child module. Whereas in inferior layering, any bug in any node can have effect on any other module. And, when we see there is a failure, we don't know what is the module, that is having the bug? But in superior layering, if we observe a bug in any leaf node, we know that either the bug is in that node or in the previous parent node or in any upper node. In superior layering, we do not have to look on all nodes.

(Refer Slide Time: 10:05)



A bad design can come up with a very complex set of module interactions. We can see in the slide that a complex module interaction has shown. The slide diagram, is not really a set of modules and interactions, it is a diagram for some other purpose. But here we are showing it for illustration, that it can look extremely odd and here for understand one module, you might have to understand all the modules associated with the module we want to understand. So, for understanding one module we need to trace all the modules and it becomes very difficult to understand. So we can say that to understand a module we need a nice tree like diagram.

(Refer Slide Time: 10:55)



So, far we said that modularity is a set of independent modules and the we said that independent modules are actually not practical. There will be some interactions between modules, but can we give a more meaningful characterization of modularity. We had intuitive understanding of what is modular design? that modules are almost independent of each other, don't depend too much represented in a tree like diagram and so on.

Let's try to give a better characteristic or more precise characteristic of what is a modular design? And, we will do that in terms of two concepts: one is called as cohesion and the other is called as coupling. A modular design should have high cohesion and low coupling.

And, if we understand this terms cohesion and coupling and we can measure them, then for a given design solution, we can check whether it has high cohesion and low coupling.

(Refer Slide Time: 12:25)

The slide has a title 'Modularity' at the top. Below it is a bulleted list:

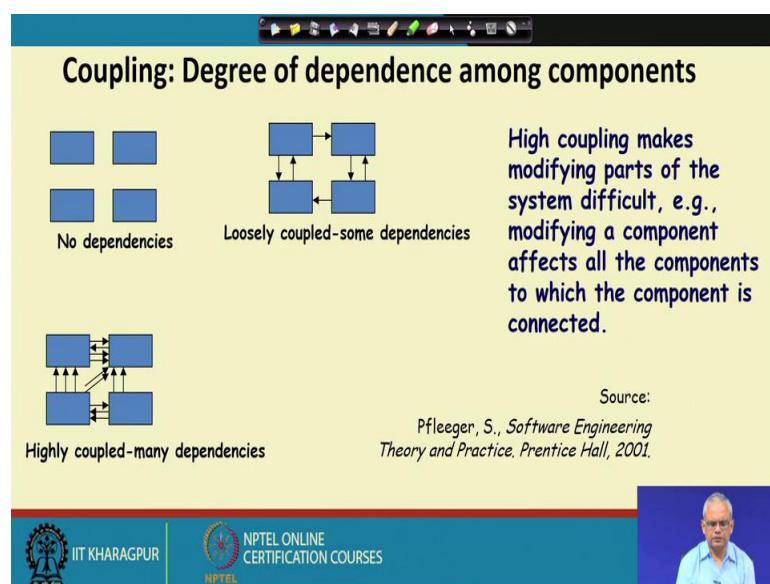
- Arrangement of modules in a hierarchy ensures:
 - Low fan-out
 - Abstraction

To the right of the list is a hand-drawn diagram showing a central module connected to several lower-level modules, illustrating a hierarchical structure. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video player showing a person speaking.

So, let's try to understand how to measure, define cohesion and coupling? The module should be layered. A good layering means a tree like diagram. But then let me just mention briefly, that in a tree like diagram, if, we want to let say understand one module, then we see that it calls only the lower modules and for that maybe we will have to look at lower modules. If, we want to understand any intermediate module then we do not even have to look at upper module. Because, we know that intermediate will call only its lower layer modules. So in tree like structure, we can understand each module through top to down hierarchy.

But, if it is a module structure like 2nd drawn diagram in slide, then to able to understand one we need to just go across all other modules and so on. So, it becomes very difficult to understand each module. So, a tree like hierarchy, we have low fan-out that is it depends on a smaller number of modules. And, also there is an abstraction. Abstraction in the sense that in tree like structure, for implementation, we can start from the leaf layer modules and then look at more concrete module and so on.

(Refer Slide Time: 14:54)



Above slide, is an illustration of how bad it can become if the modules are having lot of dependencies. In slide, first one is showing no dependencies, second one has very less number of dependencies, but look at the third one, there are very high dependencies across modules and it becomes very difficult to understand. So, maintain this kind of design like 3rd one is very difficult. So, we would rather go for 1st one if not, we will go for 2nd one.

(Refer Slide Time: 15:32)

The slide has a yellow header and a blue footer. The title 'Cohesion and Coupling' is at the top. The main content is a bulleted list:

- Cohesion is a measure of:
 - functional strength of a module.
 - **A cohesive module performs a single task or function.**
- Coupling between two modules:
 - **A measure of the degree of interdependence or interaction between the two modules.**

The footer contains the IIT Kharagpur logo, the NPTEL logo, and a photo of a speaker.

Now, let see the concepts of cohesion and coupling. If, we can determine the cohesion and coupling of a given design solution, we can tell that whether that's a good design or bad design. We say that a module is cohesive, if it performs a single task or function. If, it tries to do too many things then it's not cohesive. On the other hand, the coupling is defined between two modules. Two modules are coupled if they dependent on each other. Dependent on each other means they exchange some data items, call each other etc. So, this is two important concepts. Cohesion is defined it for a single module.

(Refer Slide Time: 16:43)

The slide has a yellow header and a blue footer. The title 'Cohesion and Coupling' is at the top. The main content is a bulleted list:

- A module having high cohesion and low coupling:
 - **Called functionally independent** of other modules:
 - A functionally independent module needs very little help from other modules and therefore has minimal interaction with other modules.

The footer contains the IIT Kharagpur logo, the NPTEL logo, and a photo of a speaker.

And then we say that a single module is cohesive, if it performs a single task, if it does many things then that is not really cohesive. Coupling is that how much dependent one module is on another module. If, they exchange too many data items call each other frequently exchanging data items and so on then, they are highly coupled. But if they don't call each other, there is no data exchange between two modules, we say that these two modules are independent. So, if there is a high cohesion and low coupling then we say that this is a good design.

Now, let's look at how do we determine the cohesion and coupling? Before, that if a design solution we find that it has high cohesion and low coupling, then we say that the design is a functionally independent set of modules. So, in the design, each module is doing some function independently that means it's not dependent on others. And the term that we use for this is functionally independent.

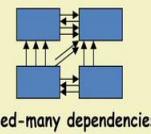
So, the goal of any good design technique is to come up with a functionally independent set of modules.

(Refer Slide Time: 18:32)

Advantages of Functional Independence

- Better understandability
- Complexity of design is reduced,
- Different modules easily understood in isolation:
 - Modules are independent


No dependencies


Highly coupled-many dependencies

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL



But we should be clear why we want a functionally independent set of modules? As, we said that functionally independent results in better understandability compared to a highly dependent set of modulus.

If, we have a functionally independent set of modules the complexity in the design is reduced and we can take up one module at a time quickly to understand. Similarly, if there is a bug in the 1st diagram of the below slide, we can easily know that which module contains the bug, otherwise we might see that whether it has come from, may be some module to which it has called. But if there is a bug while executing the 2nd design module of the below slide, we can't easily identify the module which contain the bug.

(Refer Slide Time: 19:43)

The slide has a title 'Why Functional Independence is Advantageous?' and two bullet points:

- Functional independence reduces error propagation.
 - degree of interaction between modules is low.
 - an error existing in one module does not directly affect other modules.
- Reuse of modules is possible.

Below the second bullet point is a diagram showing four blue rectangular boxes arranged in a 2x2 grid. The top-left box has a single arrow pointing to the bottom-right box. The top-right box has a single arrow pointing to the bottom-left box. The bottom-left box has a single arrow pointing to the bottom-right box. The bottom-right box has a single arrow pointing to the top-left box. This indicates a fully connected graph where every node is connected to every other node. Below this diagram is the text 'No dependencies'.

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

So, if the modules are functionally independent, debugging becomes easy and also understandability become easy and not only that, we can re-use module.

Let say, in the 1st diagram of above slide, we want to use any module in another project, we can just take the module and use it there. But here observe it if we want to use any module of 2nd diagram in another project. We will see that the needs help from another module and another module in turn needs help from another and so on. So, they are all coupled. We cannot take just one module and use it we have to take across all these modules that entire project we have to take and that becomes infeasible. So, if we have a functionally independent set of modules, any module that we need in another project, we can easily re use that.

(Refer Slide Time: 20:56)

Advantages of Functional Independence

- A functionally independent module:
 - can be easily taken out and reused in a different program.
 - each module does some well-defined and precise function
 - the interfaces of a module with other modules is simple and minimal.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Each module in a functionally independent design, has well defined precise functions. And, it does not interact with other modules and it interfaces with other modules simple and minimal. And therefore, becomes easy to re use modules.

(Refer Slide Time: 21:20)

Functional Independence

- Unfortunately, there are no ways:
 - to quantitatively measure the degree of cohesion and coupling;
 - At least classification of different kinds of cohesion and coupling:
 - will give us some idea regarding the degree of cohesiveness of a module.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

But, how do we measure functional independence? Even though we know that a functional independent set of module has high cohesion and low coupling. Can, we look at a design and say that whether there is a high cohesion and low coupling?

Let's try to explore this further; we will classify the different types of cohesion and coupling. So, that given a design we will try to see, which class of cohesion and coupling it has and based on that we will say that, whether it is a cohesive, highly cohesive and low coupling et cetera.

(Refer Slide Time: 22:09)

Classification of Cohesiveness

- Classification can have scope for ambiguity:
 - yet gives us some idea about cohesiveness of a module.
- By examining the type of cohesion exhibited by a module:
 - we can roughly tell whether it displays high cohesion or low cohesion.

First let's look at the cohesiveness and let's try to classify what are the type of cohesion? So, that if we look at a design, we will see that what set of cohesion it has.

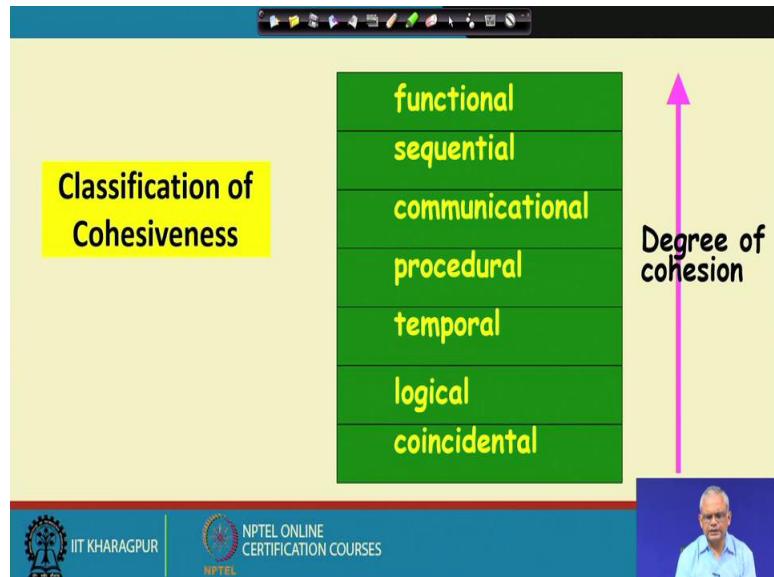
(Refer Slide Time: 22:26)

Logical cohesion

- All elements of the module perform similar operations:
 - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
 - a set of print functions to generate an output report arranged into a single module.

Let see the different types of cohesion. We can look at a design and we can say that it has any one of these seven types of cohesion (below slide).

(Refer Slide Time: 22:32)



We can look at a module and we can say that this module has either this coincidental, logical, temporal, procedural, communicational, sequential or functional cohesion.

If, it has functional cohesion then it has high cohesion means it's a good design. But if it has coincidental cohesion on the other hand, it is not a good design. And, something in between functional and coincidental is moderate cohesion. We want to look at all modules and, see what type of cohesion are there. If most of them are high cohesion or a slightly moderate cohesion we will say that the design is good or ok. But if they have low cohesion, we will see that we will say that it's not a good design.

(Refer Slide Time: 23:41)

The slide has a yellow background. At the top center, the title 'Coincidental cohesion' is displayed in bold black font. Below the title, there is a bulleted list of points:

- The module performs a set of tasks:
 - which relate to each other very loosely, if at all.
 - That is, the module contains a random collection of functions.
 - functions have been put in the module out of pure coincidence without any thought or design.

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there are two logos: IIT Kharagpur and NPTEL. Next to the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES' is written. On the right side of the bar, there is a small video window showing a man in a blue shirt.

First let's look at the worst form of cohesion, it's called as the coincidental cohesion.

Here, if we look at the module for look through what functionality it has, we will see that the functions are rather random collection of the functions. There is no thinking or plan behind putting some functions in the module, we just taken out random functions and made them into a module.

(Refer Slide Time: 24:15)

The slide has a yellow background. At the top center, the title 'Coincidental Cohesion - example' is displayed in blue font. Below the title, there is a code snippet:

```
Module AAA{  
    Print-inventory();  
    Register-Student();  
    Issue-Book();  
};
```

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there are two logos: IIT Kharagpur and NPTEL. Next to the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES' is written. On the right side of the bar, there is a small video window showing a man in a blue shirt.

Just to give an example, let say we look at one module named as AAA and then it contains the functions: print-inventory, register-student, issue-book.

So, this actually correspond to the functionalities required by various other, various requirements. Print-inventory() is part of some inventory functionality, Register-student() is part of some registration, Issue-Book() is part of library management system. So, these are random set of functions, they do not neither belong to the same requirement nor there is any relationship between these functions.

And, if we ask that what does the module AAA achieve? We cannot really answer. We will say that it has some part of the inventory, does some registering student, it also does issue book et cetera. So, there is no single thing, single function that this module achieves. It Does various things here and there. Somebody has just put these functions randomly into this module AAA and this we call as coincidental cohesion.

(Refer Slide Time: 25:45)

Logical cohesion

- All elements of the module perform similar operations:
 - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
 - a set of print functions to generate an output report arranged into a single module.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

NPTEL

We, consider logical cohesion module has logical cohesion, if all elements, all functions in the module performs similar operation. For example, all functions in a modular doing error handling or all functions in a modular reading the data input or all functions in a modular just print kind of thing, data output. Then, we say that this modular design has a logical cohesion that all the functions it has are logically related that they are doing similar things and that's why designer has put all the functions in a modular design. And, this logical cohesion is not a very good form of cohesion slightly better than coincidental cohesion, but still it's not good.

(Refer Slide Time: 26:43)

The slide title is "Logical Cohesion". The content shows a code snippet for a module named "print". The module contains three functions: "void print-grades(student-file){ ... }", "void print-certificates(student-file){...}", and "void print-salary(teacher-file){...}". The code ends with a closing brace "}".

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

Let's look at an example of logical cohesion. Let say we have a module, name is print and then it does various types of printing, across various functionalities: printing grade, printing certificates, printing salary et cetera. All these printing functions are part of various other functionalities or requirements.

And, then we say that this module has logical cohesion, it's not the bottom most cohesion, but still it's not a good cohesion.

(Refer Slide Time: 27:23)

The slide title is "Temporal cohesion". The content lists characteristics of temporal cohesion:

- The module contains tasks so that:
 - all the tasks must be executed in the same time span.
- Example:
 - The set of functions responsible for
 - initialization,
 - start-up, shut-down of some process, etc.

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

Next, we will look at the temporal cohesion. We will stop here and will continue in the next lecture.

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

Lecture – 21
Classification of Cohesion

Welcome to this lecture. In the last lecture, we were discussing some very general concepts about a good design. And, we had said that design is an iterative procedure and the designer can come up with different designs, but the different designs can vary in their quality.

And, we are trying to find out how to distinguish between two designs? When can we say that a design is good or bad? We identify some characteristics of a good design. One of the very important characteristic is modularity. The different modules in the designs should be more or less independent and we said that functional independence is an important characteristic in good design. Now, we said that to identify functional independence, we need to look at the cohesion and coupling among the modules.

Now, let's proceed from that point.

(Refer Slide Time: 01:42)

Cohesion and Coupling

- A module having **high cohesion and low coupling**:
 - **Called functionally independent** of other modules:
 - A functionally independent module needs very little help from other modules and therefore has minimal interaction with other modules.

A good design where the modules are functionally independent, these have high cohesion and low coupling. If they have high cohesion and low coupling, we say that a

module is functionally independent. But then we must define the terms cohesion and coupling.

(Refer Slide Time: 02:18)

Advantages of Functional Independence

- Better understandability
- Complexity of design is reduced,
- Different modules easily understood in isolation:
 - Modules are independent

No dependencies

Highly coupled-many dependencies

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Functional independence has many advantages. One advantage is understandability. This is an example (on the above slide) of modules where there is no dependence. So, the modules are independent and here we can check one by one, we can understand, there is no dependency. Whereas, in another modules there is high dependency. And, here we can see that a module has dependency structure with many other modules. And therefore, to understand one module will have to understand many other modules and task becomes extremely difficult. But if there is functional independence, we can easily understand the modules and also any error condition in a module is isolated. If, we find an error in a functional independence module, we can easily identify the error. But if they are dependent on each other heavily, we will have to keep on tracing the sequences and it may lead to just going round and round without really able to determine the error.

So, the advantages of functional independence is that the complexity of the design is reduced, it becomes easy to maintain debugs and also we can take out any one module here and re use in another application. Whereas, in the dependent module all these may not be possible because, if we want to take this module for re using, we might have to take the other modules on which it dependence or calls the functions of other modules.

(Refer Slide Time: 04:50)

Why Functional Independence is Advantageous?

- Functional independence **reduces error propagation.**
 - degree of interaction between modules is low.
 - an error existing in one module does not directly affect other modules.
- **Also: Reuse of modules is possible.**

No dependencies



Now, let's list the advantages of functional independence: one is that it reduces the complexity of the system, so it makes easy to understand the system. Another advantage is that it reduces error propagation any error in a module when the system fails, you can easily trace it to this module, whereas if they are heavily dependent and if there is high coupling, you can just go round and round trying to figure out where the error is, and becomes very difficult to debug. The third reason why functional independence is advantageous is that re use of modules becomes easier. If, we have developed an application consisting of functional independent modules, we can just take out any module and re use in another application without much problem. Where as if our modules structure is like highly dependent module then for taking one module, we will may have to also take out the entire application.

(Refer Slide Time: 06:18)

Reuse: An Advantage of Functional Independence

- A functionally independent module:
 - can be easily taken out and reused in a different program.
 - each module does some well-defined and precise function
 - the interfaces of a module with other modules is simple and minimal.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Reuse is a major issue in software engineering, because it reduces cost if we can reuse parts of one application. It is very advantageous. And therefore, in the design stage, we have to design such that reuse is facilitated and that becomes possible if the module structure is functionally independent.

(Refer Slide Time: 06:55)

Measuring Functional Independence

- Unfortunately, there are no ways:
 - to quantitatively measure the degree of cohesion and coupling:
 - At least classification of different kinds of cohesion and coupling:
 - will give us some idea regarding the degree of cohesiveness of a module.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Now, we have been saying that functional independence is good, it has many advantages and so on. But given a design structure it may not be that all modules are totally independent. That's a very idealistic design where no module ever interacts with any

other module. In real applications modules do interact with each other, they call each other's functions and so on.

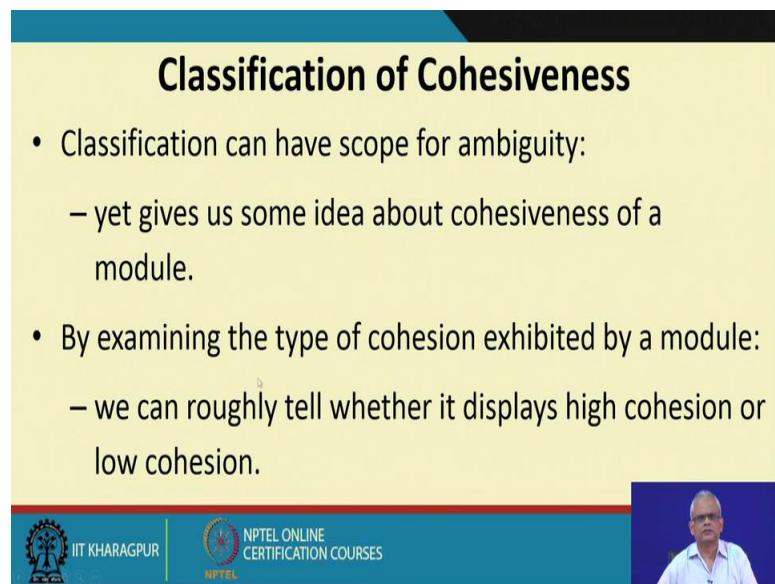
But then given an arbitrary application design can we measure the functional independence of that design? So, after measuring that we can say that one is a better design than another design. It would have been very nice if we could have come up with very quantitative measure of the degree of independence. It is very difficult to come up with quantitative measure, but we can do one thing is that, we can classify the cohesion and coupling and based on that, we can approximately say which is a better design than another.

So, let me just repeat that point that quantitative measure of functional independence in terms of their cohesion and coupling, there is no accepted practice, there is no accepted technique by which we can measure the functional independence and everybody will agree with that. But we can do the next best thing, that is we can characterize the cohesion and coupling in terms of certain classes. And, depending on the class to which an application belongs the cohesion and coupling, on the basis of that we can say a design is very good, moderate or bad.

(Refer Slide Time: 09:07)

Classification of Cohesiveness

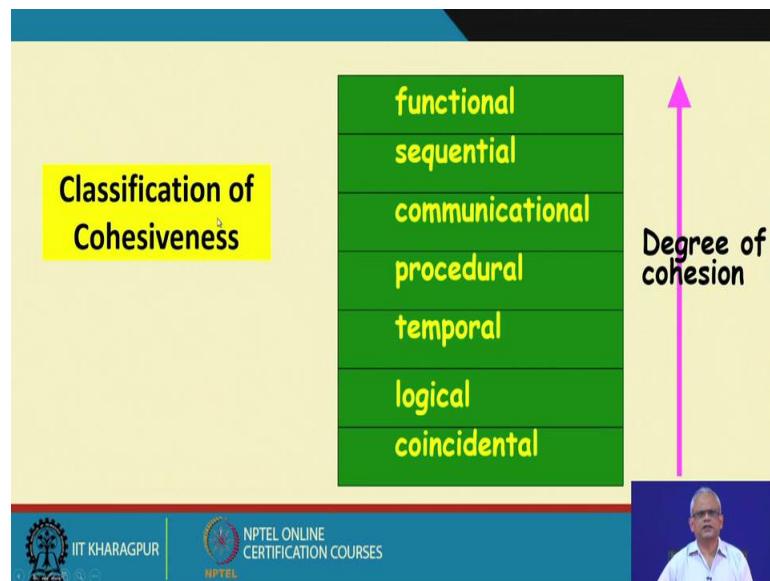
- Classification can have scope for ambiguity:
 - yet gives us some idea about cohesiveness of a module.
- By examining the type of cohesion exhibited by a module:
 - we can roughly tell whether it displays high cohesion or low cohesion.



Let us see, how we can classify the cohesion and coupling existing in a design? Here, we will see that there is a small scope for ambiguity, because you can argue that design belong to some particular class or some another class, we get a fairly good picture about

cohesiveness of a module. For example, based on the cohesive class of a design, we do not quantitatively say that it is a 70.1 percent. We can say, that it is between 60 to 70 or between 70 to 80 and that itself will help us to the great extent.

(Refer Slide Time: 09:55)



Now, look at cohesiveness. The cohesion existing in a module can be classified into coincidental, logical, temporal, procedural, communicational, sequential and functional. So, that is seven classes. And, the worst form of cohesion is coincidental. You, can just examine a module and then you should be able to tell, where in this spectrum does the cohesion of the module lie, is it coincidental (bad form of cohesion) or is it something like a temporal or procedural (middle form of cohesion) or is it functional (best form of cohesion).

Now, let see what are these different types of cohesion? So, that given a module structure we should be able to roughly place it in this spectrum.

(Refer Slide Time: 11:08)

Coincidental cohesion

- The module performs a set of tasks:
 - which relate to each other very loosely, if at all.
 - That is, the module contains a random collection of functions.
 - **functions have been put in the module out of pure coincidence without any thought or design.**



First let see, the worst form of cohesion that is coincidental cohesion. Here, there is no thought or designed behind putting functions into a module. We are just randomly assigned functions in to modules. The different functions existing in the program they have been randomly put into different modules, without any thought or design. If, this is the case that we just select a set of functions, randomly and assign them to a module then it is coincidental cohesion and we ask the question that can you tell us what does this module do? Will be very hard pressed to give a simple answer to what the module does? We can say that it does this and this et cetera.

(Refer Slide Time: 12:06)

Coincidental Cohesion - example

```
Module AAA{  
    Print-inventory();  
    Register-Student();  
    Issue-Book();  
};
```



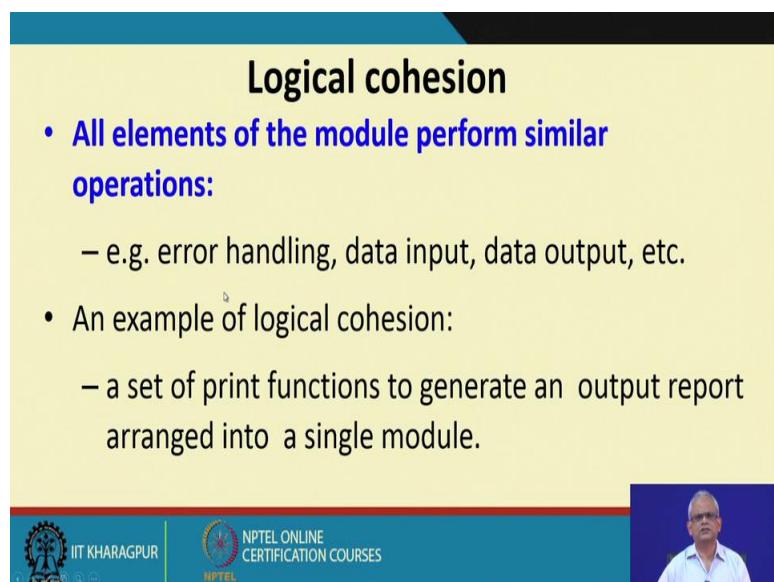
Let's look at an example. Let say we have a module named as AAA. In a library management software, there are many functions like register students, register book, collect fine, issue book, return book, and so on. We just took 3 functions here and just put them part of this module, that it does little bit of inventory: print the current inventory and also register students and also issue book.

Now, if we ask what does this module do? We will say that it does some inventory functions like print inventory, it does some student registration and it also does issue book. So, we cannot give a simple one-word description of this module. So, here are functions which are doing very different things. This is an example of coincidental cohesion and given an example, you can easily find out whether it's a case of coincidental cohesion or not.

(Refer Slide Time: 13:53)

Logical cohesion

- All elements of the module perform similar operations:
 - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
 - a set of print functions to generate an output report arranged into a single module.



Now, let's look at the slightly better form of cohesion called as logical cohesion. Here all elements of the module performs similar functions for example, all functions do error handling, all may do let say, print statements or read data, scan statements or all print statements et cetera.

So, here all the functions do similar things and just because the functions do similar things, we just decided to put them into one module. For example, in an application, if we put all print functions that generate an output report into a single module then we will say that the module has logical cohesion.

(Refer Slide Time: 14:46)

Logical Cohesion

```
module print{
    void print-grades(student-file){ ...}

    void print-certificates(student-file){...}

    void print-salary(teacher-file){...}
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



This cohesion also we can unambiguously and easily identify. For example, if we find a module named print, which prints various things like it prints grades, it prints certificates, it prints salary slip, it prints inventory details and so on. So, just because all these functions do printing, we decided to put them into one single module and call it print. And, then this is a logical cohesion, not a very good form of cohesion, but better than coincidental or random cohesion.

(Refer Slide Time: 15:26)

Temporal cohesion

- The module contains functions so that:
 - **all the functions must be executed in the same time span.**
- Example:
 - The set of functions responsible for
 - initialization,
 - start-up, shut-down of some process, etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Slightly better form of cohesion is the temporal cohesion. Here we group functions together if they are executed within the same time span.

For example, let say during initialization certain functions are executed one after other and before the initialization any other functions executed or let say, we have a certain shut down procedure et cetera. So, here these functions do very different things, but then they have been put into one module just because they run during the initialization time.

(Refer Slide Time: 16:17)

```
init() {  
    Check-memory();  
    Check-Hard-disk();  
    Initialize-Ports();  
    Display-Login-Screen();  
}
```

Temporal Cohesion - Example

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Just to give an example, let say we have a function called as init() and then we do check memory, check hard disk, then initialize the ports and then display some login prompts on the screen. So, here the functions are for very different purpose: some displays on the screen some message, some initialize port, then checking whether memory is alright, hard disk is alright, et cetera, but then they are run on the same time span and that is the reason we have decided to put them in one module called as init(). So, this is an example of temporal cohesion and here functions are execute in the same time span.

(Refer Slide Time: 17:04)

Procedural cohesion

- The set of functions of the module:
 - all part of a procedure (algorithm)
 - certain sequence of steps have to be carried out in a certain order for achieving an objective,
 - e.g. the algorithm for decoding a message.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



A still better form of cohesion is the procedural cohesion. Here the different functions in the module, they are part of some algorithm. For example, that they carry out some steps, let say for decoding a message they do some steps like initialize and then let say discrete cosine transfer and then let say entropy calculation and so on. So, these are set of steps, which is part of an algorithm and we have just put them together, but even they are part of the same algorithm, they do very different things.

(Refer Slide Time: 18:01)

Communicational cohesion

- All functions of the module:
 - Reference or update the same data structure,
- Example:
 - The set of functions defined on an array or a stack.



IIT KHARAGPUR

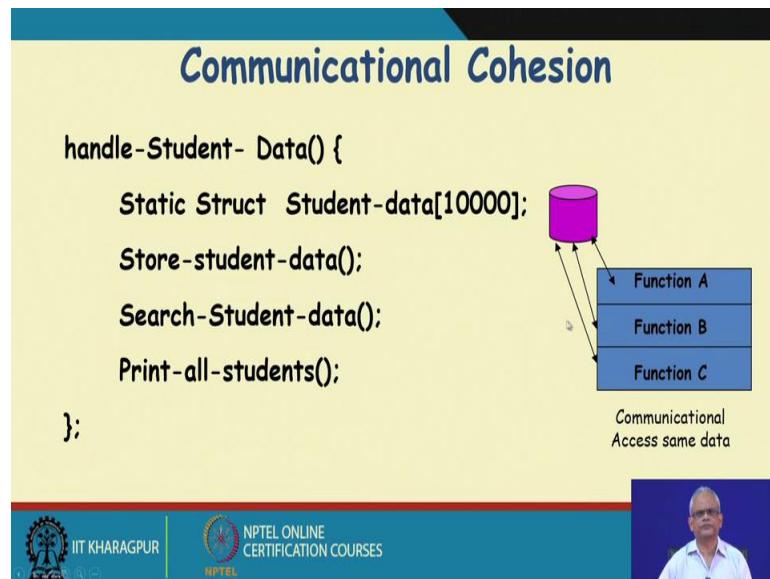


NPTEL ONLINE
CERTIFICATION COURSES



A better form of cohesion is communicational cohesion. Here the different functions they operate on the same data structure. So, here the result is shared between the different functions. Just consider the set of functions defined on an array or stack, let say stack push pop et cetera. So, here the functions on the stack they all operate on the stack and use the data structure. And, this we can call as a communicational cohesion.

(Refer Slide Time: 18:41)



This an example of a communicational cohesion is handle-student-data is the name of the module. And, then there is an array here Student-data. The size of the array is 10000. So, this is the data here. And, then we have various functions there in the module, which are operating on this. For example, search student data, print all data and store student data, update student data and so on.

So, this is a better form of cohesion than the cohesion that we talked about so far is the communicational cohesion.

(Refer Slide Time: 19:24)

Sequential cohesion

- Elements of a module form different parts of a sequence,
 - output from one element of the sequence is input to the next.
 - Example:

```
graph TD; sort[sort] --> search[search]; search --> display[display]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And a still better form of cohesion is a sequential cohesion. Here also the different functions in the module share data, but then they just do not randomly update the data here, the data is passed from one function to the other in a sequence. For example, first do sorting then, search then, display and so on.

(Refer Slide Time: 19:51)

Functional cohesion

- Different elements of a module cooperate:
 - to achieve a single function,
 - e.g. managing an employee's pay-roll.
- When a module displays functional cohesion,
 - we can describe the function using a single sentence.

Now we will discuss the best form cohesion that is a functional cohesion. Here, the different functions they shared data, but then they work towards achieving a single function. For example, if all the functions to manage the employees pay roll, we just put

them together into a module like compute overtime, compute the current months' pay, change the basic salary everything put into a module. So, these are all towards managing the employee's payroll. And, here one of the distinguishing characteristics is that by looking at the module structure, we can give a very simple name. Here we give a name like employees pay roll. Because all functions work towards doing some parts of the managing employees' payroll. So, one test whether module has functional cohesion is that we should be able to describe the function of the module or what the module achieves by using a very simple sentence.

(Refer Slide Time: 21:23)

Write down a sentence to describe the function of the module

Determining Cohesiveness

- If the sentence is compound,
 - it has a sequential or communicational cohesion.
- If it has words like "first", "next", "after", "then", etc.
 - it has sequential or temporal cohesion.
- If it has words like initialize,
 - it probably has temporal cohesion.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's see how to identify the cohesiveness of a module? Let say we are given a module and we are asked to find that, what is the cohesion here? Is it a good form of cohesion, bad form of cohesion or what?

Now, we know the seven classes of cohesion starting from very bad case of coincidental cohesion. And, then the best form of cohesion that is the functional cohesion. We can imagine that there may be some ambiguity, whether to consider it as a sequential cohesion or let say procedural cohesion and so on. But then if it is a bad case of cohesion, we can easily identify or whether it is a somehow ok level of cohesion or a very good case of cohesion we can easily identify.

One hint about the identifying the cohesion is that first look at the module and the functions that it has and try to describe what is the function of the module? What does

the module achieve? And, then we write this in a sentence and if we find that if the sentence is compound sentence that it does this and this and so on, then it has a sequential or communicational cohesion.

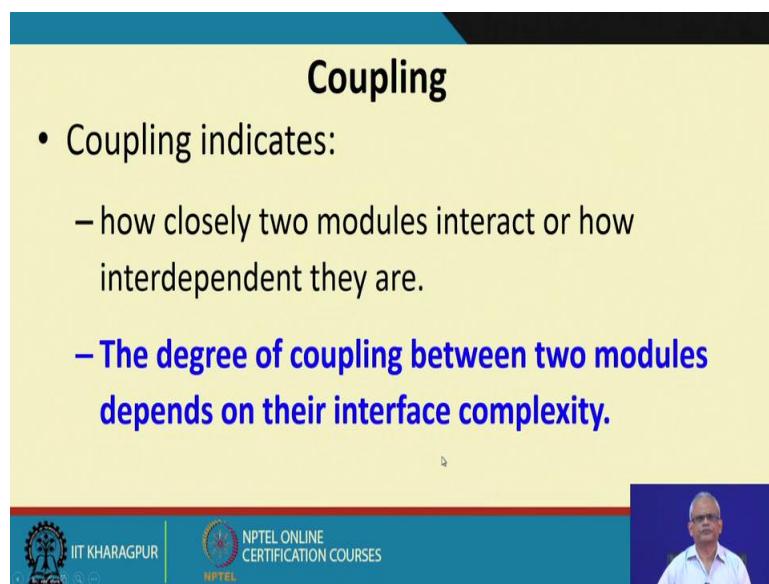
We are not mentioning here the coincidental cohesion, because that's easily identified and that's a bad case of cohesion, that the functions are totally unrelated. But these are the middle cases here we are just trying to give some hint. How to identify? If, it is a compound sentence, then we can suspect that it has sequential or communicational cohesion.

If, the sentence was words like first it does these, then it does these and afterwards it does this and so on. Then it is either a sequential or temporal cohesion. If it has words like the initialize, shut down procedure et cetera, then it is possibly temporal cohesion.

(Refer Slide Time: 24:03)

Coupling

- Coupling indicates:
 - how closely two modules interact or how interdependent they are.
 - **The degree of coupling between two modules depends on their interface complexity.**



So, far we looked at the cohesion classification and given a module how to identify the cohesiveness? Now, let's look at coupling, given two modules can we identify what is the extent of coupling between these two modules has? We say that two modules are highly coupled, if they have a very complex interaction among each other. If, there is no interaction, then we will say that there is no coupling. But then the cases where one module calls function of another module, then will say that there is a coupling, but then will see how to identify the class of coupling or how complex is the coupling? Roughly,

we can say that the coupling between two modules can be identified by looking at the interface complexity.

(Refer Slide Time: 25:11)

Coupling

- There are no ways to precisely measure coupling between two modules:
 - classification of different types of coupling will help us to approximately estimate the degree of coupling between two modules.
- Five types of coupling can exist between any two modules.



Now, just like we did the classification or cohesion, we will do a classification of coupling, there are five types of coupling exists:

(Refer Slide Time: 25:23)

Classes of coupling

data
stamp
control
common
content

Degree of coupling



data coupling, stamp coupling, control coupling, common coupling, and content coupling. The data coupling is a simple form of coupling and it is a low coupling and it's a good case of coupling. And, if it is a stamp coupling then there is complex data

interchanged between module. In simple data coupling, only simple data items like integer or character et cetera simple data items are exchanged between two modules.

In stamp coupling, more complex data items like an array or a big structure, list et cetera are exchanged between different modules. And, this is still an ok form of coupling. Worse form of coupling is control coupling, where one module decides the control path in another module. In common coupling, they share some common data and then the very bad case of coupling is content coupling. One fact about content coupling is that earlier assembly routines and machine language programming content coupling was possible, but now all high-level languages, it is difficult to write a program where there will be content coupling. So, the content coupling code can be written in high-level languages, because the high-level languages have been designed such that this coupling does not occur, because it is a very bad case of coupling and make the program extremely complex.

We will stop at this point and will continue in the next lecture and we look at for a given module or two modules how to identify, what is the extent of coupling or which class of coupling out of these five classes exists between those two modules?

We will stop now and continue in the next lecture.

Thank you.

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

Lecture – 22
Classification of Coupling

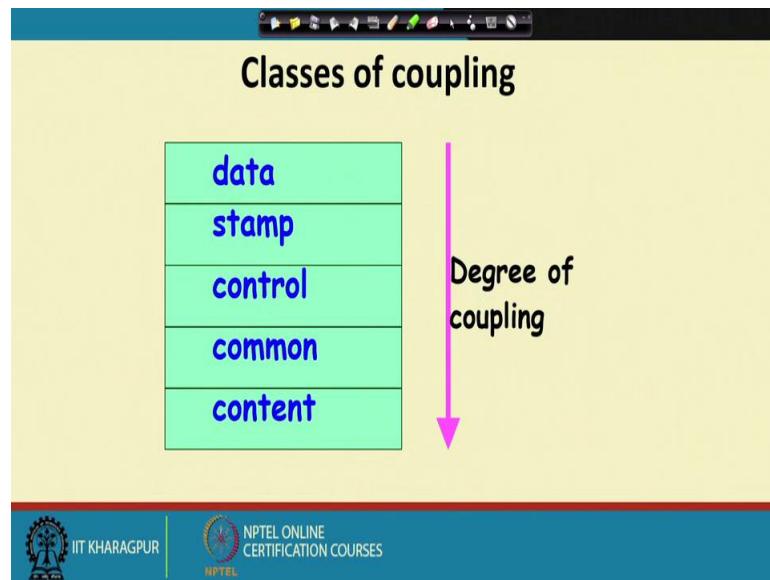
Welcome to this lecture. In the last lecture, we are trying to classify the cohesiveness and the coupling existing between different modules, because that will give us a hint is to how good is a design?

We saw how to check whether a design is functionally independent or not? If, it is a functionally independent design then we will have a case of very high cohesion and low coupling.

In the last lecture we had discussed that by looking at a module structure can we identify, what is the class of cohesion that exists in the module? For that we looked at seven classes of cohesion and we said that by examining the functions in module, we can say approximately which class of cohesion does the module belong or the module has which class of cohesion. And, then we are trying to do the same thing with respect to coupling between two modules. And, we said that coupling is the degree of dependence between two modules or the degree to which they interact or exchange data with each other. If there is no interaction between two modules then there is no coupling.

If, they interact only by exchanging some very simple data items like an integer or a character, then there is very low coupling called as a data coupling. But if they interact by exchanging complex data like; arrays, trees, lists, very complex structures and so on then we called it as a stamp coupling. Now, let's proceed that can we by looking at the structure of two modules that are given to us, can we say that how good or bad is the coupling between these two modules? So, let's proceed from that point.

(Refer Slide Time: 02:41)



We said that the coupling existing between two modules can be classified into five classes: data coupling, stamp coupling, control coupling, common coupling and content coupling. Content coupling is the worst form of coupling and nowadays the modern programming languages, they don't allow programmers to write program, where module have content coupling. Now, let's look at these different forms of coupling and we even look at the case of content coupling just to be aware of the issues involved.

(Refer Slide Time: 03:24)

The diagram shows a slide titled "Data coupling". The slide contains a bulleted list of characteristics of data coupling:

- Two modules are data coupled,
 - if they communicate via a parameter:
 - an elementary data item,
 - e.g an integer, a float, a character, etc.
 - The data item should be problem related:
 - not used for control purpose.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". On the right side of the slide, there is a video player showing a man speaking.

In data coupling, two modules they interact, but then they exchange only very simple data items, for example, just an integer, a two integers or may be a floating point number, may be some character and so on. And of course, one thing we must be sure that these are used like data members in the other module to which it is invoked and this should not be used to set a control parameter in the other module. In that case, we call it as a data coupling.

(Refer Slide Time: 04:11)

Stamp coupling

- Two modules are stamp coupled,
 - if they communicate via a composite data item
 - or an array or structure in C.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Slightly worse form of coupling is the stamp coupling. This also easy to identify a given a module structure whether it have stamp coupling or not. If, we find that one module invokes another module and interacts by passing or receiving a complex data structure like an array, linked list, structure et cetera. Then, we say that it is a case of stamp coupling.

(Refer Slide Time: 04:44)

The slide has a yellow background with a title 'Control coupling' in bold black font. Below the title is a bulleted list of points. To the right of the list is a hand-drawn diagram in red ink. The diagram shows two rectangular boxes. The left box contains the text 'f1()' and the right box contains 'switch()'. An arrow points from the word 'switch()' in the right box to the right side of the word 'switch()' in the left box, indicating a control flow or dependency.

- Data from one module is used to direct
 - order of instruction execution in another.
- Example of control coupling:
 - a flag set in one module and tested in another module.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video feed of a speaker on the right.

A still worse form coupling is the control coupling. Here, the two modules pass data items between each other, but then the data item passed by one module is used as a control element in another module. For example, it may be that they send an integer which is used as a flag tested in another module. So, let's say we have two modules here. And, let's say this module invokes a function let's say f_1 and f_1 passes an integer i and here this integer i is used as a switch on integer; one of the functions switches on integer i and depending on the integer value i , it does different things. So, the value that is passed here is used as a control element in the other module. And, this is also a bad case of coupling. It makes understanding the design very difficult and we call this as a case of control coupling.

(Refer Slide Time: 06:19)

Common Coupling

- Two modules are common coupled,
-if they share some global data.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A video player window shows a man speaking.

Another bad form of coupling is the common coupling. Here two modules are coupled, but they are coupled through some global data. Normally, if we have a module here and another module here (on the above slide). These two module have their own data and the different functions. Let say one module is M1 and another module is M2 (on above slide) and these module has some data that is visible within the module. Let say there are functions f1, f2 in module M1 and let say, f5, f6 functions in M2. Now, let say there have some global and these two module us those global data. And, different functions within the module access the global data and communicate among by using the global data. So, this type of coupling is bad case of coupling, where, we have the modules communicating with the help of some global data. This type of coupling we call as the common coupling and it's a bad case of coupling. If there is a common coupling, then we say that the modules are highly coupled.

(Refer Slide Time: 07:53)

The slide has a yellow header with the title 'Content coupling'. Below the title is a bulleted list:

- Content coupling exists between two modules:
 - if they share code,
 - e.g, branching from one module into another module.
- The degree of coupling increases
 - from data coupling to content coupling.

On the right side of the slide, there is a small hand-drawn diagram showing two rectangular boxes labeled M1 and M2. A red arrow points from M1 to M2, and another red arrow points back from M2 to M1, illustrating the concept of content coupling where one module branches into another.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a person speaking.

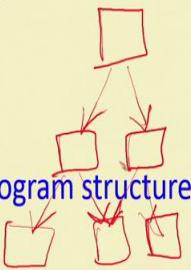
And, the worst form of coupling is the content coupling. Modern programming languages, does not allow this kind of coupling. So, you cannot even write a code using modern programming languages, where they do have content coupling. But earlier in machine language and assembly language program in old computers, it was possible to have two modules which have content coupling.

For example, we have two module M1 and M2 (on the above slide) and these modules share code. It is possible for M1 to execute some piece of code of M2 and come back to its own code. So, just because we don't want to write a certain amount of code in M1, M1 just had a jump here to M2 module and use the code of M2 and, then again jump back. So, this a very bad form of coupling, it makes understanding this module extremely difficult. And, it's a bad programming which is not allowed in the modern languages. Using modern programming language, you cannot just jump into another module and executes some parts of another module. Let say, we have a function f1 in M2 and we jump into the middle of this function and run few statements of f1 and come back in M1. So, that form of thing is not permissible. Either you call the full function or don't use anything here. You can not just jump into one module, run part of the function and come back. So, this is the case of a content coupling and largely disallowed in modern programming languages.

(Refer Slide Time: 09:46)

Hierarchical Design

- Control hierarchy represents:
 - organization of modules.
 - control hierarchy is also called program structure.
- Most common notation:
 - a tree-like diagram called structure chart.



The slide is titled "Hierarchical Design". It contains two main bullet points. The first point discusses the control hierarchy representing the organization of modules, mentioning that it is also called "program structure". The second point states that the most common notation for this is a tree-like diagram called a "structure chart". At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

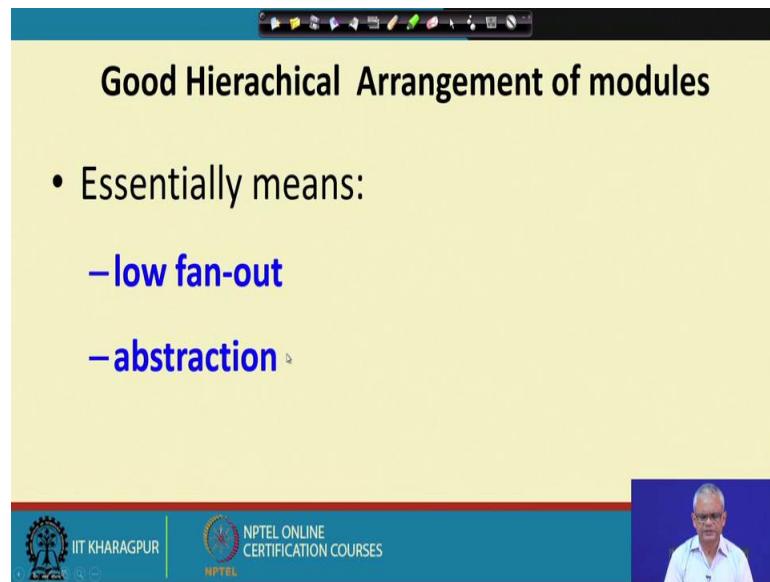
Now, let's look at other aspects of a good design: one is the hierarchical design. Here, we look at how the modules call each other and represent that as a control hierarchy (Diagram on the above slide). In the above diagram we can see, first top module calls below two modules and these two modules in turn they call other modules. And, this we call as the control hierarchy and this is also called as the program structure and typically represented by a structure chart.

We look at the structure chart in next couple of lectures. And, this program structure is a tree like structure and it's hierarchical. Using a hierarchical structure a good design is organized. So, we say that a good design has a hierarchical module structure and it has a well-defined control hierarchy.

(Refer Slide Time: 11:05)

Good Hierarchical Arrangement of modules

- Essentially means:
 - low fan-out
 - abstraction

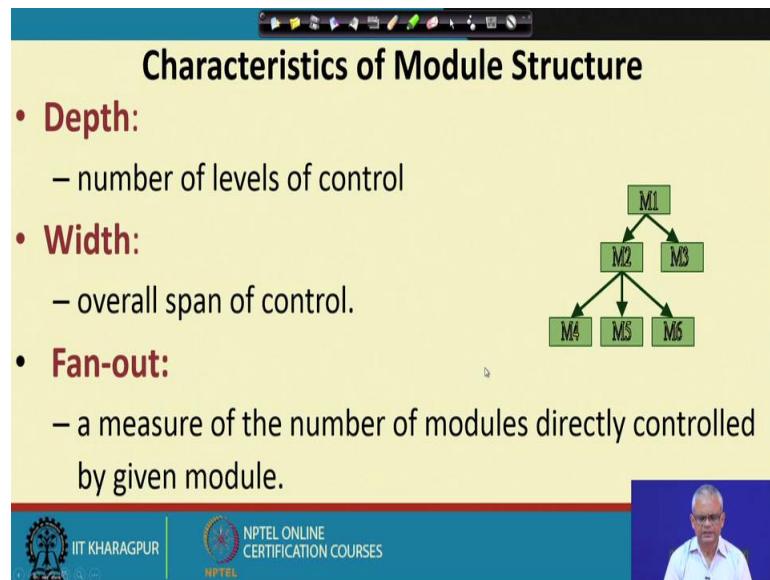


But then given a control structure, how do we say that whether the control structure is good or bad? For that we need to define two terms: fan-out abstraction, fan-in abstraction. A good design should have low fan out, it should have abstraction, what we mean by that?

(Refer Slide Time: 11:33)

Characteristics of Module Structure

- Depth:
 - number of levels of control
- Width:
 - overall span of control.
- Fan-out:
 - a measure of the number of modules directly controlled by given module.

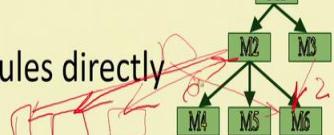


Let say this is a module structure of an application (on the above slide). How they call each other? And, into how many layers they are organized or levels of controlled?

The fan-out of every module is how many other modules it calls. For example, the module M1 fan-out is 2 here. For module M2 the fan-out is 3 here, for module M3 it is 0. So, the term fan-out says how many other module it invokes in a hierarchical structure.

(Refer Slide Time: 12:39)

Characteristics of Module Structure

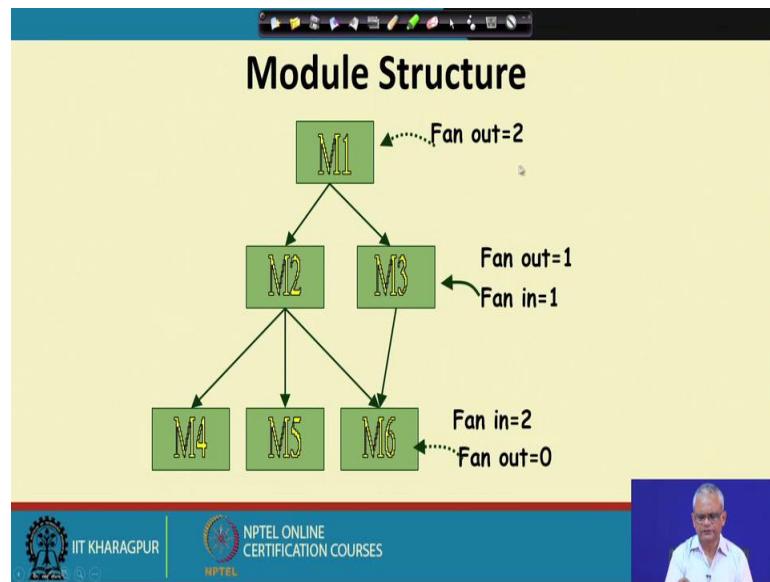
- Fan-in:
 - indicates how many modules directly invoke a given module.
 - High fan-in represents code reuse and is in general encouraged.

IT Kharagpur | NPTEL Online Certification Courses



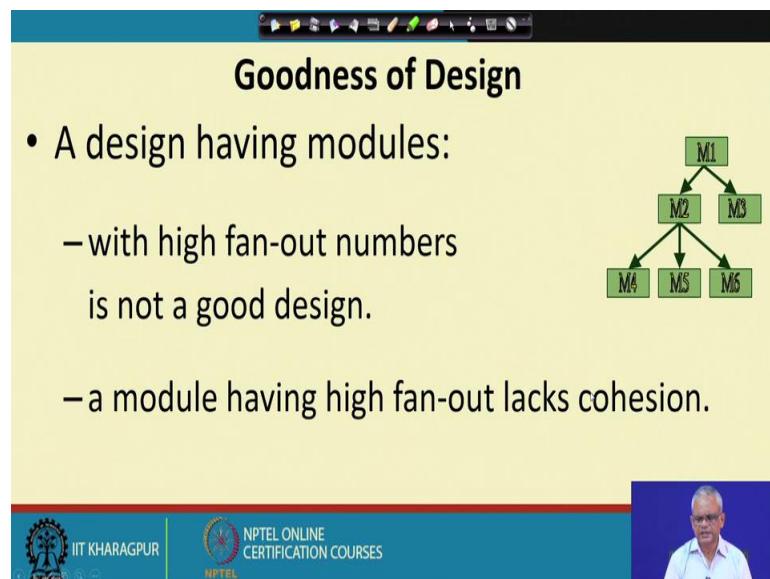
Now, what about fan-in? Fan-in for a module is how many different modules call this module. For example, here the fan-in of every module is 1. But let say we also had M3 calling module M6. Then the fan in of module M6 will be 2. Now, one thing to remember is that a very high fan-out is not good. Let say, we can see from the diagram that M2 fan-out is 3. If M2 calls let say another three modules, then we say the fan-out of module M2 is 6. A high fan-out is bad because M2 is depending on too many other modules. And, it's a hint or it's likely that M2 has a very bad cohesion. It just doing too many things and that's why it is needing many modules. A high fan-out is bad, but a high fan-in is a good thing, because there are many modules which will depend on a high fan-in module. Let say there are other modules also, which are depending on M5. So, a high fan-in is good because it means that we are able to achieve reuse of code and that's a good design. High fan-out is bad, but a high fan-in is good.

(Refer Slide Time: 14:35)



This is an example where the fan-out of this M1 is 2, for this M3 fan-out is 1 and fan-in is also 1, for M6 fan-out is 0 and fan-in is 2.

(Refer Slide Time: 14:53)

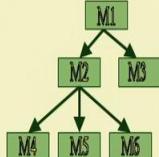


We can compute the fan-in and fan-out and if we have a very high fan-out for some modules, we will say that it indicates that the design lacks cohesion. But if there is high fan-in then that's not bad, it's actually good design. If there is a high fan-in then good reuse can be observed in that application.

(Refer Slide Time: 15:31)

Large Fan Out

- A module that invokes a large number of other modules:
 - likely to implement several different functions:
 - **not likely to perform a single cohesive function.**



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

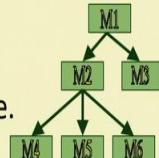


A large fan-out as we are mentioning is that it is an indication, that a module is doing very different things and its cohesion will be poor. Because, it's needing too many other functions, it's invoking them different modules.

(Refer Slide Time: 15:54)

Control Relationships

- A module that controls another module:
 - said to be superordinate to the later module.
- Conversely, a module controlled by another module:
 - said to be subordinate to the later module.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

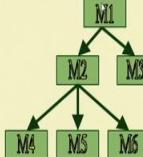


There are few other terminology with respect to the control relationship. From the above diagram on slide, we can say that M1 is superordinate of M2 and M3 or we can say that M1 controls M2 and M3. In other words, we can say that M2 and M3 are subordinate of M1 and M4, M5 and M6 are subordinate of M2.

(Refer Slide Time: 16:26)

Visibility and Layering

- A module A is said to be visible by another module B,
 - if A directly or indirectly calls B.
- The layering principle requires:
 - modules at a layer can call only the modules immediately below it.



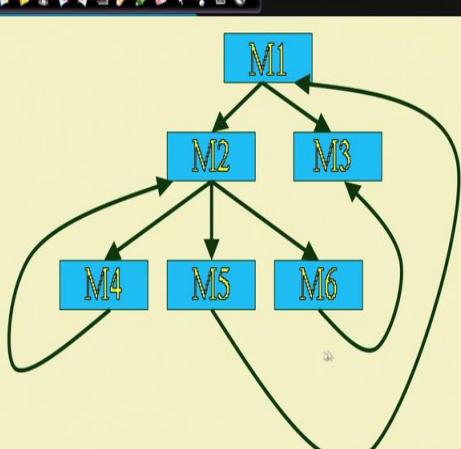
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



M1 calls M2 and M3 and therefore, M2 and M3 are visible to M1, but M1 is not visible to M2 and M3. Similarly, from M2 the visibility is M4, M5, and M6. So, we say that superordinate modules are actually abstractions to subordinate modules and they are not visible to subordinate modules. In our above diagram, M1 is the most abstract module and M4 is the most concrete module and this is a layered structure. First of all, to have a good design we must have a layered structure. That is the control hierarchy, we should be able to organize into distinct layers and then we can identify the fan-in, fan-out and so on.

(Refer Slide Time: 17:38)

Bad Design



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



In the above slide, we can see a bad case of the design. First of all, there is no layering and also it's not a simple tree like structure and it violates the principle of abstraction. For example, in our previous design the most concrete module was M4, M5, and M6. And, they should not be able to see the higher-level modules M1 et cetera. They should not be visible here, but in this diagram, M5 is invoking M1, so M1 is visible to M5. So, this will be a very bad case of design where it will be very difficult to debug, because we can see from here that it has some cyclical relation. For cyclical relation, it may happen, for a given bug, we may just go round and round and still not be able to trace the bug.

But, if it was a strict tree like structure with layering. If we observe a bug in any module then that is either on that module or with its subordinate modules. So, understandability, debugging, reuse et cetera are facilitated by a layered design. And, if we have a design like cyclic relation which violates the principle of abstraction, layering and it's a case of bad design.

(Refer Slide Time: 19:16)

Abstraction

- **Lower-level modules:**
 - Perform input/output and other low-level functions.
- **Upper-level modules:**
 - Perform more managerial functions.

The diagram illustrates a hierarchical module structure. At the top level is a box labeled 'M1'. Two arrows point downwards from 'M1' to two boxes labeled 'M2' and 'M3'. From 'M2', two arrows point downwards to two boxes labeled 'M4' and 'M5'. From 'M3', one arrow points downwards to a box labeled 'M6'. This represents a violation of abstraction principles as M5 is shown to invoke M1, which it should not be able to do in a good design.

In a good design, the modules are layered and the top level modules are the abstract modules. Top level modules do more managerial function, they just invoke the lower level modules to do some work. Top level modules invoke some other modules which we call as the middle managers. Middle managers again invoke other modules which are the lowest level modules. Lowest level modules do input output. So, lowest level modules take some input and after processing the input, give some output.

(Refer Slide Time: 20:01)

Abstraction

- The principle of abstraction requires:
 - lower-level modules do not invoke functions of higher level modules.
 - Also known as layered design.

```
graph TD; M1[M1] --> M2[M2]; M1 --> M3[M3]; M2 --> M4[M4]; M2 --> M5[M5]; M3 --> M6[M6]
```

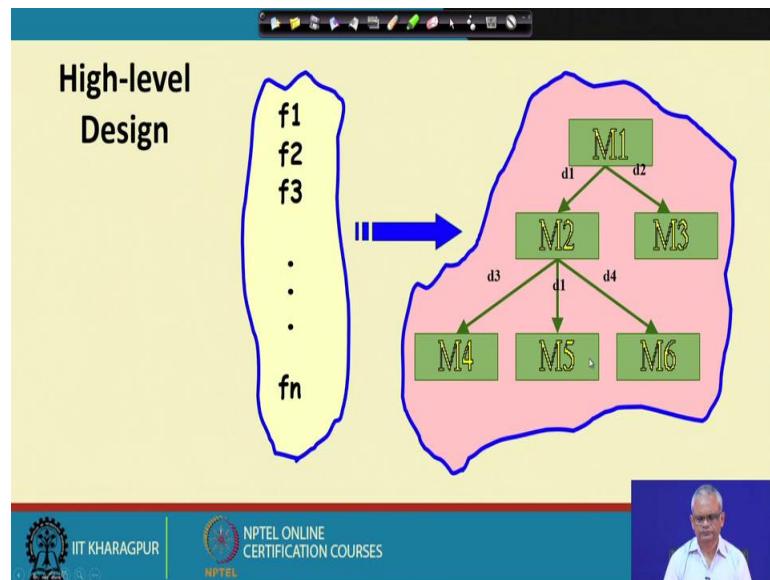
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



So, M1 calls M2, M3 and M2, M3 do some processing, again get data from M4, M5, M6 and do some processing and pass it onto a M1 and so on. In a good layered design, a lower layered design should not be able to invoke an upper level design and that is invisible, that should be invisible to the lower level modules.

(Refer Slide Time: 20:23)



We can therefore say that if we write a program and identify that a set of functions f1 to fn are to be written, and then the task of high level design is to organize those functions into a module structure. We should be able to assign these functions to different module such

that each module has high cohesion and there is low coupling among modules and also they are organized in a tree like expression. And, this is the crux of the procedural design approach, as well as for the object-oriented design approach. This principle is used for both design approach. So, here observe that we identify what are the functions in our applications to be written, and then these functions are organized into different modules such that each module has high cohesion and low coupling and also, these are organized in a nice tree like expression.

In next few lecture, we will do some procedural design. There we will identify the functions by using a data flow diagram, structured analysis principle and, then by using a structure chart, structure design, we will be able to come up with this module structure and a tree like diagram.

(Refer Slide Time: 22:03)

Design Approaches

- Two fundamentally different software design approaches:
 - Function-oriented design
 - Object-oriented design

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Two design approaches are popular one is the function-oriented design and other is the object-oriented design. Given a design, what are desirable of a design? So, we can say that this design should be functionally independent, tree like structure, layered and so on. We will look at two approaches: the function-oriented design and object-oriented design, to arrive at a good design structure. But before we look at the nitty gritty of these two design techniques, let's have some very overall understanding of what is involved in functionally-oriented design and object-oriented design.

(Refer Slide Time: 22:50)

Design Approaches

- These two design approaches are radically different.
 - However, are complementary
 - rather than competing techniques.
 - Each technique is applicable at
 - different stages of the design process.

These two design techniques even though popularly they are believed that they are computing approaches, but not really as we proceed, we will see that they are complimentary approaches, because object-oriented design also uses the principles of procedural design. So, the procedural design techniques are also used as part of the object-oriented design, we will examine those results.

(Refer Slide Time: 23:29)

Function-Oriented Design

- A system is looked upon as something
 - that performs a set of functions.
- Starting at this high-level view of the system:
 - each function is successively refined into more detailed functions (top-down decomposition).
 - Functions are mapped to a module structure.

First let's looks at the function-oriented design. Here, by looking at the description of the problem we will identify the set of functions which needs to perform. Once we identify

the set of functions, then we will map this into a module structure. Identifying the set of functions to be performed is called as the structure analysis. And, then these identified set of functions, will map into a module structure. This module structure is also called as structure design. We will also call it as the top down principle, because initially we consider the system as performing various large granularity functions. And, then we will split this into small functions. Will successively refine into more detailed functions or smaller functions. And, then there will be a large number of functions and each of those we will map into the module structure. And, that's the reason why it is called as a top down decomposition principle. As, we will study this function-oriented design techniques in more detail, it will become clear why it is called a top down decomposition. Once, we achieve the top down decomposition we map the function structure into a module structure.

(Refer Slide Time: 25:20)

Example

- The function **create-new-library-member**:
 - creates the record for a new member,
 - assigns a unique membership number
 - prints a bill towards the membership

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Just to give an example of a top down decomposition, let say in a library software we have a function named as create new library member. Now, we can see that the create new library member requires some activities to be performed. For example, we need to first create a record for the new member, we will have to assign a membership number and also we will have to print a bill towards the membership. So, we can consider that the function create new library member to be consisting of smaller functions or sub functions, that is create record, assign a unique membership id and then print bill. Similarly, how we say, for every function, what are the activities to be performed under

that? For that we can split the function into simpler set of functions. And, we can do that recursively in the sense that we can again look at the sub functions and find if this can be split into still simpler functions.

(Refer Slide Time: 26:40)

The slide has a yellow background and a blue header bar. The title 'Function-Oriented Design' is centered in the yellow area. Below the title is a bulleted list of characteristics:

- The system state is centralized:
 - accessible to different functions,
 - member-records:
 - available for reference and updation to several functions:
 - create-new-member
 - delete-member
 - update-member-record

At the bottom left, there is a logo for IIT Kharagpur and text 'IIT KHARAGPUR'. To its right is a logo for NPTEL and text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the slide, there is a small video window showing a man speaking.

This is one of the important principle of function-oriented design that we look at what are the functions that the system performs and then split those into simpler functions. And, the other thing that to notice about function-oriented design is that all these functions, they operated some centralized data structure. For example, in a library system, all the functions may be needing to operate on the set of books or the set of member records and so on.

So, these are two distinguish characteristic of a function-oriented design: First one is we need to identify the functions that the systems performs, break those function into simpler functions and second one is identify the data that are centralized and all these functions operated on that. We will take one example and see how we go about doing the function-oriented design. Later we will look at this technique in more detail, we will look at the data flow diagram in technique to do the structured analysis and, then we take the structure chart and see how to do the structure design.

We will stop here and continue from this point in the next lecture.

Thank you.

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

Lecture – 23
Introduction to structured analysis and structured design

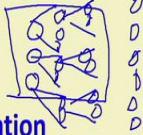
Welcome to this lecture. In the last few lectures we had looked at requirement specification and we saw that, that it is a very important issue. Before starting to develop it's very important to know the exact customer requirements, find out if there are any difficult is with the requirements and then eliminate them and finally, document that in an acceptable format. And we had seen the IEEE 830 format which is accepted across all industries and academics. And we had seen the different elements of the requirements specification, the functional requirements, non-functional requirements and so on and then how to document this.

Now, let's start looking at the structured analysis and design. We had also seen some very basic concepts about design namely the high-level design, detailed design and also what can be considered is a good design in terms of modularity, cohesion, coupling, layering and so on. Now, let's start to discuss how to go about doing procedural design of a problem. We assume that the requirement specification is done to complete documentation and we would like to start doing the procedural design for the problem. Let's see how we go about and address this issue. In this lecture and next few lectures, we will address the topic of Structured Analysis and Design.

(Refer Slide Time: 02:25)

Introduction

- Structured analysis is a top-down decomposition technique:
 - DFD (Data Flow Diagram) is the modelling technique used
 - Functional requirements are modelled and decomposed.
- Why model functionalities?
 - **Functional requirements exploration and validation**



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Structured analysis is a top down decomposition technique, in the sense that we will start with very gross or course level requirements that is as specified in the SRS document. The functionalities that the system should perform we start from them and then we do a top down decomposition; meaning that we break the required functionalities of a system into finer levels. And for this purpose, we will use the data flow diagram technique. We will use the data flow diagram (DFD) technique to do the functional requirements analysis. We will model the requirements and we will decompose into smaller functions.

But before we started it's good to know that why we need to model the functionalities? We need to model the functionalities because we will do the requirements exploration and validation. By this we mean that the functional requirements as specified in the SRS document will take exactly the same functional requirements and we will break them down into simpler functions in a hierarchical manner. What we mean by that is that if we have SRS document for our system and we have in the functional requirement section (Various function listed here), then we will take each function and we will use DFD to model this system. We will represent all these functions and also, we will decompose this into finer functions. Each of the functions will be decomposed. Sub functions also decomposed through the DFD technique and finally, we will have a set of fine granularity functions. So, this we call as the functional requirements exploration because we starting from very gross level requirements specified in the SRS document, we decompose them into functions which are very small in size.

(Refer Slide Time: 05:26)

Introduction

- Structured analysis is a top-down decomposition technique:
 - DFD (Data Flow Diagram) is the modelling technique used
 - Functional requirements are modelled and decomposed.
- Why model functionalities?
 - **Functional requirements exploration and validation**
 - **Serves as the starting point for design.**



Once we achieve this decomposition, we will be ready for starting the design. So, in the structured analysis we do the functional requirements exploration that is break the functions into fine functions and then once we have the fine set of functions, we then start the design.

(Refer Slide Time: 05:55)

Function-oriented vs. Object-oriented Design

- Two distinct style of design:
 - **Function-oriented or Procedural**
 - Top-down approach
 - Carried out using Structured analysis and structured design
 - Coded using languages such as C
 - **Object-oriented**
 - Bottom-up approach
 - Carried out using UML
 - Coded using languages such as Java, C++, C#



But before we start looking at the structured analysis and design, we will do a small digression. We will contrast the function-oriented and the object-oriented styles because right now, in this lecture and next few lectures, we will be looking at the function-

oriented design and later we will be concentrating on the object-oriented design. function-oriented design and the object-oriented design are two distinct design styles. The function-oriented or the procedural design is a top-down design technique as we are saying that we start with the requirements, take the functions, and decompose them hierarchically into a fine set of functions.

After the decomposition, we map them into a design structure and we use the structured analysis and structure design for the procedural design technique. And then once we have got the design then we can easily code it using a language such as C.

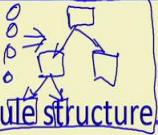
On the other hand, object-oriented design technique is a bottom-up technique. Here based on the requirements, we first identify the objects and then we see the object relations if there are any, then we model the object relations to build a hierarchy.

So, we start with the basic objects and then we group the objects into classes and then classes into an inheritance hierarchy and so on. So, this is a big difference that function oriented is very intuitive top down approach and object-oriented is a bottom up approach. In object-oriented approach we start with objects and build from there and we will use the UML notation to do the object-oriented design. Once we have the design ready then this can be easily coded in a language like Java, C++ or C# .

(Refer Slide Time: 08:35)

Structured analysis and Structured Design

- During Structured analysis:
 - High-level functions are successively decomposed:
 - Into more detailed functions.
- During Structured design:
 - The detailed functions are mapped to a module structure.



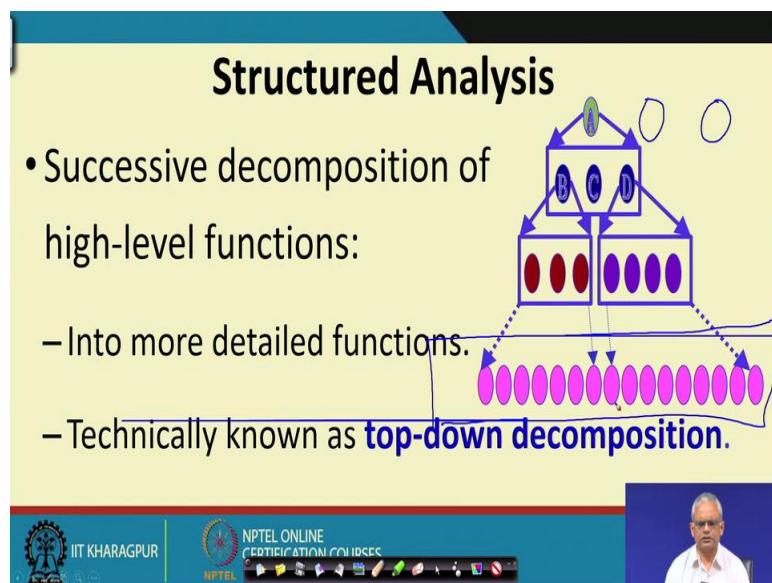
 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES 



So, now let's get into our discussion on structured analysis and design. So, here as we are saying that we will start with the high-level functions as specified in the SRS document and then we will decompose into detailed functions. This we will call as the structured analysis. So, this is the structured analysis activity where we do a functional decomposition.

On the other hand, immediately after the functional decomposition we will start the structure design and here we take the functions identified in the structured analysis. After identifying functions in the structured analysis, we will map these set of functions into a module structure. We will discuss a technique called as a structured design technique, which gives a methodology how to map these set of fine level functions into a module structure and this will form our high-level design. We will first discuss the structured analysis where we identify the fine-grained functions and next, we look at the structure design, where we discuss, how to map the fine-grained functions into a module hierarchy.

(Refer Slide Time: 10:28)



Now, for the next hour or so we look at the structured analysis where we do a top down decomposition starting with a function identified in the SRS document, we split into sub functions. This diagram (On above slide) just explains this concept. From the diagram, we can see functions are decomposed into sub functions.

For decomposition we will use the DFD. We will use DFD notation for modeling decomposition. And then each sub function into further sub functions and same thing will do for the other high-level functions mentioned in the SRS document. And at the end we will have the set of fine-grained functions and this we will mapped into the design structure.

(Refer Slide Time: 12:00)

SA/SD (Structured Analysis/Structured Design)

- SA/SD technique draws heavily from the following methodologies:
 - Constantine and Yourdon's methodology
 - Hatley and Pirbhai's methodology
 - Gane and Sarson's methodology
 - DeMarco and Yourdon's methodology
- SA/SD technique results in:
 - high-level design.

We largely use

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES** | **NPTEL**

The structured analysis and structure design technique is well accepted by the developers. Anybody wanting to do a procedural design would do the structured analysis and structured design, but then there are various flavors of these design. For example, someone may use the Constantine and Yourdon's methodology or Hatley and Pirbhai's methodology or Gane and Sarson's methodology or DeMarco and Yourdon's methodology.

So, the main point is that there is no standardization of the methodologies. We can find in different industries, using different methodologies. These are largely the same, but then there are small differences in the notations. For our discussion we will be using largely the Hatley and Pirbhai's methodology. We will see that the different tools that are available, they also support number of methodologies.

Hatley and Pirbhai's is a methodology which has come much after Constantine and Yourdon and Gane and Sarson's methodology. So, this Hatley and Pirbhai's methodology is a more recent methodology and obviously, they had the privilege of

making use of the ideas on the Constantine and Yourdon, Gane and Sarson's, and DeMarco Yourdon methodology. In our discussion we will be using the Hatley Pirbhai's methodology to a large extent. Once we do the structured analysis and structure design, we will have the high-level design. In the high-level design we will have the module structure and in the module structure we can use this methodology for coding and the final implementation.

(Refer Slide Time: 14:17)

Functional Decomposition

- Each function is analyzed:
 - Hierarchically decomposed into more detailed functions.
 - Simultaneous decomposition of high-level data
 - Into more detailed data.

Now, let see how do you achieve functional decomposition because this is one of the main objectives of the structured analysis. We do functional decomposition through a hierarchy of levels where we decompose each function into sub functions and so on. And as we decompose the sub functions at the same time, we will observe that the data also gets decomposed. Starting from the very high-level data that are mentioned in the SRS document, we decompose that simultaneously and automatically into detailed data. And finally, we have primitive data elements at the lowest level in terms of integer, character and so on.

(Refer Slide Time: 15:20)

Structured Analysis

- Textual problem description converted into a graphic model.
 - Done using **data flow diagrams (DFDs)**.
 - DFD graphically represents the results of structured analysis.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

For doing the structured analysis, we will use the DFD technique. DFD stands for data flow diagrams and this is a very intuitive and simple technique. It takes hardly few minutes to get familiar and get productive and get started using this technique. And because of its simplicity and utility this technique is extremely popular not only in the procedural design, but also in many other applications. For example, in the SRS document itself in many projects they develop the DFDs because this gives a good understanding of the functions into a finer level of activities. And also, the DFD model is a very intuitive model and by just looking at it we get a good idea about what the system needs to do.

(Refer Slide Time: 16:35)

- The results of structured analysis can be easily understood even by ordinary customers:
 - Does not require computer knowledge.
 - Directly represents customer's perception of the problem.
 - Uses customer's terminology for naming different functions and data.
- Results of structured analysis:
 - Can be reviewed by customers to check whether it captures all their requirements.

Structured Analysis

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

As we were saying that the structured analysis technique is a very simple technique, don't need to be even knowledgeable in computer; anybody can start using this technique. And it represents the customers perception of the problem. It's not really a design technique, it just keeps on elaborating what the customer wants of the problem. So, DFD is a detailed model of what the customer really wants of the problem and for this we also use the customers terminology for giving various names to the data, to the functions and so on.

So, this DFD technique is easily understood. The models developed using the DFD are easily understood by the customer, even though customer may not have computer exposure. Therefore, once the designers they come up with the DFD model of the system, it can be reviewed by the customers to check whether it missed any of their requirements and whether it has correctly captured their requirements or not. We will learn this technique and start modeling various problems in a very short time.

But let me just tell you that it is easy to learn this technique, but then for finally, using this technique to model various problems we need bit a practice.

(Refer Slide Time: 19:03)

Structured Design

- The functions represented in the DFD:
 - Mapped to a **module structure**.
- Module structure:
 - Also called **software architecture**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

A video player shows a man speaking.

Now, once we have completed the structured analysis, we will start the structured design. And here we will map the fine-grained functions into a module structure and this module structure is also called as the software architecture.

(Refer Slide Time: 19:21)

Structured Analysis vs. Structured Design

- Purpose of structured analysis:
 - Capture the detailed structure of the system as the user views it.
- Purpose of structured design:
 - Arrive at a form that is suitable for implementation in some programming language.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

A video player shows a man speaking.

One thing we must be clear is that the structured analysis is just an elaboration of the things that the customer wants. On the other hand, in structured design we arrive at a form that can be easily implemented, that can be easily coded into a solution. Obviously, if we are done only the structured analysis we cannot derive a code from that. But once

we have converted the structured analysis into a high-level design using the structured design technique that can be easily coded into the solution.

(Refer Slide Time: 20:17)

Structured Analysis: Recap

- Based on principles of:
 - Top-down decomposition approach.
 - Divide and conquer principle:
 - Each function is considered individually (i.e. isolated from other functions).
 - Decompose functions totally disregarding what happens in other functions.
 - Graphical representation of results using
 - Data flow diagrams (or bubble charts).

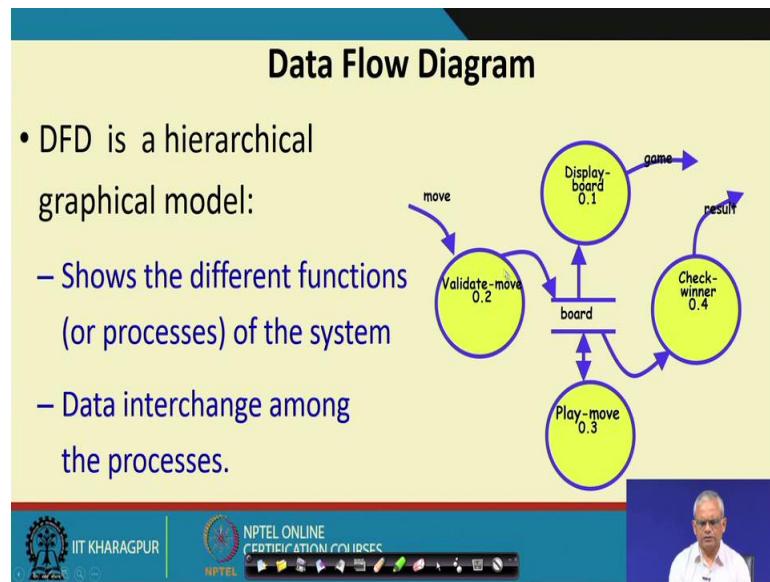
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL



Now we will just recapitulate what we discussed. We already discussed that the structured analysis is a top-down decomposition approach. We know what we mean by top-down decomposition and it's a divide and conquer principle. It uses the divide and conquer principle because, we look at all the functions in the SRS document, but take each function one by one and then at a time we decompose one function. Decompose one function means decompose it into sub functions and again at a time we look at one sub function and again decompose it and that is exactly is the divide and conquer principle.

(Refer Slide Time: 21:14)



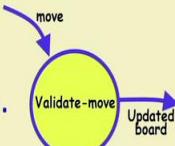
For this decomposition, we will use the DFD which is also called as bubble chart to represent our result. The DFD allows us to do a hierarchical decomposition and that is why the DFD is a hierarchical graphical model. We will have various levels of model. Some are very high-level model and then these are further decomposed into finer or more detailed levels. And at each level, we will represent the functions and also the data interchange among the processes. This is an example of a DFD model (On the above slide). In DFD, we will use the circles to represent the functions or the processes. Either we can call these as bubbles, processes or functions and then these bubbles or functions takes some input data produced some output data. So, it represents how the data flows within the system through the various functions.

Please remember that this is not a control flow model, it does not say that which flow occurs first and then which one occurs next and so on. It just represents the data flow basically. It represents what are the functionalities at this level and what data they consume and what data they produce. We will see the other notations that are used here in few minutes.

(Refer Slide Time: 23:10)

DFD Concepts

- It is useful to consider each function as a processing station:
 - Each function consumes some input data.
 - Produces some output data.

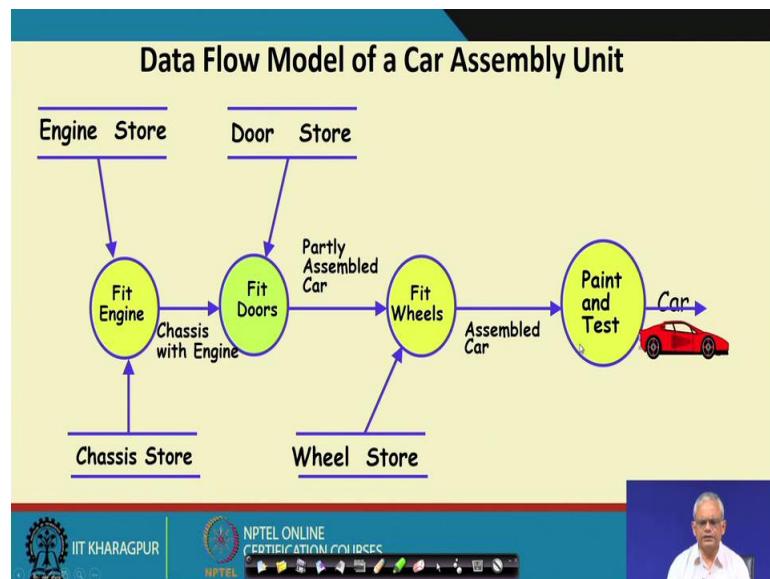


IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES



As we were saying that each of this function is represented using a circle and this represents some processing. It takes some input data, does some processing and produces some output data.

(Refer Slide Time: 23:37)



The model is extremely simple, we can easily understand a model developed using this. Now let just look at this car assembly unit modeled using the DFD notation. From the above slide, we can see the processes are fit engine, fit doors, fit wheels, and paint and

test. We can observe here that each of the process represents some activity and that's why these are starting with the verb form fit engine, fit door, fit wheel, paint and test etc.

So, every function represents an activity or processing and is named using a verb form. And then each process represents by bubble here. All these process takes some data as input. For example, 'Fit engine' process takes engines from the engine store, takes chassis from the chassis store and then fits the engine to the chassis. The chassis with the engine goes to the next processing station 'Fit doors' and here the doors are obtained from the door store and to the chassis with engine the doors are fitted. And this we call as the partly assembled car and then in 'Fit Wheels' the wheels are fitted by taking the wheels from the wheel store and then we have the assembled car. This is painted and tested in 'Paint and Test' process and the final output is the car.

(Refer Slide Time: 25:43)

The slide has a yellow background and a blue header bar. The title 'Pros of Data Flow Diagrams (DFDs)' is centered in the header. Below the title, there is a bulleted list of four points:

- A DFD model:
 - Uses limited types of symbols.
 - Simple set of rules
 - Easy to understand --- a hierarchical model.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video window showing a person speaking.

So, the model is very intuitive, just by looking at the model we can make out that what really happens. It's a very simple technique, as we are saying. We will see that in very small amount of time, we can start doing the DFDs for various problems. One of the main reasons why it is very simple technique is there are only five types of symbols. We can learn the five symbols in no time. There have some set of rules by which we combine these symbols to model. We start doing something very simple and slowly add more details to it and therefore, while even developing a very sophisticated model we find that it becomes easy.

So, with this discussion about the DFDs structured and introduction to structured analysis, we will stop here. We will continue from this point in the next lecture. We will see how to use the DFD technique, what are the nitty-gritty of the DFD technique and how to use this to model any given problem.

Thank you.

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

Lecture – 24
Basics of Data Flow Diagrams (DFD)

Welcome to this lecture. In the last lecture, we had started discussing how to performed structure analysis. In structured analysis, we take the functionalities from the SRS document and then we decompose the functions into a fine set of functions. We use the DFD technique to represent this. The DFD is the modeling technique and we had mention that DFD is very simple technique, we can learn it in no time and DFD is productive to model any given problem.

We just started to discuss about the DFD technique.

(Refer Slide Time: 01:07)

Pros of Data Flow Diagrams (DFDs)

- A DFD model:
 - Uses limited types of symbols.
 - Simple set of rules
 - Easy to understand --- a hierarchical model.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



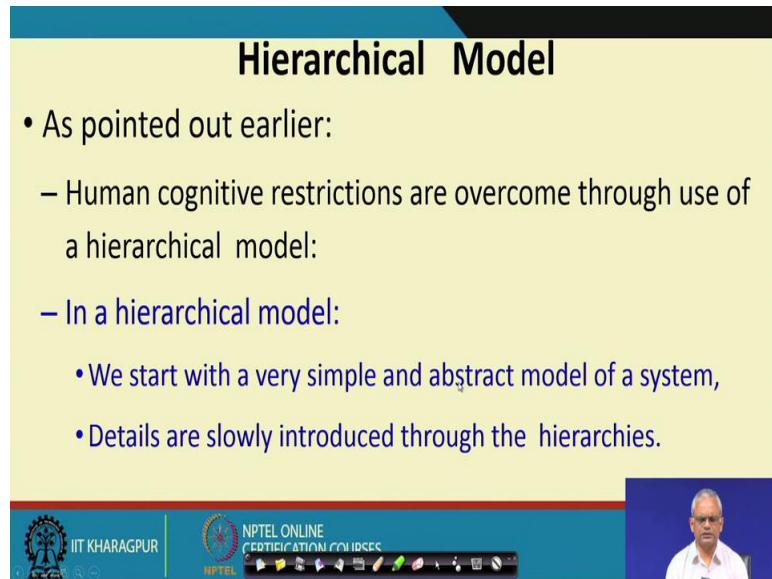
And we said that it uses a very limited set of symbols, five symbols to be precise. The set of rules in DFD are very simple and it is a hierarchical model. Because it's a hierarchical model we start with a very simple representation and then we slowly elaborate that into more detailed levels. For this reason, even when the problem is extremely complicated, we can come with very simple representation. Coming up with the first level representation is extremely simple and then each time we only add few details and

therefore, we don't even realize that we could so easily model even a very, very sophisticated system.

(Refer Slide Time: 02:03)

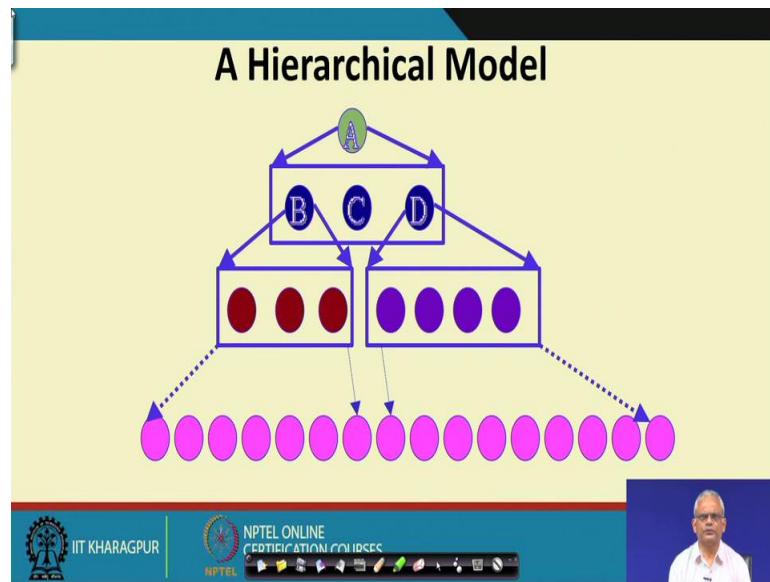
Hierarchical Model

- As pointed out earlier:
 - Human cognitive restrictions are overcome through use of a hierarchical model:
 - In a hierarchical model:
 - We start with a very simple and abstract model of a system,
 - Details are slowly introduced through the hierarchies.



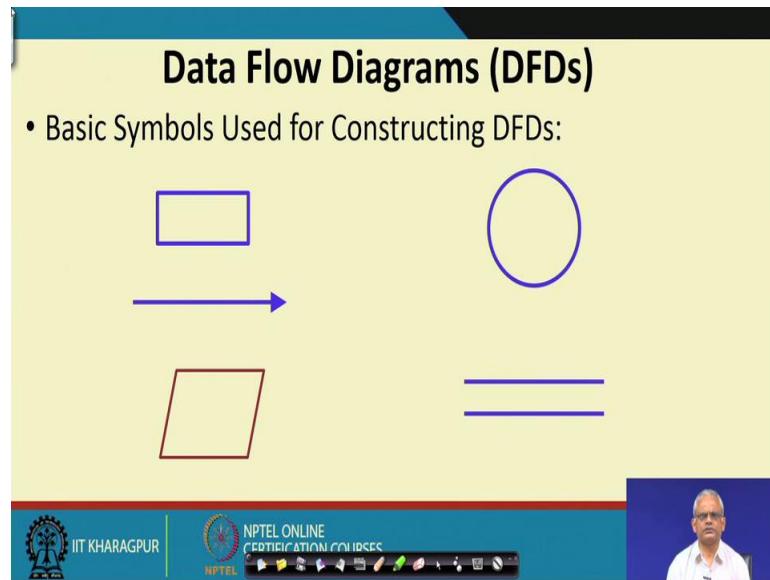
One of the main reasons, why this technique is very simple is that it is a hierarchical model and we had discussed in the early part of our lectures that it's the human cognitive restrictions which prevent us from easily understanding a very detailed description. The number of elements that we can recognize easily is restricted to five or seven and if our problem is represented in such a way that we start with the very simple representation of the problem and then over different hierarchies, we keep on adding small details to the problem. We can easily overcome the human cognitive restriction with the DFD model and that's the reason why DFD is a very simple model and very effective model.

(Refer Slide Time: 03:15)



DFD has various level: level 0, level 1, level 2, level 3 and so on and at any level we will add only very small amount of feature to the previous level, but at the end we will get a very detailed model of our system. Shown on above slide.

(Refer Slide Time: 03:48)



Now, let's look at the symbols that are used. There are only five symbols: rectangle, circle, an arrow, parallelogram and two parallel lines and once we know the meaning of all these symbols, we can straight away start using them.

(Refer Slide Time: 04:09)

External Entity Symbol

- Represented by a rectangle
- External entities are either users or external systems:
 - input data to the system or **Librarian**
 - consume data produced by the system.
 - Sometimes external entities are called **terminator, source, or sink.**

NPTEL ONLINE
CERTIFICATION COURSES



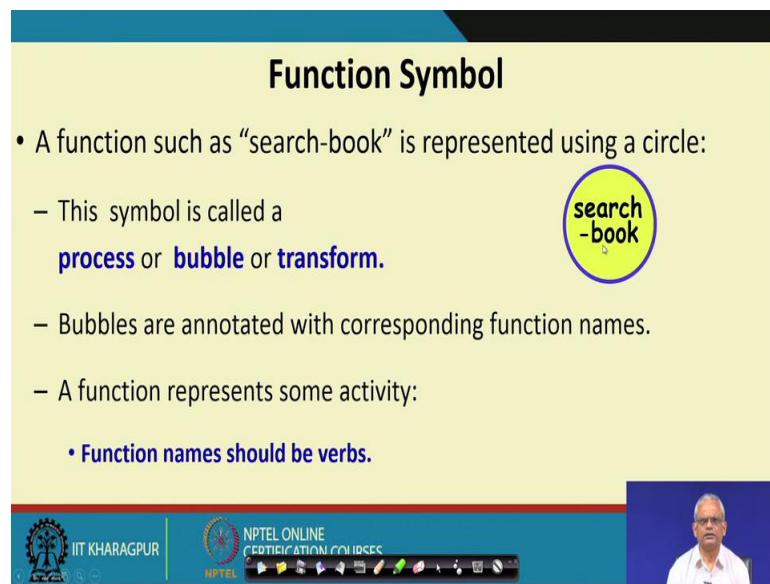
The first symbol we look at is the rectangle. Rectangle represent the user. We use it even to represent an external system for example, we might have a remote computer. We will represent the remote computer also using the rectangle and we write the name of the user or the external system inside the rectangle. For example, if librarian is a user for a library software, we just draw the rectangle and write the librarian there. And each user produces some data for the system and also it can consume some data that is produced by the system. And that's the reason why an external entity is also called as a terminator, a source or sink.

When we start using the tools, we will see that these terminologies are often used and, in some books, or in research papers these are referred as sometime by external entity, some time by terminator, source of data or sink of data. But the symbol is very simple: a rectangle represents an external entity.

(Refer Slide Time: 05:48)

Function Symbol

- A function such as “search-book” is represented using a circle:
 - This symbol is called a **process** or **bubble** or **transform**.
 - Bubbles are annotated with corresponding function names.
 - A function represents some activity:
 - Function names should be verbs.

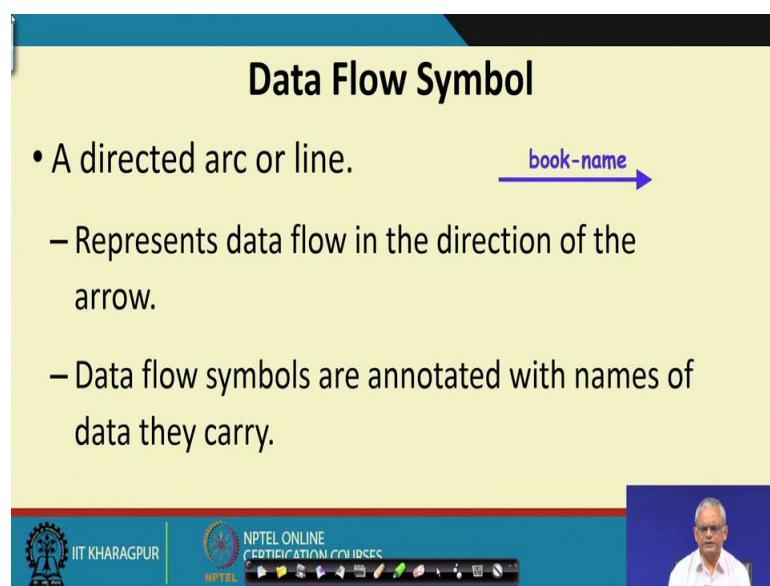


Another very important symbol here is a circle and this is called as a process, a bubble or a transfer. We write the name of the process or the function here inside the circle and as we mentioned, that since the processing represents some activity therefore, it has to be named using a verb form. For example, ‘Search book’ is a proper name for this function symbol or a process. But if we give a noun form here like let say, ‘search register’ or just ‘book search’ et cetera that is not proper. So, we have to give ‘search book’ which is a verb form.

(Refer Slide Time: 07:02)

Data Flow Symbol

- A directed arc or line.
- Represents data flow in the direction of the arrow.
- Data flow symbols are annotated with names of data they carry.



The third symbol is an arrow, which represents flow of data. A data flow symbol is used when it is produced by somebody and consumed by somebody. The data may be produced by a user and consumed by a process or it may be produced by a process and consumed by another process or may be another user. So, when data is produced by something and consumed by something, we use the arrow symbol. For example, if let say the data that is produced by an external entity let say, ‘Librarian’ and used by a process then we will use an arrow symbol from the librarian to the process.

(Refer Slide Time: 07:41)

Data Store Symbol

- Represents a logical file:
 - A logical file can be:
 - a data structure
 - a physical file on disk.
- Each data store is connected to a process:
 - By means of a data flow symbol.

The fourth type of symbol is a data store symbol, this is just two parallel lines and we name the data store. The data store actually represents a data structure it can be a physical file on the disk for example, we might have an array to contain the details of the book or array of structures to contain the book details. So that can be represented by two parallel lines and the name of the array of book details is the ‘book-details’ (On above slide). One thing to remember is a data store is not used by the end user or external entity it’s used by a process.

So, a books details maybe updated by a process or may be consumed by a process. So, each data store has to be connected to some process through a data flow symbol.

(Refer Slide Time: 09:50)

Data Store Symbol

- Direction of data flow arrow:
 - Shows whether data is being read from or written into it.
- An arrow into or out of a **data store**: Books
 - Implicitly represents the entire data of the data store
 - Arrows connecting to a data store need not be annotated with any data name.

In the above slide an example is given. Here, we have the ‘Books’ as the data store and the process ‘find-book’. The ‘find-book’ takes a book and finds in ‘Books’ data store and the direction of the arrow here shows whether the data is being read here or is being written. And one thing we must mention here that arrow implies here all the data in data store is used by the process with whom the data store connected. All the data becomes available to the process to use. It’s not that just one data from this data store is available to the process and another thing we need to mention is that since all the data are traverse on this arrow, so we don’t have to write the name of the data.

So, this data flow arrow is special which connects to a data store, we do not write the name of the data on the arrow. For all other situations, where there is an arrow connecting between two processors or between an end user and a process then, we have to write the name of the data on the data flow arrow.

(Refer Slide Time: 11:23)

Output Symbol: Parallelogram

- Output produced by the system



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

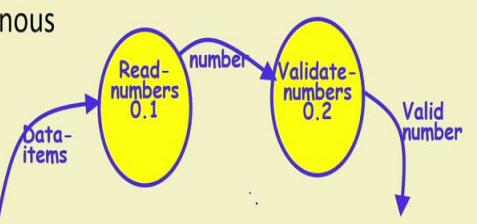
A video player interface showing a person speaking.

And lastly the fifth symbol is the parallelogram. The parallelogram represents the output produced by the system.

(Refer Slide Time: 11:35)

Synchronous Operation

- If two bubbles are directly connected by a data flow arrow:
 - They are synchronous



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A video player interface showing a person speaking.

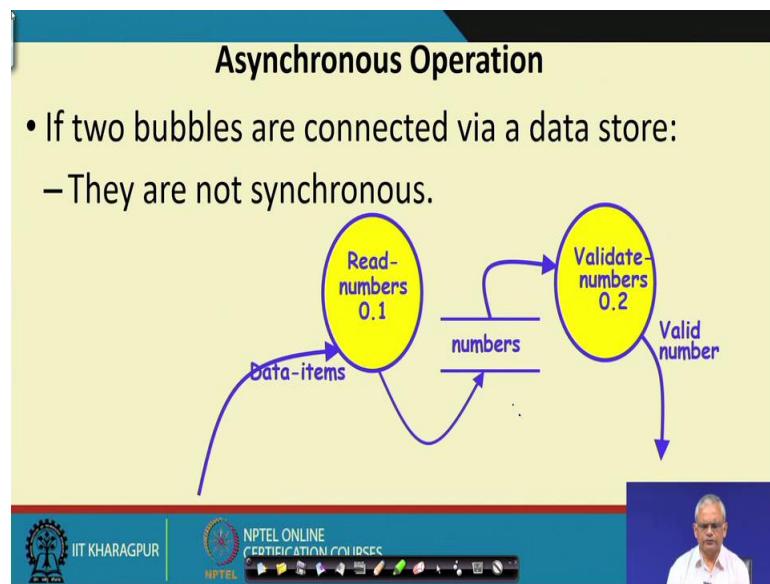
For example, a printout or a display is represent by this symbol.

So, we just saw that the number of symbols is very small, only five symbols. One symbol is for external entity. then we have the processing symbol and then we had the data flow symbol and then the data store symbol and finally, the output symbol.

Now before starting to model systems using the DFD notation, let's look at some other useful concepts. One is about synchronous operation. Let say we have this kind of connection (On the above slide) where there are two processes read number and validate number and they are connected by a data flow arrow and whenever we have a data flow arrow, we write the name of the data on that. The name of the data that flows on this arrow we write the name of that and we had mentioned that only when there is a data store, we don't write the name of the data otherwise we write the name of the data. And just see here that the 'Read-numbers' produces the number and it is consumed by the 'Validate-number'. Here we say that these two processes operate in a synchronous manner that is until the number is produced by 'Read-numbers', 'Validate-numbers' does not do anything.

Once the number is produced the 'Validate-number' starts to perform its activity and it produces the valid number. So, we can see that it's happening in synchronous manner that is the activity of this process validate number is dependent on when the read number produces the number.

(Refer Slide Time: 13:59)



On the other hand, we can also have asynchronous operation. We can also model asynchronous operation. If we model let say the same thing read number and validate number, but in between just see we have used a data store here (On the above slide).

Now, as the ‘Read-numbers’ takes the data items and produces numbers, they just kept on getting added to the store and here ‘Validate-numbers’ just takes them and starts processing to valid number. So here the speed of these two are not really the same. ‘Read-numbers’ may be producing many numbers before the validate number starts processing. This we called as asynchronous operation. So, it’s important to know that if two bubbles are connected using an arrow, we call that that is a synchronous connection between two bubbles. On the other hand, if they are connected through a data store then we say that they are asynchronously operating.

(Refer Slide Time: 15:22)

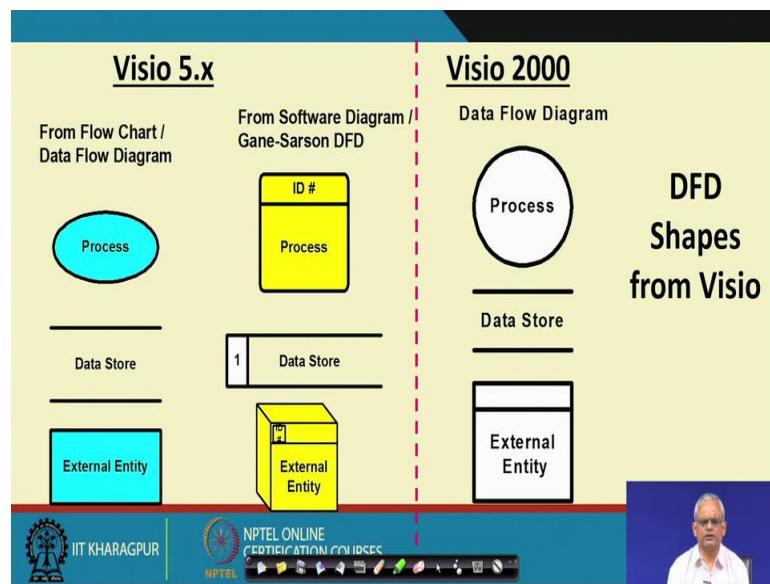
The slide has a blue header bar with the title 'Yourdon's vs. Gane Sarson Notations'. The main content area is yellow and contains the following bullet points:

- The notations that we are following:
 - Are closer to the Yourdon's notations
- You may sometimes find notations in books and used in some tools that are slightly different:
 - For example, the data store may look like a box with one end closed

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video window showing a person speaking.

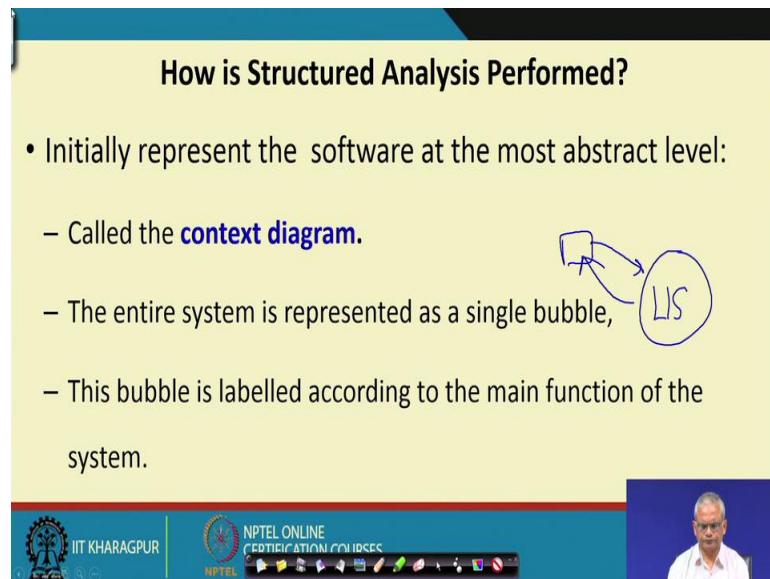
We are saying that there are many notations, methodologies in the structured analysis. Researcher have proposed different notations and slightly different methodologies for performing the structure analysis. Two major variants are the Yourdon’s methodology and the Gane and Sarson’s methodology. Even though we use here, Hatley pirbhai’s methodology which is very close to Yourdon’s notations, but you may sometimes find the other notation that is Gane and Sarson’s notation is used by some tools or some books and for that we must notice the difference between these two methodology.

(Refer Slide Time: 16:25)



For example, let say the tool Visio (On the above slide) which supports both methodologies:

(Refer Slide Time: 16:37)



Yourdon's methodology and the Gane and Sarson's methodology. From the slide of the Visio we can see, in Yourdon's methodology, the process is represented using an ellipse kind of thing and then the Gane Sarson's methodology it is represented using a rounded rectangle and the rectangle content the name of the process and some id number.

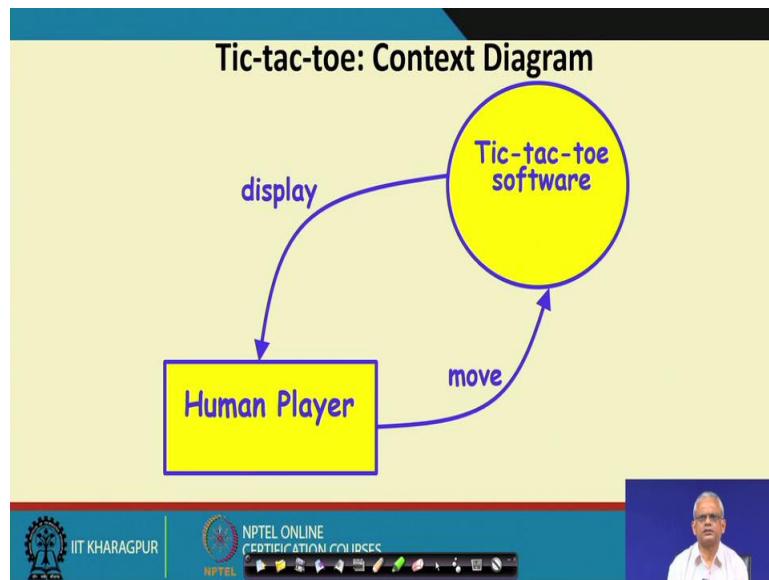
In the Yourdon's, the data store is represented by two parallel lines and external entity represented by rectangle. All these Yourdon's notions we discussed earlier. On the other hand, in the gain Sarson's, the data store represented by two parallel lines with two vertical lines also and the external entity is actually a cube kind of thing.

So, a rectangle is obviously easier to draw and the cube is more cumbersome to draw. In Visio 2000 (On the Visio slide), it uses the Yourdon's notations circle, two parallel lines and then external entity is actually slightly variant from other two we discussed. So, even though the methodologies are nearly the same and the notations are nearly the same, but then there are small variations that we must take into consideration. The reason is that there was no standardization effort. Whereas, in the object-oriented design we will see that even though there are also many variations of notations and methodologies existed, but there was a standardization effort and therefore, the unified modeling language (UML) was proposed and once that became standard, everybody used UML and therefore, the models there will appear uniform irrespective of which tool or which project you visit.

On the other hand, in procedural design, there can be small variations in the appearance of the diagrams and even in the methodology. Now let see how to do the structured analysis? As we are saying that this is a hierarchical model and we start with something very simple and then slowly elaborate this into a detailed model. The simplest model of a system is called as the context diagram this is also called as the level-0 DFD.

In the level-0 DFD the entire system is represented using a single bubble. So, we just draw one bubble and write the name of the software here. Let say the library information system. Normally that is the convention here that we first do the simplest representation of the system and for that we just draw one circle which represents the entire system we write the name of the system here and then if there are any users of the system that we identify then we draw that using rectangle and then connect them and write the name of the data that they produce and similarly the data that they consume that also we represent.

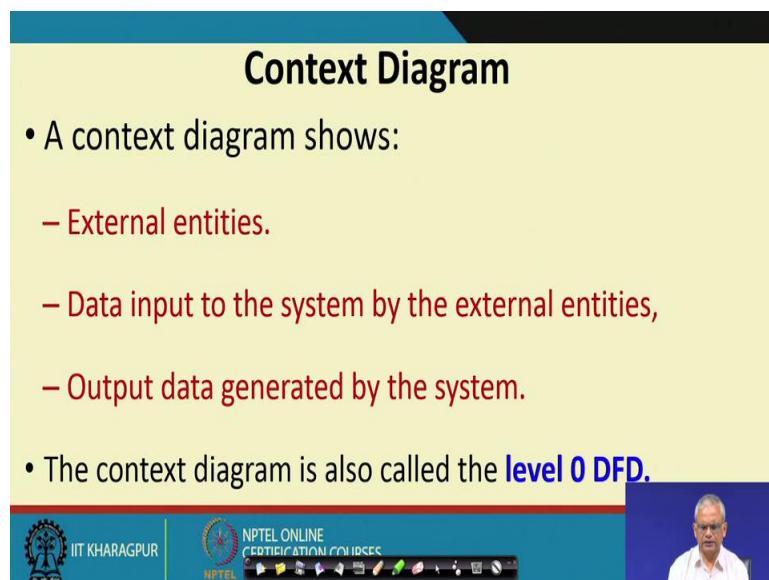
(Refer Slide Time: 21:02)



This is our first example (On the above slide) of the context diagram or the level-0 DFD. Name of the software is tic-tac-toe software. We write the name of the software in a circle and there is only one type of user here: who is the player, it's a human player. The human player enters moves into the system and the system produces the display.

So, at this level 0 or the context level we just write in what context the software exists.

(Refer Slide Time: 21:56)



The software exists in the context of its users and any other external entities. And then we also mention here what type of data they input and what type of data they consume.

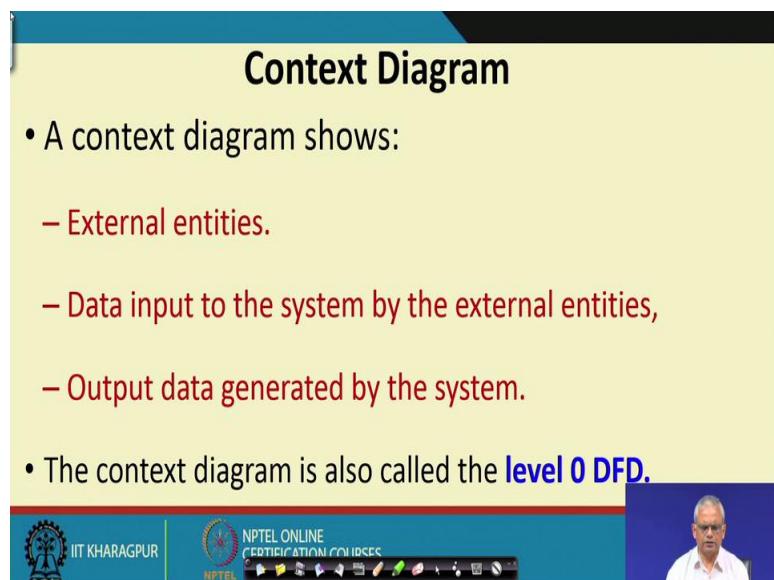
The human player inputs move to the software and in response the tic-tac-toe software produces display. We will do a couple of problems and we will see that doing the context level diagram for any system is extremely straightforward. We just go through the problem description and find out who are the users and what data they input, what data they consume. And then we draw one circle representing the system and write the different types of users here using rectangles and then write the data the input and the data they produce. We saw a simplest system (Tic-tac-toe Software), where only one type of user that is the human player.

But we will do more complex systems where there are many types of users for the software.

(Refer Slide Time: 23:20)

Context Diagram

- A context diagram shows:
 - External entities.
 - Data input to the system by the external entities,
 - Output data generated by the system.
- The context diagram is also called the **level 0 DFD**.

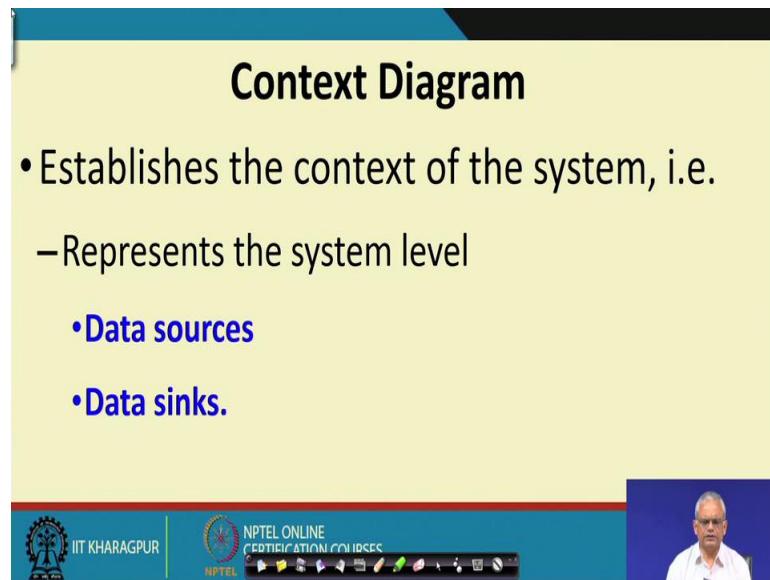


So, here we represent the external entities and the main system is drawn using one process symbol, we also represent the data input by the external entities and the data generated by the system and this we will call as the level 0 DFD.

(Refer Slide Time: 23:47)

Context Diagram

- Establishes the context of the system, i.e.
 - Represents the system level
 - Data sources
 - Data sinks.



The slide has a yellow background with black text. At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video player window showing a man in a white shirt.

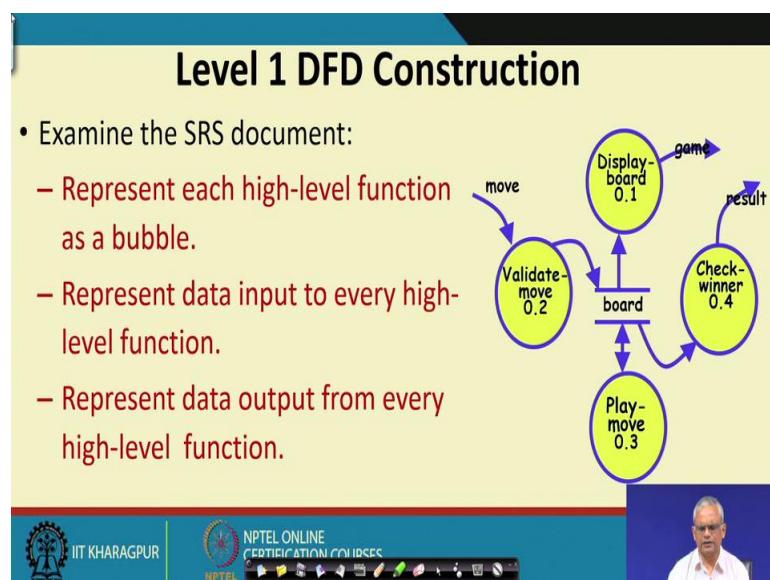
The level 0 DFD establishes the context of the system that is in the context of which users it exists and what the user does with the system, what type of data they enter and what type of response they get.

So, the context in which software exists is the context of the data sources and the data sinks which are basically the users of the system.

(Refer Slide Time: 24:24)

Level 1 DFD Construction

- Examine the SRS document:
 - Represent each high-level function as a bubble.
 - Represent data input to every high-level function.
 - Represent data output from every high-level function.



The slide has a yellow background with red and black text. At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video player window showing a man in a white shirt.

So, we saw drawing the context level diagram at the level 0 is extremely simple. Whatever be the degree of sophistication of a problem I think it should not take much

time to draw the context diagram which is the simplest representation. Context diagram just involves drawing one circle, writing the name of the system on that and then identifying who are the users representing them in rectangles and then representing what type of data the input to the system and what type of response they get from the system.

Once we do the level 0 DFD we can start to develop the level 1 DFD (On the above slide). Again, drawing the level 1 DFD is very straight forward. If we are starting to do it from a SRS document we just take up the SRS document and observe what are the high level functions there and we represent each high level function of the SRS document by a circle symbol and then we identify what data they exchanged among each other and that forms our level 1 DFD.

As an example, in the tic-tac-toe problem, if the SRS document said that there are four requirements: display board, validate move, play move and check winner. For each requirement we just draw one circle (a circle represent a function or a requirement) and then see what data each function need to consume and what data it produces and we see that what is the data that is exchanged among all these functions and then represent that in the level 1 diagram. So, we can see it is extremely simple.

But then one thing we must remember that if the SRS document contains a large number of functional requirements, let say fifty requirements for that we cannot draw fifty circles that will become very complex level 1 diagram. We need to group those into a set of higher-level functions. In a good SRS document, if there are many functional requirements these are made into subsections, related functions are made into one subsection another related functions into another subsection. So, that gives us a hint that those subsections actually become the high-level functions and typically in the level 1 DFD we should have something three to five maximum seven bubbles means maximum seven functions. And in the other extreme, if the SRS document is for a very simple system which has just one high level function then we cannot just take one function and represent in the level 1 diagram. We split that, we go through the function and find out what can be the sub functions of that and we represent them in the level 1 diagram.

(Refer Slide Time: 28:21)

Higher Level DFDs

- Each high-level function is separately decomposed into subfunctions:
 - Identify the subfunctions of the function
 - Identify the data input to each subfunction
 - Identify the data output from each subfunction
- These are represented as DFDs.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

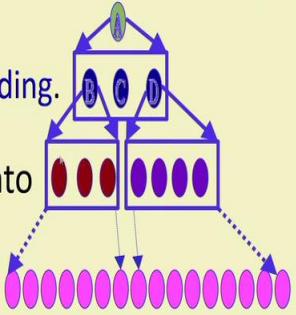
So, for level 1 DFD, we had taken the SRS document, identify the functions that are documented there and represent them as circles. Once we done level 1 DFD, if we want to do a level 2 or level 3 diagram for that we look at the one function in the level 1 diagram and then we identify the sub functions of that and then we identify the data input and the data output and represent it in a different DFD diagram.

obviously, to able to identify the sub functions of a function, we have to go through the SRS document. From the SRS document, if we clearly understand that what the function involves then will be normally be able to identify the sub functions. We will do that with the help of few examples and you will see that once you do few of that it's no big deal, you can easily identify the sub functions of a function. And in that manner, we keep on building the hierarchy and we go on decomposing until we find the functions have become very simple.

(Refer Slide Time: 30:02)

Decomposition

- Decomposition of a bubble:
 - Also called **factoring** or **exploding**.
- Each bubble is decomposed into
 - Between 3 to 7 bubbles.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

When we decompose a function, we call it as factoring or exploding a bubble (On the above slide). Each bubble is decomposed to three to seven bubbles. At this context level, we have only one function, we have factored that into about three to five functions in the level 1 diagram and then we have taken each of the function here on the level 1 diagram and we have factor that.

So, at any time we take in a diagram one of the bubbles and we develop a DFD diagram for that. So, the DFD model if you can imagine it may contain many DFD diagrams and each diagram will be linked by its previous level diagram.

So, the level 1 diagram is an elaboration of the level 0 diagram. Similarly, the set of the level 2 diagrams are an elaboration of the level 1 diagram and so on.

(Refer Slide Time: 31:21)

The slide has a yellow background with a dark blue header bar at the top. The title 'Decomposition' is centered in the header. Below the title, there is a bulleted list:

- Too few bubbles make decomposition superfluous:
 - If a bubble is decomposed to just one or two bubbles:
 - Then this decomposition is redundant.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and a video camera icon. To the right of the footer bar is a small video window showing a man speaking.

We must take care about well decomposition. What mean by a well decomposition? For example, if we take one bubble in a level, we should not decompose it into just one bubble in the next level, it should be between three to five bubbles, it should not be just one or two, otherwise the decomposition loses its purpose.

So far in this lecture we looked at the DFD symbols. Saw that there are five very simple symbols and we just identify simple rules by which they are connected. We saw dataflow arrows which connect various process or entities. We saw, when a data flow arrow connects a process to another process is called a synchronous flow and the arrow if it is connected via a data store then it represents an asynchronous operation. And we discussed also how to develop the context diagram and the level 1 diagram and we also discussed about how to do the decomposition for any level DFD, we just take one bubble at a time and then we decompose it anything between three to five sub functions and represent them for the next level DFD and so on.

We will stop here and continue from this point.

Thank you

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

Lecture – 25
Developing DFD Model

Welcome to this lecture. In the last lecture, we were discussing how to develop the DFD model of any problem. We saw that developing the context diagram is extremely simple. We just draw a circle, write the name of the system in it, identify the external users or the external entities and see what type of data they produce and what type of data they consume. Then for the level 1 diagram we just take up the SRS document and identify the functional requirements. If there is anything between three to five functions then we directly represent them on the level 1 DFD.

But if there are large number of functions in the SRS document, then we need to group some of the functions in the higher-level functions. And normally in a good SRS document that is already done that the functional requirement consists of subsections and each subsection contains several functionalities. So, we can represent those subsections in the level 1 diagram and then we identify the data flows that occur among these functions.

And then we do the decomposition to develop higher-level diagrams that is more detailed diagrams. And for decomposition we need to take one bubble at a time and then we identify are the sub functions and we need to decompose between three to five sub functions in the next level diagram. And we were discussing that if there are two or few bubbles in the next level then the decomposition does not achieve anything.

(Refer Slide Time: 02:38)

Decomposition

- Too few bubbles make decomposition superfluous:
 - If a bubble is decomposed to just one or two bubbles:
 - Then this decomposition is redundant.



Let say, if there was one bubble in some DFD and in the next DFD level again we give one bubble then there is no decomposition. And even if we decompose one bubble into two bubble then that's also not very meaningful. Typically, we need to decompose each bubble into three, five or up to seven bubbles. One thing to remember that we not be decomposed into too many bubbles because that will make difficult to understand and the hierarchical nature will be lost.

(Refer Slide Time: 03:32)

Decomposition Pitfall

- Too many bubbles at a level, a sign of poor modelling:
 - **More than 7 bubbles at any level of a DFD.**
 - **Make the DFD model hard to understand.**



So, at any level or any DFD diagram at any level we should have about seven maximum bubbles and that will be a good DFD diagram and it will be easy to understand. Otherwise if there are too many bubbles then the diagram would become difficult for somebody to understand.

(Refer Slide Time: 04:01)

Decompose How Long?

- Decomposition of a bubble should be carried on until:
 - A level at which the function of the bubble can be described using a simple algorithm.

So, in like that way we keep on decomposing. We take up one bubble in a diagram, decompose and then again take up those new decomposed bubbles and again decompose it in the next level and so on.

But to what extent we go on decomposing? We stop our decomposition when we find that it is decomposed to a function, which can be described using a very simple algorithm and its coding should not take more than few statements.

(Refer Slide Time: 04:44)

Example 1: RMS Calculating Software

- Consider a software called RMS calculating software:
 - Reads three integers in the range of -1000 and +1000
 - Finds out the root mean square (rms) of the three input numbers
 - Displays the result.

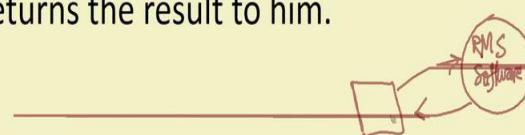


Now, let's do one simple example problem and then we will do more sophisticated problems. One simple example we will do is the RMS calculating software. It is a rather trivial software which just reads three numbers between -1000 and +1000, computes their root mean square or RMS and then displays the result. So, it is very simple.

(Refer Slide Time: 05:30)

Example 1: RMS Calculating Software

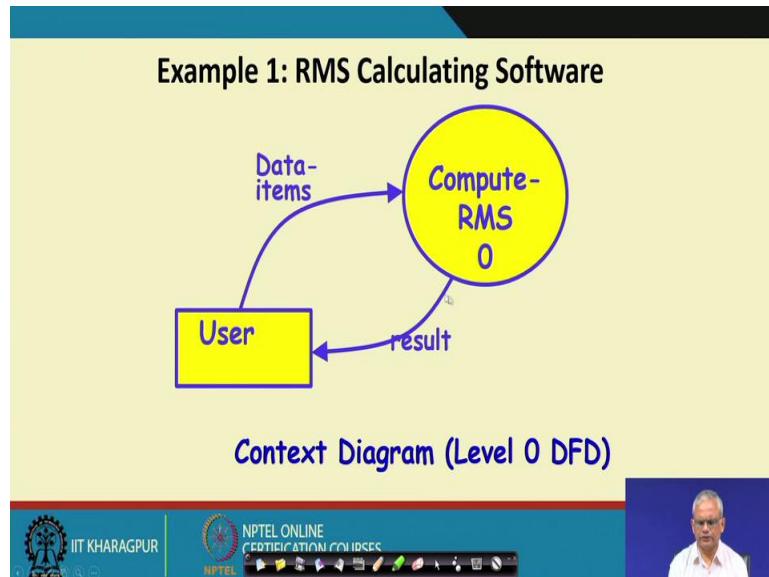
- The context diagram is simple to develop:
 - The system accepts 3 integers from the user
 - Returns the result to him.



Now, let's try to do the context diagram. In the context diagram as you said we need to just draw a circle and represent the system and we write the name of the system 'RMS software'. Only at the context level a noun form of the process is allowed, in the other

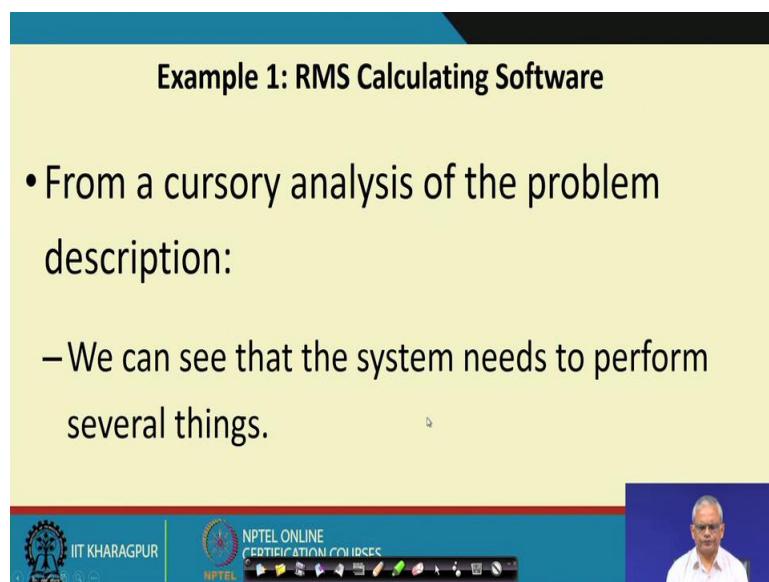
levels we typically give a verb form to the process. And then it accepts 3 integers and produces the result and we have the user for giving input.

(Refer Slide Time: 06:29)



This example shown on the above slide. The name of the software is ‘compute-RMS’ and then there is this user who enters data items and the data items are three integers and it produces a result. So, this is our context diagram or the level 0 DFD as shown on the above slide and as we can see, it is extremely simple to draw.

(Refer Slide Time: 07:00).



So, the example we discussed is just consists of one function. Any decomposition of this into level 1, level 2 et cetera will become highly artificial because this is a very simple problem.

(Refer Slide Time: 07:18)

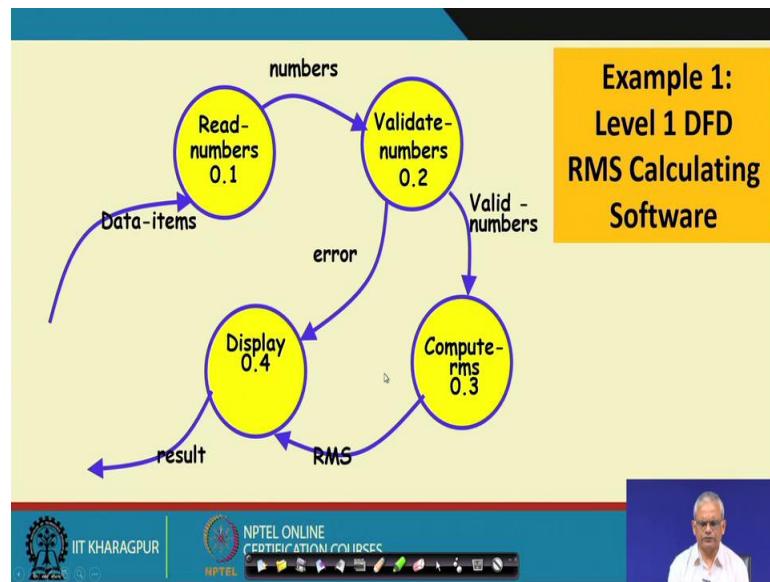
The slide has a yellow background with a black header bar. The title 'Example 1: RMS Calculating Software' is centered in the header. Below the title is a bulleted list of four items:

- Accept input numbers from the user:
- Validate the numbers,
- Calculate the root mean square of the input numbers
- Display the result.

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. Next to it is the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. To the right of the footer is a small video window showing a man speaking.

But then if we really want to decompose, we will see that for performing the RMS calculation, it first needs to read the numbers from the user then do some validation to check whether they are within -1000 to +1000 and then perform the computation root mean square and then display the result. So, these are the sub functions of the level 0 context diagram because there is only one functionality in level 0, we do not need represent those sub functions. So, in the level 1 we decompose the function represented in the level 0 and then we can represent it in the form given in below slide .

(Refer Slide Time: 08:00)



Just look at the level 1 diagram, here we don't represent the external entities. The external entities are represented only in the context diagram or the level 0 diagram. The data items are entered into the 'Read-numbers', then it reads the numbers and inputs the number to 'Validate numbers'. 'Validate numbers' validates the input and then gives the valid numbers to the 'Compute-rms'. If the inputs are not valid then 'Validate numbers' produce an error message. The error message is given to the 'Display' bubble. 'Compute-rms' takes the input and compute the RMS and then pass the result to 'Display'. So, the 'Display' process either display an error message or display the RMS result.

So, we have seen the level 1 diagram of RMS calculating, but as I was saying that this is a highly simple problem and therefore, a level 1 diagram becomes a very superfluous and very artificial. Nobody would really do a decomposition of a compute RMS because it is a very simple function by itself. And normally we would not decompose in real problem DFD modeling, but just to give an illustration how decomposition can be done, explained that with the simplest example.

(Refer Slide Time: 09:52)

Example: RMS Calculating Software

- Decomposition is never carried on up to basic instruction level:
 - A bubble is not decomposed any further:
 - If it can be represented by a simple set of instructions.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

A video thumbnail of a professor speaking is visible in the bottom right corner.

So, I just mentioned here that we don't really decompose to this level. Compute-RMS itself is a very simple function and that can be represent and that can be coded using only very few instructions.

(Refer Slide Time: 10:11)

Data Dictionary

- A DFD is always accompanied by a data dictionary.
- A data dictionary lists all data items appearing in a DFD:
 - Definition of all composite data items in terms of their component data items.
 - All data names along with the purpose of the data items.
- For example, a data dictionary entry may be:
 - **grossPay = regularPay+overtimePay**

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

A video thumbnail of a professor speaking is visible in the bottom right corner.

Whenever, we do a DFD model a data dictionary must accompany it. The data dictionary lists all the data items that appear in the DFD and then write the meaning of that data item. So, remeber that whenever we have a DFD model we must have a data dictionary accompanying that.

(Refer Slide Time: 11:38)

Importance of Data Dictionary

- Provides the team of developers with standard terminology for all data:
 - A consistent vocabulary for data is very important
- In the absence of a data dictionary, different developers tend to use different terms to refer to the same data,
 - Causes unnecessary confusion.

The data dictionary gives the meaning of all the data items that appear on the DFD. As we know that a DFD model can consist of large number of DFDs and all the data names appearing in all these various DFDs are listed in the data dictionary and their meaning is given in the data dictionary. If it is a composite data item, it's expressed in terms of the simpler data items, but for the simplest data item we just write its purpose. So, that by looking at the data dictionary we should be clear about what are the meaning of the data that appear on various DFDs.

The data dictionary is a very valuable document that is produced during structured analysis. The team members refer to the various data using the name and the decomposition given in this the data dictionary. If we don't have a data dictionary, there is a chance of misuse. The team members may misunderstand the data that is represented on the DFD and also while they refer to different data items, they may use inconsistent names and that will make it very confusing.

So, the data dictionary standardizes the names of the data and explains their purpose and what the data consist of. The complex data items may consist of some simpler data and it will be mentioned in data dictionary.

(Refer Slide Time: 12:41)

Importance of Data Dictionary

- Data dictionary provides the definition of different data:
 - In terms of their component elements.
- For large systems,
 - The data dictionary grows rapidly in size and complexity.
 - Typical projects can have thousands of data dictionary entries.
 - It is extremely difficult to maintain such a dictionary manually.



For very simple systems, the data dictionary is quite manageable. If you have ten or twenty data items then you can just write down on a piece of paper or something and that will serve the purpose. But then in real projects the data dictionary can have tens of thousands of entries and if you start to manually write this then it becomes very difficult. You may do mistakes or there can be redundant entries and when you want to search the name of some data item becomes very difficult to search it manually. And therefore, all DFD tools or structured analysis tools capture the data that appears in the DFD and build the data dictionary and maintain it in a relational database. As there is a relational database for data dictionary for that query et cetera becomes easy.

(Refer Slide Time: 13:55)

Data Dictionary

- CASE (Computer Aided Software Engineering) tools come handy:
 - CASE tools capture the data items appearing in a DFD automatically to generate the data dictionary.

For generating data dictionary, CASE tools come handy. CASE tools captures the data automatically from a DFD and they help to automatically search the data items.

(Refer Slide Time: 14:12)

Data Dictionary

- CASE tools support queries:
 - About definition and usage of data items.
- For example, queries may be made to find:
 - Which data item affects which processes,
 - A process affects which data items,
 - The definition and usage of specific data items, etc.
- Query handling is facilitated:
 - If data dictionary is stored in a relational database management system (RDBMS).

(Refer Slide Time: 14:22)

• Composite data are defined in terms of primitive data items using simple operators:

- **+**: denotes composition of data items, e.g
 - **a+b represents data a together with b.**
- **[,,]:** represents selection,
 - Any one of the data items listed inside the square bracket can occur.
 - For example, **[a,b] represents either a occurs or b**

$C = \underline{\underline{ab}}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

The CASE tools we will see that they can also answer query like a which data item is affecting which processes. The CASE tools also maintain query like where a process affect which data items, which specific data get assigned values and where and so on. It becomes easy to answer this kind of question when these are maintained by an automated tool using a relational database system.

Now, let see the grammar by which these primitive items are combined into more complex data items. The rules and the symbols here very simple. If we write ‘a+b’, then we mean that ‘a’ together with ‘b’ and if we write ‘c = a+b’ then that mean ‘c’ consists of ‘a’ and ‘b.’ If we use square bracket ([]) it means option, for example, if we write [a,b] it means that either ‘a’ or ‘b’.

(Refer Slide Time: 15:59)

Data Definition

- **()**: contents inside the bracket represent optional data
 - which may or may not appear.
 - **a+(b)** represents either a or a+b
- **{}**: represents iterative data definition,
 - **{name}5** represents five name data.

If we write ‘a+(b)’, then it represents that either ‘a’ or ‘a+b’. If we use curly bracket ({{}}) then it represents the repetition. If we write {name}5 that means, five name data items.

(Refer Slide Time: 16:26)

Data Definition

- **{name}* represents**
 - zero or more instances of name data.
- **=** represents equivalence,
 - e.g. **a=b+c** means that a represents b and c.
- *** *:** Anything appearing within * * is considered as comment.

If we write, {name}* it represents zero or many name data items. '=' represents equivalence. So, 'a=b+c' means 'a' consists of 'b' and 'c'. So, 'a' is a composite data item containing 'b' and 'c'. As you can see that the data definition is very simple using few operators like +, [], {} and so on. And we can also write comments here within * *, for example, *anything within this will be treated as comment*.

(Refer Slide Time: 17:09)

Data Dictionary for RMS Software

- numbers=valid-numbers=a+b+c
- a:integer * input number *
- b:integer * input number *
- c:integer * input number *
- asq:integer
- bsq:integer
- csq:integer
- squared-sum: integer
- Result=[RMS,error]
- RMS: integer * root mean square value*
- error:string * error message*

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Above slide is an example of data dictionary for RMS software. This is just one example here, ‘a’ is a primitive data and ‘a’ is written as an integer. In our example, ‘numbers’ is a composite data and ‘valid numbers’ is a synonym for ‘numbers’ and it consists of three components ‘a’, ‘b’, and ‘c’. In our example, ‘Result’ is a data which sometimes can be RMS and other times it can be error and RMS and error are primitive data item. Here, ‘error’ is used to represent an error message, ‘RMS’ is used to represent the root mean square value. We have earlier mentioned about comment. In our example, *root mean square value* and *error message* are the example of comments.

(Refer Slide Time: 17:59)

Balancing a DFD

- Data flowing into or out of a bubble:
 - Must match the data flows at the next level of DFD.
- In the level 1 of the DFD,
 - Data item c flows into the bubble P3 and the data item d and e flow out.
- In the next level, bubble P3 is decomposed.
 - The decomposition is balanced as data item c flows into the level 2 diagram and d and e flow out.

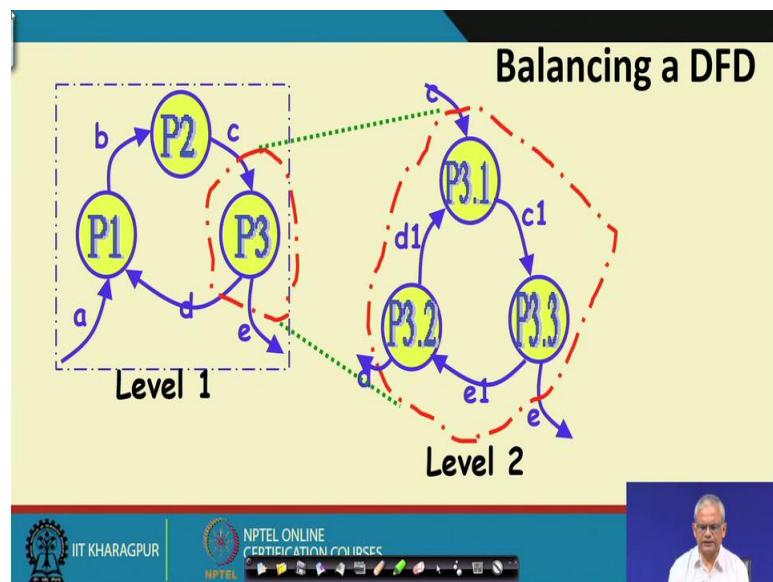
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now, if we practice few more examples, we will see that given a new example we can easily draw the DFD model. But before we do that let's look at another important concept in the DFD modeling called as balancing a DFD. Here we check, if we decompose a DFD into a lower level DFD then the data that is input to the DFD and the data that is output of the DFD are matches in the next level.

What we mean here is that let's say we have a DFD model (example on upper slide) and now, let's say we take one bubble and we decompose it into a DFD diagram. So, we again let's say decompose the same bubble into three bubbles. Both the decomposition takes some data and produces some data. Now, let's say our first decomposition takes 'a' and produces 'b'. Similarly, in the next level, we must also have taken 'a' as input and this let's say produces 'c' and 'd', then it must be ' $b = c + d$ '.

This is balanced, because the first bubble here takes input 'a' and produces output 'b' and when we have decomposed it into the next level, the data that is flowing into the next level diagram is 'b' and 'c' and 'd' is coming out as 'output' from the diagram and ' $b = c + d$ '. So, for balancing we see that the input is balanced and for the output to be balanced if ' $c + d = b$ '. This rule of balancing is used in all decomposition levels.

(Refer Slide Time: 20:42)



Above slide contains another example of balancing. Let's say we have a level 1 DFD representation. In that level 1 DFD, P1, P2, P3 are the three processes and they are exchanging some data. And 'a' is consumed by P1, P2 consumes 'b' and P3 consumes

'c' and P3 produces 'd' and 'e'. Now, let say in the next level we were decomposing this P3 bubble. So, in the next level diagram P3 again decompose into three bubbles. In this next level, there are some data that are internally input across this new level bubbles.

But just see here that to this bubble P3.1, 'c' is input and bubble P3.2 gives output 'd' and P3.3 gives output 'e'. Now, we check the previous input, output with the bubble P3 of level 1 diagram and see it is a balanced decomposition because P3 is also taking input 'c' and producing 'd' and 'e' as output

(Refer Slide Time: 22:17)

• Number the bubbles in a DFD:

- Numbers help in uniquely identifying any bubble from its bubble number.

• The bubble at context level:

- Assigned number 0.

• Bubbles at level 1:

- Numbered 0.1, 0.2, 0.3, etc

• When a bubble numbered x is decomposed,

- Its children bubble are numbered x.1, x.2, x.3, etc.

**Numbering
of Bubbles**

0.1.2

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another concept is the numbering of DFD. So, far we saw that we just write the name of the process in the bubble, but if we write a number that will help us in identifying the bubble. For example, if we write 0 is for the context level, can easily find out which bubble it is.

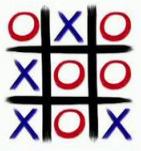
In the next level, if we in the level 1 we write 0.1, 0.2 then by just looking at the number we would know which diagram and that's the reason why the bubbles are normally numbered. Both by the tools and even if we are doing it manually, the context level is assigned the number 0 that's the convention and at level 1 we decompose into a set of bubbles. Let say we decompose into three bubbles, then we number them 0.1, 0.2, 0.3.

And similarly, if we let say find a bubble whose number is 0 .2 .1 and then we know that the level 1 diagram the second bubble 0.2, this is the parent of this bubble 0.2.1 and for 0.2 the parent is the 0. So, we can understand that this 0.2.1 is a level 2 diagram.

(Refer Slide Time: 24:44)

Example 2: Tic-Tac-Toe Computer Game

- A human player and the computer make alternate moves on a 3 X 3 square.
- A move consists of marking a previously unmarked square.
- The user inputs a number between 1 and 9 to mark a square
- Whoever is first to place three consecutive marks along a straight line (i.e., along a row, column, or diagonal) on the square wins.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

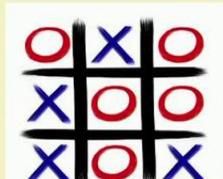
Now, let's do one more problem which is the Tic-Tac-Toe game. Let's first understand the Tic-Tac-Toe game. We all are familiar with this kind of game. Here it is played between the computer and the user. Let's say the player marks red zero and the computer makes blue cross (Shown on above slide). So, the player mark a square and then the computer will mark something, in response to that the user will mark another square and so on.

The player wins if he can occupy three consecutive elements in the row or column or on the diagonal. So, before the player can mark all along this diagonal the computer will try in a way so that the player cannot win straight away. And if the game all the squares are over and no one has got any mark along a row or a column or a diagonal then the game is considered draw.

(Refer Slide Time: 26:30)

Example: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
 - A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
 - And all the squares on the board are filled up,
 - Then the game is drawn.
- The computer always tries to win a game.



So, there are nine places to fill and each player that is the human player or the computer they place their marks in those nine places.

Now, can we draw the context level diagram of this? The game is intuitively clear and we writing the software for this and we want to draw a context level diagram for this problem. Please try this on your own and we will discuss the solution in the next class. And as I was mentioning that if we can do few problems, we can develop DFD model for any given problem, even a very sophisticated problem we can easily develop its DFD model.

We will stop now. Will continue from this point in the next class.

Thank you.

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

Lecture – 26
Examples of DFD Model development

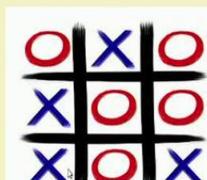
Welcome to this lecture.

We had discussed about the DFD model. We had discussed about the symbols that are used. We also saw how to draw the different levels of the DFD diagram and we had also looked at some very simple problems. In the last lecture, towards the end we are trying to develop the DFD model for the tic-tac-toe problem. The tic-tac-toe is a simple computer game.

(Refer Slide Time: 00:51)

Example: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
 - A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
 - And all the squares on the board are filled up,
 - Then the game is drawn.
- The computer always tries to win a game.



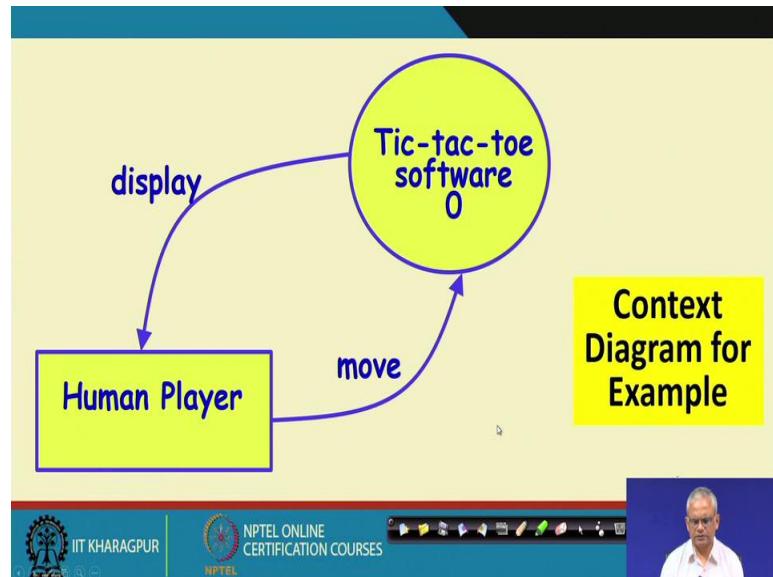
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

In a tic-tac-toe game, a three by three square is used where the computer and the human player take turns to mark one square and whoever gets three consecutive marks, on either a diagonal or on a row or on a column, wins the game and if all the squares get filled up without anybody getting three consecutive symbols in the game is considered as drawn.

Now, we want to develop a software for this we want to develop the structured analysis document and the DFD model. The first thing to do is the context diagram. In the context diagram, we draw one circle, write the name of the software there and then identify who

are the users. There is only one user and that is the player. The player enters move and receives the response.

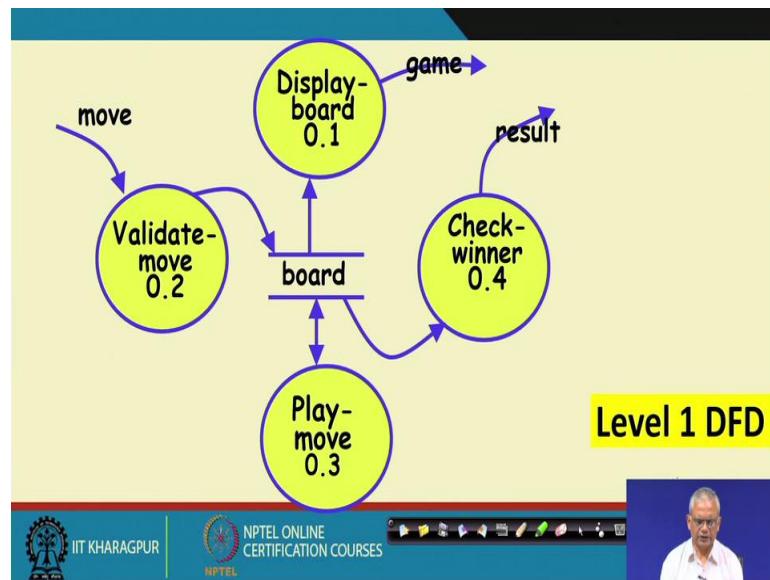
(Refer Slide Time: 02:16)



In above slide, context diagram of tic-tac-toe software has shown. The diagram is very simple here, that write the name of the software Tic-tac-toe software and the number is 0. The external entity is human player who enter moves and receives the display. Now, how do we develop the level 1 diagram? In the level 1 diagram, we identify the high-level functionalities and represent them as bubbles, typically between three to five bubbles.

If we look at the problem description one is about reading the move and validating that. The other is about playing the computer move. The computer move need to see the board position and then need to identify what would be the move for the computer and then display those.

(Refer Slide Time: 03:45)



So, we can model that by adding bubbles. One bubble is Validate-move. When a move comes validate move then the move is marked on the board and then display board and then the computer needs to play move and the check winner. So, once the human player makes the move, the move is validated that is updated on the board. And then the Check-winner bubble must be there just to check whether the human-player won or not. If human player won then the result is displayed otherwise, the computer makes the move. Again, the winner is checked and the display board is made.

One thing to remember that this is the data flow model and therefore, we don't represent control aspects like which bubble will operate after which one. These control aspects are not really important to our design. Here, we are just identifying what are the sub functions of the handle move and then we just represent them here.

(Refer Slide Time: 05:41)

Data Dictionary

Display=game + result
move = integer
board = {integer}9
game = {integer}9
result=string



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Now, we develop the data dictionary. The display variable of the data dictionary is ‘game + result’. In the level 0 diagram, if you look at the display, there is one data that is produced by the system whereas, in the level 1 diagram there are two data that are produced ‘game + result’. Therefore, to balance must right ‘display = game + result’. Here, move is an integer, the board is nine integers and the game is also nine integers and the result is a string.

(Refer Slide Time: 06:36)

Example 3: Trading-House Automation System (TAS)

- A large trading house wants us to develop a software:
 - To automate book keeping activities associated with its business.
- It has many regular customers:
 - They place orders for various kinds of commodities.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Even the tic-tac-toe problem was a very simple problem. Now, let's do a slightly more sophisticated problem which is the Trading-House Automation System.

In this there is a trading-house which wants us to develop a software to automate. It is a bookkeeping activity. The trading house has many regular customers who place orders or various kind of commodities.

(Refer Slide Time: 07:11)

Example 3: Trading-House Automation System (TAS)

- The trading house maintains names and addresses of its regular customers.
- Each customer is assigned a unique customer identification number (CIN).
- As per current practice when a customer places order:
 - The accounts department first checks the credit-worthiness of the customer.



Now, the trading-house has a set of registered customers called them as regular customers and each customer has an identification number. And when one of the registered customers places order then the account department checks the creditworthiness of the customer.

(Refer Slide Time: 07:40)

Example: Trading-House Automation System (TAS)

- The credit worthiness of a customer is determined:
 - By analyzing the history of his payments to the bills sent to him in the past.
- If a customer is not credit-worthy:
 - His orders are not processed any further
 - An appropriate order rejection message is generated for the customer.



The creditworthiness is checked by finding out the payment history of the bills sent in the past. If the account department says that the customer is not credit worthy then, his orders are not processed further and the rejection message is issued that is like ‘Sorry we are not be able to handle your request’.

(Refer Slide Time: 08:07)

Example: Trading-House Automation System (TAS)

- If a customer is credit-worthy:
 - Items he/she has ordered are checked against the list of items the trading house deals with.
- **The items that the trading house does not deal with:**
 - Are not processed any further
 - An appropriate message for the customer for these items is generated.



But if the account department says that the customer is credit worthy then the orders are checked against the list of items that the trading-house deals with. For those items, which

the trading-house does not deal with, just does not process them further and just sends a message saying that ‘Sorry, we do not deal with those items’.

(Refer Slide Time: 08:32)

Example: Trading-House Automation System (TAS)

- The items in a customer's order that the trading house deals with:
 - Are checked for availability in inventory.
- If the items are available in the inventory in desired quantities:
 - A bill with the forwarding address of the customer is printed.
 - A material issue slip is printed.



And for those items that the trading-house deals with, it checks its inventory. If the items are available in the desired quantity then a bill is printed and sent to the customer. And also, a material issue slip is printed, the customer can present the material issue slip and take possession of the items.

(Refer Slide Time: 09:04)

Example: Trading-House Automation System (TAS)

- The customer can produce the material issue slip at the store house:
 - Take delivery of the items.
 - Inventory data adjusted to reflect the sale to the customer.



And once the bill has been raised for a certain item then the inventory data is automatically adjusted.

(Refer Slide Time: 09:21)

Example: Trading-House Automation System (TAS)

- If an ordered item is not available in the inventory in sufficient quantity:
 - To be able to fulfil pending orders store details in a "pending-order" file :
 - out-of-stock items along with quantity ordered.
 - customer identification number



And if some item is not available in the inventory then there is a pending order file is created for those non-available items. And the number of items a customer ordered are stored there along with a customer identification number who has raised that request.

(Refer Slide Time: 09:43)

Example: Trading-House Automation System (TAS)

- The purchase department:
 - would periodically issue commands to generate indents.
- When **generate indents** command is issued:
 - The system should examine the "pending-order" file
 - Determine the orders that are pending
 - Total quantity required for each of the items.



The purchase department periodically issues commands to generate the indent. And here for those items which are already demanded by the customer but which are out of stock,

are taken into account. And the required quantity for each item is placed against the vendors who supply those items.

(Refer Slide Time: 10:23)

Example: Trading-House Automation System (TAS)

- TAS should find out the addresses of the vendors who supply the required items:
 - Examine the file containing vendor details (their address, items they supply etc.)
 - Print out indents to those vendors.



The vendor details are maintained in a file, like their contact address, the items they supply etc. are maintained in a file. And for a specific item the indents are raised against the specific vendor who supplies that item.

(Refer Slide Time: 10:45)

Example: Trading-House Automation System (TAS)

- TAS should also answer managerial queries:
 - Statistics of different items sold over any given period of time
 - Corresponding quantity sold and the price realized.



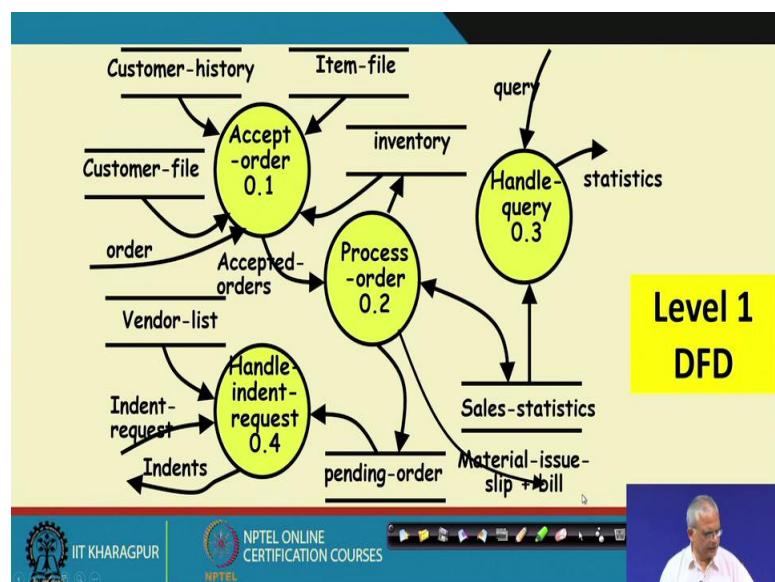
Other than handling requests from the customer it also answers managerial queries like, What is sales statistics? What is the volume of items sold? What is the price realized?

And so on. So, if we want to do a structured analysis of this problem, we need to first do the context diagram. We already know that a context diagram consists of drawing one circle with the name of the software in that and identifying the context in which the system exists that is the different users who would use the software and what data they would input and what data they would receive.

Here if we read the problem, we will find that for this system the name of the software is trading-house software. Just write that in a bubble ‘trading-house software’ (Shown on above slide) and then we identify all the users who will interact with the software. The customer is one of the user and the manager is another user and also we have the account department as an user and we have the purchase department who raised indents they are also another user.

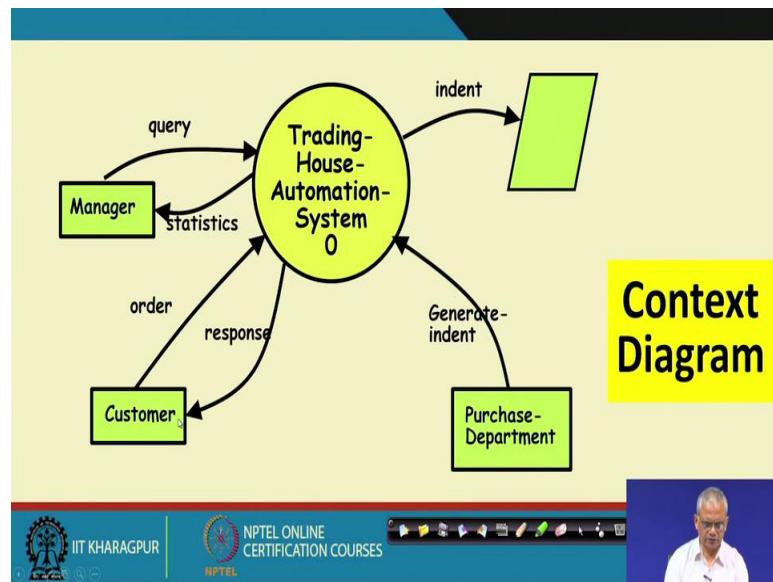
The customer places orders and receives the material issue slip. The account department looks at the order and checks the creditworthiness. The purchase department looks at the pending orders and issues the indents. The manager requests for statistics requests and gets the statistics response. All these shown in level 0 diagram which is drawn on above slide.

(Refer Slide Time: 13:47)



Now, we will see level 1 diagram. Drawing the level 1 diagram is rather straightforward. Level 1 diagram shown on above slide. Before that, we will see the level 0 diagram or context diagram in details.

(Refer Slide Time: 13:49)



The customer, the manager shown on the above context diagram. Account department not shown separately actually the accounts department role is done by the software. So, the account department is not shown as a different entity here, automatically that is done in the software by looking at the credit worthiness and therefore, no specific extra input required by the account department.

The creditworthiness of the customers is checked within the software. So, the customer places orders and receives response which is the material issue slip. Customer also got some message for those items which not dealt with by the trading-house system. The manager can raise statistical queries and get statistics. The indents are placed on the vendors and the purchase department generate indent.

Now, the level 1 DFD if we read through the problem description and identify the functional requirements, we will see that the major functional requirements are the high-level functions. One high-level function is Accept-order. So, the order is placed by the customer and then Accept-order checks whether, it's a valid customer by checking at the customer file. And also checks the customers payment history and automatically decides whether the customer is credit worthy or not. And then checks the item file, if the item is actually dealt with by the trading-house then the accepted order is given to the Process-order.

The Process-order looks at the inventory, if they are available then it would issue a material issue slip bill and it would update the sale statistics and also update the inventory. But if the item is not available then it will record that in the pending order file and it will update the pending order file. Sometime the handle indent requests will be operated by the purchase department. They will give indent request and then against the vendor list and the pending order the indent will get generated. The managerial queries concern only the sales statistic and that is once the query comes the appropriate statistics are obtained from Handel-query and displayed. So, this is about the level 1 DFD.

(Refer Slide Time: 18:01)

Example: Data Dictionary

- response: [bill + material-issue-slip, reject-message]
- query: period /* query from manager regarding sales statistics*/
- period: [date+date,month,year,day]
- date: year + month + day
- year: integer
- month: integer
- day: integer
- order: customer-id + {items + quantity}*
- accepted-order: order /* ordered items available in inventory */
- reject-message: order + message /* rejection message */
- pending-orders: customer-id + {items+quantity}*
- customer-address: name+house#+street#+city+pin

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

We then need to do the data dictionary. Every data item that appears in either in the level 0 or in the level 1, we just write the data item and for the primitive items we just directly write what purpose they are used and for composite items we write the component items.

(Refer Slide Time: 18:31)

The slide contains a yellow callout box with the text "Example: Data Dictionary". Below it is a list of data items:

- item-name: string
- house#: string
- street#: string
- city: string
- pin: integer
- customer-id: integer
- bill: {item + quantity + price}* + total-amount + customer-address
- material-issue-slip: message + item + quantity + customer-address
- message: string
- statistics: {item + quantity + price }*
- sales-statistics: {statistics}*
- quantity: integer

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a navigation toolbar.

So, this again is the data dictionary for trading-house-automation-system which is becoming quite long.

(Refer Slide Time: 18:41)

The slide has a title "Observation" at the top. Below it is a bulleted list:

- From the discussed examples,
 - Observe that DFDs help create:
 - Data model
 - Function model

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a navigation toolbar.

So, based on the problems that we have so far modeled using the DFD, we see that DFD is a very simple model. And it helps us to achieve decomposition, we perform a detailed function model using DFD and at the same time a detailed data model is also built.

(Refer Slide Time: 19:11)

Observation

- As a DFD is refined into greater levels of detail:
 - The analyst performs an implicit functional decomposition.
 - At the same time, refinements of data takes place.

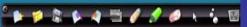


The DFDs help us to achieve functional decomposition and at the same time refinements of data takes place.

(Refer Slide Time: 19:24)

Guidelines For Constructing DFDs

- Context diagram should represent the system as a single bubble:
 - Many beginners commit the mistake of drawing more than one bubble in the context diagram.

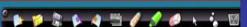


Now, we will discuss some of the mistakes to avoid while constructing a DFD. The context diagram contains only one bubble, that is the simplest representation. If we draw more than one bubble in the context diagram that's not correct.

(Refer Slide Time: 19:46)

Guidelines For Constructing DFDs

- All external entities should be represented in the context diagram:
 - External entities should not appear at any other level DFD.
- Only 3 to 7 bubbles per diagram should be allowed:
 - Each bubble should be decomposed to between 3 and 7 bubbles.



The external entities appear only in the context diagram and in level 1, level 2 et cetera, the external entity should not appear. In any DFD level only three to seven bubbles per diagram should be allowed.

(Refer Slide Time: 20:10)

Guidelines For Constructing DFDs

- A common mistake committed by many beginners:
 - Attempting to represent control information in a DFD.
 - e.g. trying to represent the order in which different functions are executed.



In DFD, only the data flow is represented, we should not represent things like which function operates before which function et cetera. We just represent here each function, what data it needs and what data it produces.

(Refer Slide Time: 20:34)

Guidelines For Constructing DFDs

- A DFD model does not represent control information:
 - When or in what order different functions (processes) are invoked
 - The conditions under which different functions are invoked are not represented.
 - For example, a function might invoke one function or another depending on some condition.
 - **Many beginners try to represent this aspect by drawing an arrow between the corresponding bubbles.**

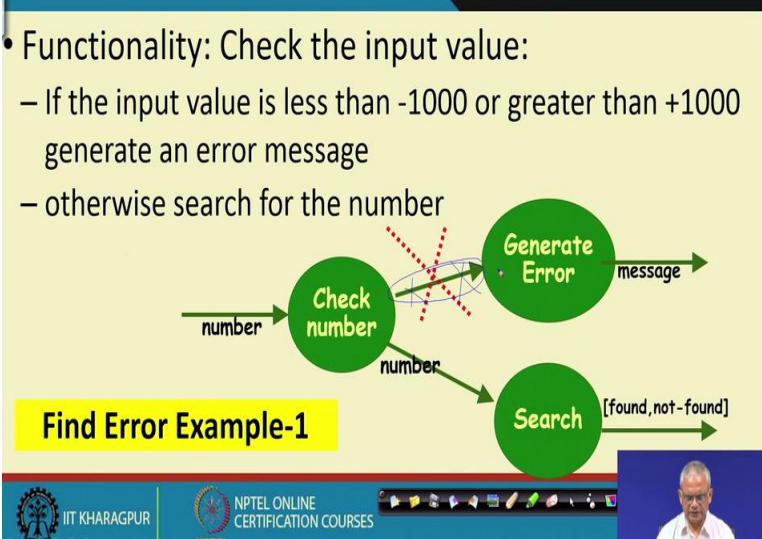


IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another common problem is use of control arrow in DFD. Control arrows indicate that which one operates after which one and so on and that's not correct.

(Refer Slide Time: 20:58)

- Functionality: Check the input value:
 - If the input value is less than -1000 or greater than +1000 generate an error message
 - otherwise search for the number

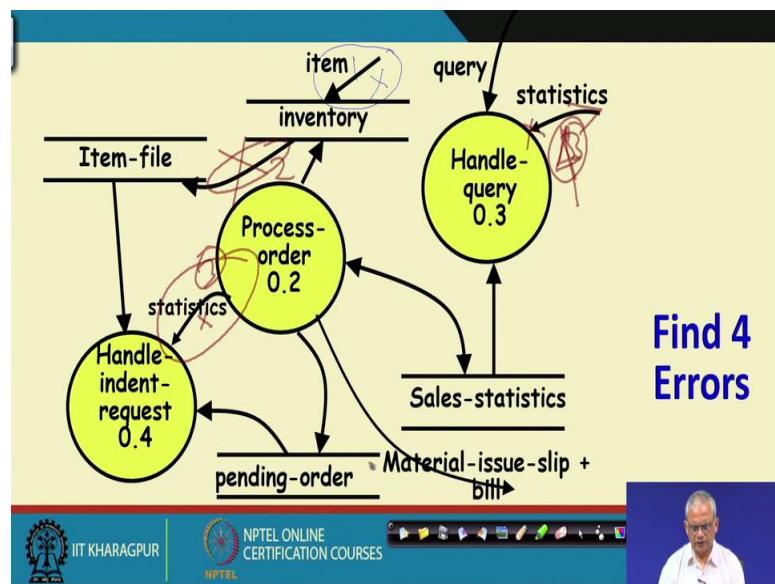


IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Just to give an example, let say we have a developer who has drawn the above slide diagram that check number. And for check number, if the number is valid that is between -1000 and +1000 then the number is searched in 'Search' bubble. So, 'Search' is another function. The number is passed on to the search and which generates one of these messages 'found' or 'not found'. Whereas, if it is not a valid number then 'Generate

Error' generates an error message. But we have drawn an arrow from 'Check number' to 'Generate Error'. But 'Generate Error' does not need any message or any data. An arrow without data mention on it actually represent control information because it does not represent data. So, here this control arrow just represents that 'Generate Error' will work after 'Check number' is completed but that is not correct. So, this arrow should not be there. This control arrow represents only control information, no data flow is indicated by this arrow and that's a mistake.

(Refer Slide Time: 22:41)



Now, we will look into the above slide diagram and will find errors in the diagram. There are four errors here. If we carefully observe the diagram, the first error that we can find here is that an item is coming in 'inventory' data store. As we know, an item can't directly come in a data store. The data store can be updated or read by a process only.

Second error here is that data flowing shown from 'inventory' data store to 'Item-file' data store. As we know, data cannot flow from one data store to another data store automatically, there has to be a process through which it flows. So, this is the second mistake. Now, are there any further errors? There are many other errors for example, the 'Process-order' error enters the statistics, but then statistics is not really required by 'Handle-indent-request'. So, this is the third error. Fourth error is 'Handle-query' should produce the statistics, not consume the statistics. So, the direction of this arrow is wrong.

Now we will see, common mistakes in constructing DFDs. The common mistakes mainly are committed by beginners.

(Refer Slide Time: 25:13)

Common Mistakes in Constructing DFDs

- If a bubble A invokes either bubble B or bubble C depending on some conditions:
 - Represent the data that flows from bubble A to bubble B and bubbles A to C
 - Not the conditions depending on which a process is invoked.

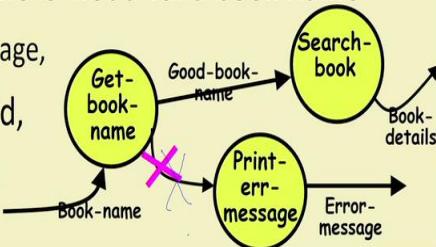


Now, we will see some example of common mistakes. For example, if conditionally something is done and represent an arrow there and that's not really required because, we should represent only data flow..

(Refer Slide Time: 25:40)

Find Error Example-2

- A function accepts the book name to be searched from the user
- If the entered book name is not a valid book name
 - Generates an error message,
- If the book name is valid,
 - Searches the book name in database.



Just to give an example, in the above slide diagram, ‘Get-book-name’ process, reads the book name and if the name is proper, it passes the book name to the ‘Search-book’ which

produces book details. But if the book name is not proper it should print an error message. So, this arrow from ‘Get-book-name’ to ‘Print-err-message’ is incorrect because print error message produces some standard error message, it does not need any input data. And therefore, this is a control arrow and this is not really required.

(Refer Slide Time: 26:47)

Guidelines For Constructing DFDs

- All functions of the system must be captured in the DFD model:
 - No function specified in the SRS document should be overlooked.
- Only those functions specified in the SRS document should be represented:
 - Do not assume extra functionality of the system not specified by the SRS document.



Now, let's proceed further. We should not miss any functions in the SRS document. If there are some functionalities that are mentioned in the SRS document, they must have been represented in the DFD. And also need to be careful that DFD should also not represent extra functionality, which is not required in the SRS document.

(Refer Slide Time: 27:10)

**Commonly
Made Errors**

- Unbalanced DFDs
- Forgetting to name the data flows
- Unrepresented functions or data
- External entities appearing at higher level DFDs
- Trying to represent control aspects
- Context diagram having more than one bubble
- A bubble decomposed into too many bubbles at next level
- Terminating decomposition too early
- Nouns used in naming bubbles

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A video player interface showing a professor speaking.

Here is a list of commonly made errors which shown on above slide. From the slide we can see the commonly made errors here are: unbalanced DFDs, forgetting to write the name of the data on a data flow. Another common error is like, suppose functions that should be there in DFD based on the SRS document are not present or data that are should be produced or consumed are not there in DFD. External entities appearing at higher level DFDs, representing control aspects, context diagram having more than one bubble all are also some common error. If a bubble decomposed into too many bubbles at the next level then it also considers as common error. The last two common error that mentioned in the above slide are: terminating the decomposition too early or too late, and using nouns to name the bubbles.

(Refer Slide Time: 28:12)

Shortcomings of the DFD Model

- DFD models suffer from several shortcomings:
- DFDs leave ample scope to be imprecise.
 - In a DFD model, we infer about the function performed by a bubble from its label.
 - A label may not capture all the functionality of a bubble.

We come at end of our lecture. We will conclude the DFD model discussion in the next lecture. And then in future lecture, we will look at how to convert the outcome of the structural analysis that is the DFDs into the structured design or the high-level design.

We will stop now.

Thank you.

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

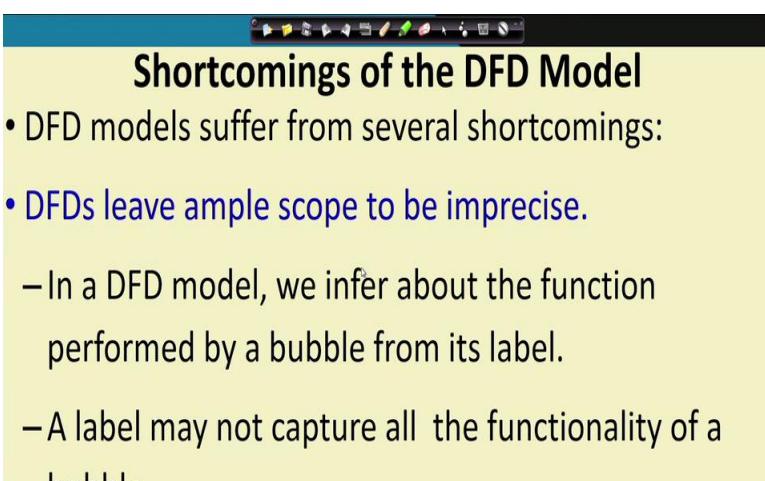
Lecture – 27
DFD Model- More Examples

Welcome to this lecture.

Over the last couple of lectures, we were discussing about the data flow diagrams. The data flow diagrams are popularly used to do the structured analysis that is capturing all the functionalities of the system and then doing a top down decomposition of the functionalities. We found that DFD is a very simple elegant mechanism and it produces a data flow model of a software. It identifies various functions at various levels and represent the interactions terms of the data exchange among those functions.

Structured analysis also popular. It's not restricted only to the design aspect, but also used for many other problems including requirements analysis in DFD's. But let's find out are there any shortcomings of the DFD is present or not and that will give us some idea about for what applications DFD's are not suitable and what needs to be done to the DFD's so that it can be applicable to those areas. Now, let's look at the shortcomings of the DFD model.

(Refer Slide Time: 02:16)



Shortcomings of the DFD Model

- DFD models suffer from several shortcomings:
- DFDs leave ample scope to be imprecise.
 - In a DFD model, we infer about the function performed by a bubble from its label.
 - A label may not capture all the functionality of a bubble.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

The first difficulty in DFD is that here the functions are only identified by the names written on the bubble and therefore, can be very imprecise, just by looking at the label we have to infer about what all is involved in the function and therefore, many things will remain ambiguous incomplete.

(Refer Slide Time: 02:50)

Shortcomings of the DFD Model

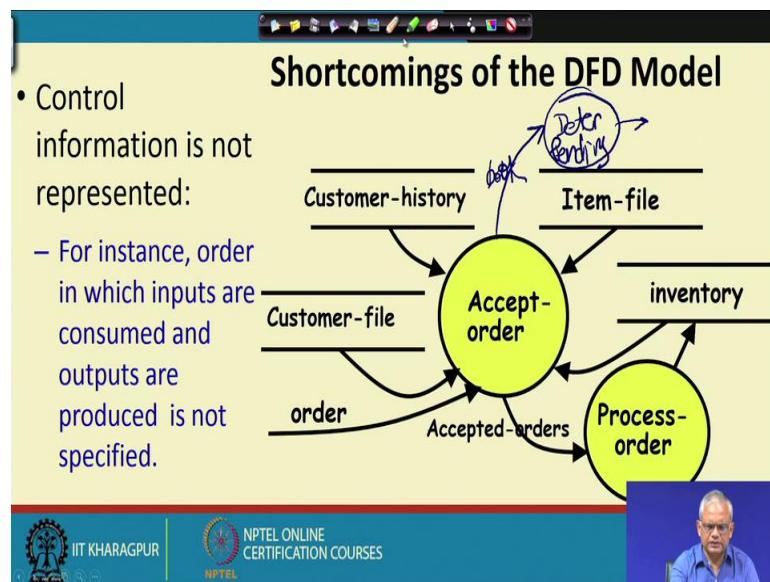
- For example, a bubble named **find-book-position** has only intuitive meaning:
 - Does not specify several things:
 - What happens when some input information is missing or is incorrect.
 - Does not convey anything regarding what happens when book is not found
 - What happens if there are books by different authors with the same book title.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** | **NPTEL**

A hand-drawn circle containing the text "find BookPosition".

Just to give an example, let say we have a bubble and the name of the bubble is 'find-book-position'. So, we have 'find-book-position' as a bubble. So, we can imagine from the bubble name that for a given book, the 'find-book-position' outputs where it is located in a library, but that is a very rough idea. For example, it does not know, how do we describe the book position, how do we give the input, do we give the name of the book or do we give the author of the book and so on. It also not tell, what would happen if the book is not there and also what would happen if for given a book name, multiple entries in the library match with the given book title, but different authors or maybe same author and multiple books does it display all of them or does it display one. So, those things are not clear and therefore, the DFD model the first glance is imprecise.

(Refer Slide Time: 04:53)



Another major problem with the DFD representation is that, control flow aspects are not represented. For example, in above slide, we have this DFD model now in what order does the accept-order take the data? Does it take first customer history, then the item file, then the customer file and then the order or first it gets the order and then the customer file and so on. This is especially important in case of real time systems where the timing issues need to be analyzed. For example, in above slide, two bubbles they execute parallelly or they executed sequentially but let say we have another bubble ‘Determine-pending’ connected with ‘Accept-order’. So, would ‘Accept-order’ execute first or would ‘Determine-pending’ execute first. So, we can understand these are the control aspects. Control aspects are not represented in DFD, we just write what data flows in DFD. So, all the data exchanges among the different entities and the bubbles are captured in DFD not the control aspect. And that’s the reason why timing analysis using a data flow diagram is very difficult. But the DFD diagram can be extended with control flow notations to indicate in what order the processing takes place and that is for doing the timing analysis. But we will not discuss about those aspects in this lecture series.

(Refer Slide Time: 07:17)

Shortcomings of the DFD Model

- Decomposition is carried out to arrive at the successive levels of a DFD is subjective.
- **The ultimate level to which decomposition is carried out is subjective:**
 - Depends on the judgement of the analyst.
- **Even for the same problem,**
 - **Several alternative DFD representations are possible:**
 - **Many times it is not possible to say which DFD representation is superior or preferable.**

There are also other problems present in DFD. One of the problems is that we achieve a functional decomposition at different levels of the DFD, but then there is no guideline about to what depth the decomposition should be carried out. We have only mentioned through a very loose statement that the decomposition should be carried out until we reach a function which is very simple. But then that's a vague statement, because it may possible that a function is simple to one person may not be the same for another person and therefore, different designers can carry out decomposition to different levels.

Another problem is that for different designers they can come up with very different DFD representation for the same problem and also many times we don't have any way to tell which DFD representation is superior and therefore, we have to explore multiple DFD representation.

(Refer Slide Time: 08:46)

Shortcomings of the DFD Model

- DFD technique does not provide:
 - Any clear guidance as to how exactly one should go about decomposing a function:
 - One has to use subjective judgement to carry out decomposition.
- Structured analysis techniques do not specify when to stop a decomposition process:
 - To what length decomposition needs to be carried out.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Previously, we say that bubble should be decomposed in the next level. Take one bubble from a layer level and then decompose it. But then we never said how to decompose it? We just said that look through the activities that are to be carried out by that process or that bubble and then identify the sub activities and those sub activities become the processes or bubbles in the next level. But then that subjective because different designers may come up with different sub functions and they will come up with different DFD representations.

(Refer Slide Time: 09:55)

DFD Tools

- Several commercial and free tools available.
- Commercial:
 - Visio
 - Smartdraw (30 day free trial)
 - Edraw
 - Creately
 - Visual analyst
- Free:
 - Dia (GNU open source)

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Beside all these drawbacks, DFD is a very useful technique very easily learnt and once you learn it, you will find that it will come to use not only in software design, but in many other areas like, in software development, in testing and so on and also in non software development areas. There are many tools that are available using which we can develop the DFD model. These tools simplifies the task of developing the model and produces good diagrams. There are several commercial tools which are popular for example, Smartdraw which includes a 30 day free trial, Edraw, Creately, visual analyst and so on and there are some free tools like Dia which is the GNU open source tool.

Please use some of the tools for better learning of DFD model. We find that these tools are extremely easy to learn and based on the concepts we discussed, we can just start using the tool. In all those tools, different DFD symbols would be available as menu items. We can just pick them and paste wherever necessary and draw the dataflow arrows between bubbles and almost every tool they develop has the data dictionary.

(Refer Slide Time: 12:12)

Word of Caution

- Tools can be learnt and used with some effort.
- **But, too much focus on SA/SD case tools does not make you any more a good designer:**
 - Than an expert knowledge of the Word Package making you a famous writer of thriller stories.

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES** | **NPTEL**

A small video frame in the bottom right corner shows a man with glasses and a blue shirt, likely the speaker from the previous slide.

But one thing to remember that, understanding the concepts are more important than using those tools. If you trying to master the tool without knowing the basic concepts it may be counterproductive. If you are hoping that just by learning the tools well you will become a good designer then your hope will be misplaced, because just to give an analogy that let say you want to become a famous writer of a thriller stories and then let say you learn the word processing package very well that will not make you the writer of

thriller stories. A good writer of thriller stories what you need is how to write thriller stories the storyline and so on. Just learning the word processing package does not lead you or does not automatically make you a good story writer. In the same way, just by learning the structure analysis and structure design tool you do not become a good designer. You need to understand the concepts well and of course, you need to practice with several problems and that will make you a good designer.

(Refer Slide Time: 14:09)

Structured Design

- The aim of structured design
 - Transform the results of structured analysis (DFD representation) into a structure chart.
- A structure chart represents the software architecture.
 - Various modules making up the system,
 - Module dependency (i.e. which module calls which other modules),
 - Parameters passed among different modules.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

Now, once the data flow diagram model is complete, based on the model we develop the high-level design and we call the process as the structure design. The structure design takes the results of the structured analysis and it transforms this structured analysis into a structure chart. For example, suppose we have set of data flow diagrams: the root level, level 1, level 2 diagrams and so on and through a structure design we transform those DFD models into a structure chart. So, we will see the methodologies by which we analyze the DFD model and come up with a structure chart. We will see those methodologies. The methodology which is used for DFD analysis is structure design methodology using which we can transfer the data flow diagram into a structure chart. And in the structure chart representation we will have the different modules of the system. The module dependencies of the system will be represented in the form of arrows and also, we will represent different data items that are passed between the modules.

(Refer Slide Time: 16:16)

Structured Design

- The aim of structured design
 - Transform the results of structured analysis (DFD representation) into a structure chart.

```
graph TD; root["root"] -- d1 --> ProcessOrder["Process-order"]; root -- d2 --> HandleIndent["Handle-indent"]; root -- d3 --> HandleQuery["Handle-query"]
```

- A structure chart represents the software architecture:
 - Various modules making up the system,
 - Module dependency (i.e. which module calls which other modules),
 - Parameters passed among different modules.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Above slide contain an example of a structure chart of a problem and all in the rectangle box are the modules of our design and for each we will write a module in the code. The arrow represents the call relationship between two modules. In our diagram, the root calls all other three modules. We will also represent the data flow that occurs between the different modules through some notation, we will write the name of the data d1, d2 and so on.

(Refer Slide Time: 17:10)

Structure Chart

- Structure chart representation
 - Easily implementable using programming languages.
- Main focus of a structure chart:
 - Define the module structure of a software,
 - Interaction among different modules,
 - Procedural aspects (e.g, how a particular functionality is achieved) are not represented.

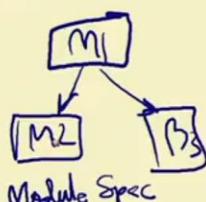
```
graph TD; root["root"] --> ProcessOrder["Process-order"]; root --> HandleIndent["Handle-indent"]; root --> HandleQuery["Handle-query"]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The structure chart identifies the module structure and we can easily program the structure chart. It is actually the high-level design and in the next step, based on the structure chart we do a detailed design where for each module we identify the algorithms, data structures and so on and that is directly transferred to code.

Structure Chart

- Structure chart representation
 - Easily implementable using programming languages.
- Main focus of a structure chart:
 - Define the module structure of a software,
 - Interaction among different modules,
 - **Procedural aspects (e.g, how a particular functionality is achieved) are not represented.**



So, in the structure chart we represent all the modules and the call relationship among the modules. In above slide, three different modules M1, M2, M3 and the call relationship among modules and the data exchanged among modules are shown, but we don't represent in the structure chart, what exactly happens in M2 or M1, what are the algorithms used in M2 or M3. All these things we do in the detailed design, where we identify the algorithms, the data structures that are used and we develop a module specification for each module.

So, the summary of the above part is that the structure chart representation is called as the high level representation, the high level design and based on the high level design we carry out the detailed design, where for each module we develop a module specification which contains the data structures that are to be used in the module and also the algorithms that would be used in this module.

(Refer Slide Time: 19:36)

Basic Building Blocks of Structure Chart

- Rectangular box:
 - A rectangular box represents a module.
 - Annotated with the name of the module it represents.

Process-order

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

A video player window shows a man speaking.

Now, let see what are the basic building blocks or elements of the structure chart. Of course, the central part of a structure chart is the modules. The modules are rectangles and the name of the module is written very simple. So, we draw a rectangle for each module and we just write the name of the module on the rectangle.

(Refer Slide Time: 20:07)

Arrows

- An arrow between two modules implies:
 - During execution control is passed from one module to the other in the direction of the arrow.

```
graph TD; root["root"] --> ProcessOrder["Process-order"]; root --> HandleIndent["Handle-indent"]; root --> HandleQuery["Handle-query"];
```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

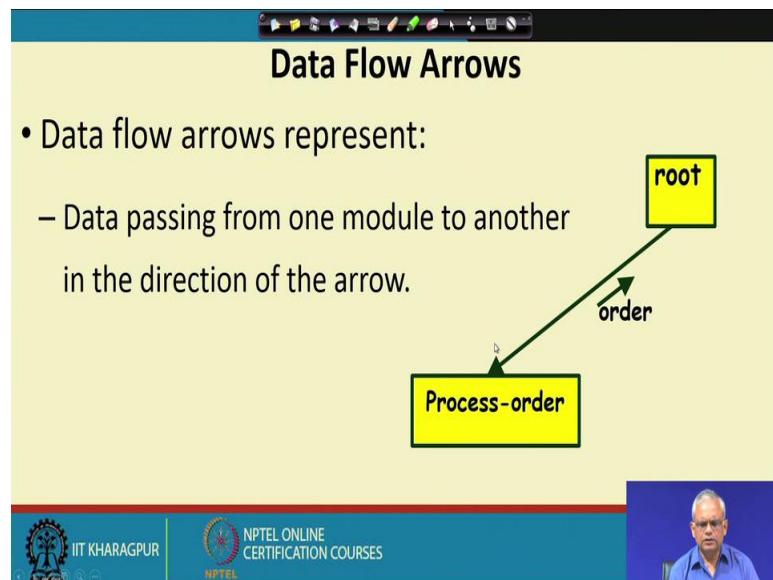
A video player window shows a man speaking.

The second element of a structure chart is the arrows. The arrow represents the invocation relation that is which module calls which module. And when there is a call from one module to another module, then the control is passed from one module to the

other. So, if root calls process order (shown on above slide) then the control is passed from root to the process order and then once the process-order completes the control is back with root and then the root may call handle-indent and then the handle-query.

But the sequential aspect of calling means whether it will call first Process-order, next Handel-indent, and third Handle-query is not really represented here. So, this just says that they are called, but the order is not indicated here. It may first call Handle-indent and then Process-order and so on.

(Refer Slide Time: 21:22)



The third item or the third element is the data flow. Data flow represent by arrow. In above slide, we draw an arrow from Process-order to root to show the direction of the data flow. So, what this represents is that the root calls the process order and the process order gives back the order to the root. It's a very simple notations.

(Refer Slide Time: 22:09)

The slide has a title 'Library Modules'. Below it is a bulleted list:

- Library modules represent frequently called modules:
 - A rectangle with double side edges.
 - Simplifies drawing when a module is called by several modules.

Below the list is a diagram of a library module, represented as a pink rectangle with double vertical lines on its left and right sides, containing the text 'Quick-sort'.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a video window showing a man speaking.

Now, there is one more element in the drawing of the structure chart which is library module. It's a module actually so it's a rectangle, but we need to draw two parallel lines within rectangle (Shown on above slide) and this indicates that some of the functions of a library module are called frequently by other modules and to represent this we just draw library modules in this notation.

(Refer Slide Time: 22:53)

The slide has a title 'Selection'. Below it is a bulleted list:

- The diamond symbol represents:
 - Each one of several modules connected to the diamond symbol is invoked depending on some condition.

Below the list is a diagram of a selection structure. It shows a yellow diamond labeled 'root' at the top, with three arrows pointing down to three yellow rectangles labeled 'Process-order', 'Handle-indent', and 'Handle-query'.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a video window showing a man speaking.

There are other few notations for example, to represent selection we draw a diamond (Shown on above slide) and it indicates selection. So, in our above slide in root we used

selection symbol that means it will call one of the module from below three module depending on some decision.

(Refer Slide Time: 23:28)

Repetition

- A loop around control flow arrows denotes that the concerned modules are invoked repeatedly.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

One more element here is the repetition. On the invocation arrow if we draw an arc shaped arrow (Shown on above slide) that indicates that these modules (Process-order, Handel-indent, Handel-query) are called many times in a loop.

(Refer Slide Time: 23:52)

- There is only one module at the top:**Structure Chart**
 - the **root module**.
- There is at most one control relationship between any two modules:
 - if module A invokes module B,
 - Module B cannot invoke module A.
- The main reason behind this restriction:
 - Modules in a structure chart should be arranged in layers or levels.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

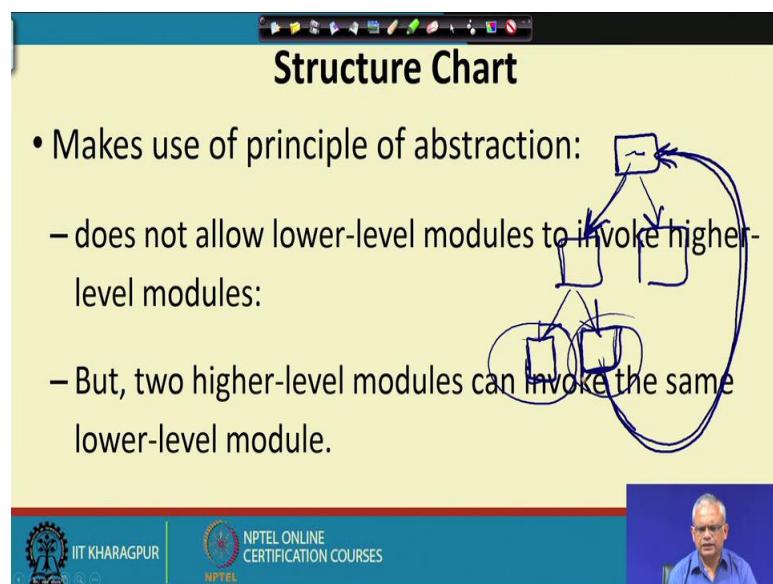
Now, in the structure chart there will be only one module as the root and then there are control relationship between two modules. If the module invokes the other module and

based on the structure design methodology we always come up with a structure like this (shown on above slide). In the above slide, we can see it's a tree like structure and arranged in levels, but it does not have lower level module calling a higher level module and for that, we don't have arrow from lower level module to higher level module.

If a lower level module call higher-level module then this is an example of a bad design. We had discussed while discussing about the good characteristics of design and we said there that the design has to be layered and higher layer module can call a lower layer module not vice versa.

We will see how the design methodology will give us automatically result in a layered representation or a layered design and there is no back arrows in that. If you are by chance while applying any methodology you come up with arrows which invoke the higher-level modules mostly that's not a good design unless there are some exceptional situation. Invoking higher-level module indicates that it is a bad design and we have to be careful.

(Refer Slide Time: 26:25)



The main reason why the lower level design modules should not call the higher-level modules is, because of the principle of abstraction. This we had also discussed earlier that due to the principle of abstraction at any time we are concerned about only some modules, we don't have to get concerned about all the modules together and if our module structure is a tree like diagram and we are in bottom modules of the tree then we

do not need to concerned about the upper layer modules, because the results are produced by the upper-level modules is passed to lower-level modules.

But just imagine that if we have a back arrow from lower-level module to upper-level module (shown on above slide), will violate the principle of abstraction. And therefore, understanding these design will be difficult, because when we try to understand the lower-level module which called upper-level module (Shown on above slide), we see that we need to understand the one which is root here that the lower-level module calls and then we need to understand also those modules which is called by the root module and we end up just going round and round and same thing happens while debugging.

In this case (Above slide case), if there is a failure we are trying to debug and if we try to find which module has the error, we see that we are looking at all modules round and round and we will have difficulty in debugging and understanding and so on. And therefore, the layering of the module is an instance of the principle of abstraction and helps us to develop a design which is easily understandable and we can easily debug that.

We are almost end of this lecture we will stop here and we will continue in the next lecture.

Thank you.

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

Lecture – 28
Essentials of Structure Chart

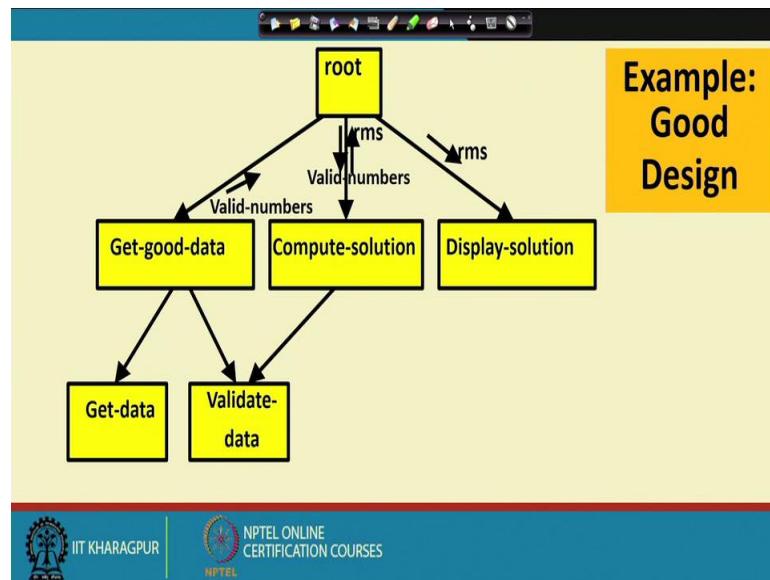
Welcome to this lecture.

In the last lecture, we were discussing about structure design. We have discussed some very fundamental aspects of structure design. We said that the result of the structured analysis that is the DFD model and through methodology we come up with the structure chart representation of the design. Therefore, the output of the structure design is a structure chart. The structure chart is also called as the high-level design or the software architecture and this high-level design is next transformed into detailed design. In high-level design for each module we write the module specification that is the algorithms to be used and the data structure to be used.

But towards the end of the last lecture we were discussing about what is a good structure design. And we said that a good structure design should have a layered structure and lower layer module should not call a higher layer module. If a lower layer module call a higher layer module then that will violate the principle of abstraction. The violation of abstraction principle will make the design difficult to understand the and also debugging will be difficult.

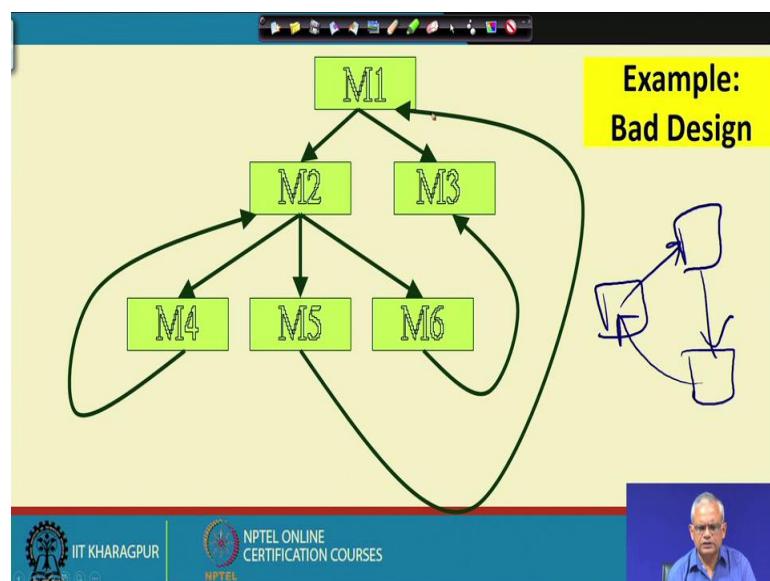
Now, let's look at how a good structure design should look like.

(Refer Slide Time: 02:02)



So, if we apply structure design methodology we should typically come up with this kind of a structure as shown in above slide. In above slide, we can see we have nice layering where the different modules are arranged and also there is no back arrow so that means it is not violating the principle of abstraction. A module at a layer can be called by different modules and if many modules are using a module then that's a library module.

(Refer Slide Time: 02:55)

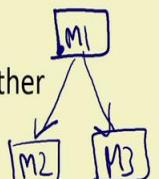


In the above slide, an example of a bad design is shown. We can see here that even though this design have some layering, but it have back arrows which is violating the

principle of abstraction. So, a bad design is one where we either don't have layering rather we just have arbitrary module structure and also there can have back arrows. So, either layering is not present or back arrows are present this will be called or this will be considered as a bad design.

(Refer Slide Time: 03:44)

Shortcomings of Structure Chart

- By examining a structure chart:
 - we can not say whether a module calls another module just once or many times.
- Also, by looking at a structure chart:
 - we can not tell the order in which the different modules are invoked.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES



Before we see how to come up with a structure chart representation, let's be aware of what are the shortcomings of a structure chart. If we have a call relation represented between two modules, for example, the module M1 calls M2 and M3(Shown on above slide). But it does not tell us that how many times M1 call M2, does it call five times, does it call ten times et cetera. This type of information is not represented in structure chart. And another shortcoming is that by looking at this type of representation we don't know whether M1 calls M3 first or M2 first. So, that information is also not represented here.

(Refer Slide Time: 04:52)

Flow Chart (Aside)

- We are all familiar with the flow chart representations:
 - Flow chart is a convenient technique to represent the flow of control in a system.
- A=B
- if(c == 100)
- P=20
- else p= 80
- while(p>20)
- print(student mark)

```
graph TD; A((A=B)) -- yes --> B((P=20)); A -- no --> C((P=80)); B --> D((dummy)); C --> D; D -- yes --> E((Print)); D -- no --> F(( ));
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Just to complete our discussion we will see how does structure chart compare with a flowchart. Even though this is a bit of digression, but for sake of completeness we will see that how does a structure chart differs from a flowchart. We know flowchart is a very popular technique represents the control flow as different statements are executed. It's a very popular notation that every engineering student knows about this. As we know that in flow chart, we have decision loops, sequential flow, decision and back arrows and so on.

(Refer Slide Time: 05:59)

Flow Chart versus Structure Chart

1. It is difficult to identify modules of a software from its flow chart representation.
2. Data interchange among the modules is not represented in a flow chart.
- 3. Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Given a flowchart representation the first thing we would notice is that the flowchart does not tell about what are the modules that means the module structure of the software is not represented in a flowchart. The second difference is that the flow chart represents only the control flow among the statements, but what data is exchanged between these is not represented. And also, looking at a flow chart we can trace through how the execution occurs. So, there is a sequential nature that is we can identify from flow chart but in a structure chart there is no such order and this is suppressed in a structure chart.

(Refer Slide Time: 07:10)

Transformation of a DFD Model into Structure Chart

- Two strategies exist to guide transformation of a DFD into a structure chart:
 - Transform Analysis
 - Transaction Analysis

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's look at the methodologies to transform DFD representation into a structure chart representation. Given a DFD model depending on the context or depending on the specific DFD we will either apply the transform analysis or the transaction analysis. These two are two different methodologies and therefore, by looking at the DFD we have to first understand which methodology is applicable and then we must apply that methodology. So, the construction of structure chart from a DFD is a systematic procedure.

Now, we will see what is this transform analysis methodology and by looking at the given DFD model can we make out whether to apply transform analysis and similarly we will look at what are the steps in the transaction analysis? And also, by looking at a given DFD, can we identify whether to apply transaction analysis? Over the next few minutes we will be addressing all these issues.

(Refer Slide Time: 09:09)

The slide is titled "Transform Analysis". It features a hand-drawn DFD model on a yellow background. The model consists of four main components: an "Input" oval containing two smaller ovals labeled P1 and P2, a "Logical Processing" section with two ovals labeled P1 and P2, a "Output" section with an oval labeled P1, and a central connector. Arrows indicate data flow from the input to processing and from processing to output. Below the diagram, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a professor.

First let's look at the transform analysis this is much simpler than the transaction analysis. Here given a DFD model we divide it into three parts: input, logical processing and the output. So, given some DFD model, we will divide this DFD into three parts based on some observation which we will discuss. So, the first part is the input part and as the name implies it takes the input from some processor, some from user and it transforms it from physical input to some logical form. For example, it may take input as a text and come up with a tree structure or it may take numbers and develop a table structure of the number.

The second part is the processing part that represents the processing that is done on the input data and also from other data elements. And then finally, the output part it transforms data from a logical representation like a table or a tree or something and it produces the output which is nicely formatted output. So, from our discussion we can say that the input part is responsible for reading the input and carrying out on that input some validation and structuring into some logical form. The output part is does just the reverse of the input part. Output part has some modules, which convert the logical data into physical part and it takes care of formatting and displaying it nicely for the user visualization. The rest of the processes are the central processing.

(Refer Slide Time: 12:19)

Transform Analysis

- Input portion in the DFD:
 - processes which convert input data from physical to logical form.
 - e.g. read characters from the terminal and store in internal tables or lists.
- Each input portion:
 - called an **afferent branch**.
 - Possible to have more than one afferent branch in a DFD.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

So, the input portion of the DFD it converts input data. It reads data from another process or from the user and converts from physical form to logical form. For example, it may read characters from a terminal and store in a list. Technically we call input part as an afferent branch. So technically the input part of a DFD is called as afferent branch. Given a DFD model how to apply the transform analysis that we will discuss next. Right now, we are just looking at the methodologies of the transform analysis and for that we need to identify the input part or afferent part, the output part and the central processing in a given DFD.

Above slide contain a DFD model which is given to us and we see that ‘Validate-move’ takes the input here and ‘Display-board’ are producing output. ‘Validate-move’ takes the data from another process or maybe from the user and we can easily identify that this is the input part or the afferent part.

(Refer Slide Time: 14:24)

Transform Analysis

- Output portion of a DFD:
 - transforms output data from logical form to physical form.
 - e.g., from list or array into output characters.
 - Each output portion:
 - called an **efferent branch**.
- The remaining portions of a DFD
 - called **central transform**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



‘Display-board’ is the one that is producing output and this output part also call as the efferent part. So, the afferent part takes the input and it stores in some logical form and the efferent part takes the logical form and it produces a nice output properly formatted and all the rest module in our example DFD we call as central processing part

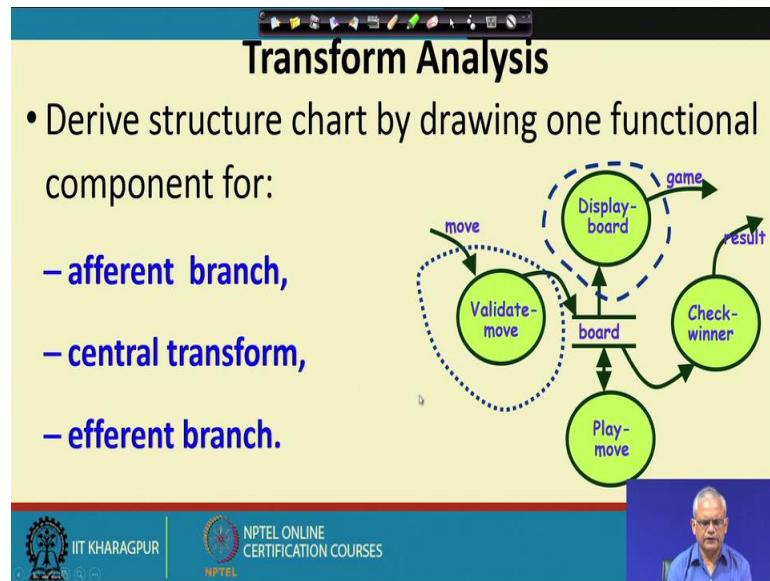
But if we carefully observe the DFD we can see that ‘Check-winner’ which is a central processing but producing some output. So, a question can be arise that why not make it a output or efferent part? You can make it as an afferent part, but then that will not become a good design because the main tasks done by the ‘Check-winner’ is processing. Even though it produces an output, but still we will consider as a processing part this is the discretion that we have to use based on the experience. A module which takes input data produces output data just by observing many times it is not sufficient to label them as afferent or efferent branch. We need to see that what main task it does. The ‘Check-winner’ even though it produces some output, but the main work it does is processing. So, even though ‘Check-winner’ produce some output for the main processing task we classify this ‘Check-winner’ as central processing part.

So, given a DFD if we want to do transform analysis, first we have to divide the DFD into three parts: the input part, the processing part and the output part. And once we have done that, we just need to represent that in the form of a structure chart. So, for a given DFD if we identify input part , output part and the central processing part then from this

coming up with the module structure is straight forward. In structure chart, we just draw root which will call these three modules: the validate move or get move, display board and let say check winner or we will call it as process move.

So, the transform analysis is really very simple. We just identify the input part, output part and the central processing and in the structure chart we draw a root level and from there we call these three modules.

(Refer Slide Time: 19:22)



So, the output of transform analysis is a structure chart, where we just draw the three modules corresponding to the afferent branch, central transform and the efferent branch.

(Refer Slide Time: 19:38)

- Identifying input and output transforms:
 - requires experience and skill.
- Some guidelines for identifying central transforms:
 - Trace inputs until a bubble is found whose output cannot be deduced from the inputs alone.
d1 → P1 → d2 → P2
 - Processes which validate input are not central transforms.
 - Processes which sort input or filter data from it are.

As we have seen that it's not really totally mechanical, it requires some experience and skill. Even though input data to a process indicates that it should be part of the input, but then many times it may not be. And similarly, just because this produces output may not be an output part, it maybe central transform or something.

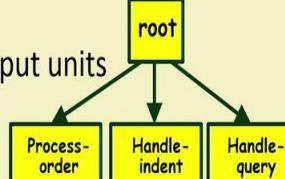
But there are some guidelines to identify the central transform. The guideline essentially says that the central transform part main activity should not be to just read data from user and convert it to logical form. To identify the central transform, we trace the input until a bubble is found whose output cannot be deduced from the input alone so that means, that it's not a input branch. In the input branch, we just read the input and it does some transformation on that.

So, in the input branch, the output is determined based on the input to this, but for a central processing unit, it will use some previous data and just by looking this data will not be able to identify simple function to convert input to output. And of course, the processes which just validate input are not central transform. Those processes are part of the input part. But if those processes do some processing on the input like filtering, sorting, sorting etc. then those can be central transforms.

(Refer Slide Time: 22:15)

Transform Analysis

- First level of structure chart:
 - Draw a box for each input and output units
 - A box for the central transform.
- Next, refine the structure chart:
 - Add subfunctions required by each high-level module.
 - Many levels of modules may required to be added.



 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES

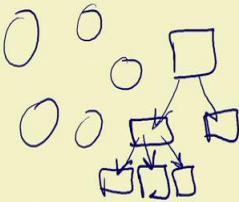


So, once we identify the input part, output part and central transform we just represent them three different modules and have a called relationship. And once we have the module structure like this (As shown on above slide), next we refine it by adding any subfunctions that may be necessary to carry out the work. For example, in above slide structure chart we need to add validate then we have to format it nicely.

(Refer Slide Time: 23:16)

- The process of breaking functional components into subcomponents.
- Factoring includes adding:
 - Read and write modules,
 - Error-handling modules,
 - Initialization and termination modules, etc.
- Finally check:
 - Whether all bubbles have been mapped to modules.

Factoring



 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES

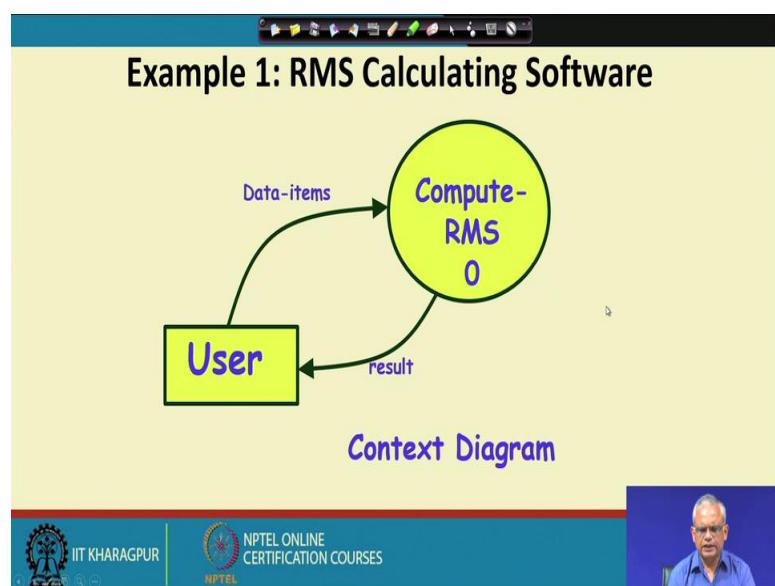


Once we have the basic structure of a structure chart, we factor the lower level modules. We might add read write modules, error handling modules, initialization modules and so

on. And the final check whether we have correctly transformed a DFD into a structure chart is to observe and need to verify that each of the process present in the structure chart belongs to either input part or the output part or the central transform of the DFD. It should not be the case that we have not included a process neither in input, output nor in the processing part.

So, for the factoring of the basic module structure, we add modules at the lowest level. For example, we can add read write modules which can be read from terminal, read from file, read from network and so on, error handling initialization et cetera. So, we can add a couple of layers if necessary and added layers are the lower level modules. The lower level modules are called by the upper level module to do some activities. By factoring, we enhance the basic skeleton of the structure chart.

(Refer Slide Time: 25:05)

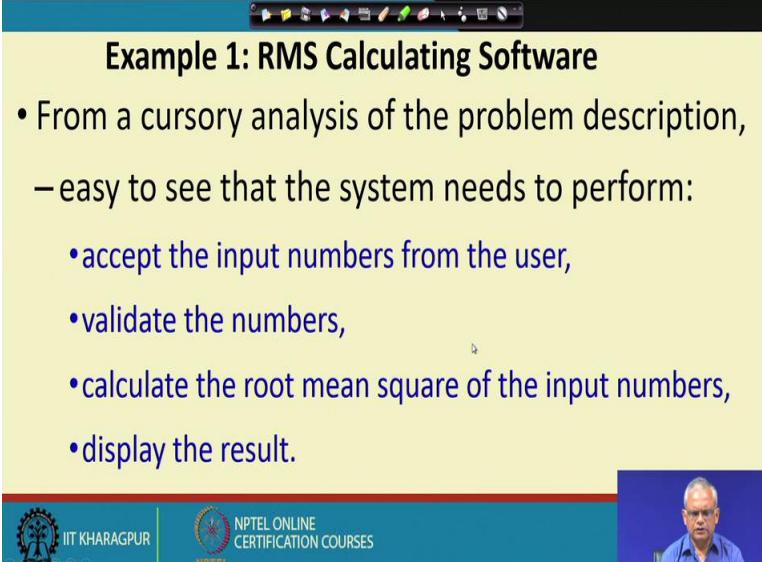


Now look at an example of Compute-RMS. This is the simplest example we had discussed it earlier for developing the DFD representation. And the context diagram if you remember we had a single user who would input data items and this will return the result which is the root mean square.

(Refer Slide Time: 25:36)

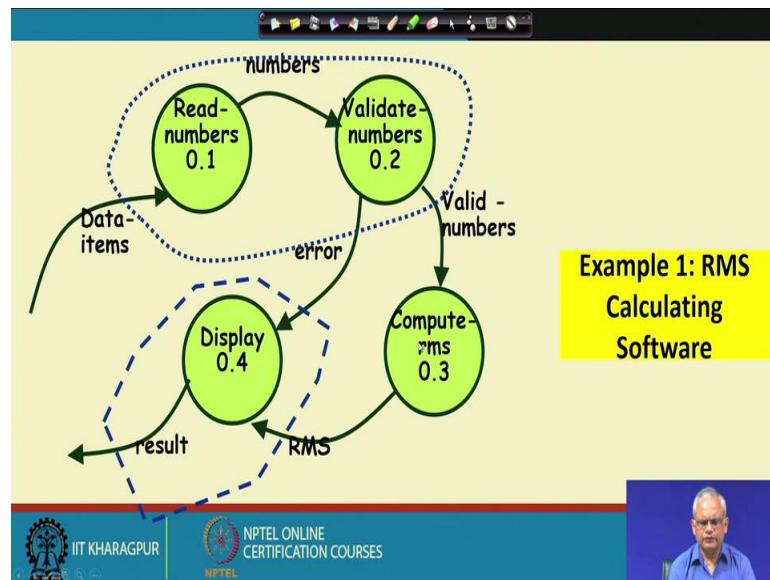
Example 1: RMS Calculating Software

- From a cursory analysis of the problem description,
 - easy to see that the system needs to perform:
 - accept the input numbers from the user,
 - validate the numbers,
 - calculate the root mean square of the input numbers,
 - display the result.



And we had also developed the level 1 diagram where we identified the activities of the RMS software. Where the activities are accept input, validate the number to check whether the number lie between -1000 to +1000, calculate the root mean square, and then display the result.

(Refer Slide Time: 26:01)



Based on those activities we had come up with such a DFD representation (Shown on above slide). Here, read data items are passed on to 'Validate-numbers' and after validation check these are passed on to the 'compute-rms', which produces the result and

the result passed on to the ‘Display’. If the number comes out invalid then the ‘Validate-numbers’ may also produce some error messages which are passed onto the ‘Display’.

(Refer Slide Time: 26:38)

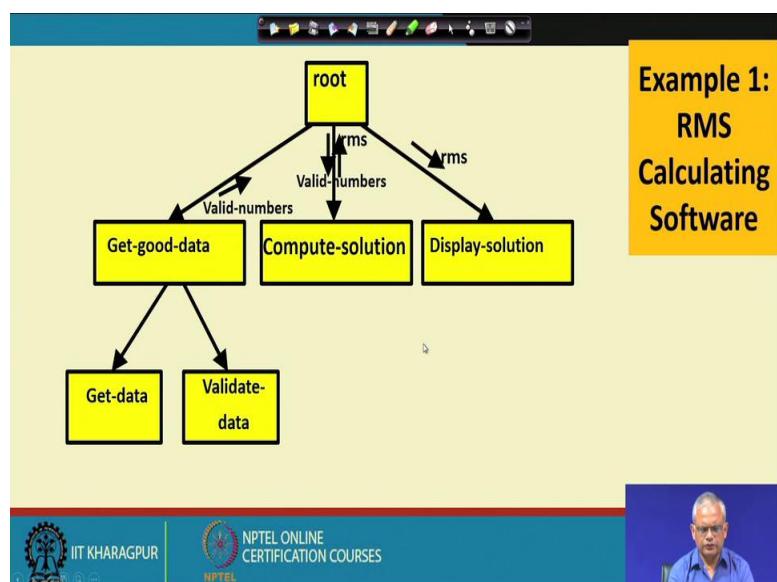
Example 1: RMS Calculating Software

- By observing the level 1 DFD:
 - Identify read-number and validate-number bubbles as the afferent branch
 - Display as the efferent branch.

The slide has a blue header bar with standard window controls. The main content area has a light beige background. At the bottom left, there is a logo for IIT Kharagpur and NPTEL Online Certification Courses. On the right side, there is a small video window showing a man speaking.

Now, if we apply the transform analysis on our ‘Compute-RMS’ DFD, we will make ‘Read-numbers’ and ‘Validate-numbers’ as the input part, because validate, read et cetera are part of the input. ‘Display’ is part of the output and the ‘Compute-rms’ is the processing part. So, in structure chart we will draw three modules for the input part, output part, and the central processing part.

(Refer Slide Time: 27:14)



‘Get-good-data’, ‘Compute-solution’ and ‘Display-solution’ are three module of the structure chart and we might factor ‘Get-good-data’ into ‘Get-data’ and ‘Validate-data’. Observe that these two ‘Get-data’ and ‘Validate-data’ were two different bubbles on the DFD representation. We have made them into two modules in structure chart, but that’s not really done all the time. It’s just a guideline that if you have many bubbles in DFD mapped on to a module of structure chart, then we need to factor that.

We are almost at the end of this lecture. So far, we looked at the transform analysis methodology to transform a DFD into structure chart representation. Next, we look at the transaction analysis and how to come up with the module structure for a DFD based on the transaction analysis. And we will also discuss when to apply transform analysis and when to apply the transaction analysis.

We will stop now and continue in the next lecture.

Thank you.

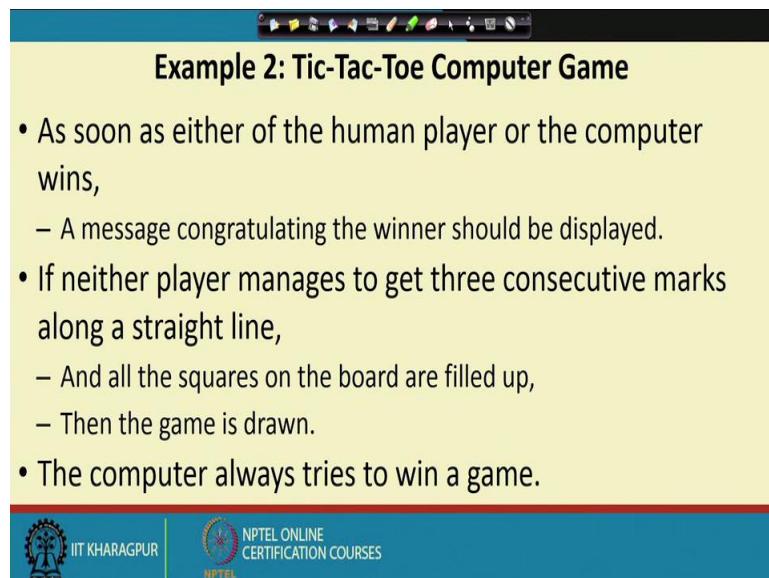
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 29
Structure Chart Development

Welcome to this lecture. In the last lecture we had discussed about the transform analysis. Next, we will discuss about transaction analysis and we will also see, when to apply transform analysis and when to apply transaction analysis. Before we proceed, we will just take one more example to see how the transform analysis is applicable? Because, the last example that we saw, that is the root mean square computation is a very simple trivial example, we will take slightly more non trivial example and see how transform analysis can be applied.

So, let us look at the Tic-Tac-Toe Computer Game.

(Refer Slide Time: 01:04)



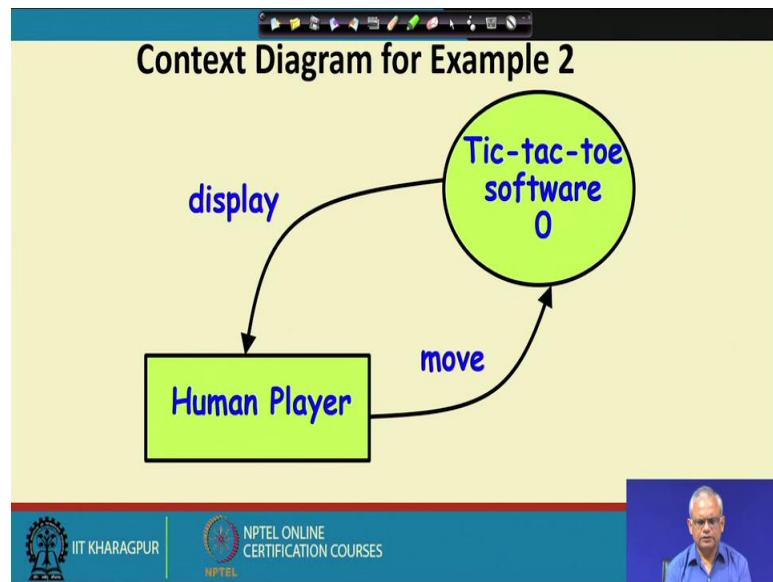
Example 2: Tic-Tac-Toe Computer Game

- As soon as either of the human player or the computer wins,
 - A message congratulating the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
 - And all the squares on the board are filled up,
 - Then the game is drawn.
- The computer always tries to win a game.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

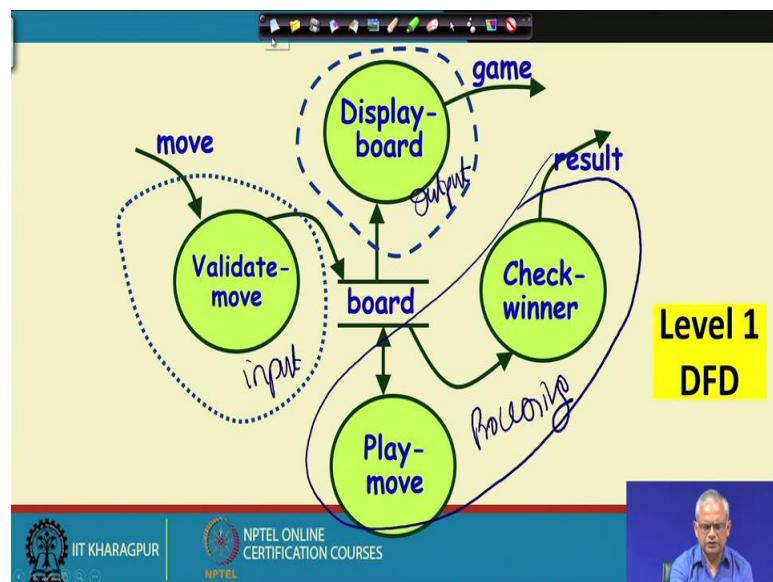
We had developed the DFD representation of the Tic-Tac-Toe Computer Game. We had seen the problem statement earlier, that human player and the computer alternatively make moves. One player wins by marking on conjugative boxes in a row or a column or on the diagonal. After a win, a winner message should be displayed, but if no one gets that and all the board elements are filled up then the game is drawn and message that the game is drawn is displayed.

(Refer Slide Time: 01:54)



We had drawn context diagram for the Tic-Tac-Toe software. The human player gives moves to the software and it checks whether the human wins then it will display a win message congratulating the human player. Otherwise it makes its own move checks. If its own moves makes it winner, then it will display the corresponding message or it just displays the updated board and prompts the human player to make the next move.

(Refer Slide Time: 02:47)

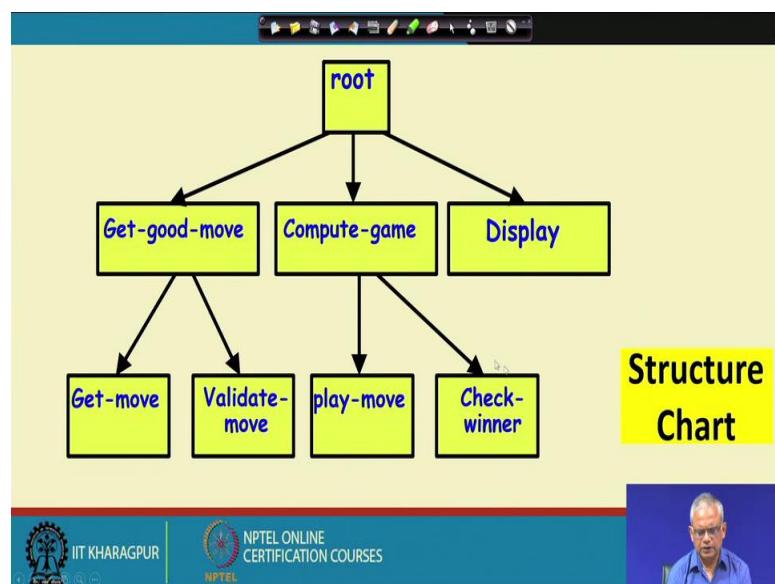


In the level 1 DFD, we had seen that the move is read and it's validated and then the board data structure is updated. The board data structure is an array of nine elements that

we had discussed and based on the input, the check winner is executed and if there is a result it's displayed otherwise, the computer makes move and finally, the board is displayed.

So, how do we do the transform analysis of this? We see that 'Validate-move' in the Level-1 DFD is the input part and it takes the input and converts physical data into logical form and the 'Display-board' is the output part. And, the 'Check-winner' and 'Play-move' are the processing part and even though there is bubble in the processing part, it produces some output, but then we observe that its primary role is processing. Processing part task is not to convert logical data into physical form, its main work is to do processing. So, we classified DFD in different part. And, once we have done that we can easily draw the module structure.

(Refer Slide Time: 05:03)



In the above slide the module structure chart is shown. Get-good-move, Compute-game, Display all are the modules. And, also observe here that Get-good-move and compute-game are again factored. We can see from the structure is that Get-good-move is factored into Get-move and Validate-move. Similarly, Compute-game consists of two different activities play-move and check-winner. If you observe the DFD see that Validate-move represent by bubble, but it's not really necessary that each bubble that is mapped into a module here becomes a sub module here.

(Refer Slide Time: 05:48)

Transaction Analysis

- Useful for designing transaction processing programs

– **Transform-centered systems:**

- Characterized by similar processing steps for every data item

processed by input, process, and output bubbles.

– **Transaction-driven systems,**

- One of several possible paths through the DFD is traversed depending upon the input data value.

NPTEL ONLINE
CERTIFICATION COURSES

Now, let's look at transaction analysis, but before we look at the transaction analysis, let see a general idea, when to apply transform analysis and where to apply transaction analysis? In transform analysis, normally whatever data is input into the DFD they are all use some similar processing and finally, produce the output. In the slide, the data d_1 is input and undergoes similar processing. Typically DFD takes just one data, but it is possible that it may take multiple data items and they undergo similar processing steps.

Where is in a transaction system, some data are processed by one bubble. So, this is one transaction $d_1 \rightarrow d_4$ (Shown on 2nd diagram of above slide) and another data d_2 may be processed through a different bubble. So, each of these will become a transaction. In the slide, we can see d_1 data is processed through one bubble and d_2 through another bubble. So, they are two different transactions.

(Refer Slide Time: 07:50)

Transaction Analysis

- **Transaction:** Any input data value that triggers an action:
 - For example, different selected menu options might trigger different functions.
 - Represented by a tag identifying its type.
- Transaction analysis uses this tag to divide the system into:
 - **Several transaction modules**
 - **One transaction-center module.**

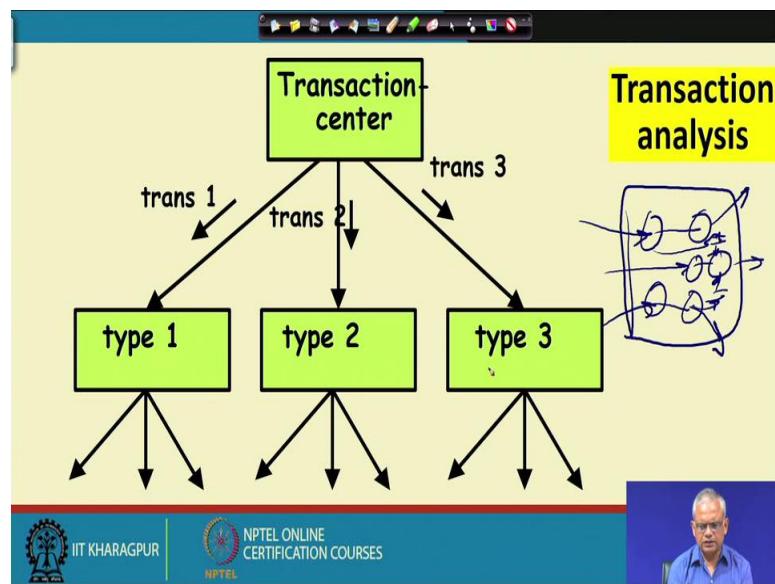
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Typically, when there are menu options, depending on the chosen menu option, some functions are executed.

For example, if you want to draw a package, you want to do editing, so some specific functions will be invoked. If you want to do printing, some other functions will be drawn that will be executed, if you want to do save, some other functions will be executed. So, typically the data that is input to a DFD, they are processed by different functions and each of those functions we call as a transaction and we can represent this in the form of a structure chart.

In the 1st diagram of above slide, two transactions are shown. Transactions are consists of functions. In 1st transaction, data is processed through 1st transaction functions and similarly, data processed through the functions of 2nd and 3rd transaction. After the transaction analysis, these 3 transactions represented in the structure chart (2nd diagram of above slide). In the 2nd diagram of above slide, we can see, we have a root module or the transaction centre module and the three transactions become T1, T2, and T3 transaction here. And, then we again do a transaction analysis or a transform analysis on each of the transactions. And finally, we will get a module structure, will see the details as we proceed.

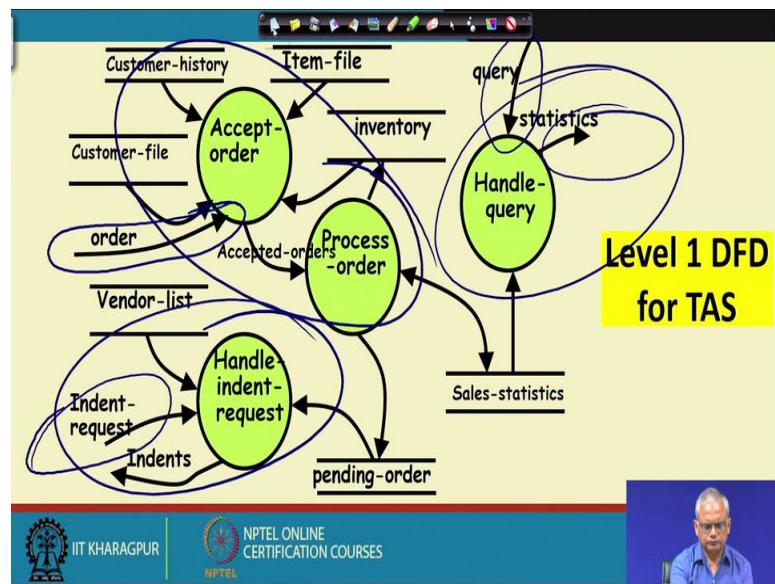
(Refer Slide Time: 10:07)



So, the main point here is that if we observe a DFD and see that it takes different data, and process the data, then, we call this as the transactions and this transaction process maybe takes data, produces output, and may be they share some data and so on.

So, we identify the transactions here and we write a transaction centre module and then we draw the three transactions (Shown on above slide), and this each transaction needs to be decomposed into further modules.

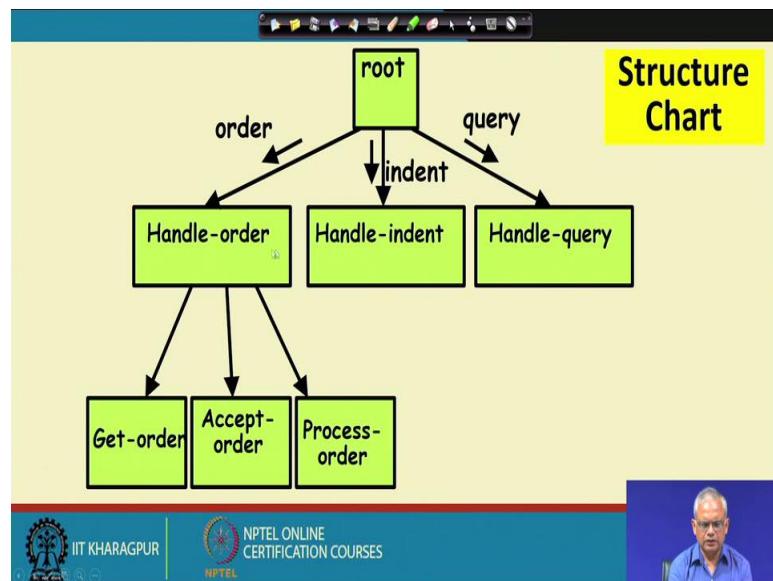
(Refer Slide Time: 11:26)



For example, if we have this kind of a DFD (Shown on above slide). We see here that it takes many data items. For example, it takes one data item here in Handel-indent request and it is processed here based on the pending order and it produces an output. Whereas a query is a data item which is processed through Handel-query and the output is produced and the order is another data item which is processed through Accept-order and process-order and an output is produced.

So, we will identify three transactions here and once we identify the transaction in DFD, we will draw a transaction center module in structure chart and then the next level will draw the three transactions.

(Refer Slide Time: 12:28)



So, this is what the three transactions: handle-order, handle-indent, handle-query and then we factor Handel-order into get-order, accept-order and process-order.

(Refer Slide Time: 12:43)

Summary

- We discussed a sample function-oriented software design methodology:
 - Structured Analysis/Structured Design (SA/SD)
 - Incorporates features from some important design methodologies.
- SA/SD consists of two parts:
 - Structured analysis
 - Structured design.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we saw that the structure chart is obtained as a result of the structured design. We first do a structured analysis given a problem, where we look at the requirement specification and then we develop the functional decomposition.

We come up with the DFD model, typically consists of many levels and then the next step we do the structured design. In the structured design we take the DFD representation and for each DFD we identify whether we apply transform analysis or transaction analysis. Typically, very simple processing that is the lowest level of the DFDs they become transform analysis. So, if we have a DFD which is the level 1 DFD and then we based on the level 1 DFD, for each of the bubble here in the level 1 DFD, we develop the level 2 DFD and then the level 3 DFD if necessary and so on.

So, given a design like the context diagram, we don't really apply structure design on it. Apply of structured design start with the level 1 diagram, unless it's a very simple system like let say the Tic-Tac-Toe or this compute-RMS et cetera.

Typically the level 1 DFD we observed, the transform analysis needs to be performed. Only for very simple software level 1 DFD will lead to a transform analysis, but typically transaction analysis needs to be done for any non trivial software. Because a non trivial software will provide different functionalities. But for trivial software the case is different, just observe that both compute RMS and the Tic-Tac-Toe have only one user and could give only one type of move.

But, when there are multiple users like let say library software or any non-trivial software, here each user can give different types of data to the system. For example, a user in a library software might issue a book or return a book or search for a book and based on which functionality in books, different bubbles will get activated.

And therefore, the level 1 DFD for any non-trivial software will normally be a transaction processing. So, we will have to apply transaction processing and even the level 2 is often a non-trivial software, different transactions are possible here.

And, normally at the lowest level of the DFD, the transform analysis is applied, because those represent the simplest processing. And, once the module structure is obtained, then the processes are identified in DFD for transactions and do the next level of the structure chart and so on.

And, then we look at the next level. The transform analysis were corresponding to the bubbles which map on to structure chart. This process will continue until we reach the lowest level DFD.

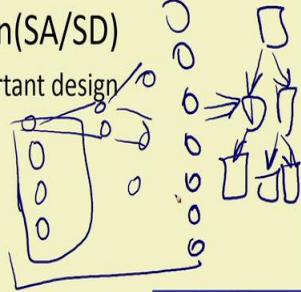
So, as a summary of our discussion, we can say, first we looked at the basics of procedural design, where starting with the requirement specification. Then, by taking the requirement specification we identify the functions and draw the DFD model. And, from the DFD model we get the structure chart or high level design. And, once we get the high level design we do the detailed design. In detailed design, for each module we develop the module specification in terms of the data structure and the algorithms.

In the next topic, we will take up is the object-oriented design. The object-oriented design is completely different then the procedural design, because we will see that the procedural design is a top down design. It start with this function specified in the requirements documents then split them, decompose them into sub-functions and so on. And finally, we get the structure chart representation.

(Refer Slide Time: 20:06)

Summary

- We discussed a sample function-oriented software design methodology:
 - Structured Analysis/Structured Design(SA/SD)
 - Incorporates features from some important design methodologies.
- SA/SD consists of two parts:
 - Structured analysis
 - Structured design.



 IIT KHARAGPUR

 NPTEL ONLINE
CERTIFICATION COURSES
NPTEL



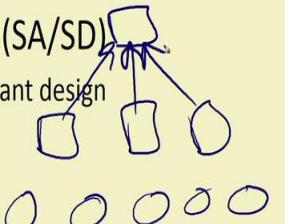
So, we start with the functions represented in the requirements document and then achieve the decomposition of that and at the end of the data flow diagram we get a set of simple functions and through a structured design, we map this into a module structure.

So, the procedural design is a top down design on the other hand will see that in the object oriented design, we have a bottom up design.

(Refer Slide Time: 20:55)

Summary

- We discussed a sample function-oriented software design methodology:
 - Structured Analysis/Structured Design(SA/SD)
 - Incorporates features from some important design methodologies.
- SA/SD consists of two parts:
 - Structured analysis
 - Structured design.



 IIT KHARAGPUR

 NPTEL ONLINE
CERTIFICATION COURSES
NPTEL



In the object-oriented design, we will first identify the objects and then, once we identify the objects we will build the classes out of those objects. And, then will build the class relations in terms of aggregation, in terms of inheritance and so on. And, then will do few other diagrams like structure chart and so on and then we will get the code structure.

So, we can say that the procedural design is a top down design where the object-oriented design is a bottom up design. So, the two design techniques appear to be entirely different but then we will see that even during the object-oriented design, we will use the concepts of the procedural design, because after all for every class will have methods. And, some of the methods are very complex method and we need to design those using the ideas from procedural design.

We will stop now and will continue in the next lecture.

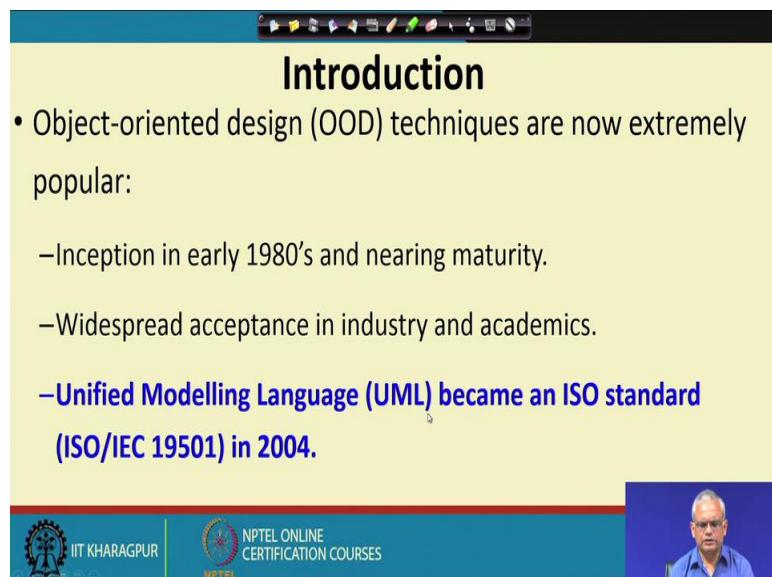
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 30
Structured Design Examples

In this lecture, we will see very initial of the Object-Oriented Design.

In last few lectures, we have discussed about the procedural design. And, now we will look at the nitty-gritty of object-oriented design. For, object-oriented design we will use UML as the notation.

(Refer Slide Time: 00:43)



Introduction

- Object-oriented design (OOD) techniques are now extremely popular:
 - Inception in early 1980's and nearing maturity.
 - Widespread acceptance in industry and academics.

**–Unified Modelling Language (UML) became an ISO standard
(ISO/IEC 19501) in 2004.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Nowadays, object-oriented design techniques have become extremely popular. Initial work in object-orientation is started in 1980s and now its nearing maturity. Object-oriented design is widespread acceptance in industry and academics. UML is the modelling language using which object-oriented design is done. UML stands for unified modelling language and that has become an ISO standard in 2004. So, we can say object-oriented design is much more standardized than the procedural design. Now, we will discuss about the various aspect of UML model and will see the design using UML model.

(Refer Slide Time: 01:49)

Object Modelling Using UML

- UML is a modelling language.
 - Not a system design or development methodology
- Used to document object-oriented analysis and design results.
- Independent of any specific design methodology.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

As the name says UML stands for unified modelling language and therefore, the name implies it is a modelling language or we can say that it is a language. UML is a language using which we can document the model and the design. Let's be clear that UML is just a language and it is not a design and development methodology. So, It's just a language, using that we can document the design, but the design will be obtained using a design methodology, and we use UML as a documentation for that design.

One good thing about the UML is that it is independent of any design methodology. UML is used for documenting the design, but the design can be obtained using any methodology.

(Refer Slide Time: 03:41)

UML Origin

- OOD in late 1980s and early 1990s:
 - Different software development houses were using different notations.
 - **Methodologies were tied to notations.**
- UML developed in early 1990s:
 - To standardize the large number of object-oriented modelling notations that existed.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we will see how UML came and how it gets popular. We already said that the object-oriented design was first used in 1980s and when the object-oriented design came up, different researchers proposed different methodologies. There were a large number of methodologies for object-oriented design in the 1980s and 90s. And, one thing about these methodologies is that each of them uses different notations. And therefore, it became very difficult in companies or for the students to come up with a same standard design notion. Because in a company there may be many projects and each project maybe using, different methodology for documenting so the result in different notations. And therefore, one project cannot really reuse or understand the design of another team and it led to lot of confusion. It was felt that there needs to be standardization and in the early 90s an attempt was made to standardize all these different design methodologies and notations.

(Refer Slide Time: 05:14)

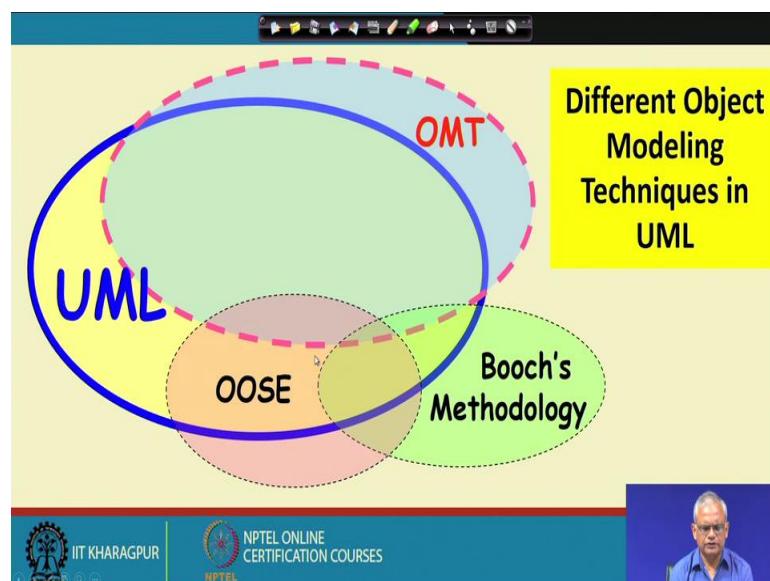
UML Lineology

- Based Principally on:
 - **OMT** [Rumbaugh 1991]
 - **Booch's methodology** [Booch 1991]
 - **OOSE** [Jacobson 1992]
 - **Odell's methodology** [Odell 1992]
 - **Shlaer and Mellor** [Shlaer 1992]

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Those days extremely popular methodologies were object modelling technique(OMT) by Rumbaugh, Booch's methodology by Grady Booch, object-oriented software engineering(OOSE) by Jacobson, Odell's methodology by Odell, Shlaer and Mellor methodology. These methodologies are differed from each other in notations. And when there was an effort to standardize these methodologies, they had to look at these methodologies and the notations and based on that the UML was proposed.

(Refer Slide Time: 06:01)



As, we can see in the above diagram that the notations for the UML have been borrowed largely from the OMT, which was extremely popular design technology having it's own notations. Some of the notations are taken from Booch's methodology, some of the notations are taken from the object-oriented software engineering (OOSE) and some of the notations were not there in any of these methodologies and these are new.

As a part of our discussion about UML, we must remember that most of the notations here are borrowed from some of the popular methodologies and, few of the notations on its own. As we proceed, we will point out from where these notations have been taken.

(Refer Slide Time: 07:13)

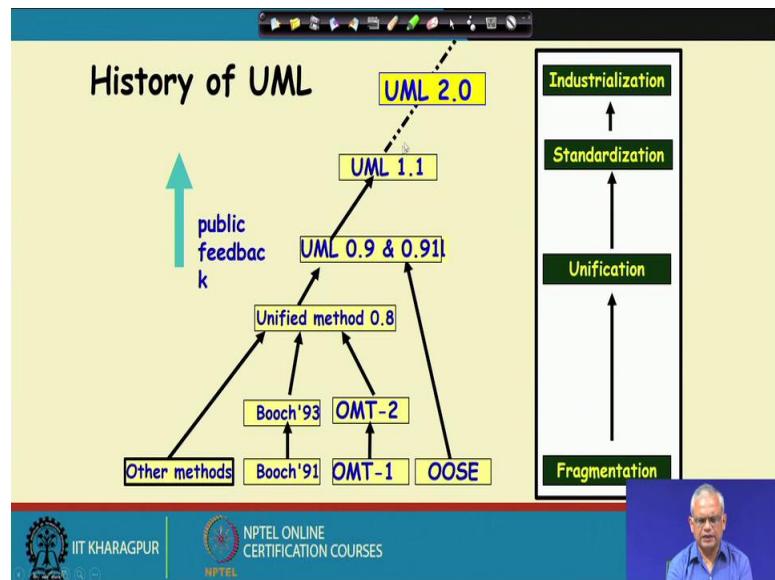
UML as A Standard

- Adopted by [Object Management Group \(OMG\)](#) in 1997.
- [OMG](#) is an association of industries
- Promotes consensus notations and techniques
- UML also being used outside software development area:
 - Example [car manufacturing](#)

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

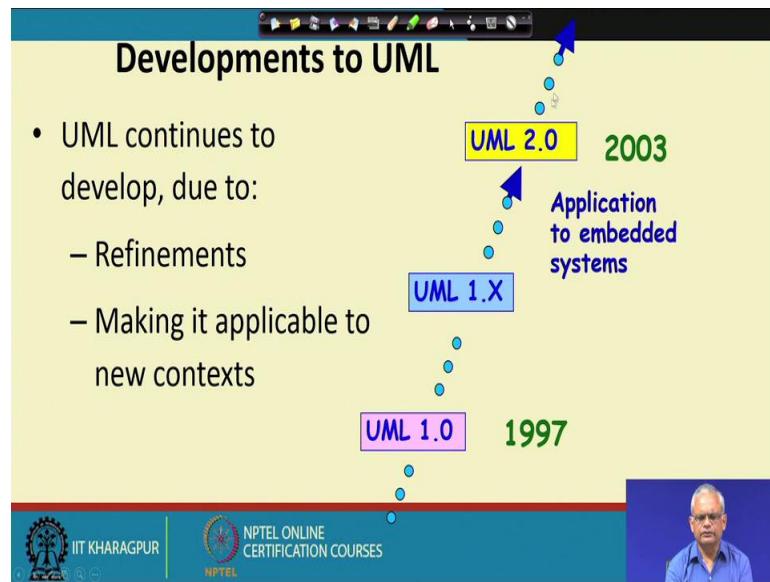
The object management group (OMG) adopted the UML in 1997 and OMG is an association of industries. It tries to promote consensus notation and techniques just to have uniformity standardization, but this is not really a standardization body it can just adopt its own notations. So, that become popular and ultimately need to lead to standardization. And, once it was adopted by the OMG, it became extremely popular and not only it is used extensively by the software development area, but also even domains which are entirely different, they are also using UML notations.

(Refer Slide Time: 08:21)



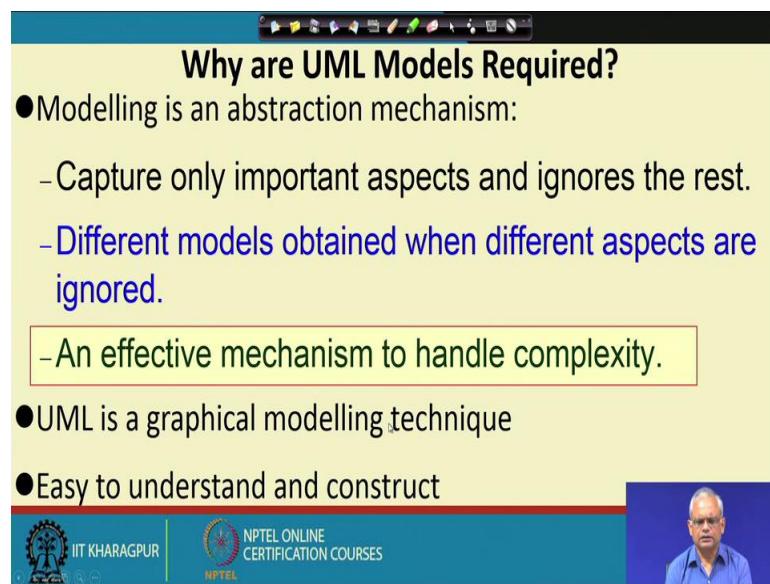
Now, we will see the development of UML. If, we look at the above diagram, we can see that various methodologies were combined in UML. The UML version one was released in 1997 is the more standardized version than all the previous UML version. So, there were a fragmentation of different methodologies in the 80s and 90s and towards the end of nineties all methodologies unified into the unified modelling language. And, then as these were being used there were need for extensions to the UML. For example, to make it use in a particular domain may be some more notations are needed and so on. So, different versions of UML came up and finally, as these were applied to different domains, there were some shortcomings of the UML were noted. For example, it does not handle events parallel processing and so on. And, UML 2.0 was released in the year 2004 and mainly to make it applicable to some domains of industry like embedded systems and so on.

(Refer Slide Time: 09:47)



So, UML 1.0 released in 1997 it was a unification of various methodologies that existed. And, then it continued to evolve until 2003 then UML 2.0 version was released and even now it continues to evolve further.

(Refer Slide Time: 10:11)



First thing let's understand before we discuss about UML modeling, that why do we need a model? Also, we need to answer the question is modelling same as designing? If, you remember the early parts of our discussion where we said that models are actually an abstraction mechanism. Abstraction is necessary to simplify complex problems. We need

to create an abstraction where we develop a simple model of a complex problem and then slowly, we go through various hierarchies then we have the entire problem modeled.

And, then once we model a problem, we can design from the model. So, model is an abstraction mechanism where we construct a very simple representation of the problem, by ignoring different aspects of the complex problem. And, model is an effective mechanism to handle complexity. Typically, we use a graphical modelling technique to model the problem domain. This graphical modelling is also called analysis model. Once we come up with the analysis model, we can convert it to a design model. So, the answer of the question ‘Is designing the same thing as modelling?’ is yes design is a model, but all models are not designs. We can have analysis models which are just models of the problem and the model of the problem can be transformed into a design model, which can be easily implemented.

(Refer Slide Time: 12:19)

UML Diagrams

- Nine diagrams in UML1.x :
 - Used to capture 5 different views of a system.
- Views:
 - Provide different perspectives of a software system.
- Diagrams can be refined to get the actual implementation of a system.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Now, we will see the diagrams that are used in UML. We said that UML is a language and it is a graphical language. In this graphical language we have different diagrams. We will see the graphical notation of UML. This graphical notation is the vocabulary of the UML and also, we will see the construction of the model.

UML diagrams are used to capture the views of a system. There are five views of a system and these views provide different perspectives of a software.

These views required to develop UML diagrams and once we developed UML diagram, then we transform it or refine it to get the actual design model and the implementation of the system.

(Refer Slide Time: 13:09)

● Views of a system:

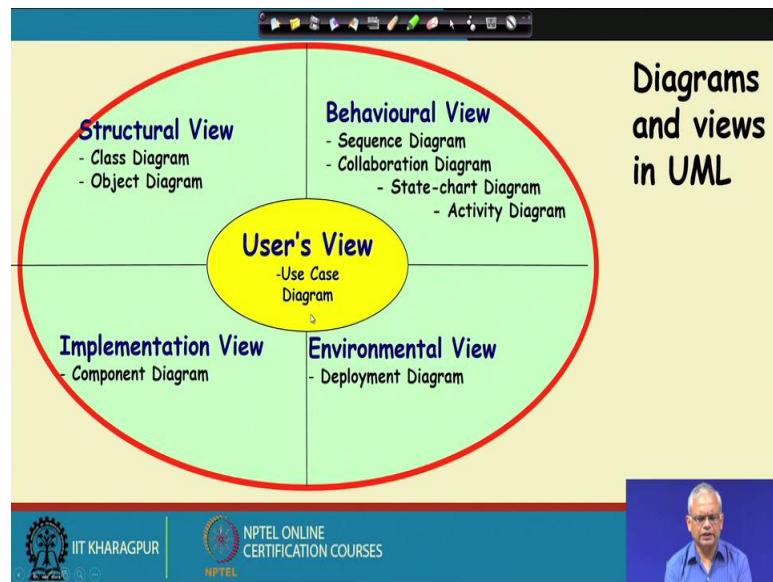
- User's view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

UML
Model
Views

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The five views of the system are: user's view, structural view, behavioral view, implementation view and the environmental view. The users view is the one which says how does the customer or the user view the system. So, this is the external view. The structural view is an internal view of the software. Structural view says about classes, objects, object relations and so on. Whereas a behavioral view represents system behavior on a given input. The implementation view says about the organization of internal structure and the environmental view depicts the different parts of the system.

(Refer Slide Time: 14:28)



In above slide, diagram of UML model views shown. We can see the user's view in the center and it is represented in the form of a use case diagram. We have drawn 'User's View' at the centre because this is the central view based on which other views and other diagrams are developed. The user's view is the starting point, because the user gives the requirements. And, the requirements are modeled using a use case diagram and based on this the other views and diagrams are developed. As user view is the starting point because of that we draw users view at center.

The structural view is represented by the class diagram and the object diagram. The behavioral view through sequence diagram, collaboration diagrams, state chart diagram, and the activity diagram. The environmental view through deployment diagram, and the implementation view through the component diagram.

(Refer Slide Time: 15:36)

Structural Diagrams

- **Class Diagram**
 - set of classes and their relationships.
- **Object Diagram**
 - set of objects (class instances) and their relationships
- **Component Diagram**
 - logical groupings of elements and their relationships
- **Deployment Diagram**
 - set of computational resources (nodes) that host each component.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let see the structural diagrams. The first structural diagram is the class diagram which consists of set of classes and their relationships. Next, is the object diagrams which are set of objects and their relations. Then the component diagrams which are a logical grouping of elements and their relations and the deployment diagram which consists of set of computational resources.

(Refer Slide Time: 16:01)

Behavioral Diagrams

- **Use Case Diagram**
 - high-level behaviors of the system, user goals, external entities: actors
- **Sequence Diagram**
 - focus on time ordering of messages
- **Collaboration Diagram**
 - focus on structural organization of objects and messages
- **State Chart Diagram**
 - event driven state changes of system
- **Activity Diagram**
 - flow of control between activities

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the behavioral diagrams. Behavioral diagrams are the use case diagram, the sequence diagram, collaboration diagram, the state chart diagram, and the activity diagram.

The use case diagram represents the high-level behavior of the system. Whereas, the sequence diagram focus on the message exchanged.

We will stop here and will continue in the next lecture.

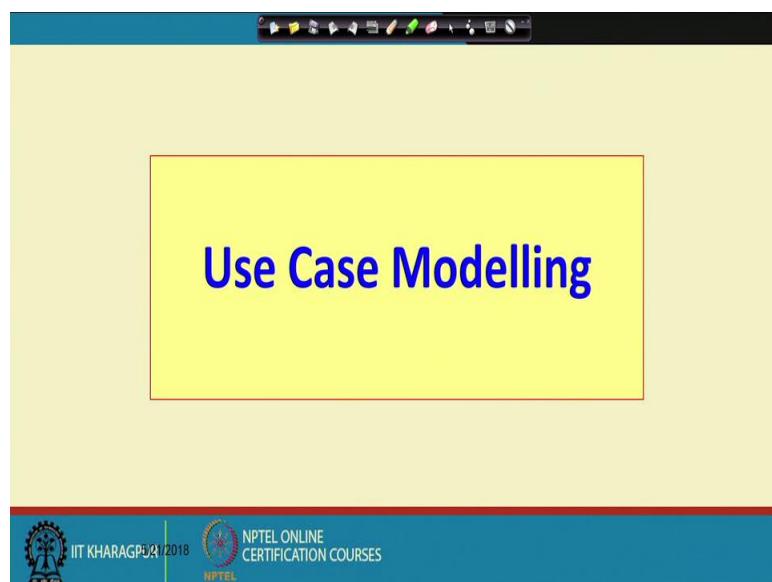
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 31
Use Case Modelling

Welcome to this lecture. In the last lecture we had discussed the introductory topic on object oriented design and we had said that the unified modeling language (UML) has become a de-facto standard for object oriented design. We saw that UML has evolved over the years from the techniques that existed in those days. We also saw that modelling using UML consists creation of many types of diagrams.

The first diagram to be constructed while modelling a problem is the use case diagram. A use case diagram is the central diagram of UML design, because that portrays the customer's view of the system. Therefore, the use case model needs to be constructed first and all other models that are constructed subsequently must conform to the use case model. Let's first discuss how to construct the use case model for a given problem

(Refer Slide Time: 01:38)



and then we will discuss the other models that need to be developed as a part of software development exercise. Let's first look at the use case modelling.

(Refer Slide Time: 02:01)

The slide is titled "Use Case Model". It features a central diagram enclosed in a red circle, divided into four quadrants by a horizontal and vertical line. The top-left quadrant is labeled "Structural View" with "Class Diagram" and "Object Diagram" listed below it. The top-right quadrant is labeled "Behavioural View" with "Sequence Diagram", "Collaboration Diagram", "State-chart Diagram", and "Activity Diagram" listed below it. The bottom-left quadrant is labeled "Implementation View" with "Component Diagram" listed below it. The bottom-right quadrant is labeled "Environmental View" with "Deployment Diagram" listed below it. In the center of the circle is a yellow oval labeled "User's View" with "Use Case Diagram" listed below it.

- Consists of a set of “use cases”
- It is the central model:
 - Other models must conform to this model
 - Not really an object-oriented model, it is a functional model of a system

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The use case model captures the user's view of the system and it consists a set of use cases. The use cases are something similar to a high level requirement, but here we have a graphical formalism to model a use case. We also have a text description for each use case that gives the specifics or the details of what happens during execution of the use case.

There are various models that need to be developed as part of a software development exercise: the structural view in terms of class diagram, the behavioral view in terms of sequence diagram etc. We have drawn the user's view at the centre to imply that this is the most fundamental view of the system. It captures the requirements of the system as the user views the system and that is basically the requirements of the system.

Therefore, for every development, the use case model is the one to be constructed first. Once the use case model is developed, the other models can be constructed and they must conform to the use case model. Another thing we need to point out that even though this is part of an object oriented development or an object oriented design technique, the users view is not really object oriented as it captures the various functionalities that the system performs. So, we can say that all other models are object oriented model in the true sense, whereas the use case model is not an object oriented model as actually we do not show the objects and so on. It is in fact, a functional model, where we model the functions those are to be performed using the system.

(Refer Slide Time: 04:30)

A Use Case

- **A case of use:** A way in which a system can be used by the users to achieve specific goals
- Corresponds to a high-level requirement.
- Defines external behavior without revealing internal structure of system
- **Set of related scenarios tied together by a common theme**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, what exactly a use case is? A use case is similar to a high level requirement; the way it is defined here is a way in which the system can be used by users to achieve some specific goals. That basically is a high level requirement; for example a library software, which has various categories of user i.e, the members of the library, the librarian, the accounts department and so on.

The users of the library i.e., the members who use the software to issue book, return book, renew book, search for book etc. So, we can say that those are the use cases for which the user is the member. As far as the librarian is concerned, the librarian also uses the library software, but he does the member record creation, member record deletion, book record creation, book record deletion, checking the availability of a book etc. Those are the use cases or the high level requirements from the perspective of the librarian. Similarly, for the account, checking the total investment in the library, the total fines collected over a period and so on are the use cases.

So, the use case is similar to a high level requirement, but here we represent it differently. We have a graphical representation of a high level requirement called as a use case which also have a text description. As any functional requirement details only the functional behavior without revealing the internal structure of the system. As we will see that each use case has a set of scenarios. We will also see how to identify and how to document those scenarios as we proceed in this lecture.

(Refer Slide Time: 07:22)

–Use cases for a Library information system

- issue-book
- query-book
- return-book
- create-member
- add-book, etc.

Example Use Cases

Diagram: A hand-drawn sketch showing three user icons (Account, Member, Librarian) interacting with a central 'Book' icon. The 'Member' icon is shown with a checkmark next to it. The 'Librarian' icon is also present.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

As we were just mentioning that for any software, we can ask the question that how do the users use it. Each software has various categories of users. We just took the example of a library information system and we said that the users are the members, the librarian and also the account. As far as the members are concerned, they issue book, query book, return book, renew book and so on. As far as the librarian is concerned, the use cases are create member, add book etc.

Given any software, we can consider it to be a black box and we can think of it as it is offering various functionalities to the different categories of user. Some functionalities are being offered by the software to a particular group of users, whereas the other functionalities are being offered to some other group of users. We can represent the specific users as per the functionalities they are assigned to. This becomes the graphical representation of the use cases, for this example we will write issue book, query book, return book, etc. as the functionalities for the member and for the librarian create book, create member, add book, delete member record etc.

So, we get a very impressive view of a system, where just by looking at the diagram we can relate that what does the system have to offer to different categories of users, who are the users and what are the different categories of functions that they can perform using the system.

(Refer Slide Time: 10:20)

The slide has a yellow header bar with a toolbar icon at the top. The main title is 'Are All Use Cases Independent?'. Below it is a bulleted list:

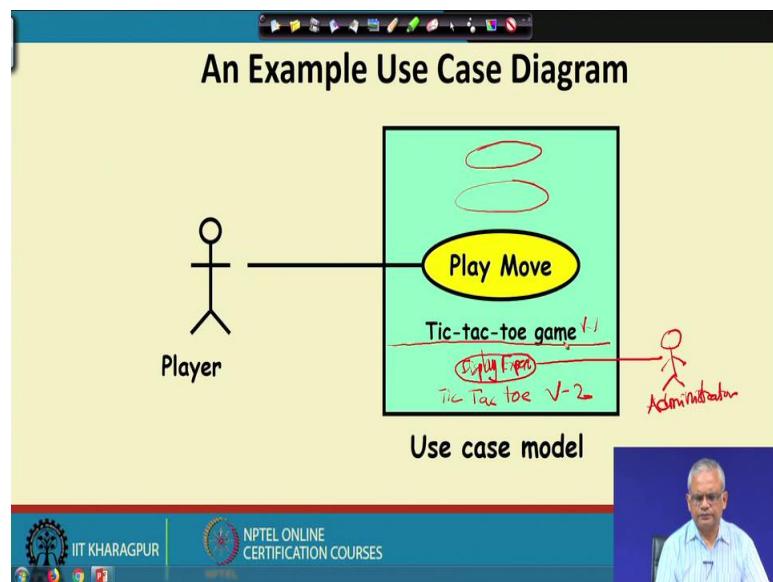
- Use cases appear independent of each other
- However, Implicit dependencies may exist
- **Example:** In Library Automation System, renew-book and reserve-book are independent use cases.
 - But in actual implementation of renew-book--- **A check is made to see if any book has been reserved using reserve-book**

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

But, one question is that, if we identify the different use cases, can we say that all use cases are independent of each other? For example, let's consider the library information system, you can issue book, return book, renew book etc. from the library member's perspective, but are these different use cases really independent? As we drew them as independent use cases, they have no dependencies that's what we drew them separately so, no dependency between them. To answer this question is that, as far as a first level view of the use cases are considered, they are all independent, but if we break them down, there may be dependencies. For example, issuing a book by a member may be dependent on another member, whether he has reserved. So, in a member reservation use case, there will be a dependency of the issue use case with the reservation use case, because if the same book is reserved by another member then it cannot be issued out. So, if we look at the details of various use cases, we might see that there are dependencies, but we do not show those dependencies here in the use case diagram. It is good to remember that there can exist dependencies between different use cases.

We just saw about renew book and reserve book this is a another example actually, that if some member wants to renew a book, then he goes and presents the book, but then in between if another member has reserved that book then he cannot renew it. So, the success of the renew book depends on whether another member has reserved the same book. So, there is a dependency between the reserve book and the renew book as renew book is dependent under reserve book.

(Refer Slide Time: 12:59)



This is the very simple and elegant model here for the use case model. Every use case is represented by an ellipse with the name of the use case in it. Typically, the name of the use case is a verb, because each use case depends on functionality or action being performed. Label the name of the system and then write the use cases and the different categories of users. For example, for a library system there may be many members, but we just draw one stick icon for members and label it as member that denotes the member category of users there is a line connecting from the user to the use case indicating that they are associated or the user can invoke the use case. There is the system boundary which indicates that, the functionalities that are available within this system boundary. If a system has multiple releases, then we can draw different boundaries for different releases. The administrator can display all the expert category of players and this will be done in the version 2. So, this boundary specifies all the functionalities in version 2. There can be other functionalities, the functionalities that will be done in version 1 and the functionality that will be done in version 2, but in any case the boundary can be drawn to indicate even if there are no versions to indicate what are the functionalities that are available. But, sometimes the boundaries are also not drawn as you can see in different books, literature, tools and so on. Sometimes the boundary is not drawn, only the use case and the actors are drawn. So, the boundary is not really a mandatory requirement for modelling a use case, it's the ellipse indicating the use case, the association relation and the user the boundary is optional, but it helps to show clearly

what are the functionalities available and we normally write the name of the software.

(Refer Slide Time: 16:55)

Why Develop A Use Case Diagram?

- Serves as requirements specification
- How are actor identification useful in software development?
 - Identifies different categories of users:
 - Helps in implementing appropriate interfaces for each category of users.
 - Helps in preparing appropriate documents (e.g. users' manual).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Before we look at the nitty-gritty of the use case diagram, let's see what are the uses of the use case diagram. We can easily guess that it serves as the requirement specification in a very elegant way. It first gives a graphical view of all the requirements and then it is also accompanied by a text description which gives the detailed behavior associated with the requirement i.e., what the system does, given an input how the system behaves or what the output are that it produces.

But, that is understandable that we identify the requirements which are the ellipses, but how to identify the different categories of users? That is the different categories of users are called as actors in the UML terminology. How does identifying the actors help in developing the software? Does it have any rule? Yes, it has rule. When we identify different categories of users, we can imagine the different user interface that each category needs, let us say software has factory workers as a category of user and the system administrator is another category of user. The factory worker need a very simple interface, because they are not very familiar with the software, very elaborate user interface where they are prompted to enter some data they enter it and so on. Whereas, the system administrator does not need a very elaborate interface. He needs an interface which would work very fast so that he can just press some keys and get things done very

fast, whereas for the factory workers there will be prompt for everything they need to do and that has to be in very simple format.

The second use of identifying the actors is that it also helps in preparing the user manual. If you can identify who are the users, the user manual can be developed by targeting those users. For example, if factory workers are the user of the software, then the user manual has to be written in their language in a very low level description, but if the software will be used by system administrators and computer experts, then the user manual can assume certain background knowledge and it can be presented at that level. So, identifying the category of users is helpful as the development proceeds. First is in developing the GUI part and second is preparing the documents specially the user's manual.

(Refer Slide Time: 20:33)

Representation of Use Cases

- Represented in a use case diagram
- A use case is represented by an ellipse
- System boundary is represented by a rectangle
- Users are represented by stick person icons (actor)
- Communication relationship between actor and use case by a line
- External system by adding a stereotype

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The representation of the use cases we had seen is drawn by an ellipse, the system boundary is drawn by a rectangle, the users are called as actor, they are represented by stick person. These are the terminologies we must be familiar while using the UML the actor, the use case and the association relation between the actor and use case. This is also called as a communication relation or association relation so, drawn by a line.

Sometimes we need to represent an external system, because one system might interact with another system. So, we can consider the other system to be a kind of a user here and

we represent the external system, using the same stick person icon and we just annotate that which we call as a stereotype saying that it is an external system.

For example, let us say in the game, backup need to be taken for every game where the backup is made on an external system. So, we represent the external system using a stick icon, but we just write here within “<< >>” symbols. These are called as the guillemet and the text within the guillemet is called as the stereotype. The external system is actually a system not a human user, but for simplicity we represent it as a stick icon and we write that, but the backup the external system is involved and this backup is taken under external system.

(Refer Slide Time: 22:57)

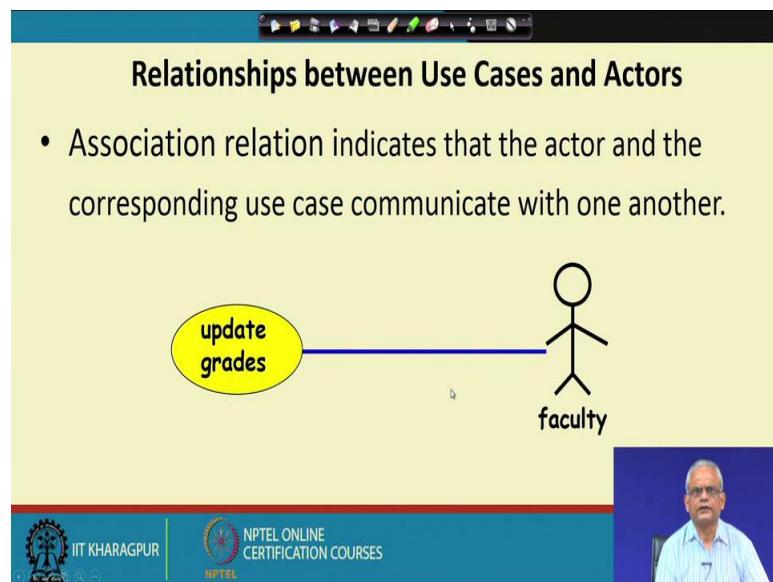
What is a Connection?

- A connection is an association between an actor and a use case.
- Depicts a usage relationship
- Connection does not indicate data flow

The diagram shows a stick figure icon representing an actor connected by a red line to a rectangular box representing a use case. Inside the box, there is an oval containing the text "Play Move". Below the box, the text "Tic-tac-toe game" is written. The slide has a blue header bar with various icons and a blue footer bar featuring the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video thumbnail of a person speaking.

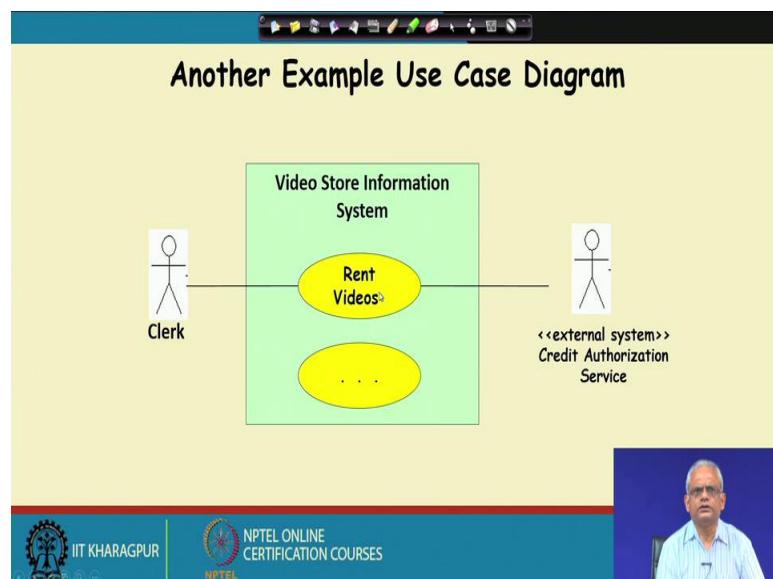
Let us see what the connection means. This is the line between the actor or the user and the use case. The line indicates an association relation or a communication relation. The user invokes the use case or the user uses the use case, but then it does not indicate that he will input some data etc., he may just invoke it. We do not capture here data flow unlike in a DFD, where we discussed about external entities and we are interested in what kind of data they input. Here, we do not capture the data flow rather we just indicate here that the user uses the use case.

(Refer Slide Time: 23:54)



This is another example, that the faculty is a category of user who can update the grade.

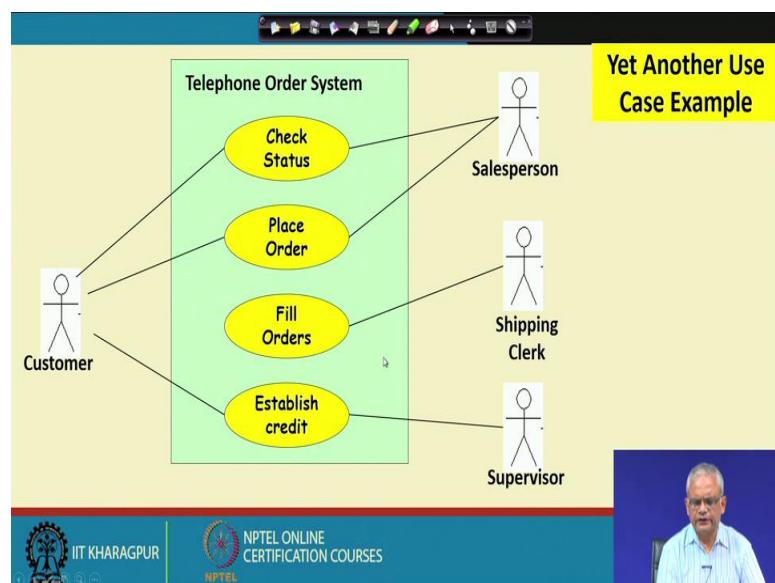
(Refer Slide Time: 24:12)



This is yet another example a video information system. There are many use cases here. We have just represented one of the use cases to explain what is indicated when two actors are connected to the same use case. It indicates that Rent Video use case can be invoked by both these actors or for the success of the use case execution. The two actors are involved. One implication is that if there is another actor here (user) they can independently invoke the use case, but here the implication is that both need to

participate. The clerk when he rents the videos, the help of an external system is taken for credit authorization. So, before a person who wants to rent the videos presents his video VCR, the clerk texts that and enters the details and checks whether the member is creditworthy by using an external system automatically. It is checked as part of this use case and then the rent video will be success, otherwise if he is not creditworthy, then the rent video will say that the video cannot be issued. So, we will often draw multiple actors using the same use case they collaborate for successful execution of the use case.

(Refer Slide Time: 26:12)



This is another example here; this is a telephone order system here. In this commerce application, we have the customer as user who can place order through telephone, can check the status of the order, and also while placing the order he needs to establish the credit possibly by entering the credit card number etc. The details of those what really happens will be given as separate text description. Here, by looking at the diagram we can see that the customer can invoke three functionalities of the telephone order system i.e, check the status, place order and establish the credit. The salesperson can help in placing the order and the salesperson also can check the status. The salesperson can himself or herself place an order, the shipping clerk fills the orders and dispatches, the supervisors he is involved in establishing the credit.

(Refer Slide Time: 27:41)

The slide has a yellow background. At the top right, the title 'Factoring Use Cases' is displayed. Below the title is a bullet point: '• Two main reasons for factoring:'. At the bottom of the slide, there is a dark blue footer bar. On the left side of the footer, there is a logo for IIT Kharagpur. To its right, the text 'IIT KHARAGPUR' is written. Next to it is a logo for NPTEL, which consists of a circular emblem with the letters 'NPTEL' below it. To the right of the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES' is written.

So, far we have looked at some very basic concepts in use case modelling, looked at the graphical representation of a use case diagram.

Right now, we are nearly at the end of this lecture we will stop here and in the next lecture we will discuss about some details of use case modelling namely; how to decompose a use case sometimes the use cases are complex and we need to represent them as separate use cases by decomposing them, we will see how to decompose, what the different techniques are and we will also see the text description how to write the text description for a use case. We will stop now.

Thank you.

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

Lecture - 32
Factoring Use Cases

In the last lecture, we had looked at use case modeling and said that that's the very basic fundamental model which is constructed first in any development and it captures the user's view. We saw that it was really the high level requirements that are called as the use cases and then we had a graphical modeling of the use cases and that would be accompanied by text description. We had discussed the graphical model of use case where we studied what the different elements indicate and given a use case model what we interpret with the system somebody showed us. Can you understand what does this use case model mean?

In this lecture we will see in more details of the use case modeling namely, given a high level requirement or use case is very complex or some are very simple. So, can we decompose this complex use cases and represent them as sub use cases? We will see how to write text description and finally we will see that how we identify the use cases for a given problem, and how to develop this use case model, because that is the ultimate objective i.e, given a problem we should be able to draw it is use case model.

Now, let us see how we factor a use case. First we understand why we need to factor. There are two main reasons why we need to factor a use case.

(Refer Slide Time: 02:21)

- Two main reasons for factoring:
 - Complex use cases need to be factored into simpler use cases
 - Helps represent common behavior across different use cases
- Three ways of factoring:
 - Generalization
 - Include
 - Extend

The first is as we proceed with design activities, we will see that unless we factor the use cases and make it into simpler use cases, the design process becomes extremely complicated. We will develop some diagrams like the sequence diagram and so on which we will develop based on the use case diagram. Unless the use cases are simple, the sequence diagrams etc. will become extremely complicated and at that time we would have to again come back and relook at our use case model and simplify. So, it is a good idea that while developing the use case model, we do not represent complex use cases rather we try to factor them or decompose them into simpler use cases.

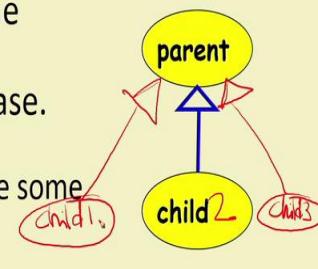
The second reason why we need to factor is that some aspect of one use case may be common to another use case. If you do not identify them in the use case model, latter there may be a duplication of the design and code. So, if there are some part of the functionality that are similar across two or more use cases, we need to identify those similar functionalities and represent them in the use case model so that we develop them only once. It helps us to reduce the design and coding effort and identify what are the common functionalities across the different use cases.

So, these are the two main reasons why we need to factor use cases. The next question comes how do we factor? We will use three techniques for factoring: one is called as generalization-specialization. The second technique is called as include and the third technique is called as extend. Let us see these three techniques for factoring a use case.

(Refer Slide Time: 04:52)

Generalization

- The child use case inherits the behavior of the parent use case.
 - The child may add to or override some of the behavior of its parent.



A UML generalization diagram showing a yellow oval labeled 'parent' at the top. Below it are three smaller ovals labeled 'child1', 'child2', and 'child3'. A blue triangle pointing upwards from 'parent' to 'child2' represents inheritance. Red arrows point from 'parent' to each of the three children, and red curved arrows point from each child back to the parent, representing overridden behavior.

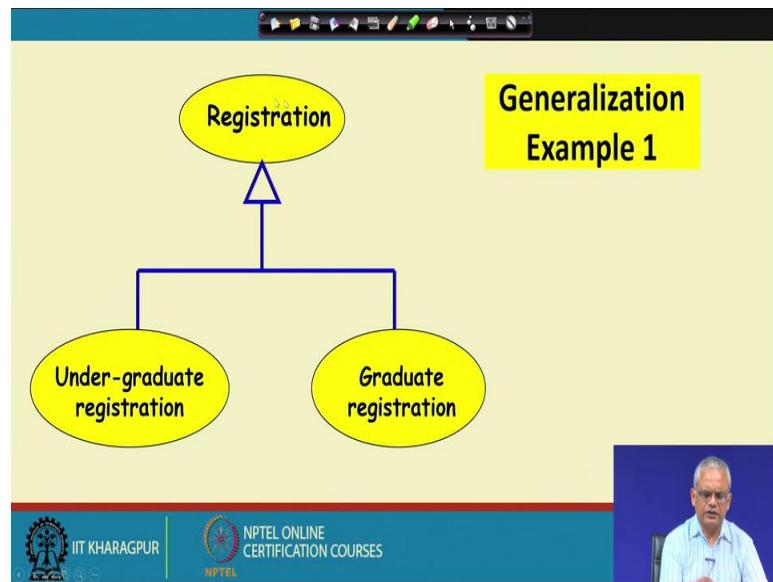
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



First let us look at the generalization here. There are some functionality, which are common to many other use cases, but there are some minor differences. We show the common functionalities as the child use case or the base use case. So, we can have different child use cases. The parent is a common functionality. Child 1, child 2, child 3 have parent common functionality, but something extra.

This helps because, we do the parent use case, design code etc. and then we use that by slightly extending that and adding some more functionality for different child use cases. If we do not do this, then we write the three parent use cases separately then we keep developing each of this which leads a duplication of effort and also these may become large. Here, once we develop the parent use case, there are only small part in the child that we need to develop.

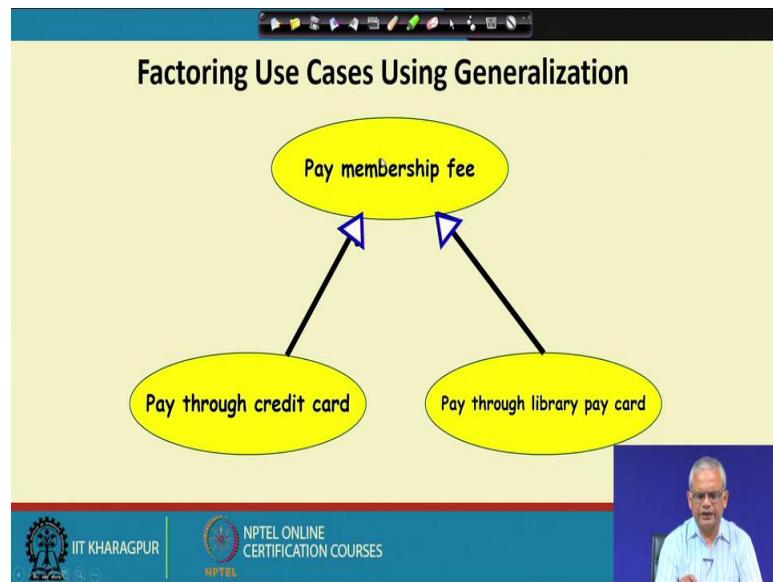
(Refer Slide Time: 06:52)



One example of generalization: let's say student registration system, before the semester starts the students need to register and we have two types of use cases available, one is called as undergraduate registration and the other is called graduate registration. In the undergraduate registration, let's say the fee needs to be paid before the registration can start whereas in the graduate registration, let's say there is an option that the fee can be adjusted from the scholarship, because all graduated students get some scholarship in this specific institute for which we are developing this diagram. Let's assume that registration is more or less same they need to enter the semester, the hostel accommodation etc., but there is a minor difference between undergraduate registration graduate registration. In undergraduate registration the registration, we won't be complete until the fee is paid whereas, in a graduate registration a question will be asked whether you will pay the fee now or you would like the fee to be adjusted from scholarship.

So, we do the registration part which is common to both of this, and then the fee payment and the question part that whether it will be adjusted or fee payment. A small change in behavior will be implemented in these two use cases and the registration use case having the basic functionalities of the registration will be implemented.

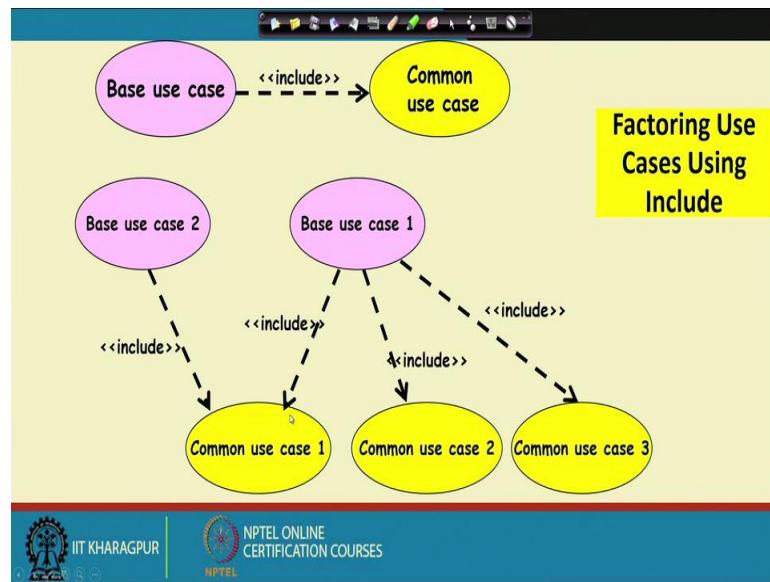
(Refer Slide Time: 08:46)



There is another example here: let's say a library software, where we have a pay membership fee use case. Actually there are two use cases here supported by the software; one is called as pay membership fee through credit card and pay membership through library pay card. The library issues pay cards where the members can deposit money in the pay card or they can pay through their credit card.

Now, most of the functionality are same for pay membership, they would have to write their membership number, see that there are no outstanding books and there are no complaints from the librarian and so on. So, this is the part that takes care of those functionalities and these two small differences between the pay through credit card and pay through library pay card are taken care through the generalization relationship, the generalization is a field is triangle shape and a solid line. So, these are two use cases, but they have common behavior there and we call this as a child use case, which are derived from the parent use case.

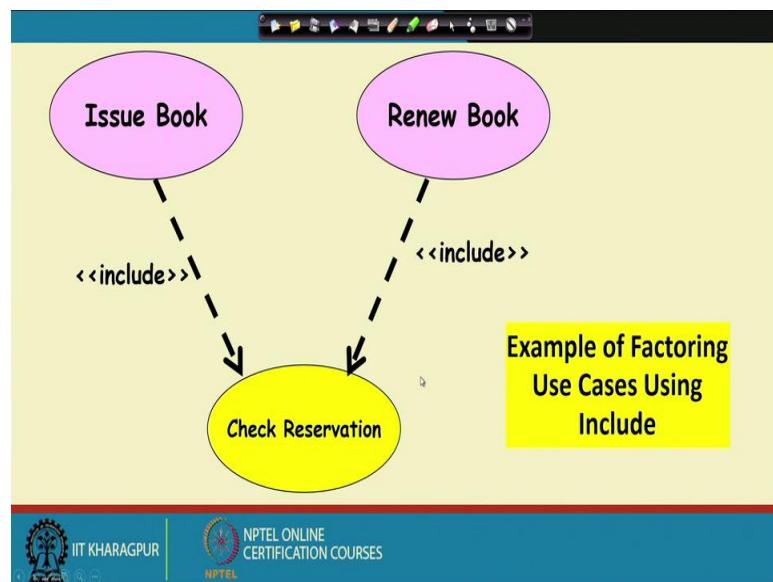
(Refer Slide Time: 10:29)



The second way to decompose a use case is by using the include relation. Here the base use case includes some use cases which are common across different use cases. The representation is a dotted arrow with stereotype “include”.

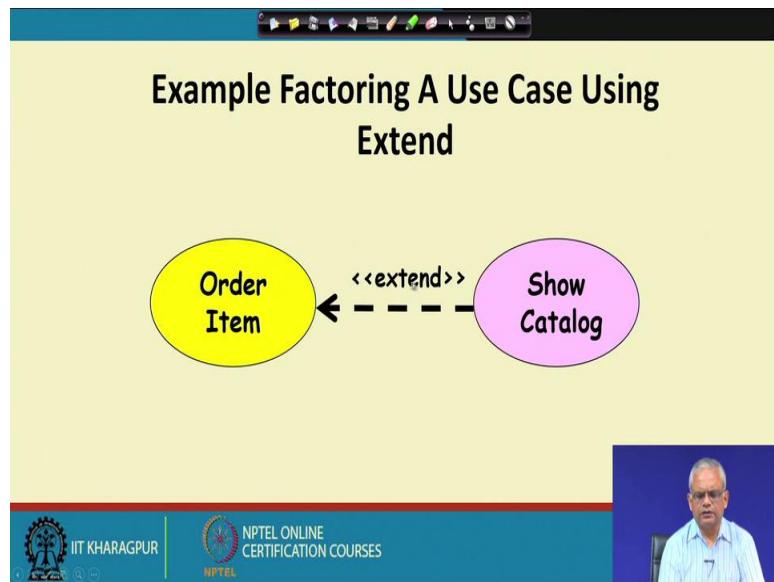
In case of a generalization specialization there are many aspects which are common in the base use case and then these are extended in minor ways in the child use case. There are small parts which are common across the base use cases and we identify that is a common use case and write there include. So, it means that there are some small part of functionalities between the base use case 1 and base use case 2 and we can develop common use case 1 once and then base use case 1 and base use case 2 will invoke the common use case 1. So, include is, it compulsorily includes common use case 1 behavior as well. If it is a large use case, it also helps us to split it into simpler use cases. We develop this separately and therefore, the base use case becomes simple.

(Refer Slide Time: 12:28)



This is an example of factoring using the include relation between use cases. Let's say we have two use cases by the library information system which is issue book and renew book, but before a book can be issued, the functionality requires that we need to check if somebody has not reserved that book. Similarly, before a book can be renewed needs to be checked whether somebody has reserved that book. We can just write this representation, it means that both these use cases require checking the reservation and this said check reservation functionality will be developed once we will have a separate sequence diagram code for check reservation and that will be invoked by both the issue book and renew book. So, it gets developed only once, it also helps to simplify these two use cases: the issue book and renew book.

(Refer Slide Time: 13:49)



This is the third way to factor a use case called as the extend relationship between two use cases. Here, Order item is the base use case and it is extended by the show catalog. Let us say we are ordering an item on a e-commerce portal. When we order the item if we are sure about the item that we are ordering, we just go on entering the details of the item, but sometimes we do not know the item, we want to see the details of the item, what is the item code, check whether warranty is given for how long and so on. Before, we can complete the order, we as part of the order item may request a show catalog and in the catalog we can check what are the different types of products that are available which you can order, what are the specification warranty and so on.

If we are receiver of the product, we know all the details and do not invoke the show catalog rather just keep on ordering. So, this is optional if you think of it. If there is extend relationship, then order item can execute as it is i.e, some user may execute the order item as it is, but some users may take help of request catalogue use case as part of execution of the order item, they will request show catalog.

So, the behavior of the order item is optionally extended by the show catalog, but just note the direction of the arrow, when it was include it was compulsorily includes the other functionalities. But here, it is optional, the stereotype is extend and the direction of the arrow opposite to that of include.

(Refer Slide Time: 16:17)

Extension Point

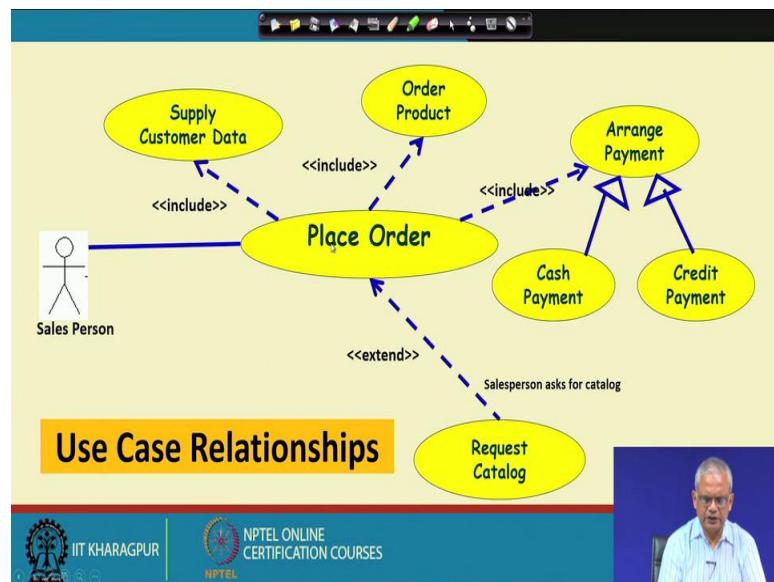
- The base use case may include/extend other use cases:
 - At certain points, called extension points.
- Note the direction of the arrow

The diagram shows a green rounded rectangle representing a use case boundary. Inside, there is a yellow oval labeled "Perform Sale After checkout". Outside the boundary, to the right, is another yellow oval labeled "Gift wrap Product". A blue dotted arrow points from the "Gift wrap Product" oval to the "Perform Sale After checkout" oval. Above the arrow, the text "<>" is written, and below it is the text "Product is a gift".

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

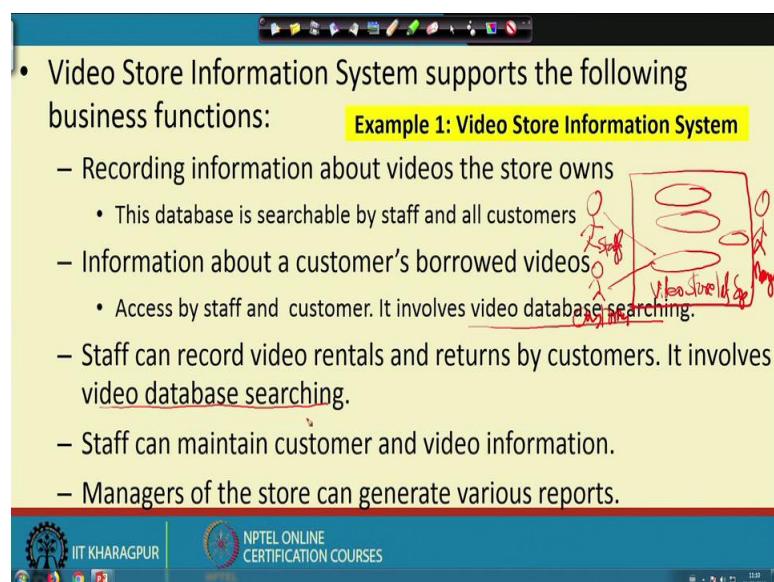
UML also provides the facility to define an extension point, it helps sometimes to say that at what point in the execution of the order or perform sale, an extend use case can be executed. Let's say during a sale somebody can request that the product is gift wrapped. So, by this notation we can say that after checkout, there will be an option that whether to gift wrap the product or not. So, somebody can invoke the gift wrapped product after checkout. So, that's what it means. This is called as the extension point at which this functionality can be invoked, but the extension point is optional i.e., we might not show in our design, in our model at what point the product the gift wrap product will be invoked.

(Refer Slide Time: 17:32)



Given a use case model let's try to understand, that what it implies. Here the sales person can invoke the place order use case. The place order use case, while execution includes the supply customer data, order product and also arrange payment. The arrange payment are actually two types; one is cash payment and other is credit payment. And also while placing the order, the sales person can invoke request catalog. So, this is extend and the direction of the arrow here it is in this direction towards the base use case, but for include, the directions are opposite.

(Refer Slide Time: 18:36)



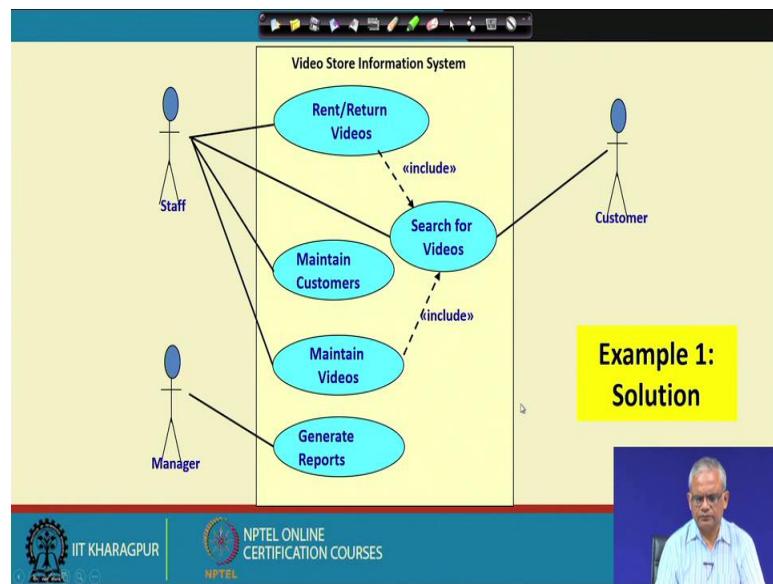
Now, let's see few examples how to develop a use case diagram. Let's take an example of a video store information system. It is given to us that it has the following functionalities. The first functionality is that all the videos that the store owns needs to be recorded and once these are recorded, this can be searched by staff and also the customer. So, the available videos that the store owns can be searched by both staff and customers and the information about a customer's borrowed videos can be accessed by staff and customer and also it involves the video database search where the customers borrowed information is recorded. The third functionality is that the staff can record the video rentals and also the return of rented video by the customers.

Again this involves database searching video database searching and the staff can maintain the customer records, they can add customer, delete customer and also they can maintain about the video information that is they should be able to record the information about the video, they can enter the data about the video and also if some video is lost, damaged etc., they should be able to delete it. The manager of the store can generate various types of reports. Now, how do we go about developing the use case model for this software? One is that we identify what are the functionalities and who are the users. We can draw the system boundary and then we represent the users and then find out what the functionalities they can represent, they can invoke and represent those functionalities. As you can see here, we draw the boundary, write video store information system and then draw the stick icon corresponding to different types of users. So, who are the users here? One is about staff and the customer. The staff and customer can query information about a customer's borrowed videos. The staff can record video rentals and also returns by customer and as part of this, it invokes video database searching. The staff maintain the customer and video information the manager can generate various reports manager of the store can generate various reports.

So, we can see here clearly that there are 3 categories of users; the first category of user is the staff, the second category of the user is the customer and the third category of the user is the manager. After having done that we can find out what are the functionalities invoked by each of this. For example, the staff we can find out that the staff can maintain customer and video information. So, you can write here maintain customer information, maintain video information, the staff can record video rentals and returns record rentals and returns and the customer can query about the borrowed videos. So, the customer

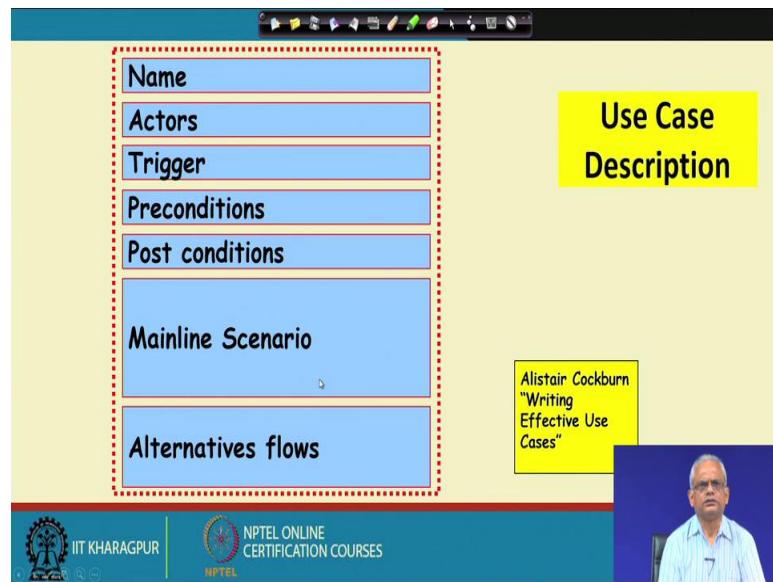
write query about borrowed videos, so, is the staff, but one thing is that all this involved video database searching, that the staff rental recording, the rental and return by the customer involves video database searching. So, this appears like a sub function which is included across different use cases. Here the video database searching is also required as part of the query information about the borrowed videos. We approximately know how to go about drawing the diagram.

(Refer Slide Time: 24:35)



So, the customer, the staff and the manager are the three categories of users, the staffs interact with the rent and return videos use case, they maintain customer and maintain video records and the manager can only generate reports, the customer can search for videos. Observe here that the maintain video, rent and return video they also use the search for videos.

(Refer Slide Time: 25:21)



Developing the graphical model of a use case is the first step, but then that does not give us lot of details, it just tells us what the requirements are, what the use cases are and so on.

But, we need to also document the use cases. There is no standard format that is prescribed by the UML, but then there are some way to document the use cases, which are considered as good practice. This is the one recommended by Alistair Cockburn in his book writing effective use cases, but there is no hard and fast rule that we need to document in this way.

We need to write the name of the use case. So, this is the text description that should accompany every use case diagram. Just giving the graphical use case model is not enough, that does not portray or give lot of information. The detailed information are given by a text description. Here, we need to write the name of the use case, who are the actors of the use case, these are also present in the graphical model. We need to write what are the triggers like what the different users, external systems need to invoke the use cases, the precondition and post condition etc.

What should be satisfied before a use case can be executed and what must hold true after a use case has completed. We will look at the scenarios in a use case and we will represent what is called as a mainline scenario and the alternate scenarios.

(Refer Slide Time: 27:18)

The screenshot shows a presentation slide titled "ATM Money Withdraw Example". The slide content is as follows:

- **Actors:** Customer
- **Pre Condition:**
 - ATM must be in a state ready to accept transactions
 - ATM must have at least some cash it can dispense
 - ATM must have enough paper to print a receipt
- **Post Condition:**
 - The current amount of cash in the user account is the amount before withdraw minus withdraw amount
 - A receipt was printed on the withdraw amount

The slide footer includes logos for IIT Kharagpur and NPTEL, and a small video window showing a speaker.

We will take some sample problems, we will try to document in this manner, this is a good way to document use case, but it's not compulsory that we need to document always like this. We can deviate it from little bit the companies they have their internal standards, where they might enforce the same one or they might have small deviations from this, but this is more or less the way the use case documentation is done..

We are almost at the end of this lecture in the next lecture we will see how to document a use case. We will take some examples and see how the documentation of the use cases can be developed.

Thank you.

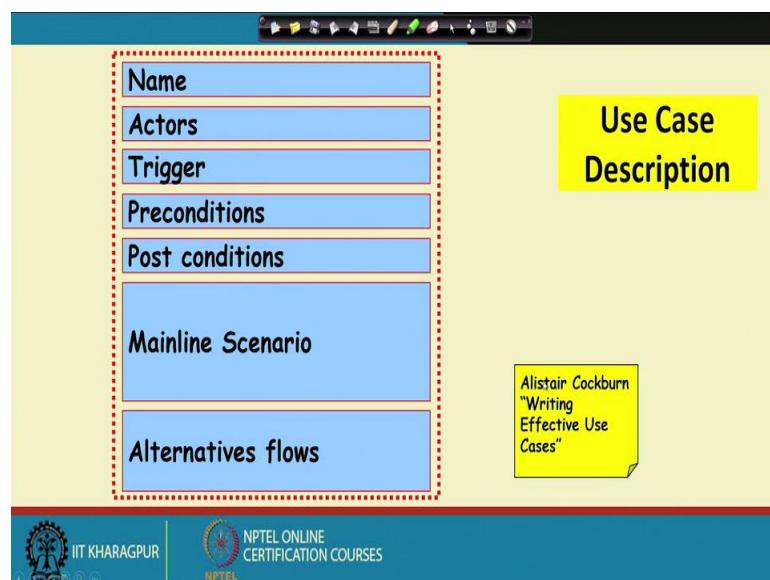
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 33
Overview of Class Diagram

Welcome to this lecture. In the last lecture, we had seen how to develop Use Case models, the graphical model and also how to decompose the complex use cases using the 3 relations: generalization, specialisation and include and extend. In this lecture, let us see how to document a use case, because just the graphical model by itself does not convey the complete picture. It just tells about what are the use cases and if they are decomposed into some use cases who are the actors and so on.

But the details of the use cases are not given and that is the reason why in a use case model not only the graphical use case model should be given, but also the text description. As far as the text description is concerned, UML does not impose any standard way of documenting the use cases. But, some good practices have come up and those are normally followed even though there is no hard and fast requirement to follow the same documentation technique. Let's see how to document use case.

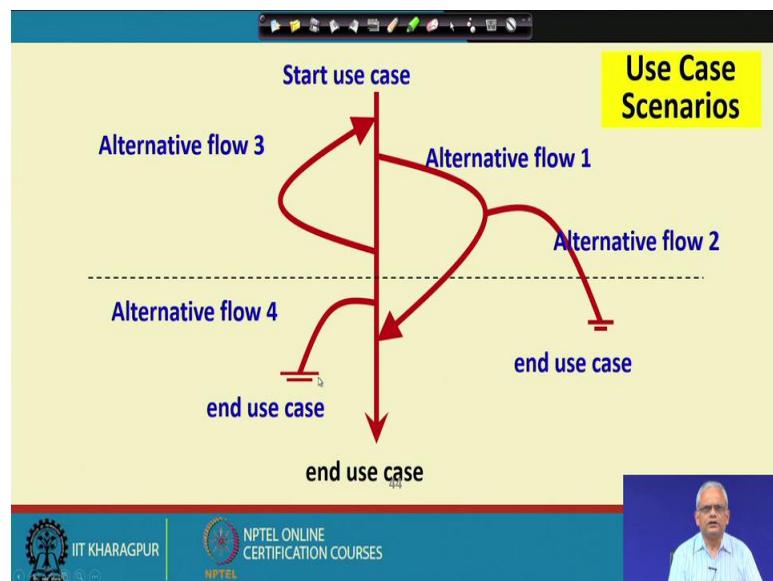
(Refer Slide Time: 01:44)



The documentation that is recommended by Alistair Cockburn in his "Writing Effective Use Cases" book says that every Use Case we must have the following documentation to

accompany: the name of the use case, the actors that would use the use case, the trigger when the use case will start to execute, the preconditions that is what must hold before the use case can be executed, the post condition that is after the use case completes what conditions must hold, the mainline scenario i.e, the typical interaction sequence between the computer and the user and the alternative flows.

(Refer Slide Time: 02:48)



Will see what the different scenarios are and what the mainline scenario, the alternative flow are and so on. In every Use Case there is a normal interaction between the computer and the user and sometimes there are alternative flows. Let's take an example to explain this. Let's take the case of a bank ATM. The user goes to the bank, inserts the ATM card and then the system prompts for the password or the pin code; the user enters the pin code; the computer prompts "do you want to withdraw from savings account or current account?" User chooses the savings account and then the system asks for the amount; user enters the amount, let say 2000 rupees and then the system asks "do you want a printed receipt" and the user enters yes and then the cash is dispensed and the printout is generated and the use case terminates. So, that is like a mainline scenario, but sometime the user when prompted to enter the amount, the user enters 550 rupees and then, system prompts "please enter in multiple of 100 rupees". So, again the user enters correct amount and so on. Sometimes as user enters 2000 rupees and the system prompt for printout and so on. But then, the system generates a message out of cash and it

terminates. Sometimes, it proceeds and the amount is entered and it says that account does not have sufficient balance and it ends and so on.

So, the mainline scenario is the typical interaction sequence between the user and the computer. The alternate scenarios occurs, but these are not so frequent. The frequent case is the mainline scenario, but then there are other scenarios which we call as Alternate flows or Alternate scenarios may occur. While documenting a use case we need to document not only the mainline scenario, but also the alternate flows. Let's see how we go about documenting that.

(Refer Slide Time: 06:20)

ATM Money Withdraw Example

- **Actors:** Customer
- **Pre Condition:**
 - ATM must be in a state ready to accept transactions
 - ATM must have at least some cash it can dispense
 - ATM must have enough paper to print a receipt
- **Post Condition:**
 - The current amount of cash in the user account is the amount before withdraw minus withdraw amount
 - A receipt was printed on the withdraw amount

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NPTEL

Let's take the ATM money withdraw example. Here, the actor is the customer, the preconditions are that the ATM must be switched on or ready state to accept transaction, ATM must have at least some cash it can dispense; otherwise, it would be displaying out of cash, ATM must have enough paper to print a receipt etc. So, these are the preconditions before the user can start to interact.

The post condition is that the current amount of cash in the user account is the amount before withdraw minus withdraw amount. So, if the use case completes successfully, the post condition says that what must hold true that is the amount in the balance in the customer's account will be the amount before the withdrawal minus the withdrawn amount and receipt was printed on the withdrawal amount.

(Refer Slide Time: 07:37)

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deducts the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

ATM Money Withdraw Mainline Scenario



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

And here, we list the mainline scenario and alternate scenario. The different scenarios basically are interactions between the actor actions and then there is a corresponding system action. The customer inserts the debit card; the system verifies customer Id. The customer chooses withdraw system asks for an operation time, operation type; the customer enters the cash amount.

The system asks for withdraw amount; the system checks if withdraw amount is legal; system dispenses the cash; System deducts the withdraw amount from the account and system prints a receipt. The customer takes the cash and the system ejects the cash and the card. This is the mainline scenario; but then there can be several alternate scenarios.

(Refer Slide Time: 08:40)

The slide has a yellow header bar with the title "ATM Money Withdraw (cont.)". Below it, there are two main sections: "Alternative flow of events:" and "Exceptional flow of events:". Under "Alternative flow of events:", there are three bullet points: "Step 3: Customer authorization failed. Display an error message, cancel the transaction and eject the card.", "Step 8: Customer has insufficient funds in its account. Display an error message, and go to step 6.", and "Step 8: Customer exceeds its legal amount. Display an error message, and go to step 6.". Under "Exceptional flow of events:", there is one bullet point: "Power failure in the process of the transaction before step 9, cancel the transaction and eject the card." At the bottom left, there are logos for IIT Kharagpur and NPTEL, and at the bottom right, there is a video frame showing a man speaking.

Let see, how we document the alternate scenario. In the step 3, the customer authorization failed as the customer has entered a wrong pin code. Then, display an error message; cancel the transaction and eject the card. So, this is one alternate flow.

Another alternate flow can occur at step 8 i.e, the customer has insufficient fund in the account; display an error message and go to step 6. Step 8, customer exceeds the legal amount that can be dispensed per customer may be 50000 is the legal amount; the customer entered 55000; display an error message and go to step 6. There can be exceptional flow of events. For example, while the transaction is going on, power failure occurred, then the transaction is cancelled and the card is ejected.

(Refer Slide Time: 09:41)

Use Case Description: Change Flight

- **Actors:** traveler
- **Preconditions:**
 - Traveler has logged on to the system and selected 'change flight itinerary' option
- **Basic course**
 1. System retrieves traveler's account and flight itinerary from client account database
 2. System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
 3. System asks traveler for new departure and destination information; traveler provides information.
 4. If flights are available then
 - 5. ...
 - 6. System displays transaction summary.
- **Alternative courses**
 4. If no flights are available then ...

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

Let's take another example of a Use Case and how it can be documented. So, this is the change flight in a flight reservation system. The actor is the traveller. The precondition is that the traveller has logged into the system and selected the change flight option. The basic course or the mainline sequence is that the system retrieves the travellers account and the flight itinerary from the client account database.

The system asks to select the segment that wants to change. System asks the traveller for the new departure and destination information and if the flight is available, then do the change and the system displays the transaction summary. This is the mainline sequence and then, there can be alternate sequences if there are no flight available for the segment, then error message is displayed and so on.

(Refer Slide Time: 10:40)

The slide is titled "Guidelines for Effective Use Case Writing". It contains a bulleted list of guidelines and a UML sequence diagram.

- Use simple sentence
- Do not have both system and actor doing something in a single step
 - Bad: "Get the amount from the user and give him the receipt."
- Any step should lead to some tangible progress:
 - Bad: "User clicks a key"

The sequence diagram shows interactions between a System and an Actor:

```
sequenceDiagram
    participant Actor
    participant System
    Actor->>System: Actor asks for money
    System->>Actor: System asks for amount
    Actor->>System: Actor gives the amount
    System-->>Actor: System produce the money
```

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, and a small video player showing a man speaking.

So, we just saw some templates for documenting use case, but UML gives flexibility in not following the exactly the same standard.

Same template, we can tailor it, but then the one that we discussed are considered as good practices and many developers follow that. Let's see some guidelines for writing effective Use Cases. The Use Case is a document to be read by a large number of stakeholders and therefore, it should be very readable. We have to use simple sentences and we must have separately the actor action and the system action documented because as you see the system and the actor actions alternate the system. Initially the actor initiates, the system responds and asks for something the customer enters it. The user enters the system responds and so on.

Therefore, we must write it in the same way what the actor action is and what a corresponding system action is. We should not mix that up that, get the amount from the user and give him the receipt. We should not club both of these rather prompt the user for the amount, the user enters the amount; then the system action is to dispense the account and print the receipt. Also we should not try trivial steps in the use case documentation. For example, the user clicks a key that is not a tangible progress in the use case execution. So, every step that we document for use case must result in some tangible progress in the use case execution.

(Refer Slide Time: 13:03)

Identification of Use Cases

1. Actor-based:

- Identify the actors related to a system or organization.
- For each actor, identify the processes they initiate or participate in.

2. Event-based

- Identify the external events that the system must respond to.
- Relate the events to actors and use cases.

Now, let's see that given a problem description; how do we identify the use cases? A popular way is to find out from the problem description who are the actors and then, for each actor we can find out what the functionalities that the actor invokes are and then against that actor, we had seen the case of the library software. We identified the actor is the member from the problem description, the librarian and the account and then for the member, we can identify the use cases that the member needs to execute that is a query book, issue book, return book. Similarly the librarian create member, delete member, create book, delete book, cheque fine etc. and the accountant check the current balance, enter book procurement price and check the fine collected, membership fee collected and so on. So, this is a popular way to find out what are the, who are the actors and each actor needs to invoke which functionalities.

The second way is an event based, where we identify what the invocations of the system are, what events the system must respond to and then we relate these events, who generate these events and then what are the use cases that execute. For some systems like the embedded controllers and so on. It is easy to identify the events based on the state machine specification of the controller and relate who are the actors, who generate that event and what is the corresponding use case. So, for some systems like embedded controllers and so on, event based may be preferable, but then the actor based one is very popular across all systems.

(Refer Slide Time: 16:04)

The screenshot shows a software window titled "Example 2: Use Case". The main content area contains the following list of requirements:

- At the beginning of each semester,
 - Each professor shall register the courses that he is going to teach.
- A student can select up to four-course offerings.
 - During registration a student can request a course catalogue showing course offerings for the semester.
 - Information about each course such as professor, department and prerequisites would be displayed.
 - The registration system sends information to the billing system, so that the students can be billed for the semester.
- For each semester, there is a period of time during which dropping of courses is permitted.
- Professors must be able to access the system to see which students signed up for each of their course offerings.

At the bottom of the window, there are logos for IIT Kharagpur and NPTEL, along with a video player showing a man speaking.

Now, let us see one example. This is a course registration software; let us see how we go about identifying the actors, the corresponding use cases and then document it. At the beginning of each semester each professors shall register the courses that he is going to teach. So, into the software, each professor will enter the course details that he is going to teach and then, once the professors have entered their courses they will teach, the students can select up to four-course offerings. During the registration, a student can request a catalogue showing various course offerings for that semester. Information about each course, professor, department and the prerequisites would be displayed at part of this catalogue. The registration system sends information to the billing system once the student completes registration. So, that the student can be billed for the semester based on the courses he is taking.

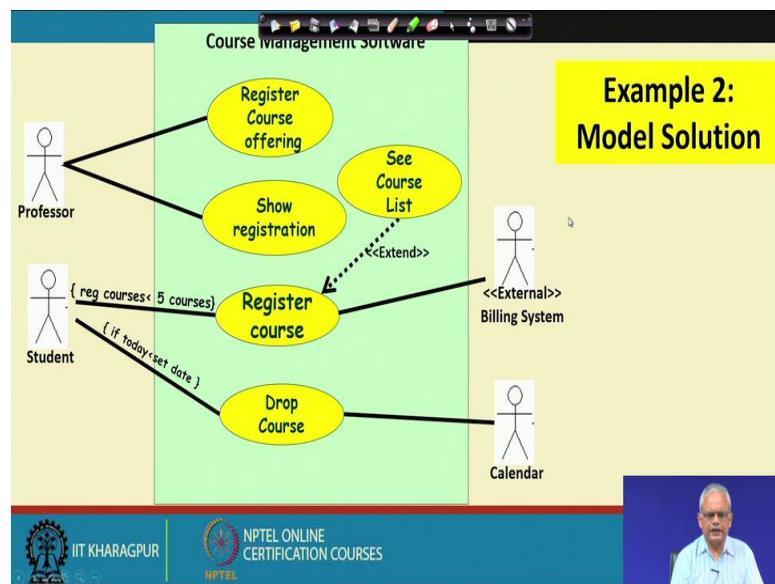
For each semester there is a time period during which dropping of courses is permitted and the professor must be able to access the system to see which students have signed up for their course offerings. How do we go about developing the use case diagram for this problem description? We need to identify the actors, which is very easily identified. Here, it says that the professor shall enter the course details. So, the professor is an actor. The student can select up to four-course offerings. The student is an actor and the registration system sends information to the billing system. The billing system is an external software may be running on a different computer or at least it is different software and this is also an actor. So, we can start identifying what are the use cases that

each of these actors can invoke. The processor can register the course detail; the student can select course offering. During the course offering i.e., during selecting a course he can request for a catalogue. So, this is the optional he may or may not request a catalogue and looks like an extension. So, course catalogue is an extension of the use case which is registered for courses. For each semester there is a period of time during which dropping of courses is permitted. So, that dropping of course, is a use case for which the student is the actor because the student initiates it; but how do we represent that there is a period of time during which dropping of courses is permitted?

When there is element of time here, we need to have an external system which is a calendar or a time that is the typical convention even though it is the system time. But, if we represent a calendar actor or a timer actor, then the design process become simpler. As we proceed will see; but then, at the moment whenever there is a element of time, we will have another actor which is a calendar and this is a use case which the professor will access to find which students have signed up.

Now, let us see this is the overall diagram that we discussed. How do we draw the diagram?

(Refer Slide Time: 20:46)

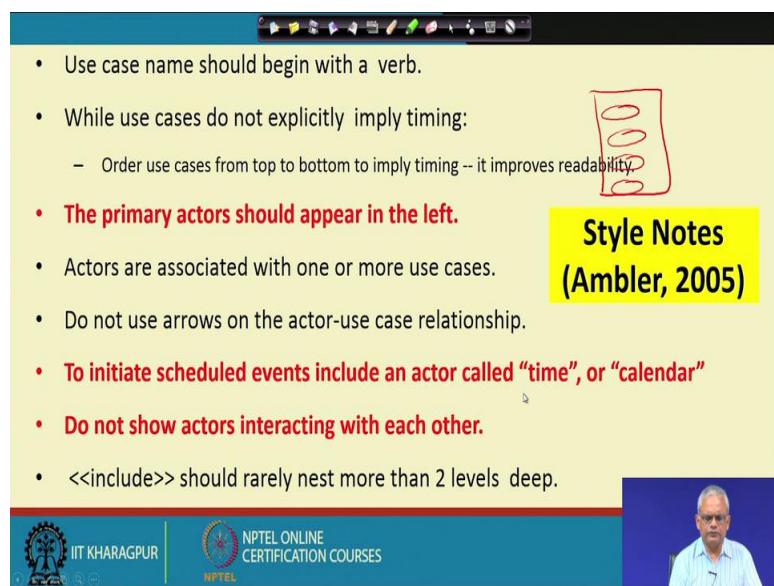


So, this is the diagram, the professor registers the course offering. He can find out how many students have and which students have registered and the student can register for courses which is less than 5 courses. We just write the form of a constraint on the

communication line and during the course registration on successful registration. The billing system is sent information to generate the bill and we write external system, the billing system.

And for dropping courses, there is a time limit and we have the calendar as an external actor and then we have written the constraint i.e., if today is less than set date. So, as long as the deadline is not passed, the student can drop a course.

(Refer Slide Time: 21:50)



Scott Ambler, in his book in 2005, gave some style notes how to document a use case. All use cases must be verbs and even though we can document that in any order. But according to Scott Ambler, as far as possible we must have an ordering of the use cases in the diagrams and that should correspond to the order in which the use cases are invoked that helps in understandability. For example, first is query book, issue book, return book and so on. We must order the use cases from top to bottom to imply timing wherever possible and the primary actor must be on the left. For example, if the student is registering for the course, the student initiates the registration, this is a primary actor; whereas the external billing system is a secondary actor. The primary actor must appear in the left and the secondary actors must appear in the right.

The relation between the actor and the use case drawn by just a line do not use an arrow and whenever there a time we must have a “time” or “calendar” as an external actor. External system which is the actor and if there is an interaction between two customers.

For example, the user gives some information to the clerk and the clerk enters it. We do not have to show that; we need to only represent the information that the actors input to the system. When decomposing the use cases i.e., the complex use cases, we should not decompose into a very deep nested sequence of use cases. The include and extend should be rarely nested to 2 level deep. The use case names should be in the perspective of the users i.e., we should use the users language.

(Refer Slide Time: 24:36)

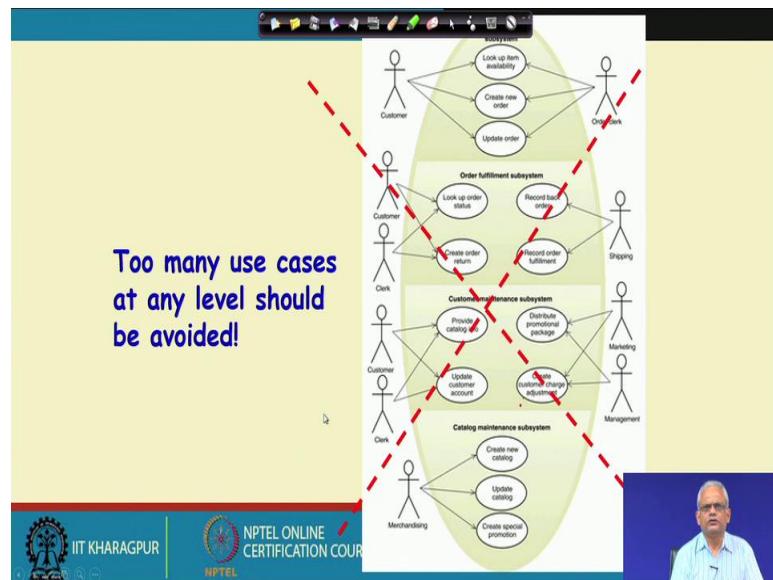
The slide is titled "Effective Use Case Modelling". It contains the following bullet points:

- Use cases should be named and organized from the perspective of the users.
- Use cases should start off simple and at as much high view as possible.
 - Can be refined and detailed further.
- Use case diagrams represent functionality:
 - **Should focus on the "what" and not the "how".**

The slide footer features the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a small video frame showing a man speaking.

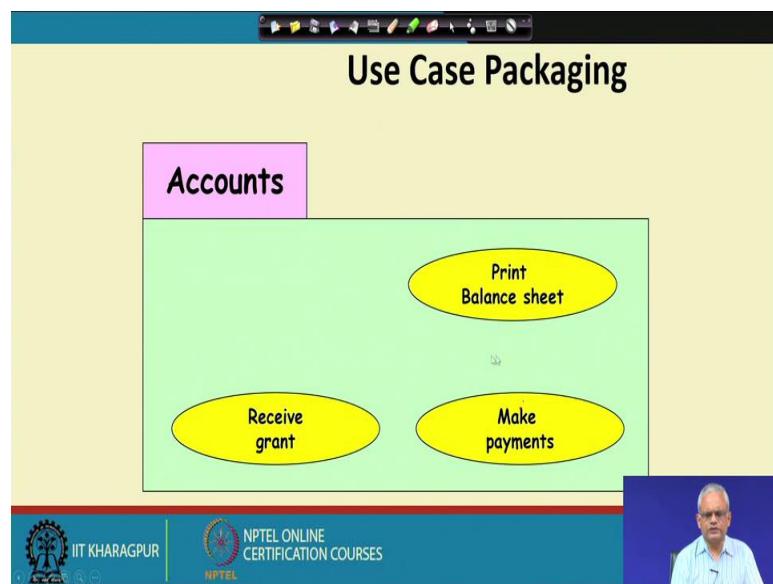
The use cases should be simple to understand and this can be decomposed wherever required and also remember that these are requirements, the use cases are basically functional requirements and just like the requirements they focus on what aspects and not how the design aspect holds for the Use Case diagrams.

(Refer Slide Time: 25:32)



We should not have a Use Case diagram that has too many use cases which becomes very confusing. When we have a situation where we have too many use cases we should use packaging.

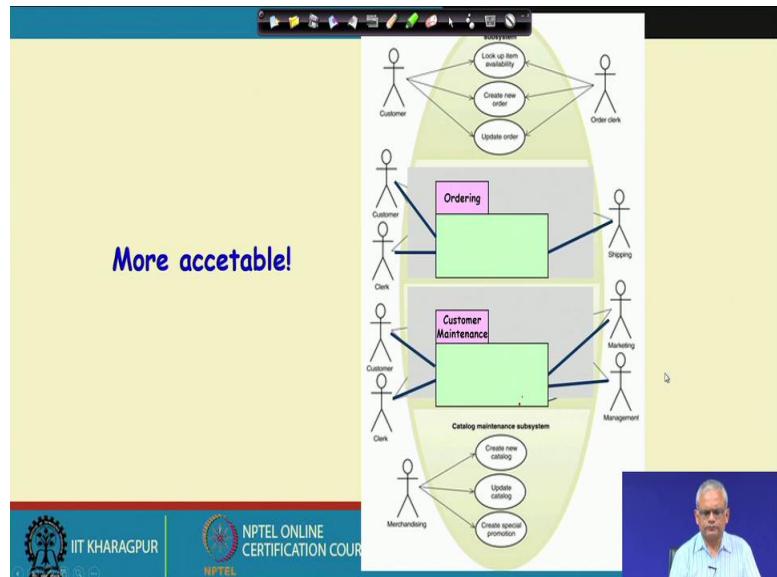
(Refer Slide Time: 25:49)



In Use Case packaging we use this folder symbol which is supported by UML and related use cases we can put inside the package and we write the name of the package i.e., in the diagram, all the use cases pertaining to the accounts actions. For example: printing the balance sheet, making payment, receiving grant etc. These are in the

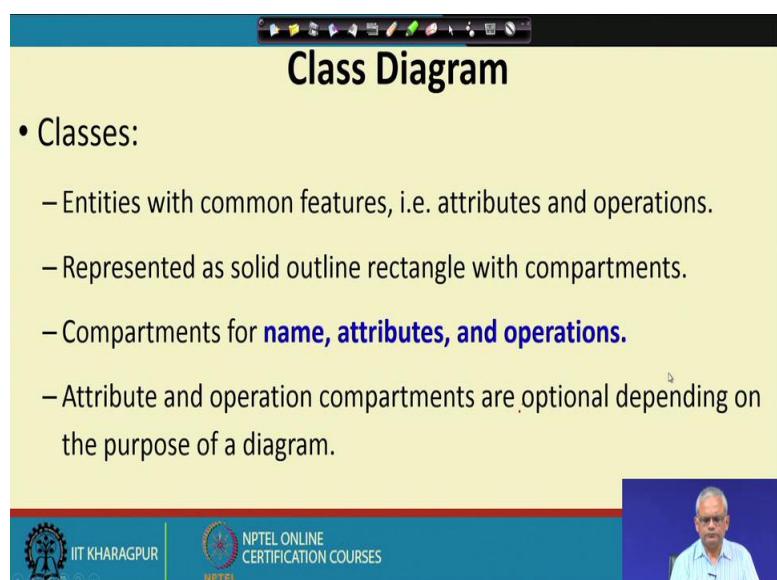
accounts package. We can use the packaging to reduce the number of use cases in a Use Case diagram.

(Refer Slide Time: 26:36)



Just see here in the diagram, we can just have the ordering here. The ordering as one of the package. The ordering related use cases are here. The customer maintenance related use cases are represented here and then, in a separate diagram we can show the use cases in the ordering and that makes the diagram easier to understand.

(Refer Slide Time: 27:03)



So, far we have been looking at the use case diagram that is a crucial diagram; but it is also one of the simplest diagrams to develop. But, it requires little bit of skill with respect to identifying the use cases; with respect to decomposing the use cases the documentation of each use case. By practising some of the assignments that we will give on developing the use case diagram to specific problems because just by understanding the concepts here, you may not get the expertise to develop a use case model. You need to practice and several problems to develop the expertise of developing good use cases.

Now, let's look at the class diagram. We know that classes are entities with common features, this have attributes and operations. Each class can be instantiated into objects; all objects have the similar attributes and operations. The classes typically accepted are represented by solid outline rectangle with compartments and the compartments are name, attribute and operations. But, we will see that there are various ways of representing the class diagram; for example, you might just write the name of the class. We might write the name and attribute into compartments or you might write all the name, attribute and operations.

We are almost at the end of this lecture and in the next lecture, we will discuss the class diagram in more detail. How to document a class diagram; the class relations and how you represent that in a class diagram, we will discuss in the next lecture.

Thank you.

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

Lecture - 34
Inheritance Relationship

Welcome to this lecture, in the last lecture we had discussed how to develop use case diagrams and given a specific problem how one goes about developing use case. We said that we need to practice that with several examples and exercises to get some insight into how to develop the use case model and to document it. In this lecture we will discuss about the class diagrams.

(Refer Slide Time: 00:52)

The screenshot shows a presentation slide titled "Class Diagram". The slide content is as follows:

- Classes:
 - Entities with common features, i.e. attributes and operations.
 - Represented as solid outline rectangle with compartments.
 - Compartments for **name, attributes, and operations**.
 - Attribute and operation compartments are optional depending on the purpose of a diagram.

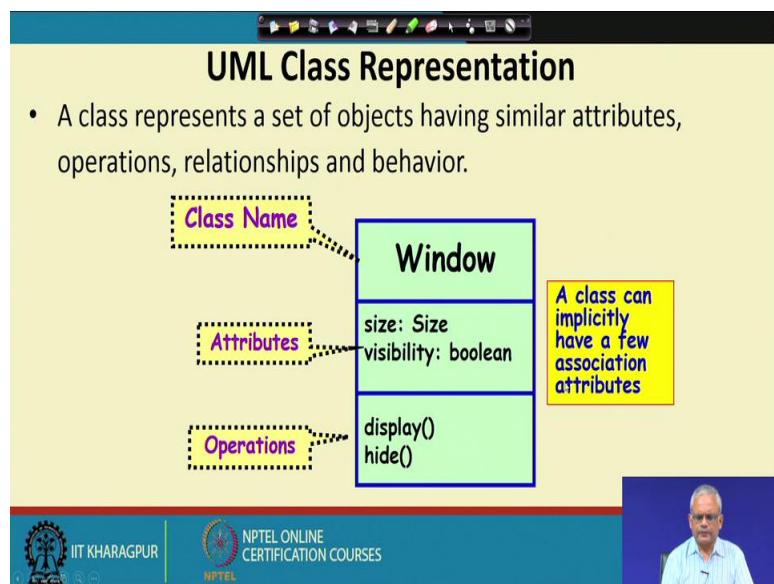
On the right side of the slide, there is a small diagram of a class named "Book" with compartments for "Name", "Attributes", and "Operations".

The footer of the slide includes the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a professor speaking.

In the class diagram with document the entities with common features i.e., all the objects in a problem which are of similar attributes and operations constitute a class. For example, in a library system the book objects have similar attributes and operations like each book has a name and ISBN number and so on and each book has similar operations like issue the book, return the book, create the book and so on. Each class is represented using the solid outline rectangle with compartments. We will use the solid outline rectangle with compartments and we will write the name of the class like book at the top compartment.

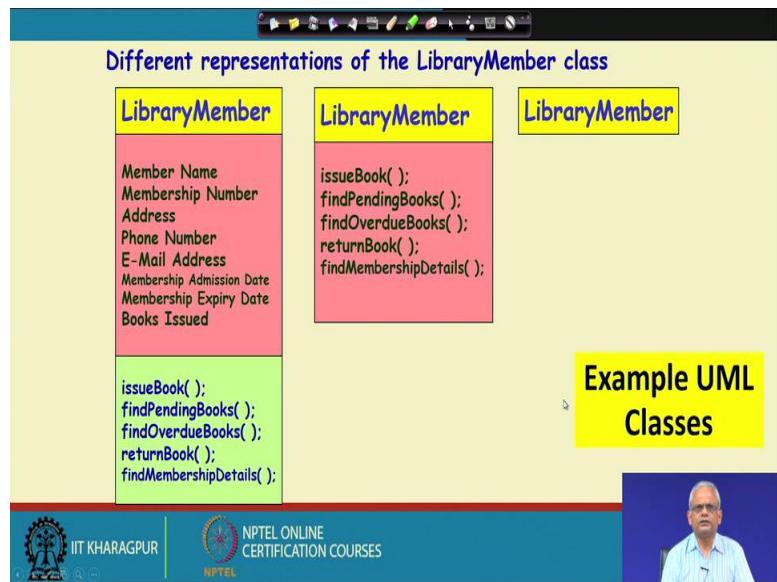
We will write the attributes like the book name, the author, the ISBN number etc. and then in the last compartment, we will write the operations for example, create book, issue book, returned book and so on. Although we don't show all the three compartments always, we might write only one rectangle containing the name of the class that is book in some cases. Just notice here that the name of the class is singular. We don't name the class books. So, we can have just one compartment just the class name, we might have the class name and the methods or the operations that are applicable on this book or we might have all of them together with three compartments.

(Refer Slide Time: 03:19)



So, this is the representation in the figure, name of the classes is window, the attribute name is size which is an instance of the size class, visibility is a Boolean. The operations are display and hide the class name, these are the attributes and these are the operations, but later we will see that there are some attributes which are not shown, but there present what we called them as association attributes. As we proceed, it will become clear.

(Refer Slide Time: 04:03)



As we were saying that the same class can be represented with different details, we might just represent the class name. Sometimes, we might represent the class name along with the operations on that class or we might represent the class name along with the attributes and the operations. But, one would ask that when does one use this representation, which is the name of the class. When does one use the second representation, which is the name with the operations and the full representation using the name attributes and the operations. To answer this question, when we start the object oriented design process as we will be seeing soon that how to go about doing the object oriented design, initially we identify only the class names that is which are the classes we identify the class.

So, in the initial stage of the design process we represent the classes using simple representation. As we proceed with the design steps, we assign the operations to the classes and then we know about which classes or which operations and in that stage of the design we represent using the class name and the operations. Towards the end of the design, we identify the attributes that the classes should have which makes these more complete and can easily be translated to code i.e., the class name, what are the attributes of this and the class member and then the methods of the class.

So, to start with the design process as we start we use the simplest representation, because we identify only the classes and much later in the design process we identify the

operations that for each class. We use this representation and towards the end of the design we have this complete representation and this can be easily translated to code.

(Refer Slide Time: 06:59)

The slide has a yellow background. At the top, the title 'What are the Different Types of Relationships Among Classes?' is centered. Below the title is a bulleted list: '• Four types of relationships:' followed by four items: '- Inheritance', '- Association', '- Aggregation/Composition', and '- Dependency'. At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL Online Certification Courses. On the right side, there is a small video window showing a man speaking.

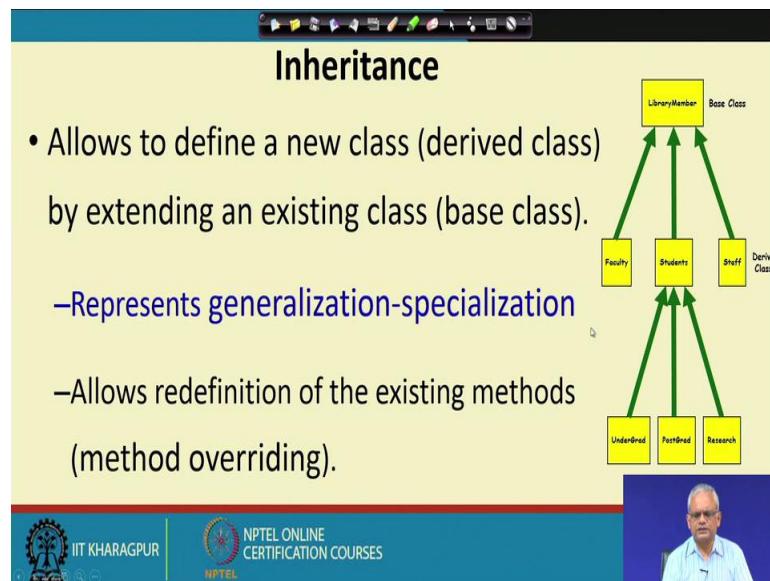
Now, let us answer this question that in any object oriented implementation there are many classes, many objects from which we construct the classes. Now a typical software might have a dozen classes or maybe several dozens or 100's of classes. But the classes are not independent, the classes are related. What are the possible relations that can exist among the classes? Actually there are 4 types of relations that may exist among the classes in a system. The first type of relation is inheritance, one class may be inherited from the base class.

So, from the base class you can inherit a derived class. Between two classes, an association relationship may exist. Please give an example of an association relationship between two classes. Couple of minutes we will just see some examples of association relationship between two classes. There may be an aggregation or composition relationship between two classes if you can give some example regarding aggregation or composition relationship between two classes.

Some realistic examples that will be really nice otherwise in couple of minutes we will see, some examples of aggregation and composition relationship and what the implication of these with respect to the implementation of the software is. The fourth type of relation between classes is dependency. So one class may depend on another

class. We will see a short while from now that, when a class is said to depend on another class. We will see that there are various reasons why a class may be dependent on another class. Now let see this 4 type of relationship in more details.

(Refer Slide Time: 09:35)

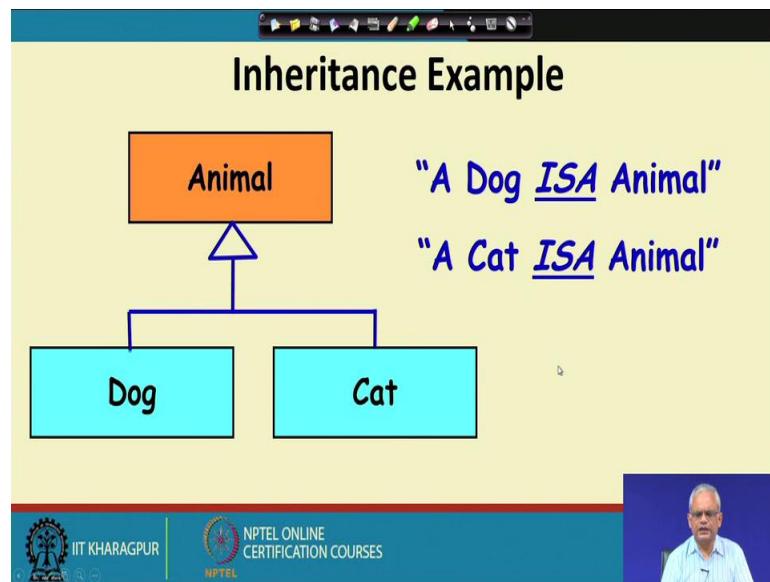


First we will look at the inheritance relationship. Inheritance is a powerful mechanism in object orientation. It allows just to define a new class called as a derived class by extending the base class. Using the inheritance we can represent the generalization-specialization relationship. The base class is the general class and the special class i.e., the specialization classes which add few more attributes or methods to the general class. While, we can add few more attributes to the general class, we can at the same time redefine some of the existing classes that is called as overriding.

Some of the existing methods of the base class and that is called is overriding. Just look at this example the library member class is a base class, there are different types of library member, but some characteristics are common among them for example, all of them have a library membership number, all of them can issue certain number of books, they can return book and so on. So, the common characteristics among different members in the figure are represented as the base class and the faculty, student and staff are the child classes or the derived classes. Each of these add few more attributes and operations to the base class and we have another hierarchy in the next level, that the students can be undergraduate student, postgraduate or research students.

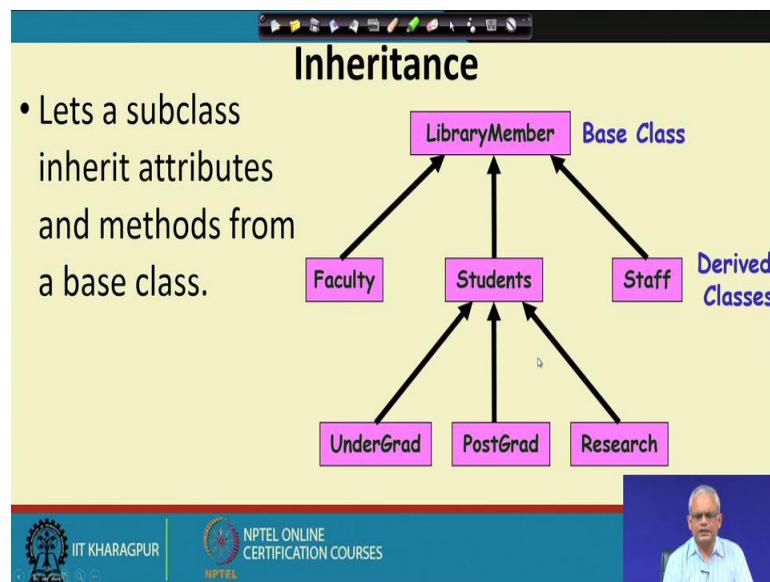
So, the characteristics that are specified in the student class that is the method and attribute are all shared or all of these classes have the same characteristics i.e., the attributes and methods, but each of these add few more attributes and this may be methods.

(Refer Slide Time: 12:14)



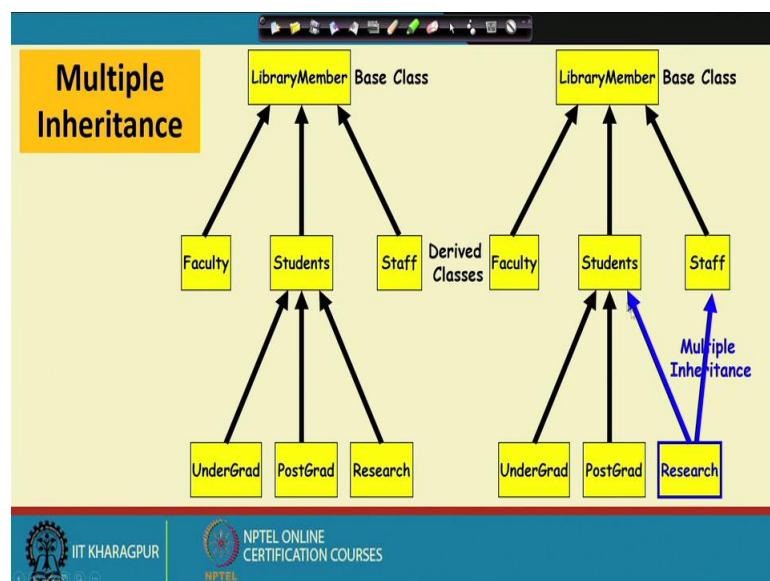
This is an example of inheritance. A dog is an animal, a cat is an animal. We have underlined the ISA in the diagram to denote that sometimes the inheritance relation is also called ISA relation. Animals, as we know they have some characteristics, they are living, they take food, they die etc. Those characteristics are all shared by the different animals. So a dog is an animal, a cat is an animal and therefore, sometimes the inheritance relationship is called as a ISA relationship.

(Refer Slide Time: 13:11)



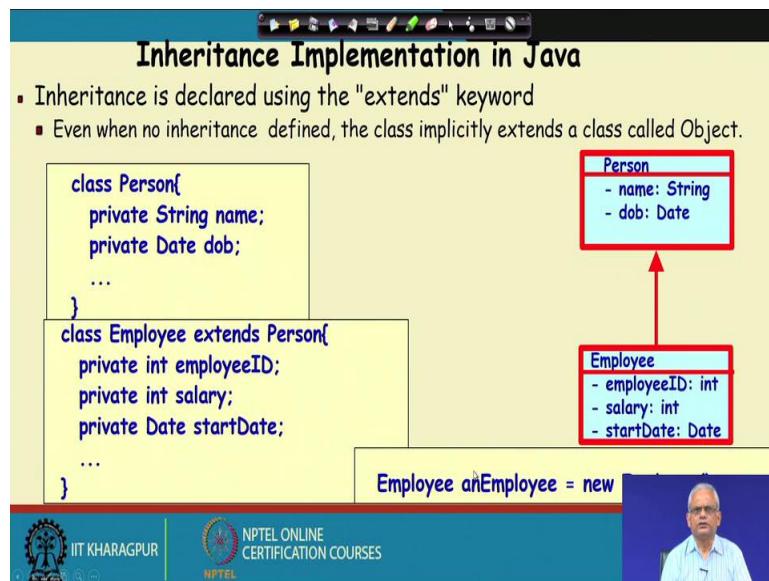
A derived class inherits the attributes of the base class and this helps in reusing the code. The library member may have some attributes and methods, we do not have to repeat those or rewrite those again in all the derived classes these are automatically inherited. When we use the inheritance mechanism and that helps in code reuse, you can have multiple inheritance.

(Refer Slide Time: 13:51)



In multiple inheritance, a class may inherit from two base classes, so the research student has the characteristic of a student and also a staff so this is called as the multiple inheritance.

(Refer Slide Time: 14:10)

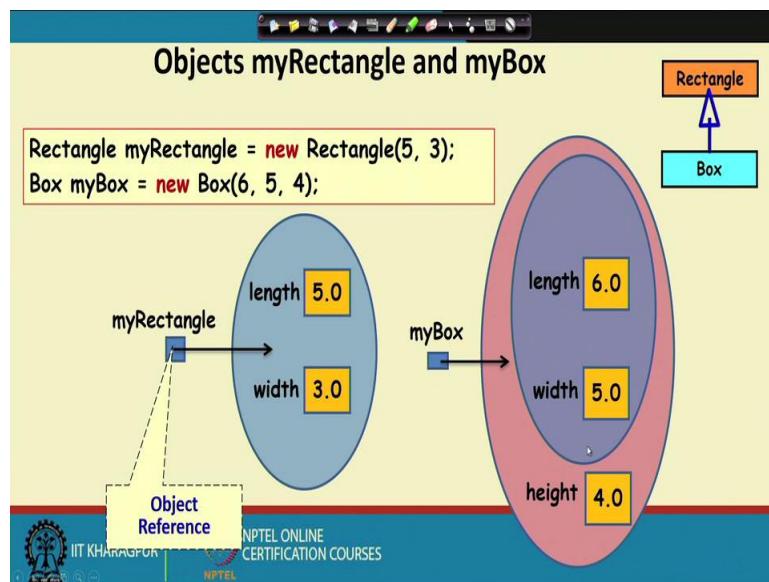


Now, let's see how inheritance relationship can be implemented in Java. Of course, if you are familiar with other object oriented programming languages that will be very similar to here just a small change in the keywords and so on but, the concepts are very similar. Let's assume that the class hierarchy that we have in our design. A person is a class and we have name and date of birth as the attributes and employee is a derived class i.e., employee is a special type of person. So person is a general class, employee is a special type of person which has not only the name, date of birth but also has few other attributes as well like employee id, salary, start date for the employment and so on.

How do we write the java code for this? That is very straight forward. We write the definition for the class person which is private string name, private date of birth. So, straight away we can translate this into the code similar is with C++ or any other object oriented language and for the derive class, we use the keyword extends in Java. The class employee extends person and if you write extends then automatically name and date of birth are inherited by the person class and we have few attributes that are defined extra in the derived class. We write them like private employee id, private int salary, private date start date.

So, inheritance implementation is straight forward we have a keyword extends in Java and we can very easily implement such a relationship in Java by simply using the extends relation, but we will see that the other type of relationship between classes like association, aggregation and so on. We do not have single keyword using which we can implement those relations we need to do something else, but it would have been nice or very simple if we just had a keyword like an extends. This is the simplest implementation of a relationship by just using a keyword and once we create the two classes, here employee class we can instantiate an employee class into objects, we write employee, an employee is equal to new employee. So, the new keyword helps us create the object of employee class and then name of this object is an employee.

(Refer Slide Time: 17:41)

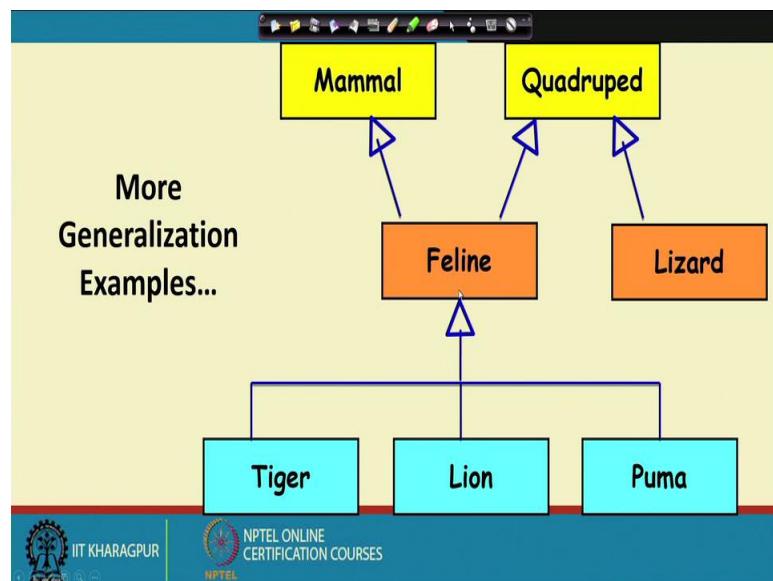


Let's say we want to create objects, `myRectangle` and `myBox`. We have the classes `Rectangle` and `Box`. Now we want to create objects `myRectangle` and `myBox`. We write `Rectangle myRectangle = new Rectangle(5, 3)` and we give the dimensions of the rectangle 5, 3 as the argument to this constructor and `Box myBox = new Box(6, 5, 4)` and we give the arguments 6, 5 and 4.

So, when we create this we write `new Rectangle(5, 3)`. The attributes in this object are assigned the value 5 and 3 and once we write the `new` operator that is we create an instance of `Rectangle` class, we get handle here that we called as the identification or the identifier of this object. Normally we call it as the object reference, so `myRectangle` once

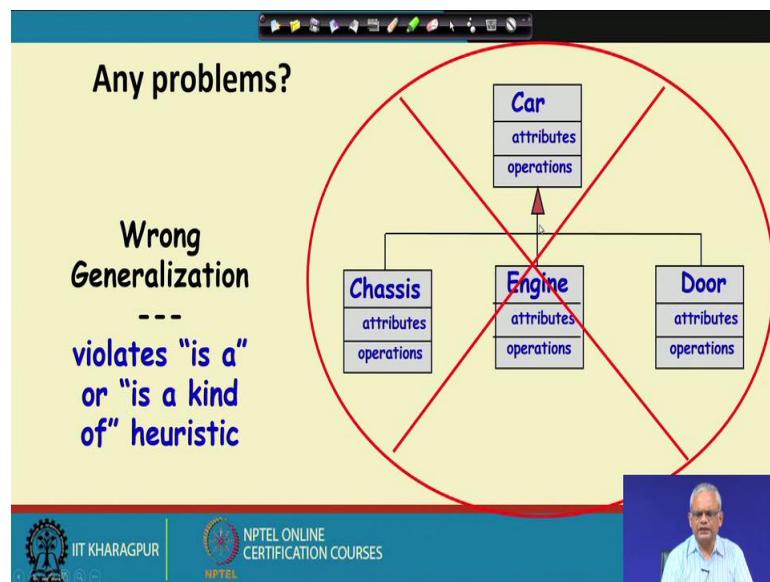
you say new Rectangle(5, 3), myRectangle is the object reference or the reference for this object that we created. Similarly once you say that myBox = new Box, myBox becomes the object reference for the box object, but the box object is an inherited object, it is a derived class and it inherits the attributes and methods of the base class. Therefore, implicitly we can think that the attributes length and width defined in the base class are also present in the derived class and you have this new attribute which is added in the derived class is the height which is also present.

(Refer Slide Time: 19:54)



So, this is another example of a generalization - specialization. There is an example of a multiple inheritance, the feline class inherits from both mammal and quadruped and then from the feline class we have the tiger, lion and puma has the derived classes, lizard is derived from quadruped.

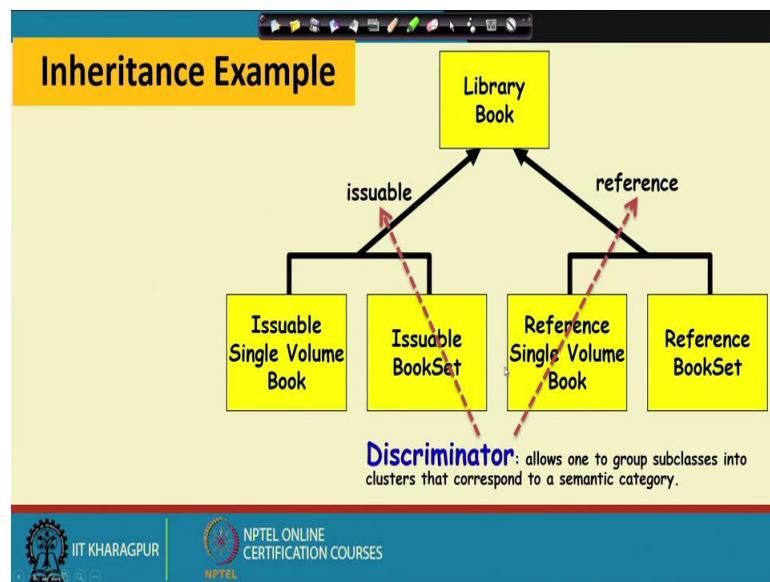
(Refer Slide Time: 20:31)



Suppose someone has drawn this inheritance relationship chassis, engine and door are derived from class. Is this correct, does it appear all right, this inheritance hierarchy that chassis, engine, door a derived from the car class? No, this is not correct, it's a wrong generalization. We had said that the inheritance relationship we characterized by ISA relationship, we cannot say that chassis is a car, car is not chassis. Engine is a car is not a proper generalization and similarly door is a car is not a proper generalization. So this is not a proper generalization - specialization relationship and we cannot represent it using an inheritance relationship, later we will see that this is actually an aggregation relationship.

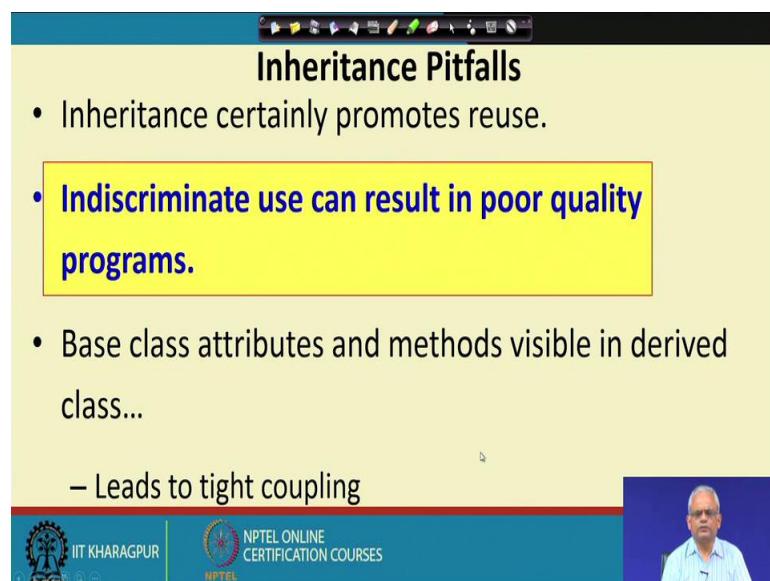
So, to check whether the relationship is correct or not, we can use this test that we can say that whether a classes is a car, engine is a car, if it is a general class and these are special class then the ISA relationship hold. For example, library member i.e., UG member, PG member and staff member. UG member is a library member, PG member is a library member and staff member is a library member. So, the ISA relationship holds. But in the above example in the figure, it does not hold. So this is a wrong construction of generalization - specialization relation between these classes.

(Refer Slide Time: 22:41)



We often have many classes derived from a base class, but it is a good idea to use a discriminator based on which the derived classes are defined for example, this the library books can be issuable books or reference books. So it is a good idea to write the discriminator based on which the set of classes are derived and the set of classes are derived that helps in readability and the supported by UML to write the discriminator while constructing a complex class hierarchy, that helps us to understand the hierarchy better.

(Refer Slide Time: 23:34)



Inheritance is a powerful mechanism that promotes reuse, simplifies the design and also easily implemented in code just by using a keyword, but in spite of its many advantages if we indiscriminately use inheritance, it will result in poor quality of programs. For example, if you have a very deep class hierarchy then it has poor cohesion and coupling because a large number of methods are visible in the leaf classes and that leads to tight coupling and it makes it very difficult to understand the leaf classes, because they have too many methods and we need to examine what are the methods it has inherited. So, inheritance should be used with some cautions.

(Refer Slide Time: 24:57)

Association Relationship

- How implemented in program?
- Enables objects to communicate with each other:
 - One object must “know” the ID of the corresponding object in the association.
- Usually binary:
 - But in general can be n-ary.

Now, let's look at the second type of relationship between classes, this is the association relationship. We need to discuss some examples of association relationship and also we need to answer this question that how are these relationship implemented in a program. In case of inheritance, we saw that there are simple keywords in all object oriented languages whether it is Java, C++ or any other language the inheritance relationship is straight forward to implement.

But how does one go about implementing the association relationship? One thing that we must remember about the association relationship is that, when two classes are associated the corresponding objects can communicate with each other as we will look at the examples it will become clear. One class can invoke the methods of another class, but to be able to invoke the method of another object, the object must somehow store the ID of the

other object otherwise how will it invoke the method of the other class. And that holds the idea of how to implement an association relationship will soon look at that. The association relationship is usually binary, but it can be a higher degree association like ternary or in general can be n-ary, let's look at an example of an association relationship.

(Refer Slide Time: 26:41)

Association – example

- In a home theatre system,
 - A TV object has an association with a VCR object
 - It may receive a signal from the VCR
 - VCR may be associated with remote
 - It may receive a command to record

```

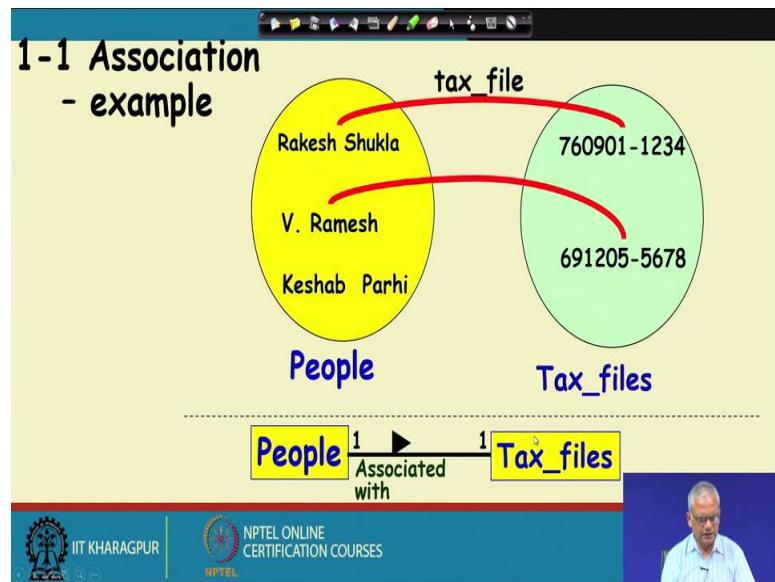
    graph LR
        Remote[Remote] -- "1 commands-->" --> VCR[VCR]
        VCR -- "1 Connected to-->" --> TV[TV]
    
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let's say VCR, a video recorder is associated with the TV and we can have a remote which is associated with the VCR. So we can say the remote object can give command to the VCR, the VCR can record what is being played on the TV. So, the TV object is associated to the VCR object, because the VCR object receives signal from the TV object and the VCR may in turn be associated with a remote and it can receive command from the remote. The way we represent the association relationship is by drawing a line, there are different ways we can draw the line sometimes, we draw an arrow sometimes, we just write this line. We will see when we draw an arrow and when we draw a line and we write the multiplicity and we write the reading direction.

We will see the details of how to represent the association relationship, what are the implication of just drawing an arrow, just drawing a line and then we write the name of the association on this line or arrow. And we write the reading direction say that the VCR is connected to TV it helps us to read or we can read the TV connects to the VCR. The remote commands to the VCR or we can read as the VCR receives commands from the remote.

(Refer Slide Time: 27:57)



So, we have reached the end of this lecture and we will continue in the next lecture to discuss about how to represent the association relationship, the multiplicity and so on; as per the UML syntax. We will stop here and continue in the next lecture.

Thank you.

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

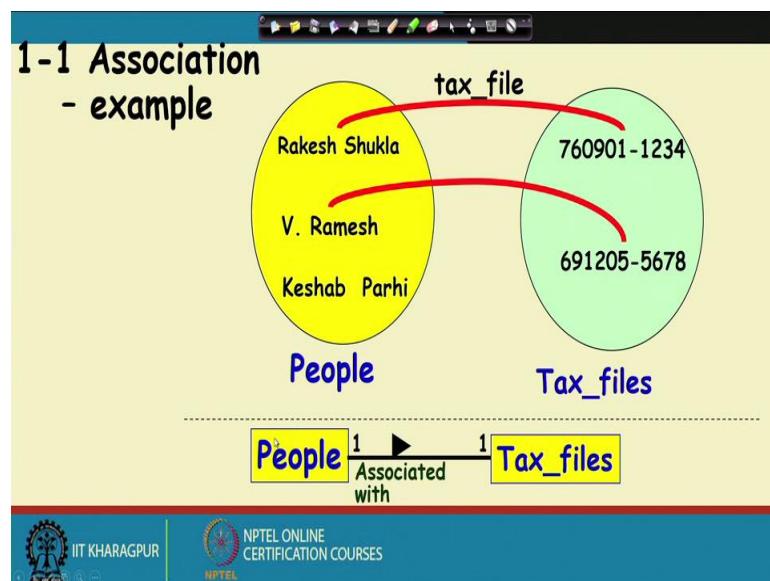
Lecture - 35
Association Relationship

Welcome to this lecture. In the last lecture we had discussed the different types of relationship that exist among the classes. Then we said that there are 4 types of relationship that can exist among classes i.e., inheritance, association, aggregation-composition and dependency.

We, looked that the inheritance relationship with some examples and we saw that inheritance is a powerful mechanism and also easily implemented in code. Just by using a keyword we can implement the inheritance relationship. And, then we started to look at the association relationship, we are looking at how to represent that in the UML syntax and also how to implement it in code and what is the implication of the association relationship between classes?

Now, let's proceed further understanding the association relationship.

(Refer Slide Time: 01:33)

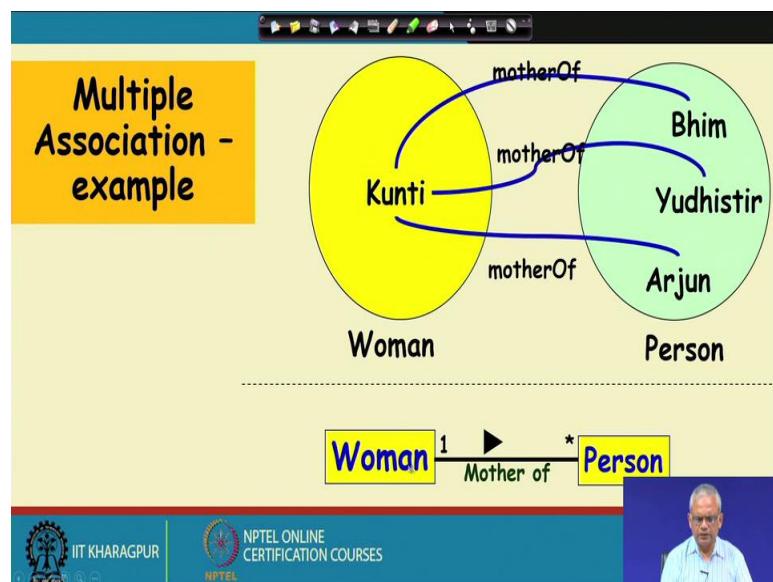


The association relationship is represented by a line here in the figure above and sometimes by an arrow which we will see later and we write a multiplicity number with

the association relationship. So, we read here like the Person or the People associated with the Tax_file. One Person associated with one Tax_file or Tax_file associated with one Person.

If, we look at the corresponding objects of the People, we will see that there are many objects Rakesh Shukla. V. Ramesh, Keshab Parhi etc., and then the Tax_file are having different numbers. So, each person here is object people object here is associated with the Tax_file. But, then there can be some objects here, which do not have or which are not associated with any Tax_file so that is permitted by this relationship. As you proceed it becomes clear that why that is the case that some of the objects, people object is associated with a tax file, but some are not.

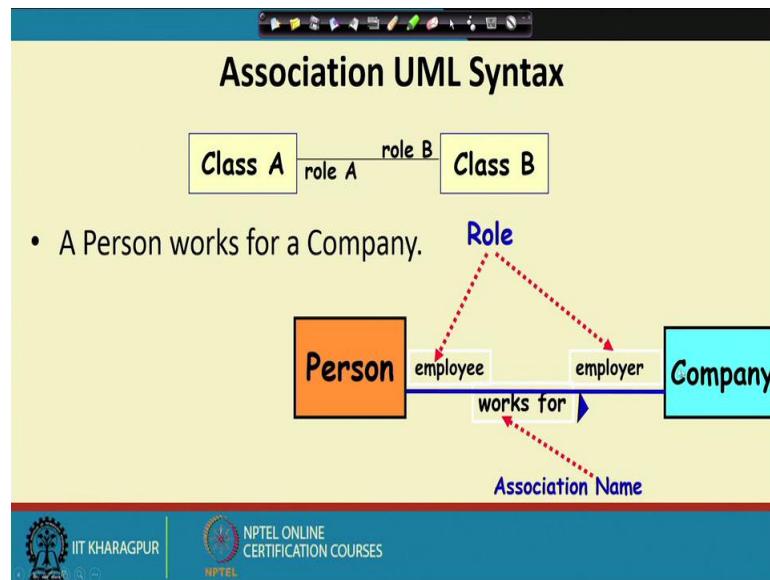
(Refer Slide Time: 03:03)



Here, a woman is mother of many persons, but we can also read it as a person has a woman as mother. We read this association relationship between woman and a person. The name of the association relationship is motherOf. We can read it as a woman is the mother of many persons that is the implication of the star.

We can also read it as a person has one woman as his mother. If we look at the corresponding object diagram that becomes clear the woman class is instantiated into many woman.

(Refer Slide Time: 04:24)

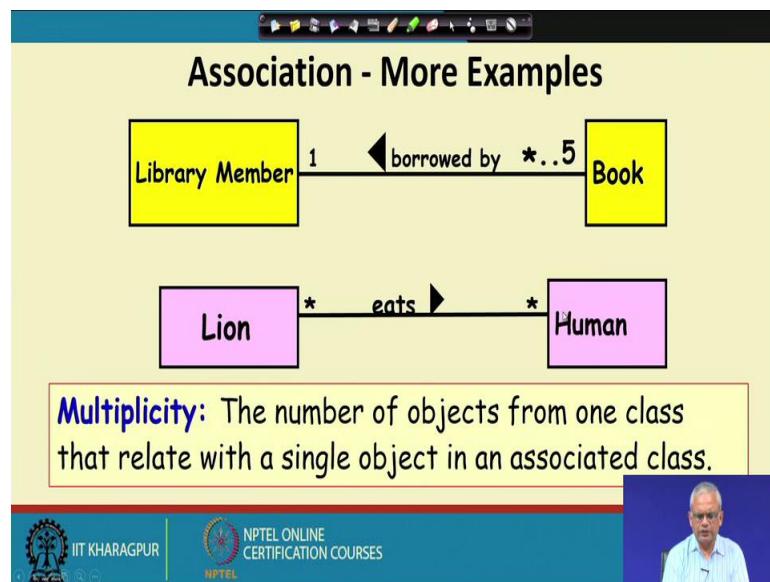


And, one of the women here is Kunti who is the mother of many persons here Bhim, Yudhistir, Arjun etc., but Arjun has exactly one mother, Yudhistir also has exactly one mother.

So, that is what it says a person has one woman as mother, but woman can have many persons, woman is the mother of many persons, this is the syntax in UML. We write the 2 classes which are related by the association relationship, we draw a line between them indicating the association relationship and we write the role there, we will see, what exactly is the role?

For example, a person and a company are associated, because the person works for the company i.e., the name of the association relationship is works for. We represent this in the form person works for Company (this is the reading direction); name of the association is works for. The role of the Person is the employee and the role of the Company is the employer. We can write that or we may not write, but then in the case tools we will see that if we write this two, it helps this become attributes of the corresponding classes. The Person works for a Company and the role of the Person is employee and the role of the Company is the employer.

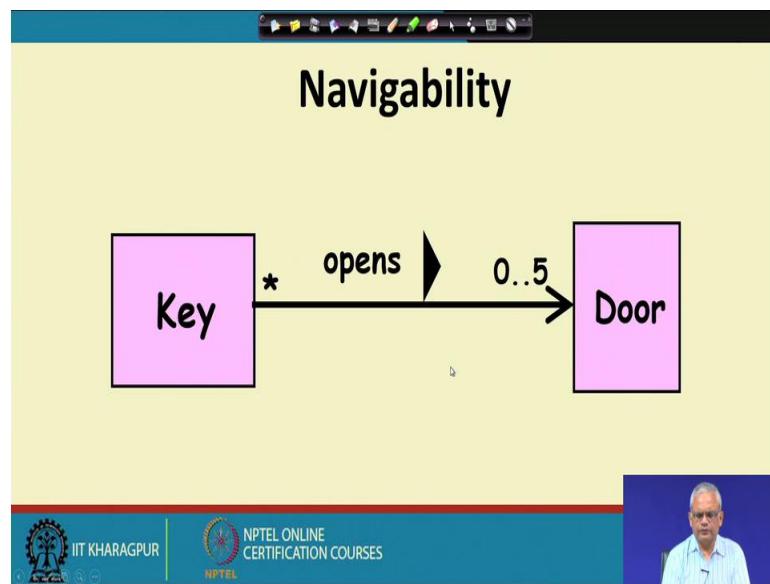
(Refer Slide Time: 06:22)



These are more examples of association, how do we read this association? A member borrows up to 5 books. A book is borrowed by a library member; a member borrows up to 5 books. I should have written 0 instead of * in the first relationship. Let me just change that it should be 0, 0 to 5. A member borrows up to 5 books or we can read in the direction, that a book is borrowed by exactly 1 library member.

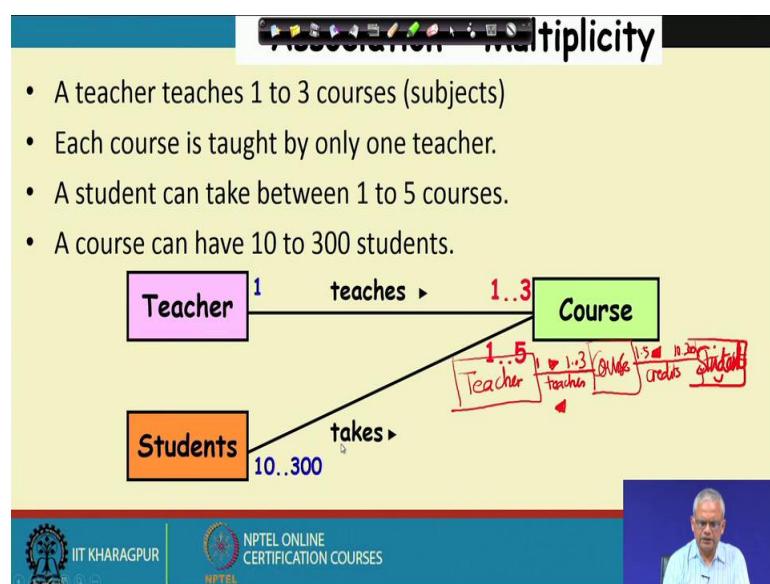
Now in the second relationship, a lion eats many humans, a human is eaten by many lions. So, the multiplicity that we write here, it indicates a single object here is associated with how many objects on the other side, This is important; the multiplicity indicates one object here is associated with how many objects on other side. A lion eats many humans, a single object on this side, a human is eaten by many lions. Similarly, the multiplicity in the first relationship is 1 and here it is 0 to 5. A, library member that is a single object here borrows up to 5 books, a single object here, a book is borrowed by a member.

(Refer Slide Time: 08:37)



Now, let's look at the arrow here. If we put arrow in the association; that means, that the key can invoke a method on the door, but the door cannot, it is navigable from key to door, but not the vice versa. If it is a simple line here or a bidirectional arrow i.e., the Navigability on both directions. Now, let's see how you can read here, a key that is a single object here, a key opens up to 5 doors, a door is opened by many keys.

(Refer Slide Time: 09:34)



Now, let's see how to draw a class diagram given the description in the figure. A teacher teachers 1 to 3 courses, each course is taught by one teacher. A student can take between

1 to 5 course and a course can have 10 to 300 student. So, the classes here are teacher, course and subject, a teacher teaches 1 to 3 courses. A course is taught by exactly one teacher. Teacher teachers up to 1 to 3 course, to write the reading direction with a arrow direction in the figure, then we write course taught by, to write this is the arrow direction in the figure with the teaches association. If we will write the relation, the association is taught by and the next one is student can take between 1 to 5 courses. A course can have 10 to 300 students. And we can write the name of the association relationship, a student credits 1 to 5 courses, with the reading direction and association name as credits.

So, that is the same diagram here, a teacher teaches 1 to 3 courses and a student takes 1 to 5 courses or a course is taken by 10 to 300 students. Given a description we should be able to identify the classes and the association relationship, write the multiplicity and the association and the reading direction.

(Refer Slide Time: 12:27)

Quiz: Draw Class Diagram

- A Student can take up to five Courses.
- A student needs to enroll in at least one course.
- Up to 300 students can enroll in a course. *Student 1..5 Course*
- An offered subject in a semester should have at least 10 registered students.

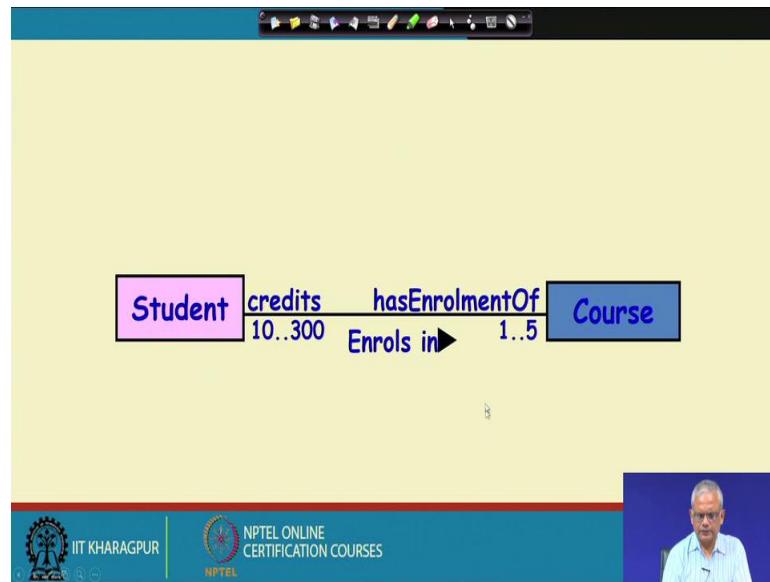
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, please do a small quiz, we need to draw the class diagram: a student can take up to 5 courses. Student names to enroll in at least one course, up to 300 students can enroll in a course and an offered subject in a semester should have at least 10 registered students please try to draw the class diagram for this.

The first thing is to identify what are the classes here. Student is a class and Course is a class and we represent them a student and the course, a student can take up to 5 courses and need to take at least one course. So, we write 1 to 5 and student takes write the

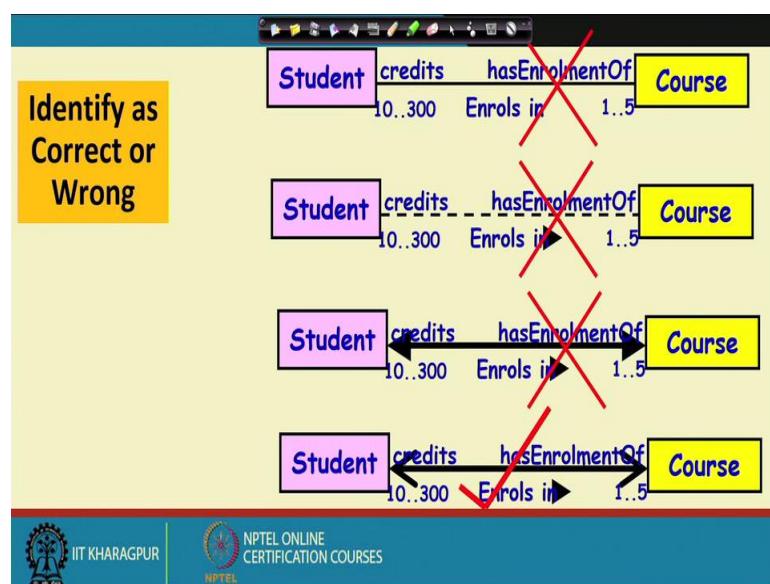
reading direction here, up to 300 students can enroll in a course and it must have at least 10 students. So, we write the multiplicity 10 to 300. So, this will be the class diagram for this description.

(Refer Slide Time: 14:11)



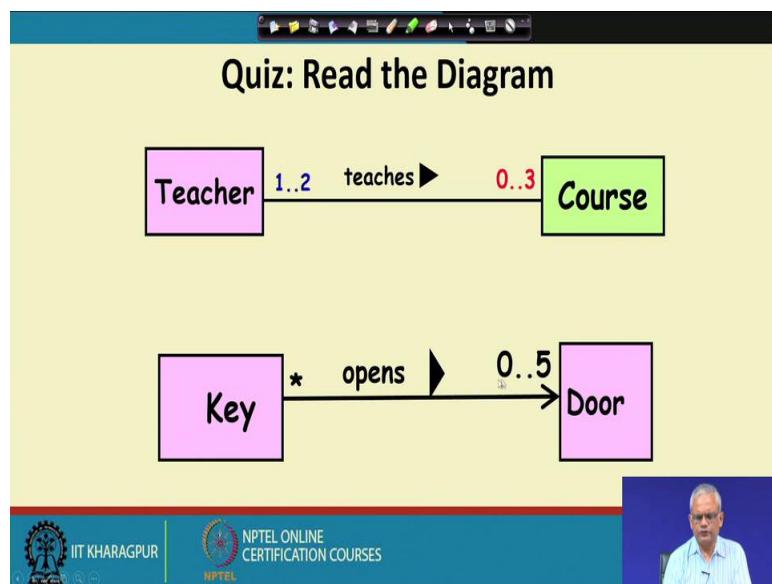
So, that is what we have written student takes are enrolls in 1 to 5 courses and a course has enrollment of 10 to 300 students. And we can also write the role, the student credits and the course has enrollment of.

(Refer Slide Time: 14:32)



How do I identify, which diagram is correct? Let's look at the first diagram above which is a round diagram, what's the mistake in this diagram? The mistake in this diagram is that we have not given the reading direction. In the second diagram, what's the mistake here? Given the reading direction here, but then the line association line is wrong used a dotted line. What about the third diagram? This is a wrong diagram, we have used a wrong arrow type. The fourth one is a correct diagram i.e., a bidirectional arrow, which is also equal to just as, which is just equivalent to a simple line, a simple line connecting to classes is same as their being connected by a bidirectional arrow of this form.

(Refer Slide Time: 15:57)



Now, there is another quiz here. Please read this diagram above, 2 classes i.e., Teacher and Course and we have the multiplicity that are written here and the association relationship and reading direction, how do we read this diagram? We can read as a teacher teaches up to 3 courses. A course is taken by either one or 2 teachers, a teacher teaches up to 3 courses, a course it is taught by either 1 or 2 teachers. Now, let's try to read second diagram. A key opens up to 5 doors; a door is opened by many keys.

(Refer Slide Time: 17:05)

Association and Link

- **A link:**
 - An instance of an association
 - Exists between two or more objects
 - **Dynamically created and destroyed as the run of a system proceeds**
- For example:
 - An employee joins an organization.
 - Leaves that organization and joins a new organization.

Diagram: A UML-style class diagram showing a directed association between 'Person' and 'Company' classes. A hand-drawn object diagram below shows a 'Person' object (labeled 'Ramesh') connected by a red line to a 'Company' object (labeled 'TCS').

Logos: IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's look at the concept of a link, so far we have seen that the association relationship exists between classes. For the objects of the 2 classes, which are associated, a link exists between them. Therefore, we can say that a link is an instance of an association exists between 2 or more objects. The association relationship between classes is a static in nature, we always represent the association. Whereas the objects as the system runs the software executes objects may get the links between objects may get dynamically created or destroyed.

For example, let's say we have an employee class and a company class. Let's say person works for a company and write the reading direction. A company has many employees write the star here, but when we look at the object diagram let's say an object like Ramesh and there are other objects and there are many companies let's say Infosys, TCS, etc. Let's say the company the person Ramesh works for Infosys a link exist between these two objects, but as the system runs, let us say Ramesh leaves Infosys and joins TCS. Then, this link gets dissolved and at the moment they have no link and let's say after few months he joins TCS. So, link gets formed between Ramesh and TCS.

So, the association relationship is static in nature between two classes, whereas the links are more dynamic in nature. And these exist between the corresponding objects of the associated classes.

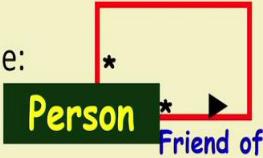
(Refer Slide Time: 20:01)

Association Relationship

- A class can be associated with itself (**unary association**).

—Give an example?

- An arrowhead used along with name:
—Indicates direction of association.



- Multiplicity indicates # of instances taking part in the association.



IIT KHARAGPUR



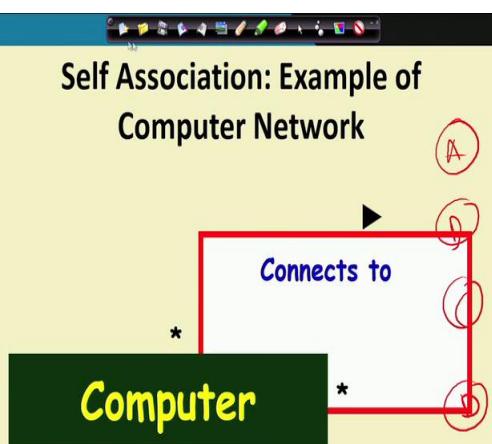
NPTEL ONLINE
CERTIFICATION COURSES



But, can association be unary i.e., we have a single class and association relationship is defined with that same class? A single class and the association relationship is defined on that same class, is it possible? If yes give an example. There are many examples possible, we can write the multiplicity and we can use the arrow head to indicate the direction of the association. And one example here is a person is a friend of many persons or a person has many persons as his friend. A person has many persons as his friend or a person be friends many persons. This is an example of a unary association; there can be many examples.

(Refer Slide Time: 21:24)

Self Association: Example of Computer Network





IIT KHARAGPUR

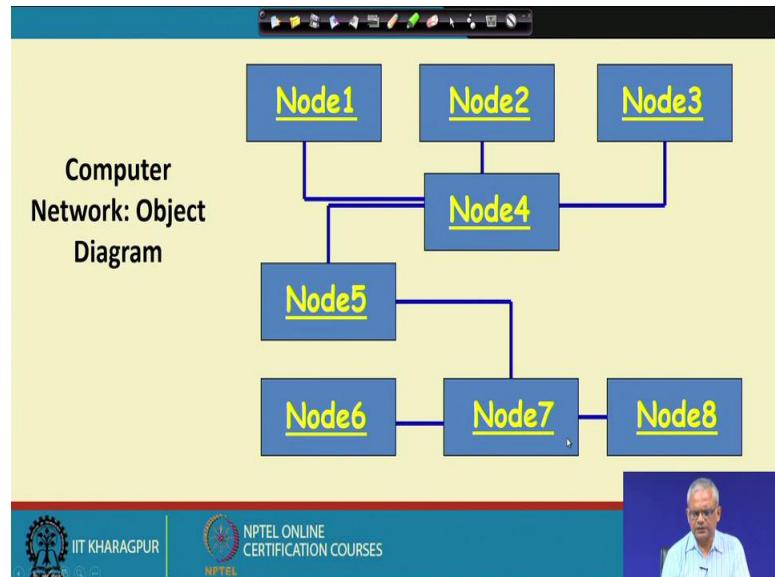


NPTEL ONLINE
CERTIFICATION COURSES



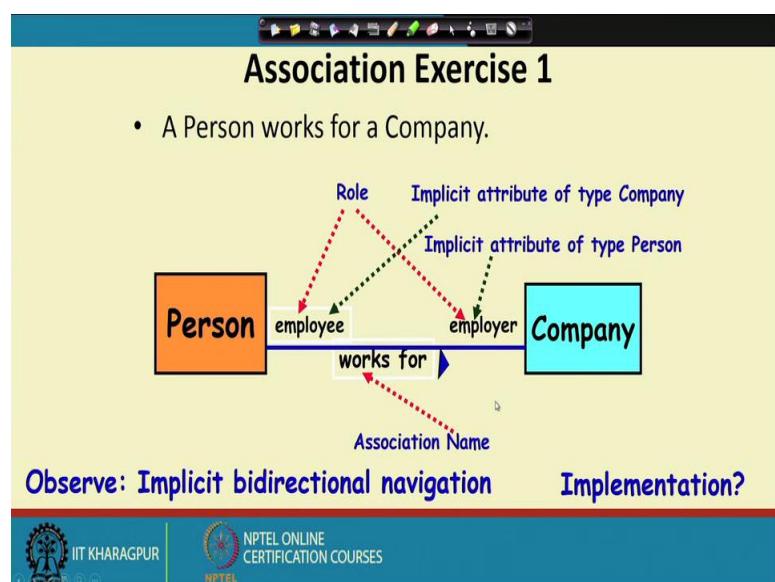
This is another example, a computer connects to many other computers or a computer is connected by many other computers, but how will the object diagram of this look like?

(Refer Slide Time: 20:20)



So, this is an object diagram, we have the computer name Node1, Node2, Node3, Node4, Node5, Node6, Node7, and Node8. So, Node1 is connected to Node4, Node5 is connected to Node4. So, the Node4 is connected by 4 computers, whereas Node6 is connected by 1 computer.

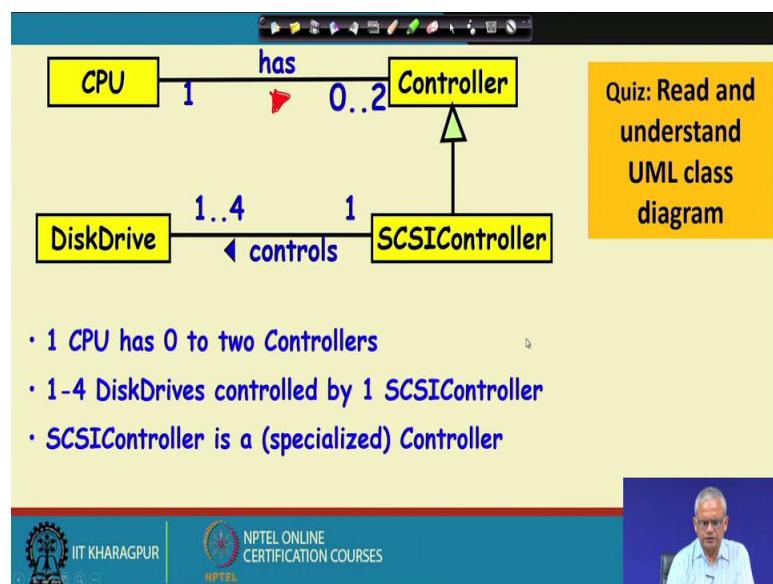
(Refer Slide Time: 22:56)



Now, let's do one excise, let's develop the class diagram. A person works for a company we draw the person class and the company class, and then draw the association relationship works for, and the role is employee here, the role of the company is the employer, these are the implicit attributes as we will see that when there is a association relationship, we need to create some implicit attributes, which will look at the very soon and this is the name of the association.

Let us see the implementation of this that if this is the class diagram that a person and a company, works for his association relationship. We will see the implementation, that how we can write the code for such a diagram, then it will become clear, that the roles become the implicit attribute type of the company. So, employer becomes an attribute type of company and employee become the attribute type of the person.

(Refer Slide Time: 24:20)

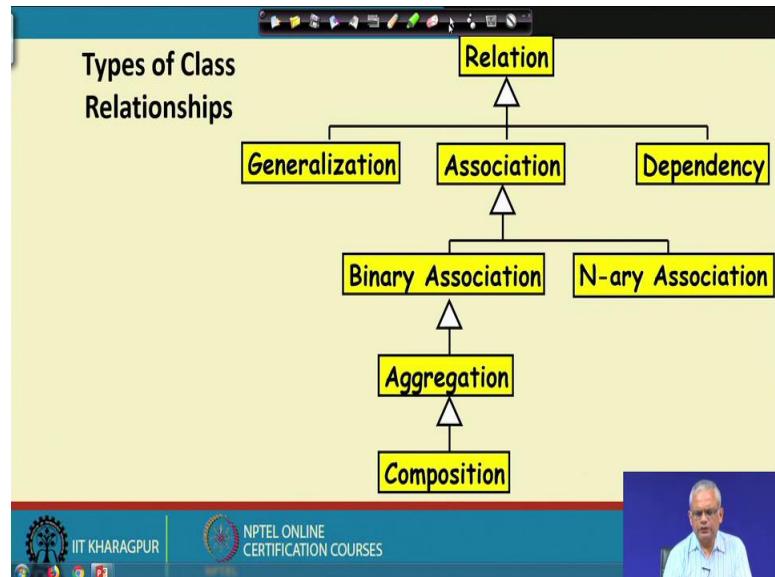


But, before that there is another quiz here, please read the following diagram, if this is the diagram given how do you read it?

We can read here in the above diagram like a CPU has up to 2 controllers. A controller is connected to 1 CPU. The controller can be a quasi-controller, if you had other inheritance relationship would have other types of controllers, but a quasi controller is a special type of controller. A controller controls up to 1 to 4 disk drives or we can also read a disk drive is controlled by 1 quasi controller. A CPU has 0 to 2 controllers, 1 to 4

disk drives are controlled by ones quasi controller, ones quasi controller controls 1 to 4 disk drives as quasi controller is a special type of controller.

(Refer Slide Time: 26:13)



So far we looked at the relations between classes, generalization, association. Aggregation-composition and dependency are not looked at. Association can be binary association or n-ary association and a special type of binary association is aggregation and a special type of aggregation is composition. We are at the end of this lecture in the next lecture we look at aggregation-composition and the dependency relations, we will stop here.

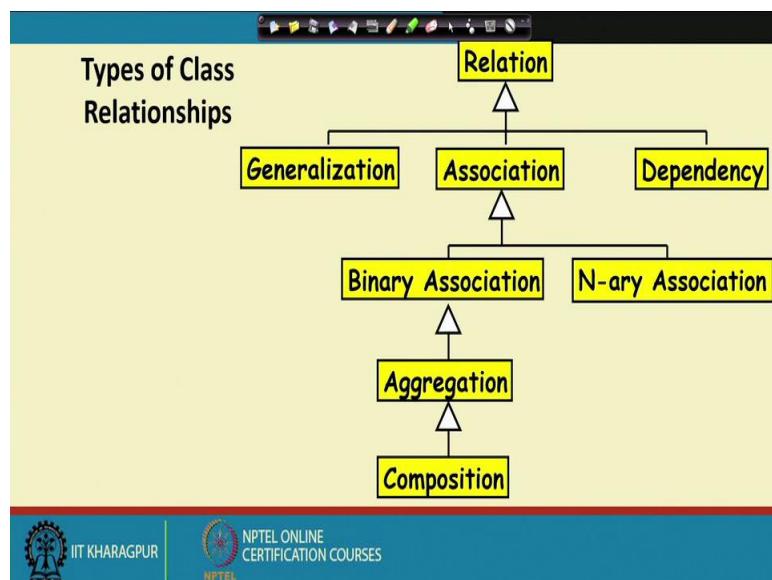
Thank you.

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

Lecture - 36
Aggregation / Composition and Dependency Relations

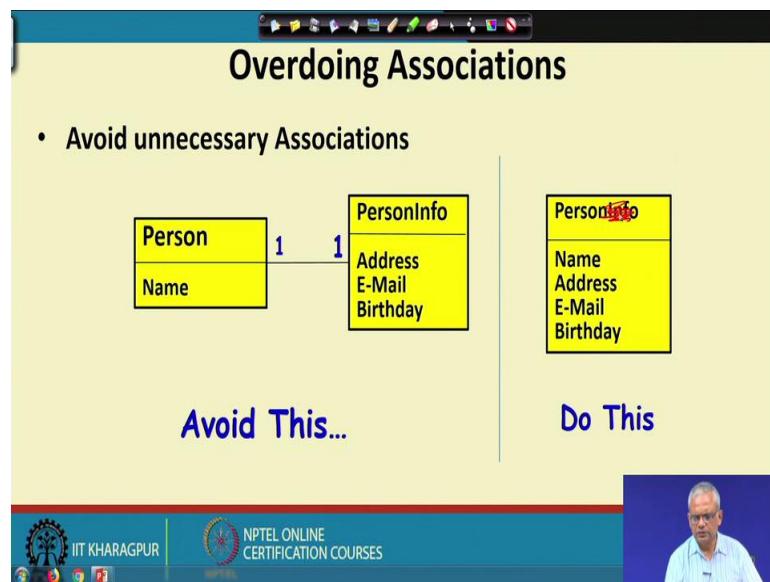
Welcome to this lecture, in the last lecture we were looking at the association relationship between classes. We saw that the association relationship can imply lot of information between the types of association between two classes.

(Refer Slide Time: 00:44)



The types of relation that so far we have discussed between two classes; we will look at the relationship they can be generalization/ specialization association or dependency. And the association can be a binary association or N-ary association and a special type of binary association is aggregation and a composition is a special type of aggregation. So, please observe that we are saying aggregation is a special type of association; it is a association with some extra characteristics. Soon, we will see what we mean by aggregation and why we consider it to be an association relationship with some extra characteristics. Similarly what exactly is composition relationship and why we consider the composition relationship to be an aggregation with some extra characteristics.

(Refer Slide Time: 01:42)



But before we look at the aggregation relationship in the composition; one thing we must mention here is that overdoing an association, unnecessarily complicates in design. For example, we might create a fancy diagram like in the figure above that a person is associated with person information, but actually this information are the attributes of the person. It would have been much better if you had written person information like name, address, email and birthday as the attributes of the Person class.

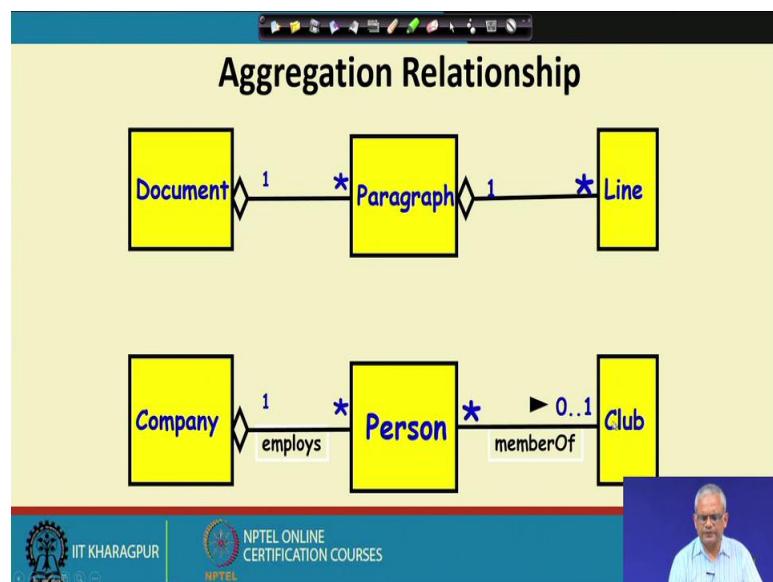
(Refer Slide Time: 02:44)



Now, let's look at the aggregation relationship between two classes; the aggregation relationship indicates a whole part relationship represented by a diamond symbol. The aggregate class creates many components and it is also true that aggregate class invokes the same operations of all the components.

We will see what we mean by aggregation and this is in contrast to plane association where a class does not invoke the same operation of all the classes to which it is associated and the aggregate classes typically the owner of the components.

(Refer Slide Time: 03:38)



Let's look at some aggregation relationship and then we will see what we meant by saying that an aggregate class is a owner of the component classes, it creates the component classes and it usually applies the same operation to all its component classes; this is an example of aggregate relationship, the document consists of many paragraphs. So, the document class creates the paragraphs and a paragraph is the aggregate of many lines, the paragraph creates the lines. If we need to search some word in a document; the document will apply the search operation to all the paragraphs and similarly the search operation each paragraph will apply to all the lines. So, that is what we meant by saying that the same operation is applied to all the components of an aggregate class.

This is another example of aggregate class and this is an example of an association; let's say a company employs many persons. If a company is a aggregate of persons, then it is typically the case that the company creates the persons, applies the same operation on all

the persons; for example, print salary etc. Whereas a person is a member of 0 to 1 club. This association relationship and a club has many members, many persons as its member. This is an association relationship in the club does not create the person whereas here the company creates the person.

(Refer Slide Time: 05:58)

The slide has a title 'Aggregation' with a 'cont...' link. It contains two main bullet points:

- An aggregate object contains other objects.
- Aggregation limited to **tree hierarchy**:

Below the second point is a note: "- No circular aggregate relation." followed by a UML diagram illustrating a circular aggregation relationship. The diagram shows a 'Paragraph' box pointing to a 'Line' box with a filled arrowhead, and a 'Line' box pointing back to a 'Paragraph' box with a hollow arrowhead. Both arrows are crossed out with red dashed lines.

The footer of the slide includes the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, 'NPTEL ONLINE CERTIFICATION COURSES', and a small video frame showing a speaker.

The aggregation is typically a tree hierarchy and no circular aggregate relation is possible; like a paragraph contains many lines, but at the same time a line cannot contain many paragraphs, this is a wrong aggregation relationship; it will have no implementation.

(Refer Slide Time: 06:25)

Cont...

Aggregation vs. Inheritance

Inheritance:

- Different object types with similar features.
- Necessary semantics for similarity of behavior is in place.

Aggregation:

- Containment allows construction of complex objects.

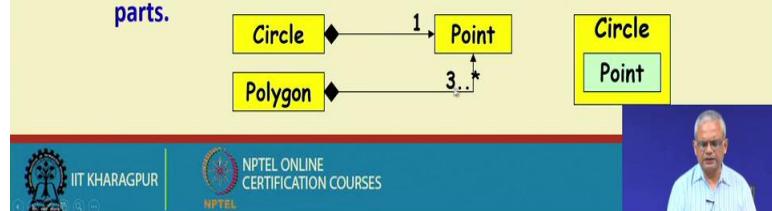


How are aggregation and inheritance compared? There are some similarities and dissimilarities. Using the inheritance relationship we create many objects that have similar features and the necessary semantics for similarity in behavior is in place because they all inherit the attributes and operations of the base class. So, there is similarity of all the objects that are created by inheritance. On the other hand, the aggregation is a containment relationship; it only helps to create complex objects, but these objects might have different characteristics.

(Refer Slide Time: 07:18)

Composition

- A stronger form of aggregation
 - The whole is the sole owner of its part.
 - A component can belong to only one whole
 - The life time of the part is dependent upon the whole.
 - **The composite must manage the creation and destruction of its parts.**

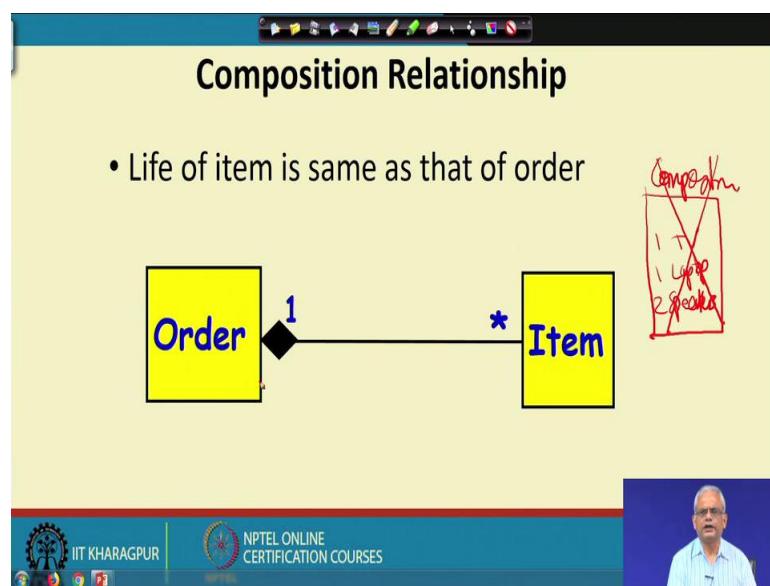


A special case of aggregation relationship is composition represented by filled diamonds; it is a stronger form of aggregation and here this is the sole owner of this point; a circle contains one point and a polygon contains three or more points.

So, you can represent a circle having one point; so, the circle class, circle object is sole owner of a point. When the circle is created along with the point object is also created and when the circle is destroyed, the corresponding point object is also destroyed. Similarly when a polygon is created, three or more point objects are created as part of the polygon and when the polygon is destroyed, the point objects are also destroyed.

So, here in composition, the composite class manages the creation and destruction of its parts. This is unlike the aggregation relationship where the aggregate relationship the aggregate class may create the component classes, but it does not destroy them; so, their life lines are different.

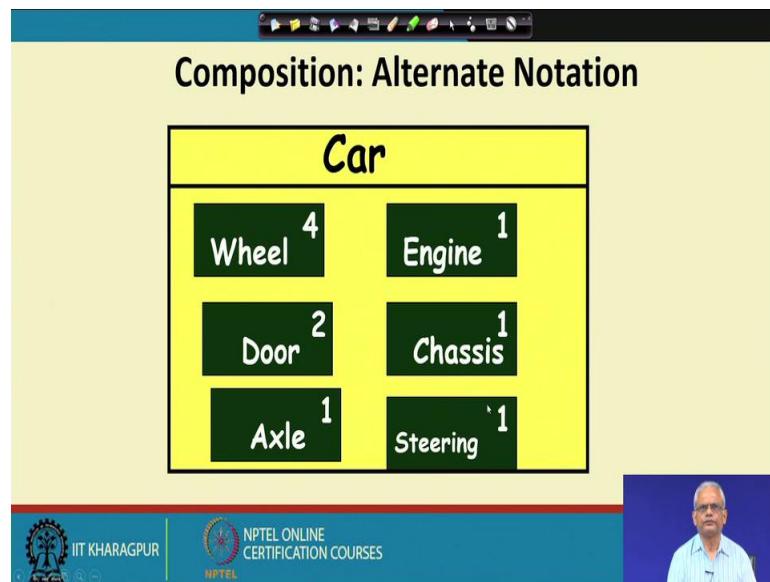
(Refer Slide Time: 09:05)



Let's just explain the difference between a composite relationship; a composition relation and aggregation relation with help of some examples. In this example, an order contains many items and when the order is created the items in it are created. And when the order is destroyed those items are destroyed, but we cannot change an item inside an order. For example, we create an order by entering the necessary data; let's say we have 1 TV, 1 laptop and 2 speakers; the order is created, but now we cannot.

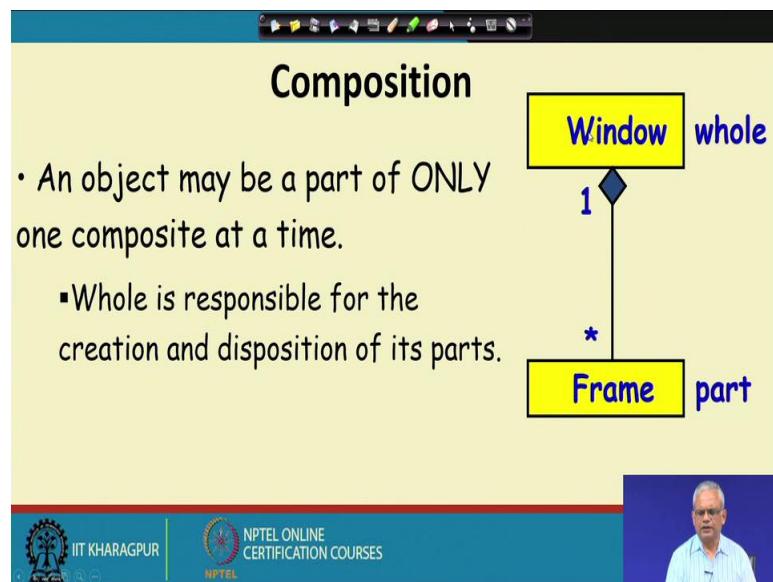
So, this is composition class, it contains 1 TV object, 1 laptop object, 2 speaker object, but we cannot really change it to 3 speaker objects; what you can do is you can delete this and you can create a new order class. But we cannot change the parts of it once it is created; the order during creation, the items are specified and the items are created along with the order. And the items are destroyed with the order and there is no change possible to the order and the items in the order in between.

(Refer Slide Time: 11:04)



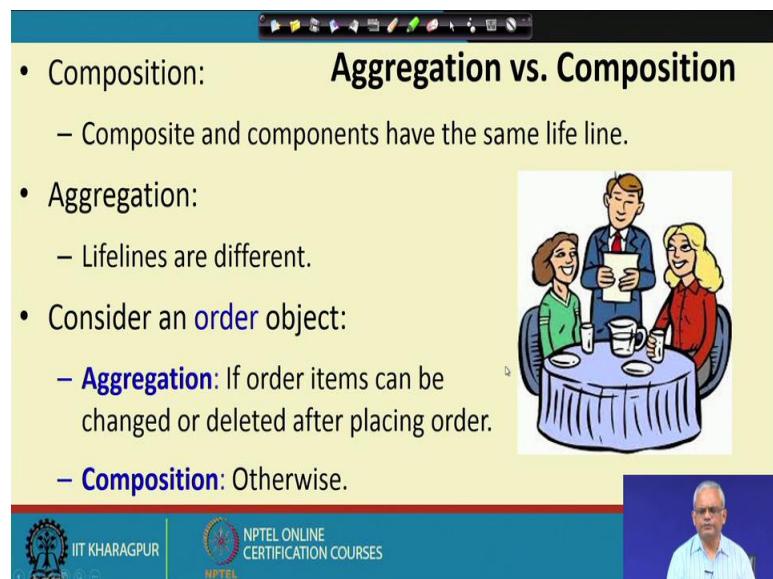
UML also supports an alternate notation we can write the corresponding classes here inside the composite class and we can write the multiplicity of this. For example, a car contains 4 wheels, 1 engine, 2 doors, chassis, 1 axle and 1 steering wheel.

(Refer Slide Time: 11:32)



In the composition relationship, an object may be part of only one composite at a time and the composite is responsible for creation and destruction of its part. For example, a window can have many frames; the window creates the frames during the creation of the window. And once the window is created no further frame alterations, creation or deletion is possible, but when the window is deleted; the frames also get deleted.

(Refer Slide Time: 12:12)

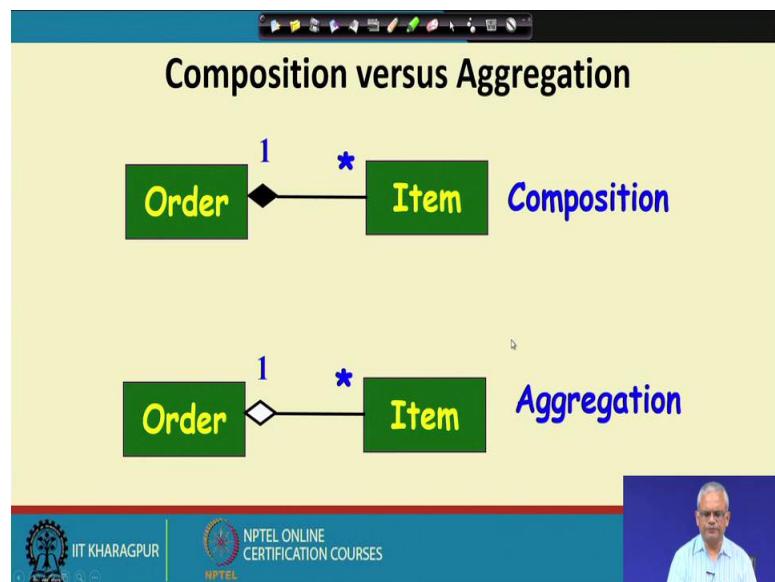


Just to explain the difference between aggregation and composition, we just explained with the help of one example. Let's say you went to a restaurant and you ordered some

menu items. And if you say that we forgot about the soup; please order the soup here and the waiter says yes I will add the soup. But if the waiter says that see your order is already entered you can't do anything about it; you can only think you can do is you can delete that order and place a fresh order and that case it is the composition.

In aggregation, you can add new menu items in the order, delete some menu items and so on; whereas in composition, once the order is created, the menu items are created with that and their lifetime is the same. You cannot change any items in between either you can delete the order and place a fresh order.

(Refer Slide Time: 13:32)



With respect to the representation; both are represented using diamond symbol, in case of composition, it is filled diamond and in case of an aggregation, it is empty diamond.

(Refer Slide Time: 13:48)

```
public class Car{
    private Wheel wheels[4];
    public Car (){
        wheels[0] = new Wheel();
        wheels[1] = new Wheel();
        wheels[2] = new Wheel();
        wheels[3] = new Wheel();
    }
}
```

Car ← 1 .. 4 Wheel

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

But how is the composition relationship implemented? Let's say a car contains 4 wheels; when the car is created, the 4 wheels must be created. So, we can have public class Car with just 4 wheels and in the constructor of the Car, we can create 4 wheels. So, when the car is created the 4 wheels are created, cannot have a car with 3 wheels. So, as long as the car exists the wheel exists and when the car is destroyed the wheels get destroyed.

(Refer Slide Time: 14:28)

How to identify aggregation/composition?

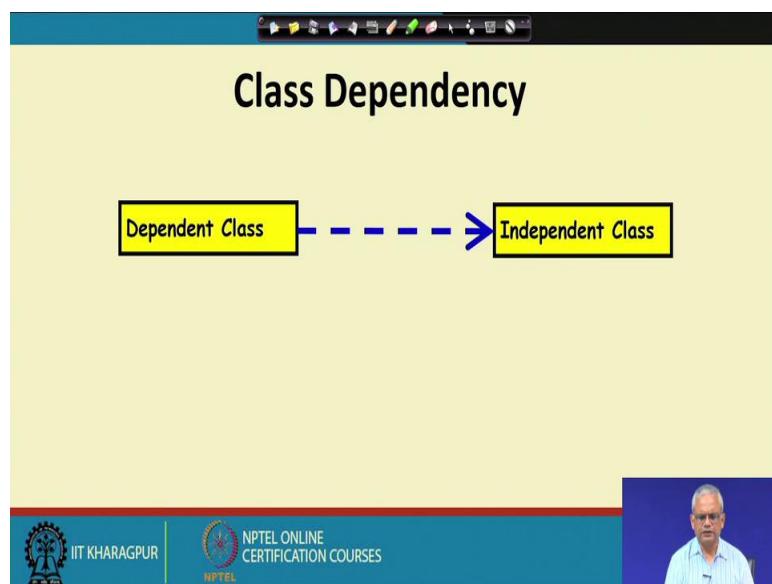
- Lifetime of part is bound within lifetime of composite
 - There is a create-delete dependency
- There is an obvious whole-part physical or logical assembly
- Some properties of composite propagate to parts (e.g., location)
- Operations applied to composite propagate to parts (e.g., destruction, movement, recording)

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, far we have been concentrating on the syntax; how to represent, what is the meaning of the symbol and so on. But now we come to the problem of given a description; how

do we identify aggregation composition relationship? If we find that the lifetime of the composite is the same as the parts, then that is a composition relationship. There must be an obvious whole part relationship and some properties of the composite propagate to the parts; then it's either an aggregation or composition. For example, let's say we have a car which is an aggregate of its parts engine, chassis etc. So, when the car moves the parts also move. When an operation is applied to the composite, it is propagated to the parts. If you destroy the composite, the parts also get destroyed. Let us say we have a polygon. Here the polygon contains many point objects. If we move the polygon by distance x then each of the component objects also move by the distance x. So, the operation is applied on the composite object is also applied to each of the component objects.

(Refer Slide Time: 16:50)



Now, let's see the last type of relationship i.e., the fourth type of relation which is the dependency relation represented using a dotted arrow from the dependent class to the independent class indicating that the dependent class is dependent on the independent class.

(Refer Slide Time: 17:12)

The slide has a yellow header bar with the title 'Dependency'. Below the title is a bulleted list: 'Dependency relationship can arise due to a variety of reasons:' followed by a sub-point '- Stereotypes are used to show the precise nature of the dependency.' A table below the list details five types of dependencies with their stereotypes and descriptions.

Type of dependency	Stereotype	Description
Abstraction	«abstraction»	Dependency of concrete class on its abstract class.
Binding	«bind»	Binds template arguments to create model elements from templates.
Realization	«realize»	Indicates that the client model element is an implementation of the supplier model element
Substitution	«substitute»	Indicates that the client model element takes place of the supplier.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a professor.

There are many reasons why dependency can arise between classes. One is due to abstraction. We have a concrete class which is dependent on the abstract class which means that if anything changes on the abstract class, the concrete classes also get affected by the changes. Whereas abstract class is an independent class i.e., if you change the concrete class the abstract class is not affected.

Similarly, is the binding i.e., if you have the template arguments to create the model element from template. Let's say we have a stack class and we can have an integer stack or we can have a floating point stack or we can have a stack of strings and so on. We can bind the specific type int, float or string etc. to the stack class and there is a dependency and the template arguments.

Similarly the interface realization, if there is an interface class and the client classes are implementation of the interface and if the interface changes, then the client classes need to change. So, there is a dependency of the client class and the interface class; similarly the substitution a class can be used in place of another object.

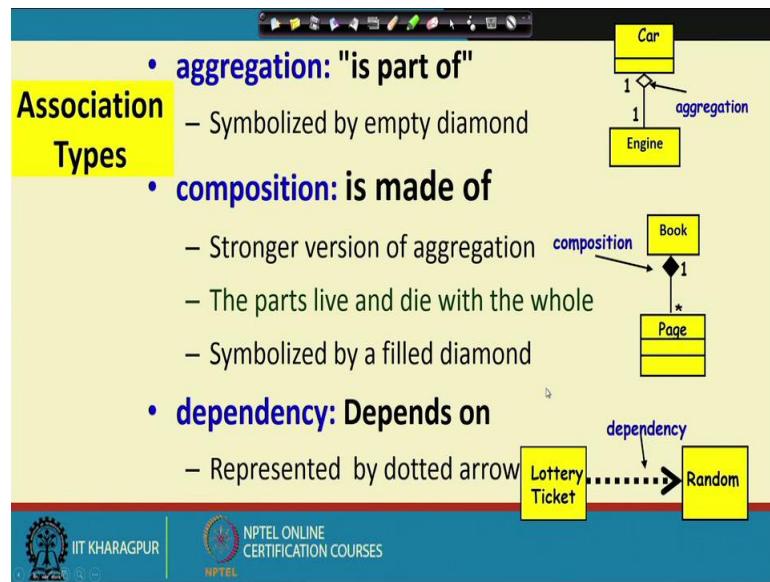
So, there are many reasons why the dependency can arise; may be there about a dozen reasons why dependency can arise and we have just listed few of them here.

(Refer Slide Time: 19:24)

The slide has a yellow background. At the top, it says "Association Vs. Aggregation". Below that, there are two bullet points: "• Is aggregation an association?" and "• Is composition an aggregation?". At the bottom, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a man speaking.

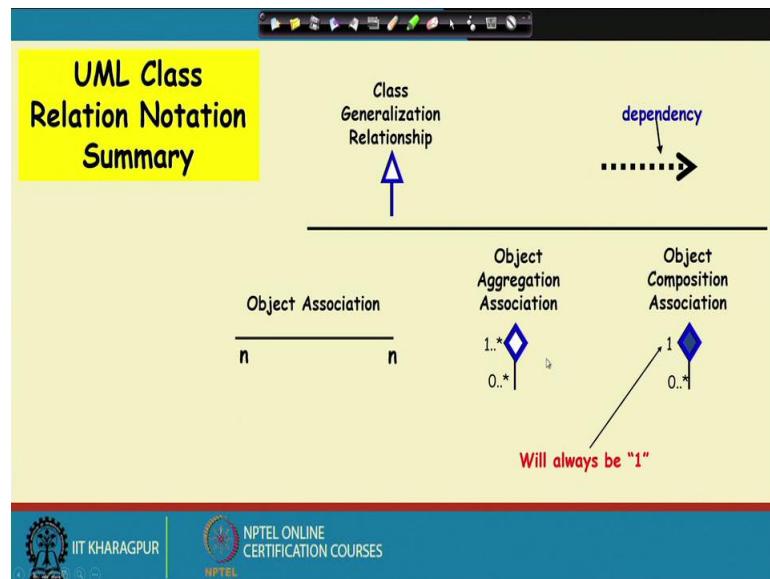
Now, is aggregation relationship an association? Yes aggregation relationship is an association because the aggregate class is associated with its component classes. It can invoke their methods. But what about a composite class, is it an aggregation relationship? Yes, a composition is an aggregation relationship, the similarity is the composite class and aggregate, the component classes have same life time, but the only difference is that the composite class creates its components during its creation and also destroys the parts on its destruction whereas an aggregate class, some of the parts can be created later on and they become part of the aggregate class or some of the items parts may be deleted.

(Refer Slide Time: 20:40)



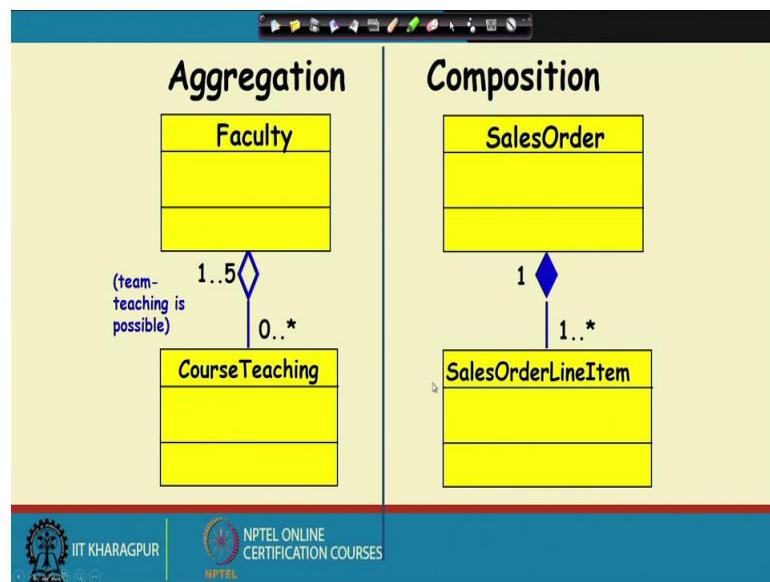
The aggregation relationship is represented by an empty diamond whereas the composition by a filled diamond and the dependency by a dotted arrow.

(Refer Slide Time: 20:58)



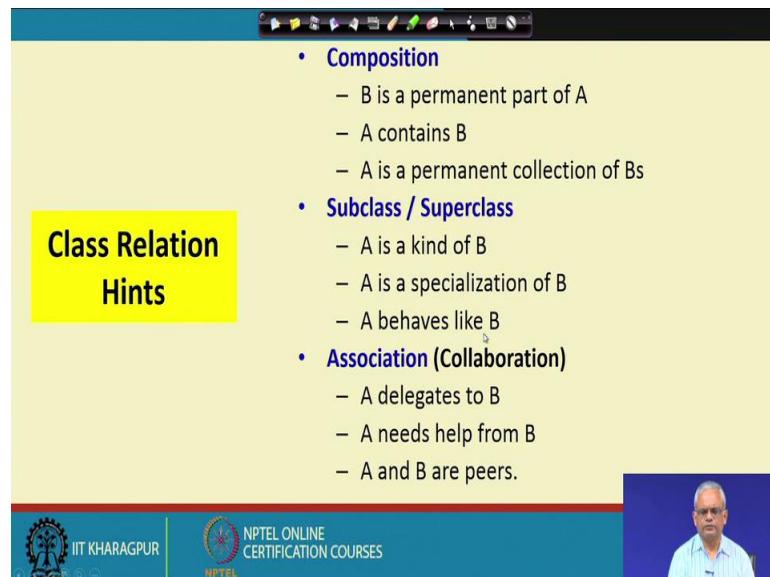
So, this is the summary of the notations we have so far seen the generalization relationship, dependency, association and this the multiplicity we write down the association and the name of the association and the association line aggregation relationship, composition relationship. Here always we have 1 with the filled diamond where as it can be 1..* with the empty diamond.

(Refer Slide Time: 21:29)



Just to give an example a faculty can teach many courses, but a course may be taught by 1 to 5. So, that is a group teaching possible, but here it is always 1 in the diagram above, the sales order contains many sales order items once the sales order created along with the sales order line items are created.

(Refer Slide Time: 22:01)



How do you identify the class relations given a description, how do you identify which is composition, which is the subclass, superclass, generalization, specialization relationship and which is an association relationship? If from the test description you can find B is a

permanent part of A. Or A contains B or we have A is a permanent collection of B, then we can say that is a composition relationship. If A is a kind of B or A is a specialization of B or A behaves like B, then we say that there is a subclass superclass relationship or generalization specialization relationship. B is the base class and A is the derived class. Similarly, if A delegate some responsibility to B; A needs help from B, A and B are peers, if the text contains this kind of terms, then we know they have association relationship between them.

(Refer Slide Time: 23:13)

Class Diagram Inference Based on Text Analysis
(based on Dennis, 2002)

- A common noun implies a class e.g. Book
- A proper noun implies an object (instance of a class): CSE Dept, OOSD, etc.
- An adjective implies an attribute e.g. price of book
- A “doing” verb implies an operation
 - Can also imply a relationship e.g. student issues Book
- A “having” verb implies an aggregation relationship
- A predicate or descriptive verb phrase elaborates an operation e.g. ISBN numbers are integers
- An adverb implies an attribute of an operation e.g. fast loading of image...

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But how do we identify the classes from a text description? Because after all during the design process we need to identify the classes that is an important step in design. When we look at the design methodology, we will see that identifying the classes from text description is an important skill.

Here is a hint how to go about identifying the classes based on the text analysis; grammar analysis you can do and identify the classes. A common noun implies a class for example, book is a common noun, a proper noun implies an object for example CSE department, Object Oriented Software Design and so on. And adjective implies an attribute for example, price of book; a doing verb implies an operation for example, student issues a book. So, issues is an operation; having implies an aggregation relationship, a descriptive verb phrase indicates an operation for example, ISBN are integer; it elaborates an operation. The descriptive verb elaborates an operation for

example, ISBN operation numbers are integers and an adverb implies an attribute or operation for example, fast loading an image. So, it says that; what the characteristics of loading it identifies an adverb fast loading identifies some characteristics of the operation and helps in implementing the operation.

(Refer Slide Time: 25:16)

- Faculty & student Association
- Hospital & doctor Association
- Door & Car Composition
~~Aggregation~~
- Member & Organization Association
- People & student Inheritance
- Department & Faculty Association
- Employee & Faculty Inheritance
- Computer Peripheral & Printer Inheritance
- Account & Savings account Inheritance

Identify Class Relations

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's practice with few simple examples how to identify the class and relation? What are the classes here in the above scenario? The first one, the classes are faculty and student, but what is the relation? The faculty teaches many a faculty teaches by teachers; many students a student taught by many faculties.

Similarly what is the relation between hospital and a doctor? A hospital employs many doctors; a doctor is employed by a hospital. So this is an association relationship. A door and a car; a door is attached to one car, a car has 4 doors. So this is an composition relationship. A member and an organization, a member belongs to one organization; organization has many members people and student. So this is an association relationship. Department and faculty: this is the association relation. Employee and faculty: this is a inheritance relationship; a faculty is a special type of employee. Computer peripheral and printer: this is also an inheritance relationship because a printer is a special type of computer peripheral. Account and savings account: this is also an inheritance relationship because a savings account is a special type of account; we might have current account, fixed deposit account and so, on.

So, given 2 classes we should be able to identify what is the relationship that exists between them; whether it is an association, aggregation, composition. This actually door and car is a composition relationship because a door is permanently attached to a car. So, we can write that as a composition relationship whereas hospital and doctor is an association relationship because doctor may leave a hospital, new doctors may join. Similarly, member and organization is an association relationship, department and faculty is a association relationship. We need more practice to identify the relationship between classes. We are almost at the end of this lecture.

We will stop here and continue in the next class. And we will give you some more exercises to practice, because after all, designing is a skill which is learnt by lot of practice. We will give some assignment. And also in the next lecture we will give you some quizzes so that you pick up the skill and identify the classes and their relationships.

Thank you.

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

Lecture - 37
Interaction Modelling

Welcome to this lecture. In the last lecture we looked at the Class diagram. We looked at the Class relations. There are 4 types of relations that can exist among classes and also we looked at how to identify the classes from a problem description and also to identify relations and to represent them in a diagram. But, we mentioned that even though it appears very simple to identify the classes just by reading a statement and also to identify the relationships, but lot of practice is needed.

(Refer Slide Time: 01:06)

The slide has a yellow background. At the top right, there is a yellow box containing the text "Identify Classes & Relations". To the left of this box is a list of five statements:

- A square is a polygon
- Shyam is a student
- Every student has a name
- 100 paisa is one rupee
- Students live in hostels

Below the list, there is a hand-drawn class diagram. It shows a rectangle labeled "Student" connected by a line with an arrowhead to a rectangle labeled "Hostel". The arrowhead is positioned above the word "live in".

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video window showing a man in a pink shirt.

Towards the end of the last lecture, we had done some practice on identifying class relations. Today, we will start with some more practice. I will display a simple statement. We need to identify the classes and also the relationship that exist between them and if we are able to identify we can represent them in a class diagram. The first one is a square is a polygon.

So, what are the classes here and what relationships exist between these two classes. This is rather straight forward. The two classes are square and polygon and there exists a generalization, specialization relationship; it is an ISA relationship we can see this is a

phrase there which quickly identifies the type of relationship between square and polygon.

But the question is which would be the base class and which is the derived class? Is square derived from polygon or polygon derived from square? So, here the polygon is the base class and square is a special type of polygon. So, the inheritance relationship between polygon and square; polygon is the base class and square is the derived class.

The next one is Shyam is a student. Please identify the class and relations. If you look at this statement, student is a class and Shyam is a object; it is not really a class just an instance of a student. So, Shyam is an instance of the student class. Every student has a name. So, what are the classes here; student is a class and name is an attribute of the student class.

100 paisa is one rupee: here, the 2 classes are rupee and paisa and there is a composition relationship. One rupee is a composite class and it consists of 100 paisa. But then, we need to be careful that this not an aggregation relationship. It is actually a composition relation because a rupee consists of 100 paisa, we cannot make it 99 paisa or 101 paisa. So, then paisa in 1 rupee is fixed and therefore, this is a composition relation and we all know how to represent the composition relationship. Please draw the diagram even though I am not drawing here; please look up the syntax and please draw the diagram rupee is 100 paisa.

Students live in hostels. So, here the 2 classes are student and hostel. We need to use the singular form; the class name is not students, the class name is student and the other class is hostel and there is an association relation between student and hostel. But then, what about the multiplicity of the relation? A student lives in one hostel and a hostel can have many students. How do I represent that in a class diagram? Let me just draw this one. Student and hostel; there is an association relation and the student is the reading direction, student lives in hostel.

But then, we have to give the multiplicity and for that we need to identify one object of one of the class is associated with how many object of the other class and vice versa. So, a student lives in one hostel. So, we can write here 1 with hostel class; but if you do not write 1, by default it is 1. But if you read it the other way, in a hostel many students live.

So, this will be star. So, the correct class relations is association relation between student and hostel and we should get this kind of relation.

(Refer Slide Time: 07:55)

The slide has a yellow background with a black header bar. In the top right corner, there is a yellow box containing the text "Identify Classes & Relations". The main content is a bulleted list of statements:

- A square is a polygon
- Shyam is a student
- Every student has a name
- 100 paisa is one rupee
- Students live in hostels
- Every student is a member of the library
- A student can renew his borrowed books
- The Department has many students

At the bottom of the slide, there is a blue footer bar. On the left side of the footer, there are logos for IIT Kharagpur and NPTEL. On the right side, there is a video player showing a man in a pink shirt speaking.

Now, let us look at the next one. Every student is a member of the library. So, here library aggregates all students is a aggregation relationship because every student is a member of the library and we know that aggregation is a special type of association and if you write this is an association relationship still it's acceptable, but then, we have to give the multiplicity, the association name, reading direction etc. correctly.

Next one, student can renew his borrowed books. Here there are 2 classes, student and book. Student can renew his borrowed books and we can draw an association relationships. So, this implies the student can renew. So, there is an association relationship between two classes, student and book there are two classes here.

Now, let's look at the next one. The department has many students. The 2 classes are department and student. But, we can debate whether it is an aggregation or a composition relation, to resolve this let's see that whether the department when it is created are the students fixed there. Because the implication of the composition relation is that when the whole is created the parts are also created and when the whole is destroyed the part is also destroyed; the parts cannot be added or deleted. But then, to the department, students come and go; students may leave the department, graduate or otherwise they may leave the department, new students join. So, the department and student is an

aggregation relationship, department is the composite class and student is the component and department has many students and that's aggregation relationship, please draw the aggregation relationship between department and student.

(Refer Slide Time: 11:19)

The slide is titled "Identify Classes & Relations". It lists three examples:

- A country has a capital city
- A dining philosopher uses a fork
- A file is an ordinary file or a directory file

Below the list is a hand-drawn UML-style diagram. It shows three boxes: "File" at the top, "ordinaryfile" at the bottom left, and "directoryfile" at the bottom right. An arrow points from "File" down to both "ordinaryfile" and "directoryfile".

The slide footer includes the IIT Kharagpur logo and NPTEL Online Certification Courses information.

Now, let us look at few more exercises. Some more exercises because as I said that we need some practice to identify the classes and the relations. The first one here is a country has a capital. The country has a capital city; the two classes are country and capital city and what's the type of relation the giveaway here is? The has a relation and we know that has a implies aggregation or composition and since the country has a fixed capital city normally, we can make it a composite relation, country has a capital city; it's a composite relation. But then, we may consider that even the capital city can change and in that case if that is allowed then, we can make it a aggregation relationship; has a is typically an aggregation relationship even though it is just one instance. Aggregation is normally many components, but here there is only one, but because of the has a, we can represent it is an aggregation relation.

A dining philosopher uses a fork. So, the two classes are the dining philosopher and uses a fork. So, the dining philosopher is a derived class of the philosopher, if you think so otherwise, we can just write the philosopher is the class and use as a fork. So, there is a association relation between the two classes fork and philosopher and the association relationship it uses.

A file is an ordinary file or a directory file. So, the giveaway here is the ISA relation. A file is the base class and ordinary file and directory file are the derived classes. So, there are three classes here; file, ordinary file and directory file. So, we can draw here file is the base class and ordinary file and the directory file, these are the two derive classes. Now, let's look at the next one. A file contains many records; contain is the giveaway here.

(Refer Slide Time: 14:41)

The slide has a yellow background with a black border. At the top right, there is a yellow box containing the text "Identify Classes & Relations". Below this, there is a bulleted list of statements:

- A country has a capital city
- A dining philosopher uses a fork
- A file is an ordinary file or a directory file
- Files contain records
- A class can have several attributes
- A relation can be association or generalization
- A polygon is composed of an ordered set of points
- A person uses a computer language on a project

At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL. On the bottom right, there is a video player showing a man in a pink shirt.

So, there is an aggregation relationship between file and record; file contains many records. Now, let us look at the next one. A class can have several attributes. Attribute is normally not a class because you do not have methods or operations and private data, public data etc. associated with attribute just one value maybe. So, this is a several attribute, it is not really a class attribute. So, there is class and attribute is part of it. Relation can be association or generalization. So, this is again a generalization, specialization relationship or inheritance relationship. Relation is a base class; association and generalization are the derived classes.

Now let us look at the next one. A polygon is composed of an ordered set of points. Here, the giveaway is the composed of and this phrase indicates that a polygon permanently contains set of points; you cannot just delete some point here, add some point here. When a polygon is created, a number of points in that is fixed and therefore, a polygon is a composite class of many points.

Now, let us look at the last one. A person uses as a computer language on a project. There are three classes here; person, computer language and project. But then, what about the relation among these? Person use as a computer language is an association and again language is used in a project. So, that is again an association, the three classes a person, computer language and project and where association relationship between the person and computer language and computer language and project.

(Refer Slide Time: 17:34)

The slide has a yellow background with a black header bar. The header bar contains several small icons. The main content is a bulleted list:

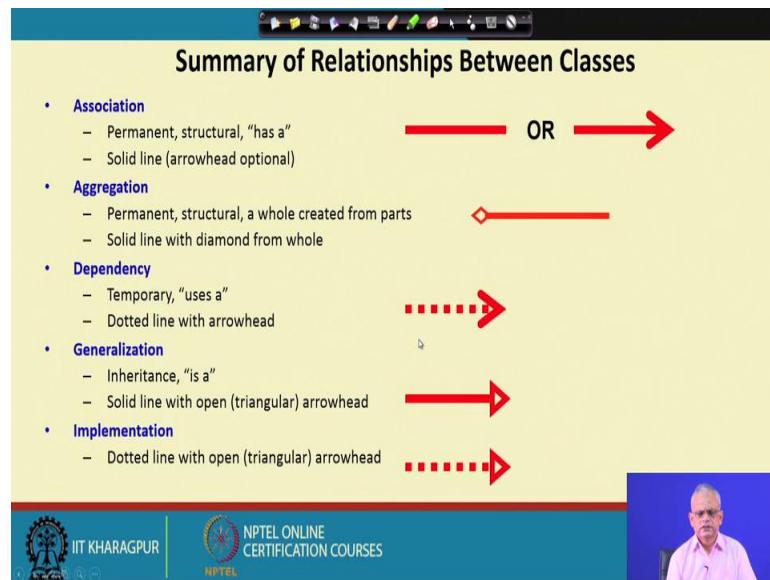
- Describes static structure of a system
- Main constituents are classes and their relationships:
 - Generalization
 - Aggregation
 - Association
 - Various kinds of dependencies

A yellow box labeled "Class Diagram: Recap" is positioned to the right of the list. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the NPTEL logo. On the right side of the footer bar, there is a video player showing a man in a pink shirt speaking.

Now, let's just recapture what we discussed in the class diagrams. The class diagrams describe the static structure of the system. Unlike the object diagrams which is dynamic because as we said that the objects may be created, deleted, linked established destroyed and so on. But the class is the static structure of a system; association, relation, aggregation etc., they are permanent between classes. Therefore, we say that it is a static structure of a system.

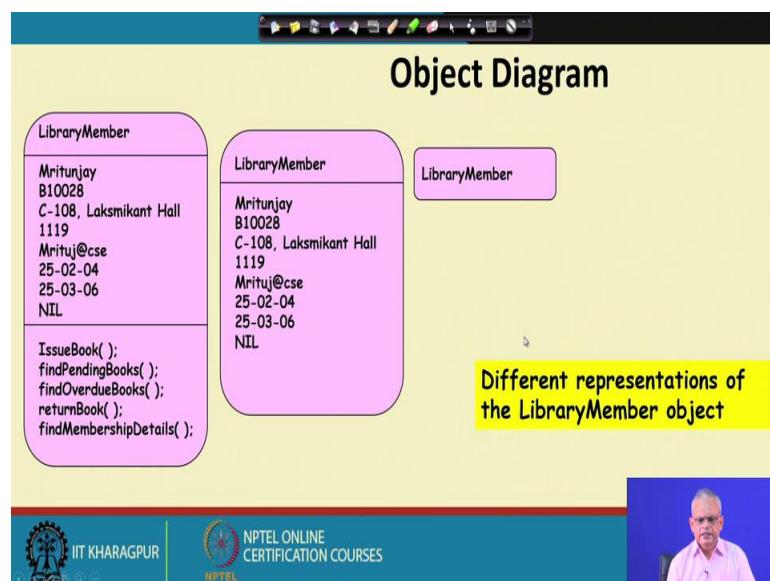
The main constituents of the class relation, the different types of class relations are generalization, aggregation, association, dependency, composition.

(Refer Slide Time: 18:29)



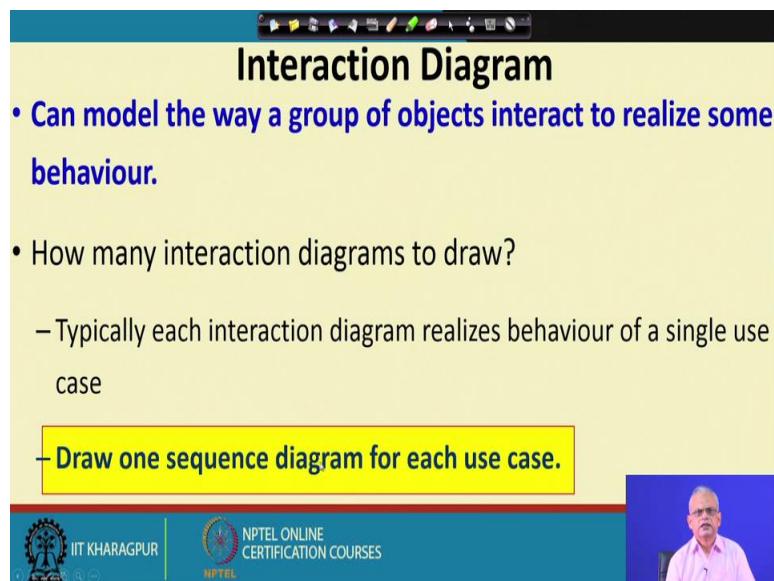
And we had seen the notations for that and Association typically appears as a has a clause in statement and then, it can be represented by a simple line or an arrow; Aggregation relationship a diamond with an arrow. A dotted line with this kind of arrowhead for a Dependency; Generalization is an arrow with this kind of dependency is a line having an arrow with this kind of symbol and then, Implements again here it is similar to a generalization inheritance relation; but here it is a dotted arrow.

(Refer Slide Time: 19:25)



The object diagram just like a class diagram, they can either appear with the name of the object or some attributes of the object or the methods that are available attributes. And here, it's a rectangle with rounded edges. So, far we have looked at the use case diagram, the class diagram and object diagram is simple we did spend too much time. Now, let us look at some behavioral view of a system and will look at the interaction diagram.

(Refer Slide Time: 20:16)



The slide is titled "Interaction Diagram". It contains a bulleted list of points:

- Can model the way a group of objects interact to realize some behaviour.
- How many interaction diagrams to draw?
 - Typically each interaction diagram realizes behaviour of a single use case
 - Draw one sequence diagram for each use case.

The slide footer includes the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". There is also a small video window showing a person speaking.

The interaction diagrams, they can model the way that objects interact during the run of the system to realize some behavior; but then, what can be a behavior of a system? The behavior is when we execute it done something for us for example, issue book, return book. So, that is a behavior of the system and if we think of it the behavior is either execution of a use case or part of a use case.

So, we model here using interaction diagram that when use case executes what are the objects that take part and what messages they interchange among each other. But the question here is that when we model a system, how many diagrams to draw; how many interaction diagrams? The answer is simple, we draw as many use cases are there because each interaction diagram captures the behavior of a single use case and therefore, the number of interaction diagram is typically equal to the number of use cases. But of course, if the use case is very complex, we might split into multiple interaction diagrams; but if we have designed well, then we might have decomposed the use cases so that none of the use cases is very complex.

So, therefore, the number of interaction diagram should be equal to the number of use cases identified and we must remember so very important statement here that we need to develop one interaction diagram for every use case. We will see that interaction diagram either a sequence diagram or collaboration diagram and we need to draw one sequence diagram for each use case.

(Refer Slide Time: 22:46)

A First Look at Sequence Diagrams

- Captures how objects interact with each other:
 - To realize some behavior (use case execution).
- Emphasizes time ordering of messages.
- Can model:
 - Simple sequential flow, branching, iteration, recursion, and concurrency.

119

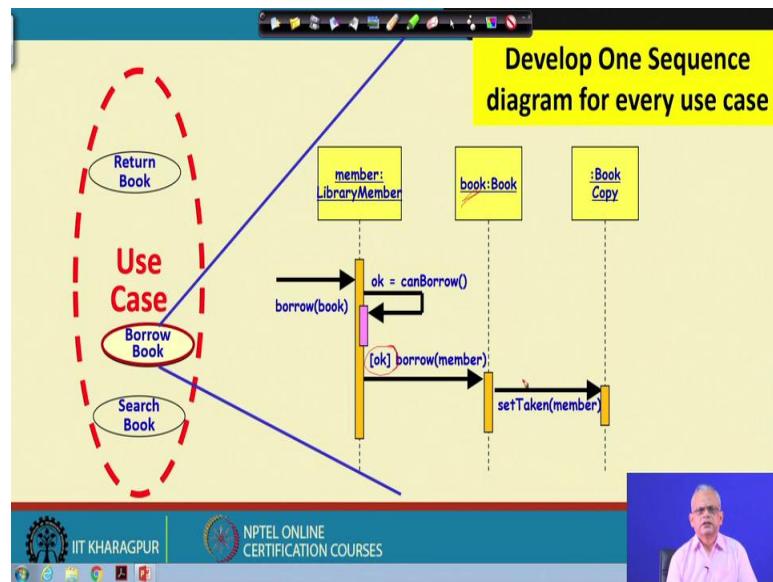
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

A small video window in the bottom right corner shows a man speaking.

The interaction diagrams capture how the objects interact during the run of the software to realize some behavior; for example, use case execution and here we need to identify what are the objects that are interacting during the execution of the use case and what is the messages they are exchanging among each other and the time ordering of the messages.

And we will see that we will have ways to represent some control information like whether there is sequential flow from one object to another object, the control just flows from one object to another object. There is a branching that is depending on some condition there; control flow occurs from one object to another object or some other object. Iteration multiple times control passes from one object to other object and returns, recursion, concurrency and so on.

(Refer Slide Time: 24:12)



But we must remember that for every use case, we need to develop one sequence diagram. Sequence diagram is one type of the interaction diagram. There are actually 2 types of diagrams; the sequence diagram and the collaboration diagram. If this is our use case model; then, we need to develop one sequence diagram for each of these use cases.

We can take any one use case and draw the sequence diagram for this, the sequence diagram will look something like this that what are the objects that participate during execution of the use case and what kind of messages they exchange and the time ordering of those messages. So, here this diagram at the top, we mention the objects that interact. Even though, it looks like a class symbol, but then just look here we have underlined here library member is a class name and member is the object when we underline that.

This one is a instance of the book class, small book is a instance of the book class and then, here just see here in this, we have not even mentioned what is the name of the object and that is because it is an anonymous object; any book copy . And the diagram is typically read from top to the bottom. This we call as the timeline and this arrow is here capture the sequence in which the messages are exchanged between various objects of these classes as the interaction between them occurs during the execution of the specific use case.

So, to start with, the user issues a borrow book request and he needs a specific type that is book as given as the argument here and then, this is a self method here and checks whether the member is eligible to borrow book; has not exhausted his limits, number of books that he can issue etc. And just see here there is a recursion here because the same one calling here and then, this small rectangle here is an indication that the object is active now it is executing its code.

And then, depending on whether he can borrow the book that we write here in the form of this control statement that ok with the condition is true then invoke the borrow member. So, the book class the specific book, I mentioned here that this is a book here and then, the book for the book class, the method borrow is activated and the member class waits here for the result to return and that is how we do not really stop this rectangle here it continues. And once it comes to the book, the control comes to the book, it gets active and then the book has many copies and the specific copy that would be issued any one of the copies the member who is the issuing it, it is the registered member there.

So, that next time query we query that who has taken the book copy, then you can find that the member has taken the specific member who was requesting has taken the book. So, this is the essence of a sequence diagram. Basically we will identify what are the objects and for execution of a use case in what way they interact; which messages the exchange and in what time ordering. We are at the end of this lecture. We will stop here and continue in the next lecture.

Thank you.

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

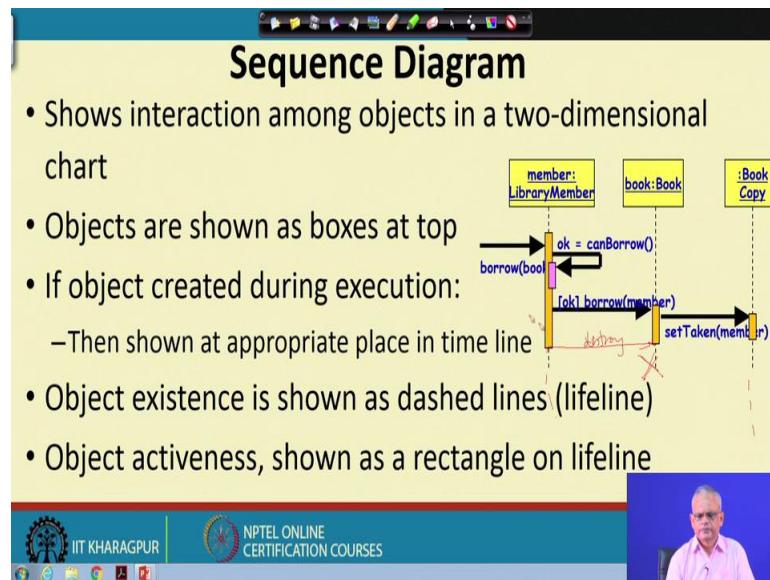
Lecture - 38
Development of Sequence Diagrams

Welcome to this lecture. In the last lecture we discussed about how to identify the classes and their relations and also started to discuss the interaction diagrams. The interaction diagrams are a very vital diagram; these are developed for every design. Even for a small system, we want to develop, we need to develop the class diagram and the interaction diagram and the use case diagram.

So, these three diagrams are developed for all systems whether it is trivial or large scale, but the other diagrams for example, the state chart diagram, activity diagram, etc. are developed for specific cases for example, if the classes have significant number of states we need a state diagram, activity diagram, when a use case is complex or we need to have a overall understanding of a very complex software, we develop activity diagram.

But, interaction diagram is developed irrespective of the problem for all types of problems we need interaction diagram, and we will discuss about the nitty gritty of the interaction diagram and see what purpose they serve in a design. There are two important types of interaction diagrams; one is called as a sequence diagram and the other is collaboration diagram. Of course UML 2.0 supports few other types of interaction diagram for example, interaction overview diagram and so on, which we will not discuss in this lecture. Let's look at the sequence diagram and see how we can develop a sequence diagram and what role it plays in a design process?

(Refer Slide Time: 02:25)



The sequence diagram as we discussed in the last lecture, it shows the interaction among the objects in a 2 dimensional chart. This is the interaction that occurs when a use case is executed, the objects are shown at the top and if an object is created during the execution, these are showed at the appropriate place in the diagram, there is a lifeline for every object. So, if the object exists, the lifeline exists and if the object is destroyed we put across on it is lifeline and it does not exist. When an object becomes active, it's shown on as a rectangle on its lifeline. So, this is an example of a sequence diagram just see here that the objects are shown at the top of the diagram, and then we have the lifeline for each object as long as the lifeline exist, this dotted line exists and the object exists.

If any time, object gets destroyed for example, another object member destroy method on that. And then we just draw a cross here and stop the lifeline there indicating that the object no more exists. So, with this call say destroy, we will put a cross here and the lifeline does not exist for this other lifeline exist. And we have this small rectangle on the lifeline indicating that the object has got the control and it is in execution state. And we write the messages on the arrow here. And as the messages are sent or method call takes place, then the control is transferred and the other object becomes active.

(Refer Slide Time: 05:05)

Sequence Diagram Cont...

- Messages are shown as arrows.
- Each message labelled with corresponding message name.
- Each message can be labelled with some control information.
- Two types of control information:
 - condition ([])
 - iteration (*)

```

sequenceDiagram
    participant member as member: LibraryMember
    participant book as book: Book
    participant copy as :Book Copy
    member->>book: borrow(book)
    activate book
    book-->>copy: [ok] borrow(member)
    deactivate book
    copy->>member: setTaken(member)
  
```

The sequence diagram illustrates the interaction between three objects: member (LibraryMember), book (Book), and copy (:Book Copy). It begins with a synchronous message 'borrow(book)' from member to book. The book object then sends a return message '[ok] borrow(member)' back to member. Finally, the book object sends a synchronous message 'setTaken(member)' to member.

IIT Kharagpur | **NPTEL ONLINE CERTIFICATION COURSES**

Now, the messages are shown as arrows. It is seen that each message is labeled with the message name, and for each message name we can have some control information. For example, depending on some condition the message will be invoked or not. There are two types of control information the condition and iteration. So, message is sent many times or a message will be sent depending on some condition. So, here we have this condition here. OK is the condition, if is true, is computed here can borrow returns here. So, borrow is the method that is executed for the library member. And then it is the return that is obtained that is how we represent here. And then based on the condition, we check whether indicates true or false and if it is true only then borrow member executes.

(Refer Slide Time: 06:25)

The diagram illustrates the Gist of Syntax for UML2 interaction frames. It shows a sequence of messages between three objects: member: LibraryMember, book: Book, and :Book Copy. The process starts with member sending a 'borrow(book)' message to book. A condition 'ok = canBorrow()' is evaluated. If true, the message '[ok] borrow(member)' is sent back to member, which then sends a 'setTaken(member)' message to :Book Copy. A callout box labeled 'Gist of Syntax' highlights the use of iteration markers (*), conditions ([]), and self-delegation.

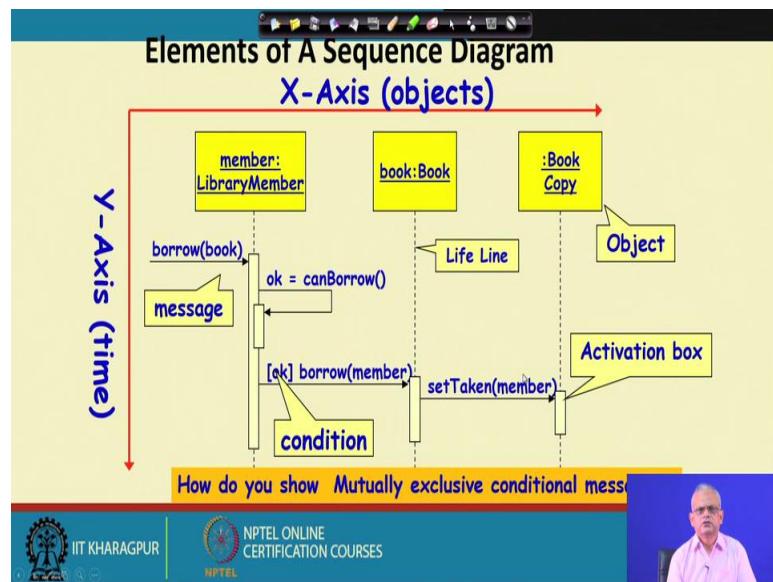
The gist of the syntax that we discussed for a sequence diagram is simple, we show star on a condition the condition may not be there also, and then we say that it is iteration. Star indicates the iteration condition for all objects etc. maybe there or may not be there. And then to represent condition we use the square bracket and the message is sent only if the condition is true, when a self delegation that is the object calls a method of its own loops and conditionals we can use either the star notation or the square bracket, but then UML 2 uses a new set of notations called as interaction frames.

(Refer Slide Time: 07:30)

The diagram illustrates Control logic in Interaction Diagrams. It shows a conditional message 'variable = value' message() and an iteration message '* [i := 1..N] message()'. A callout box highlights that the message is sent many times to possibly multiple receiver objects.

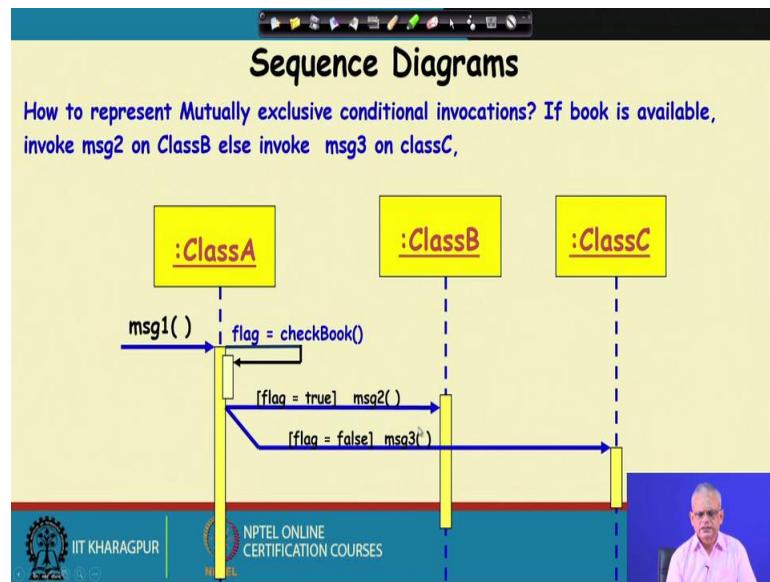
We had seen that we can use a conditional message; we can set Boolean value here in the conditional message. If variable has some value then the message is sent. Then, we have the looping here with star message. The star is the requirement for looping and the condition may not be there.

(Refer Slide Time: 08:01)



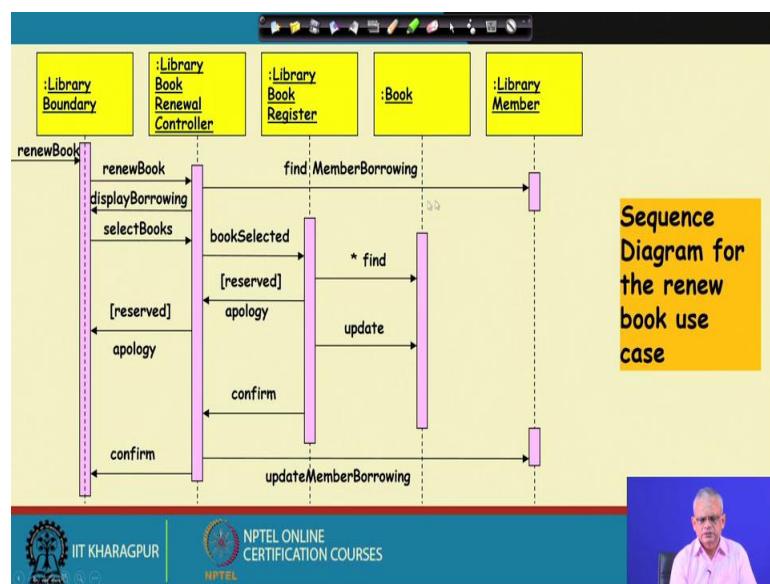
So, let's look at the sequence diagram again. This is a 2-dimensional chart, the X-Axis indicates the object, the Y-Axis is the timeline and the dotted line here is the lifeline for the objects. The objects have been drawn at the top that means, at time 0 before the use case starts the objects are already created and they exist. So, at time 0, the objects exist and therefore, they have been drawn at the top of the diagram. And time during the execution of the use case elapses from top to bottom. First the borrow book is invoked on the member object and then it invokes a self-method, method of its own that is can borrow and we read the diagram from top to bottom. So, these are the objects, this is the message, this is the conditional, and this is the activation box(as labelled in the figure above), but the question is that let's say a message will be sent either to one object or another object not to both objects. Mutually exclusive sometimes a message will be sent to anyone or to the other and not to both, how to do we represent that?

(Refer Slide Time: 09:39)



Let's say, if a book is available then we invoke some issue on class B and if not available we invoke record the request on class C, how do we represent this? So, we draw the 3 three classes class A, class B, class C and then we have a flag set which is written by execution of a self-method checkbook and depending on if the flag is true message2 is sent to class B or if the flag is false message3 is sent to class C. And therefore, both the classes will not be invoked, they are mutually exclusive. Hence, one of the classes will be invoked.

(Refer Slide Time: 10:36)



This is another example of a sequence diagram; let's see how to read this diagram. So, this diagram represents the behavior that occurs when the renew book use case is executed. We can also say in alternate one that, this sequence diagram is a implementation of the renew book use case. Here, the objects are shown at the top and just see here colon name of the class under line that means; that these are anonymous objects i.e., any object of this.

The execution starts by the renew book which is invoked and the library boundary. And this in turn in books i.e., the renew book and a renewal controller becomes active and invokes the find member borrowing on the member. It checks, what the books they have been borrowed are and just see here in the library member object that there is no return arrow here. The return arrow that is the result that is return by the library member is implicit.

But of course, if you want to show that some variable that is used later in a decision or something, we made draw an arrow back arrow here just like we have drawn back arrow here. We might sometimes draw a back arrow, but then these are normally omitted. This invokes and a result that is return is not represented normally. And see here there is a control information here that is find the book among all books. So, that is represented by iteration i.e., by condition, if book is reserved, then generate an apology and is displayed. So, we can see that the different objects that participate in the execution of the use case and the message there if seen among each other for execution of the use case and the time ordering the messages is represented here.

(Refer Slide Time: 13:31)

Example: Develop Sequence Diagram

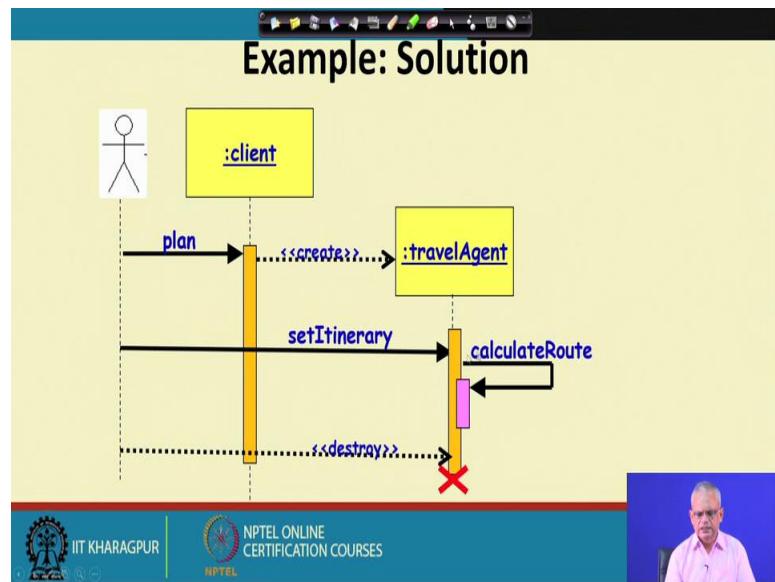
- A user can use a travel portal to plan a travel
- When the user presses the plan button, a travel agent applet appears in his window
- Once the user enters the source and destination,
 - The travel agent applet computes the route and displays the itinerary.
 - Travel agent widget disappears when user presses close button.

Now, let's look at few more examples. The idea is that given a statement like this that is a details of a use case execution, we need to identify the objects that interact with each other and develop the sequence diagram. So, let's see one example here a user can use i.e., travel portal to plan a travel. The travel portal is one object, normally the travel portal is a website and therefore the website appears on our client machine and we can even consider this as a client object. The user presses a plan button on the travel portal as the travel portal appears on the client machine and when a presses a plan button a travel agent applet gets created and it appears in his window, and then on this, travel agent the user enters source and destination.

So, the user interacts with the travel agent and enters the source and destination on it. Based on the source and destination, the travel agent computes the route and displays the itinerary. And as soon as the user presses the close button, the travel agent wizard disappears. So, this a simple sequence diagram, where there are two classes here, a travel portal and depending on when the user presses the plan button and the travel portal it will create a travel agents.

The travel agent object does not exist to start with and only when the plan button is pressed the travel agent button appears. Then the user enters the source and destination and the travel agent and the travel agent invokes a self-method to compute the route and then it displays it, and it disappears in the user presses the close button.

(Refer Slide Time: 16:21)



So, this is the user presses the plan button in the client and then it creates the travel agent object that does not exist to start with and only when the plan button is created, it gets created and appears. The creation of an object, we represent using this kind of notation i.e., a dotted arrow with the stereotype create and the point at which create is invoked the object appears in the timeline. It does not exist a time 0.

But, only when the create method is called, the user sets the itinerary on the travel agent and it calls a self-method calculate route, displays the route and that is implicit here. And then the user invokes the close or destroy and the object gets destroyed here. So, this is the representation object getting destroyed.

(Refer Slide Time: 17:37)

Return Values

- Optionally indicated using a dashed arrow:
 - Label indicates the return value.
 - Don't need when it is obvious what is being returned, e.g. `getTotal()`
- **Model a return value only when you need to refer to it elsewhere:**
 - Example: A parameter passed to another message.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | 130

Normally, we do not represent the return values, but sometimes as I said that we need to use the specific value returned in the diagram itself, then we need to use this kind of dotted arrow. The idea behind using this arrow is that we should not complicate the diagram too much and only if we need the return value in the diagram we represent it. Otherwise, it is understood or implicit. We do not represent the return values.

(Refer Slide Time: 18:20)

Method Population in Classes

- Methods of a class are determined from the interaction diagrams...

Sequence diagram illustrating method population:
Participants: `:RegistrationForm`, `:RegistrationManager`, `RegistrationManager`
Interactions:

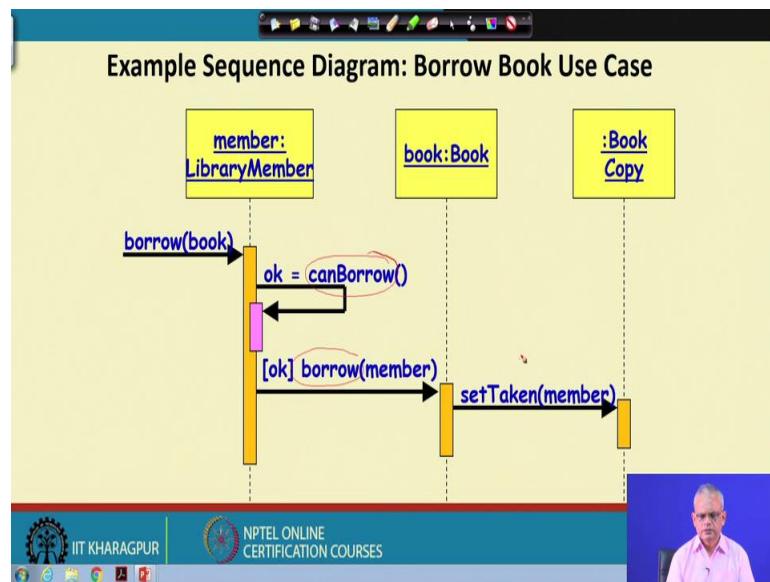
- `:RegistrationForm` sends a synchronous message `add course (joe, math 01)` to `:RegistrationManager`.
- `:RegistrationManager` returns a value (represented by a dotted arrow) to `:RegistrationForm`.
- `:RegistrationManager` sends a synchronous message `addCourse(Student, Course)` to an external `RegistrationManager` object.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see if we develop a sequence diagram, how is it useful in our design? First let's start with a very simple use case diagram. There are two objects i.e., registration

form and registration manager and then we have just added a method here between these two methods call add course with some argument. The implication of this on our design is that we have the registration manager that must have the add course method and it must take these two(joe, math 01) as it is parameters.

(Refer Slide Time: 19:08)



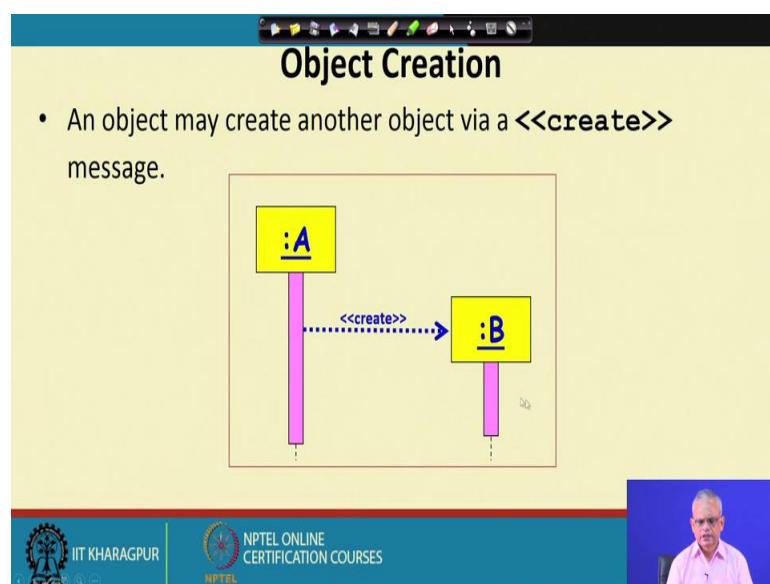
So, it must have the registration manager if you draw this kind of message. Then the registration manager must have the method add course. Only, if it has the method add course the registration form object can invoke the method and then the argument to this method at the student and the course. So, we can think that the role of the sequence diagram is to populate the methods in the classes. To start with, we have just the class name, but as we draw the sequence diagram, the methods are added to the classes and most case tools they do that very efficiently.

As, we in the case tool draw the sequence diagram, we can see that in the class diagram, the corresponding methods are been incorporated. At the end of drawing all class diagram we will know that which classes have what methods, because we will draw many sequence diagrams. As many as there are use cases and after we have drawn the sequence diagram for every use case, we will see that the same class might be participating in different use cases. Therefore, the methods in different use cases get executed and they get added here.

So, this is the borrow book use case, now can you identify what are the methods of the different classes that will be populated based on this diagram? What about Borrow book will be a method of the library member class, the name of the method will be borrow and it takes a book type as its argument. What about canBorrow? canBorrow is also a method of the library member class?

So, from here based on this sequence diagram the library member class will get two methods borrow and canBorrow, but what about the borrow here, that is the book class will also have a method called as borrow and here the argument is a member unlike the borrow method of the library class which is argument of a book, you are the borrow for the book class, the argument is a member and then what about the set taken, the set taken is a method of the book copy class.

(Refer Slide Time: 22:29)



We had seen the object creation, we need to draw the dotted line and then write the <<create>> on it and then once this is invoked, object is created an anonymous object, we write the name of the class colon name of the class underscore is an anonymous object any object of this B class, will be denoted by :B and this comes into existence at this point of time which did not exist at the beginning of the use case execution.

(Refer Slide Time: 23:05)

Object Destruction

- An object may destroy another object via a <<destroy>> message.
 - An object may also destroy itself.
- **But, how do you destroy an object in Java?**
 - Avoid modeling object destruction unless memory management is critical.

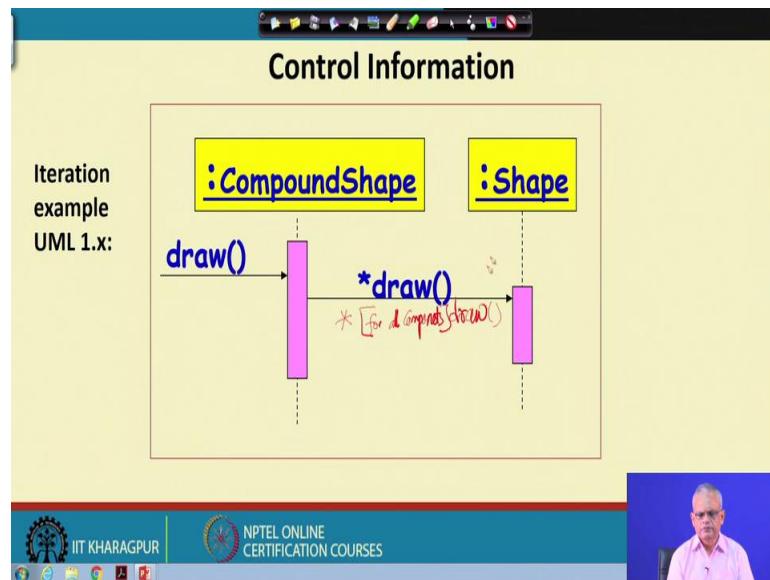
```
sequenceDiagram
    participant A
    participant B
    A->>B: <<destroy>>
    activate B
    deactivate B
```

IIT KHARAGPUR NPTEL ONLINE
CERTIFICATION COURSES

Object destruction the object B existed, but then when A invokes the destroy method on the B the class of B the object B class, then its lifeline terminates and objects see just to exist that is these the notation to show that, we draw a cross on it is lifeline and ceases to exist. And it is possible that an object may destroy itself, but then just a question, that how does somebody destroy an object in Java?

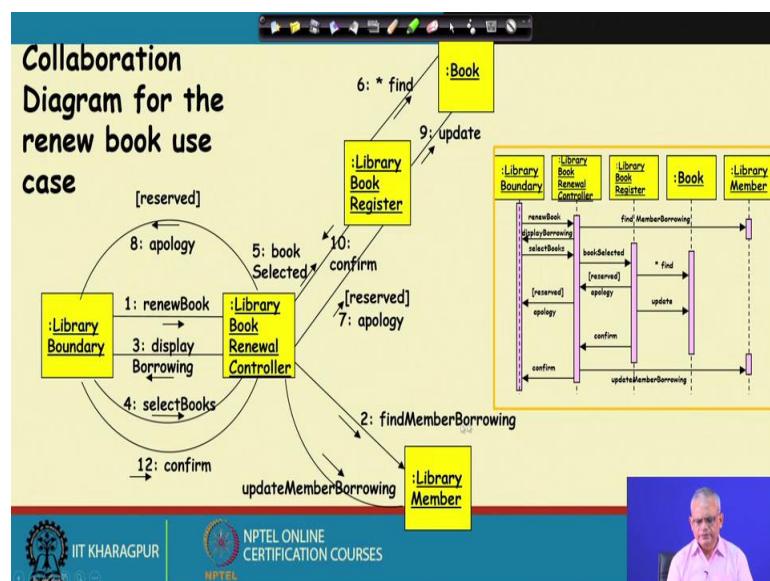
Is there a method called that you can do to destroy an object in Java? Normally the objects do not have a method to destroy, they just fall out of scope and then they destroyed and typically we do not model to that extent that when objects are destroyed and so on unless memory management is critical.

(Refer Slide Time: 24:19)



Let's see the control information that is star(*) indicates multiple times. So, the draw method when it is invoked the compound shape, it calls the draw method on all its shape, we can also write here star for all components to make it clear draw. So, for all components it draws that's the indication with this star and the compound shape consists a many shapes and the draw that is paint itself on the screen for all components.

(Refer Slide Time: 25:18)



A collaboration diagram is another form of the interaction diagram and this diagram is automatically generated from the sequence diagram. In most of the case tools by press of

a button you get the collaboration diagram and this is just another view of the sequence diagram. Let's see how the collaboration diagram is created, if this is the sequence diagram in the above figure then here we write all the corresponding objects here and then here there is a time ordering of the messages. There is no time ordering of the messages, but we add number in the front of the messages or method call to indicate the order in which it occurs. For example, the start with the boundary object invokes the renew book on the renewal controller. So, this is the first message and therefore, library boundary library book renewal control, we write one here. This is the first message to be interchanged. The next will be the renewal controller that invokes the find member borrowing and the library member. So, this is the library member and the book renewal controller next invokes the library member borrowing. Therefore, we write 2 here and normally the direction of the invocation is shown by a small arrow here, we just draw a line of cursor, an arrow drawn here it should not be, it is just lines and then we just write a small arrow here to indicate the direction of the invocation.

Then, we take the next one display borrowing between these 2 number that 3 library book controller and then library boundary we will make it 3 we draw it and so on, we take one by one and draw this. As we can see that it can be automatically drawn based on the sequence diagram, the collaboration diagram can be automatically done drawn.

But, then what purpose does it serve? Can you guess; because it supports this diagram it must serve some purpose. The purpose it serves is that it clearly shows the class associations. If it invokes the method that they are associated with and we do not even show the arrow here by just draw a plane line and remember that the association symbol.

So, as we draw the sequence diagram we get the class associations. Even though we said that associations are identified just by inspecting the problem statement, but often we cannot identify all associations and they get captured here, when we draw the sequence diagram. We are almost at the end of this lecture; we will stop here and continue from this point in the next lecture.

Thank you.

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

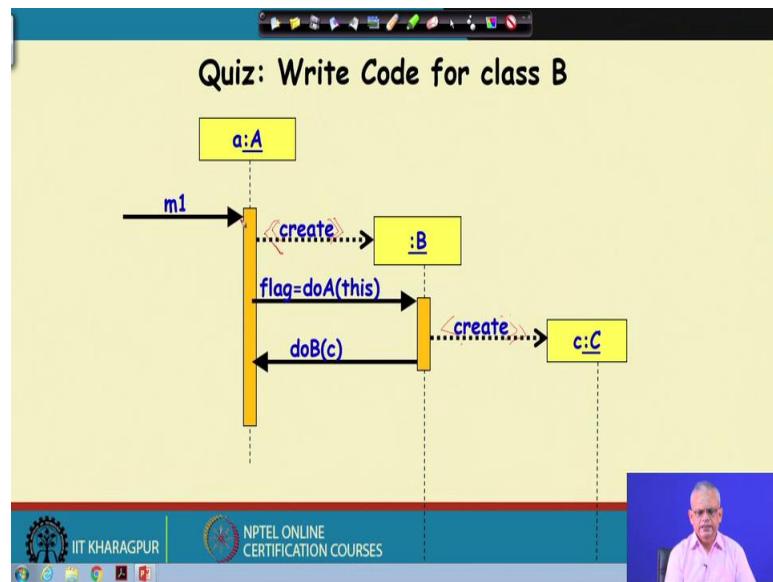
Lecture - 39
State – Machine Diagram

Welcome to this lecture. In the last lecture, we had discussed the interaction diagrams. We said that, there are two main types of interaction diagrams, the sequence diagram and the collaboration diagram. The sequence diagram captures how the objects interact during execution of a use case. We also said that the sequence diagram plays a very important role in any design process. The role is that, it helps in identifying which classes have which methods that we call as method population in the classes.

To start with the design process we identify the classes that exist and later we identify the methods and then we identify the attributes. So, the sequence diagram has a very important role in identifying which classes should support which methods. And then we saw the collaboration diagram is automatically derivable from a sequence diagram and most case tools they just display the collaboration diagram based on just a press of a button. The main purpose of the collaboration diagram is that it shows the association relation between different classes.

Now, we know that based on the sequence diagram, we can also write the code for a method, because we know that when the method is invoked on a class what it needs to do. It might call a self method or call another method some argument etc. Therefore, some code also get developed for different classes i.e., not only that the method prototypes are developed for every class, but also the method skeleton code gets developed. Now, let us do it through a small exercise that is simple enough, given a diagram would be able to develop the code, because it is very simple. Let's see that we have this sequence diagram below.

(Refer Slide Time: 02:59)



There are three objects which participate in the sequence diagram, method m1 is first invoked in object-a of the class A. And then it invokes the <<create>>. It invokes the create on the B class and anonymous object gets created here and then it calls the doA method on the anonymous object of the B class with argument this and remember this is the small a itself. And then it returns a flag and in turn the object of the B class. It creates an object of C class called as c: name of the object that is created is c, and then it calls the doB method of the A class with argument c.

So, what will be the methods for different classes and can we write some of the code here. We know that m1 is a method of A class, doA is a method of the B class of course, create that is constructed for the C class. Now, the question is that can we write the code for class B? So, that is for doA method. So, we know that class B contains doA method that takes an object of class A as argument and then inside the method it will create the class C. So, it will call the new operator in java for the class C and then call doB method of the class A.

(Refer Slide Time: 05:33)

```
Quiz: Ans
public class B {
    ...
    int doA(A a) {
        int flag;
        C c = new C();
        a.doB(c);
        return flag;
    }
}
```

The slide includes the IIT Kharagpur logo, NPTEL logo, and NPTEL Online Certification Courses text at the bottom.

So, if we write the code neatly we will have class B and then it has the doA method, it returns a flag integer and then the first it creates a c object and the name of the object created is small c. And it calls the doB method of the a object with c as the parameter and then it returns a flag.

(Refer Slide Time: 06:01)

State Machine Diagrams

The slide includes the IIT Kharagpur logo, NPTEL logo, and NPTEL Online Certification Courses text at the bottom.

So, based on the sequence diagram, we not only can populate the methods, but some skeletal code for the different methods can also be created.

Now, let's look at the state machine diagram. This is also an important diagram, even though it is not developed for every class in every problem. If the classes have significant states, then we show that in the form of a state machine. The state machine also helps us to generate some code, because our idea is that once we do the design, we should have some code written for us. Most case tools do that good case tool generate lot of code and we just need to fill few of the code there, and we saw that how the interaction diagrams help us to generate the code.

Now, let's look at the state machine diagram and see that if the classes some classes have significant states, then we can develop the state machine diagram and at the same time the code for that class will get generated. We will see the nitty gritty of the state machine diagram and also see the code that a state machine diagram can generate.

(Refer Slide Time: 07:47)

State Chart Diagram Cont...

- State chart avoids two problems of FSM:
 - State explosion
 - Lack of support for representing **concurrency**
- A hierarchical state model:
 - Can have **composite states** --- OR and AND states.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The state machine diagram as it is called in UML 2.0 is based on the state chart diagram. The state chart diagram is a refinement of the finite state machine that we know all about. The state chart diagram was proposed by David Harel long back maybe 30 years back to handle two problems of the finite state machine. Even though we use finite state machine for small problems, but if we use it in object orientation in modelling classes, systems and in many other application, it suffers from two main problems.

The FSM suffers from two main problems one is called as the state explosion, that is the number of states becomes too many 100s, 1000s, and it becomes very difficult to draw

the diagram on a page or a on a screen and also to understand the diagram. The second problem is that the state machine is just a sequential diagram; states get activated one after other we do not have concurrent states. The state chart diagram over comes these two basic problems. We will see that it is an elegant formalism and UML uses the state chart diagram and calls it as the state machine diagram.

The main improvements to the finite state machine is hierarchical state. A finite state machine is a flat state machine, if you draw an FSM, let's say we draw it like this that we have two states and then there are some transitions. It is a flat state machine with State S1, S2 and then some transition based on some event, some transition takes place and we have some action e1 causes this transition e2 causes this transition.

But, in a state chart diagram we have hierarchical states that is a state here can have further states. So, even though the top level we have this state machine, but we can look at any state and that also contains a state machine. And this may contain a state machine, but we can look at any state and that also contains a state machine and this may contain state machine. So, there is a hierarchy and if you look at this state in the second level it might also contain a state machine.

And this is the one mechanism hierarchical state to handle the problem of state explosion, earlier as the number of state variable increase the state efficient became extremely complex, number of states increased exponentially with the number of state variables. We will just see that with an example that how the states increase exponentially with state variable.

But, here we address the problem of state explosion by having a very simple diagram with few number of states and transition and then we use the abstraction mechanism. We said in the beginning of our discussion on software engineering that wherever things become very complex we use either abstraction or the decomposition to handle that. Here we use the abstraction mechanism and that is done through hierarchical state model. So, instead of making the diagram flat and very complex we have a hierarchical diagram and the top level is simple and the complexity are slowly added over the hierarchies. Here for representing the concurrency, we have composite state that is within one state we can have a concurrency that is the object can exist in two different states within one state let's we will look at that.

(Refer Slide Time: 13:20)

The slide is titled "Robot: State Variables". It lists seven state variables:

- Movement: On, OFF
- Direction: Forward, Backward, left, Right
- Left hand: Raised, Down
- Right hand: Raised, down
- Head: Straight, turned left, turned right
- Headlight: On, Off
- Turn: Left, Right, Straight

Next to the list is a drawing of a blue and orange robot with the word "HELLO" on its chest. To the right of the list is a box containing the question "How many states in the state machine model?" followed by handwritten calculations:
$$2 \times 4 \times 2 \times 2 \times 3$$
$$\times 2 \times 3$$
$$= 384$$

At the bottom of the slide, there is a footer with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video window showing a person speaking.

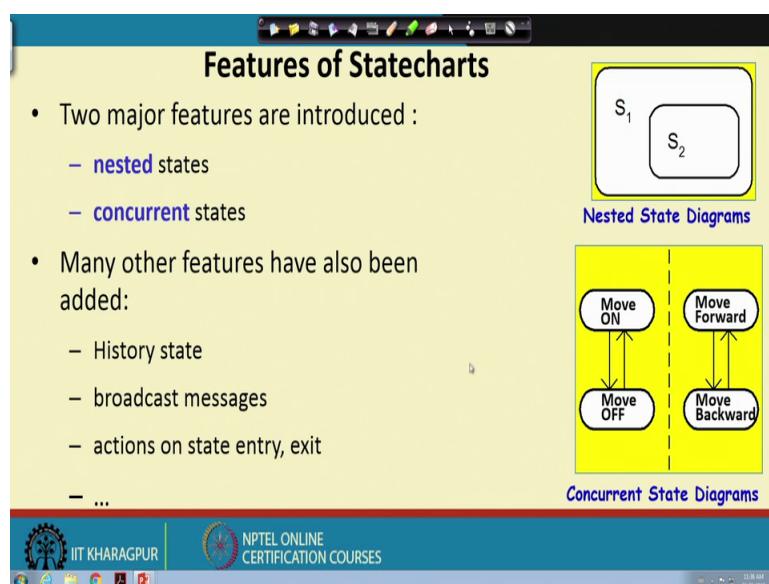
Let's first look at an example and understand the state explosion problem in a FSM. This robot can make movements, the first state variable. Let's try to compute the number of states in modeling a robot is a behavior. Let's assume that there is a switch to have the movement of the robot on or off, the robot can move if the movement is on, movement can be off also. So, there is a state can be that movement on and movement off.

Now, there is another state variable called as a direction and the direction can be forward, backward, left and right. So, this state variable direction can take 4 different values and these are 4 states. So, based on this two we will see that the state for movement there are only 2 states that is the robot movement on, robot moment off. But, now with the second state variable we have 8 states that is robot movement on with direction forward, robot movement on with direction backward, robot movement on with direction left, robot movement on with direction right, and similarly robot movement of with forward backward etc. So, with those 2 state variables we have 8 states.

Now, that can be further states. For example, the left hand raised or down. So, these are two states. Now, the number of state variable becomes $2 * 4 * 2$, because for each one here each state here we can have 2 cases that left hand raised the down. Similarly, the right hand raised the down. Similarly, will have head straight or turned left the turned right, you can have the head light turn on and off and we can have the turn left right or straight.

So, it may be proceeding straight or it may be turning to left or right. So, to model this robot, how many states we need in a simple finite state machine model, we will need 2 that is $2 * 4 * 2 * 2 * 3 * 2 * 3$. So, this becomes 8, 16, 32, 96, 192 and this becomes 384. So, just imagine efficient you draw to model this and it has 384 states. If you even if you spend a month trying to understand the behavior it becomes because very complex. But using the state chart notation we can draw a very simple diagram to represent this, the state machine for the robot, because you will use the concurrent states and the number of states will drastically come down to only a few.

(Refer Slide Time: 17:39)



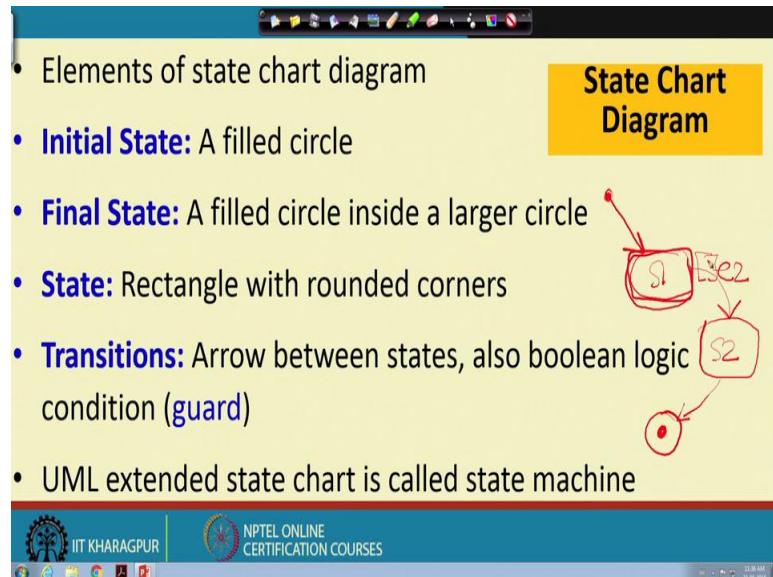
Now, let's look at the features of the state chart. So, this is the concurrent state machine for the robot we have one thing is that movement on and off and at the same time we can also set the movement to forward and backward.

Similarly, we can set the state to hand left hand raise left hand down and so on. Just see that the state machines are becoming independent and therefore the combinational explosion that was occurring there is stopped here in the state chart, because these are independent. The state variables are shown the concurrent state that is the same time it can have the state change occurring from forward to backward and on to off and so on.

The other one is the hierarchy is the nested states the state S1 contains another state machine S2 and so on. There are few other features on a state chart like history state,

broadcast message, actions on state entry, state exit etc. These part will not be discussed in this lecture.

(Refer Slide Time: 19:08)



So, to draw state the diagram, we draw an initial state which is a filled circle will draw an initial state, to start with it will come to this state let's write S1 and then so by default to start with it will start in S1, maybe go to S2 based on some event e2. There is a final state which is a filled circle inside another circle. The rectangle with rounded corners are the state representation and arrows between states these are the transition. We can have Boolean logic here to show that based on some condition the transition occurs. Even if the event e2 occurs, transition will not occur unless the condition holds true.

(Refer Slide Time: 20:24)

How to Encode an FSM?

- Three main approaches:
 - Doubly nested switch in loop
 - State table
 - State Design Pattern

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Now, let's address this question, that if we develop the state machine representation, given an FSM how can code be generated from that? Because, an FSM or a state chart is used to represent the state model of a class now, can we generate some code for the class? Actually, it is very simple if you understand it. Let's look at that, there are three main approaches one is called as doubly nested switch in loop.

So, the meaning of this is that we have one switch in C and inside switch we have case1 and inside that we have another switch event and then we have e1, e2 etc. Similarly case2 will also have a switch. So, there is a doubly nested switch i.e., there is a switch here inside another switch many switches here between nested switches and then this is within a loop while some condition.

We will get a code like this is the first approach, we will see that it is extremely simple give an FSM, you can easily draw this and develop this code. We can actually automatically generate this code. The second one is state table this is also straight forward just use the table and write the states and for each S1, S2, S3. Based on the different events, if it is in some state what occurs etc., we can also write that this very simple will discuss that, but the state design pattern will not be discussing in this lecture.

(Refer Slide Time: 22:46)

The slide has a yellow background and a red header bar. The title '3 Principal Ways' is at the top right. A bulleted list follows:

- **Doubly nested switch in loop:**
 - Global variables store state --- Used as switch
 - Event type is discriminator in second level switch
 - Hard to handle concurrent states, composite state, history, etc.
- **State table:** Straightforward table lookup
- **Design Pattern:**
 - States are represented by classes
 - Transitions as methods in classes

At the bottom left is the IIT Kharagpur logo and 'NPTEL ONLINE CERTIFICATION COURSES'. On the right is a video feed of a speaker.

So, there are 3 principal ways, the doubly nested switch in a loop, here we use a global variable to store the state. And switch on the state and the event type is the discriminator in the second level switch, but here it is bit harder to handle concurrent states, composite state history etc. The state table is a straight forward table lookup. In the state design pattern we have as many classes at the states and the transitions are methods in the class we will not be discussing this.

(Refer Slide Time: 23:30)

The slide has a yellow background and a red header bar. The title 'Doubly Nested Switch Approach' is at the top right. It shows a piece of C code with handwritten annotations:

```
int state, event; /* state and event are variables */
while(TRUE){
    switch (state){
        Case state1: switch(event){
            case event1:
                state=state2; etc...; break;
            case event2:... state=state1, n... )
            default:
        }
        Case state2: switch(event){...
    }
}
```

Handwritten notes include:

- A red circle around the line 'int state, event; /* state and event are variables */'.
- A red circle around the line 'while(TRUE)'.
- A red circle around the line 'Case state1: switch(event){'.
- A red circle around the line 'case event1:'.
- A red circle around the line 'state=state2; etc...; break;'.
- A red arrow points from the line 'case event2:...' to the handwritten note 'state=state1, n...)'.
- A red circle around the line 'default:'.
- A red circle around the line 'Case state2: switch(event){...'.
- A red circle around the line 'Event2: state=state1;'.
- A red circle around the line '}'.
- A red circle around the line 'break,'.

At the bottom left is the IIT Kharagpur logo and 'NPTEL ONLINE CERTIFICATION COURSES'. On the right is a video feed of a speaker.

First let us see the doubly nested switch approach. Here, we have the states as a global variables; the states as a global variables here and then we switch on the state. This is a loop here keep on doing this and as an event occurs first switch on the state depends on which state the same event may have different actions. If it is in state 1, again we switch on the event here and depending on the event 2 let say, you do something that state becomes current state becomes state 3 and print something and break. And if an event 2 occurs when it is in state 2 maybe we have state becomes event 2, event 2 maybe state becomes state 1 get some input from the user do something and then break.

So, this is a simple translation we will just look at the state machine and we see the state variable that is used to write that here the form of the specific type of the state whether it is an integer and so on. And then we look at the events here e1, e2, e3 and you look at the states S1, S2, S 3, and then switch that case S1 if the event e1 occurs then transit S2. And this is a simple state machine and that is it, and if it is an S2 it only switches based on e2 and their state is S2, S3 and in S3 it is transistor S1. So, this is a very simple code and the state table is also very simple we can also look at that.

(Refer Slide Time: 26:07)

```

int state, event; /* state and event are variables */
while(TRUE){
    switch (state){
        Case state1: switch(event){
            case event1:
                state=state2; etc...; break;
            case event2:...
            default:
        }
        Case state2: switch(event){...
        ...
    }
}

```

Doubly Nested Switch Approach

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

So, here is a doubly nested switch, the first one is switch on the state, the second one is switch on event and depending on the state the same event may have different actions.

(Refer Slide Time: 26:23)

The slide title is "State Table Approach". A bulleted list states: "From the state machine, we can set up a **state transition table** with a column for the actions associated with each transition". Below this is a state transition table:

Present state	Event	Next state	Actions
Light_off	e1	Light_off	none
	e2	Light_on	set red LED flashing
Light_on	e1	Light_on	none
	e2	Light_off	reset red LED flashing

At the bottom left is the IIT Kharagpur logo with the text "IIT KHARAGPUR". At the bottom right is the NPTEL logo with the text "NPTEL ONLINE CERTIFICATION COURSES".

The state table approach is also extremely straight forward. Here, we have the present state and the events that are recognized here and then what action takes place. So, which will be the next state, if light is off this is the present state and it reacts to e1 and e2. If e1 occurs when light is off.

So, there is no action basically for e1, e2 then light becomes on and set the red light flashing. Similarly, for every state we write the specific events and then what the state change are and what the actions are. So, it becomes easy to write the program whenever there is a state we just look up and the state table and set the state next state based on this and then we perform the action and that is it the code becomes very straight forward.

So, we are almost at the end of this lecture.

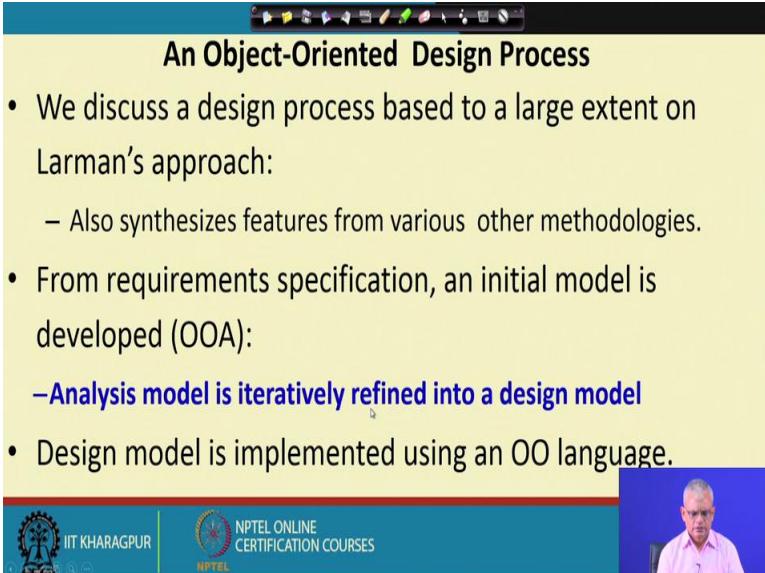
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 40
An Object - Oriented Design Process

Welcome to this lecture, in the last few lectures we looked at the UML, the UML is a modeling language and different types of views of a problem can be created. But we looked at only 3 or 4 diagrams. We looked at the use case diagram, class diagram, object diagram, sequence diagram, collaboration diagram and the state machine diagram. There are of course, other diagrams for we did not discuss those and those diagrams are not really needed in very simple problems. For specific category of problems, we need activity diagrams and so on. Now we look at a design process; the design process will tell us, will give us information about how to go about, what are the steps through which will carry out of design and we will use the UML to document the results of our process.

So, let's look at this design process and if you understand this well, for any given problem we can use this process or the number of steps that are there here and come up with a reasonably good design and the UML is of course used for documentation and helping carry out the steps.

(Refer Slide Time: 02:21)



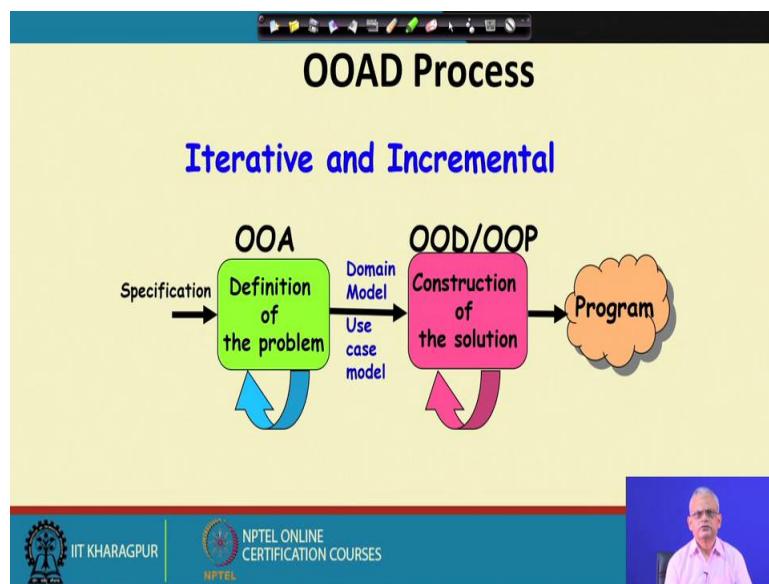
An Object-Oriented Design Process

- We discuss a design process based to a large extent on Larman's approach:
 - Also synthesizes features from various other methodologies.
- From requirements specification, an initial model is developed (OOA):
 - **Analysis model is iteratively refined into a design model**
- Design model is implemented using an OO language.

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES**

Let's look at the design process. We look at it to a large extent based on the Larman's approach, but if we look at any other design process, there are only minor variations. In our approach we also use features from other methodologies. Here we have the requirement specification available to us. Based on the requirement specification document, we develop an initial model which we called as the analysis model. And then we refine this to obtain the design model and the design model as we proceed in the design step. Good amount of code is also automatically generated and then during the implementation, we need to fill up some of the machine code.

(Refer Slide Time: 03:35)



If we diagrammatically show our approach at the very overall overview of the approach is that we have an analysis stage; call it is an object oriented analysis. Here, based on the requirement specification, we analyze the requirements and develop an analysis model and this analysis model results in a domain model and use case model. So, the outcome of the analysis here are 2 diagrams. One is the use case model and the other is the domain model and then we use these 2 diagrams to carry out the design process and also the programming object oriented design and some code is also generated during the design process. Here we construct the solution. In the analysis problem we analyze the problem and model the problem. This is not really design, because we don't generate the code from here directly; it's the analysis model. But the design model takes into consideration the specific implementation that we need and the design that needs to be performed for that.

So, it is a refinement of analysis model and here some code is also generated and based on the design we can complete the program.

(Refer Slide Time: 05:23)

OOA versus OOD?

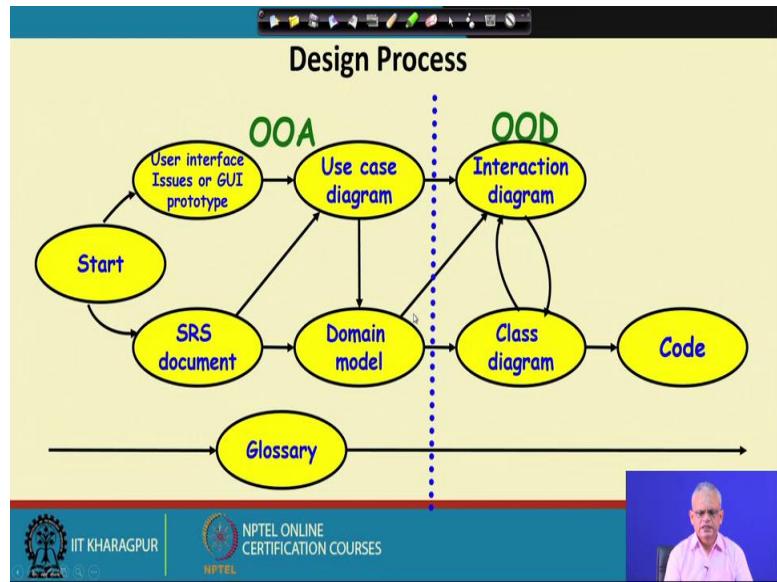
- **Analysis:**
 - An elaboration of requirements.
 - Independent of any specific implementation
- **Design:**
 - A refinement of the analysis model.
 - Takes implementation constraints into account

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us be clear how the object oriented analysis is different from object orient design. We have already discussed in some form in the previous slide discussion but now let us be clear how object oriented analysis is different from object oriented design. In the analysis, we only elaborate the requirements, we model the requirements and we create detailed models of the requirement and the analysis model can be implemented for in any situation.

So, it is a very general purpose one and it is not specific to any implementation. On the other hand in the design step we take the analysis model and look at our constraints that we have for the solution and then we create the design model. As we proceed this point will become clear that initially we develop the analysis model, this is a very general purpose model of the problem. And then depending on the specific constraints we create the design model by refining the analysis model.

(Refer Slide Time: 06:55)

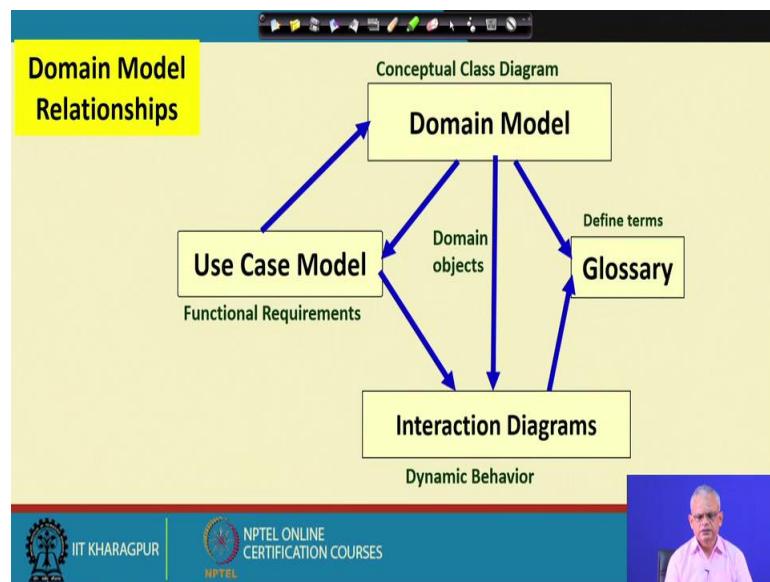


In our design process, we generate the use case diagram in the domain model during the analysis stage: these are general purpose diagram. For specific implementation, we do the object oriented design based on these two diagrams. Here we start with the SRS document available to us and we will see given the SRS document how we develop the use case diagram. For developing a use case diagram the GUI prototype is helpful to come up with a good use case diagram and we use the SRS document and the use case diagram to develop the domain model, we will see what exactly the domain model is.

Based on the use case and domain model, we develop interaction diagram, the sequence diagram and its dual that is a collaboration diagram which can be automatically obtained from the sequence diagram; we developed that the interaction diagram. Based on the interaction diagram that we develop and the domain model, we develop the class diagram. As you might remember that we were discussing the domain model has only the class names and interaction diagram as we develop it adds the methods to the classes. Then we add the attributes and so on based on the methods and we obtain the class diagram and based on the class diagram we can write the code.

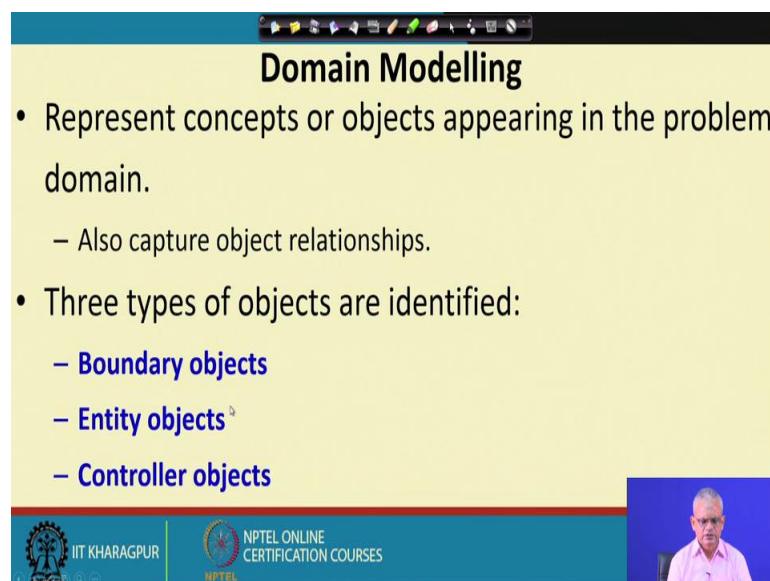
We have a glossary that is prepared throughout the design process. Here as we elaborate the requirements, we might come up with new terms, concepts etc. which you add to the glossary. During the interaction diagram and class diagram, we come up with several new terms and we write the meaning of those terms in the glossary. So, the glossary development occurs throughout the design process.

(Refer Slide Time: 09:23)



We can also model our design process in this way that initially we develop the use case model which is the functional requirement. We take the SRS document and we create the use case model out of that and from that we develop the domain model which is the conceptual class diagram. Then based on the use case model and the domain model, we create the interaction diagrams and that will lead us to the class diagram and meanwhile we keep on defining the glossary.

(Refer Slide Time: 10:06)

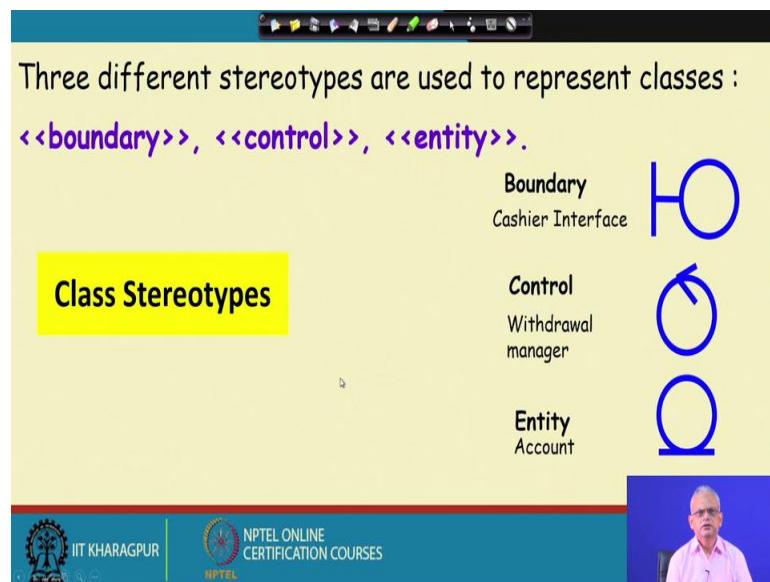


So, the outcome of the analysis model is the domain model and the use case model. We have seen while we were discussing the use case diagrams how to develop the use case model. Therefore, we will not be discussing development of use case model here, we will discuss the domain model. The domain model is the initial fast cut class diagram.

We analyze the requirements document and we look at the use case model and develop the initial class diagram. We will say that it is not very complicated, if we have the use case diagram developed well and we have the requirements document available to us; domain modeling is not very complex. While developing the domain model we look for three types of objects: the boundary objects these are the user interface objects, the entity objects and the controller objects.

So, these are the three categories of objects that will look for while doing the domain model and in the domain model we represent these three types of classes. We will discuss what exactly the boundary, the entity and the controller object are.

(Refer Slide Time: 12:03)



In every case tool we also use similar approaches to come up with the design and we look for three types of classes; that is boundary, control and entity. Since these classes are used frequently, we have stereotypes for that instead of writing boundary control and entity. We have the symbols in most of the case tools.

The boundary classes are the user interface for example, of cashier interface; we represent using the symbol in the figure above. These are not the UML symbol, but then most case tools, they use this symbols so that the design activity become simpler. The control; for example, a withdrawal manager is a control type of class and we represent using the corresponding symbol in the above figure. And the entity classes for example, and account class is represented by the symbol above.

So, if you are using a case tool and you come up with this kind of notation or in any other book or paper you can relate that in any specific problem during the analysis step we develop the domain model and in the domain model we look for 3 types of classes the boundary, controller and entity and these three we can have represent them using the symbols.

(Refer Slide Time: 13:58)

Boundary Objects

- Handle interaction with actors:
 - User interface objects
- Often implemented as screens, menus, forms, dialogs etc.
- Do not perform processing:
 - But may validate input, format output, etc.

Now, first look at the boundary objects: these are the user interface objects and the actors they interact using the boundary objects. The different types of boundary objects are screens, menus, forms, dialog boxes, check boxes and so on. So, these are basic GUI objects and here the main function of the boundary object is to collect the user input and may be to validate and also to output the display to the user and may be format the output.

So, the boundary objects are concerned with input and output. Input data from the user using specific type of forms, screens, menus etc.: these are the GUI objects and the the

boundary objects are also used for displaying the results. But remember that these don't do any processing, they are just used for collect information from the user and to display the information.

(Refer Slide Time: 15:32)

Entity Objects

- Hold information over long term:
 - e.g. Book, BookRegister
- Normally are dumb servers:
 - Responsible for storing data, fetching data etc.
 - Elementary operations on data such as searching, sorting, etc.
- Often appear as **nouns** in the problem description

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The second type of classes that we look for other entity objects: the entity objects store the information. For example, in a library we have large number of books and for each book as you create, we store the name of the book, the author, the ISBN number, the cost etc. So, this information is stored for long time. It is not that each time the book is issued etc., these are lost. Once book is created, it remains until after several years or something gets discard the book. So, the entity objects; the store data just like we saw that the book stores some data like author name, book name, ISBN number etc. Similarly book register is also an entity object. It contains what are the books available and which are issued out to whom.

Similarly we might have members register all these are entity objects. We call these objects as dumb servers because they do only very simple processing like store some data, lookup some data or may be search, sort etc. The entity objects are normally identified by reading the problem description and finding the nouns there, the nouns as you will see the mostly indicate the entity objects.

(Refer Slide Time: 17:42)

Controller Objects

- **Overall responsibility to realize use case behavior:**
 - Interface with the boundary objects
 - Coordinate the activities of a set of entity objects
- **Embody most of the business logic required for use case execution:**
- There can be more than one controller to realize a single use case

The third category of objects are the controller objects. The controller objects are in overall charge of executing the use case. As soon as a use case is executed, the user uses the boundary to enter some request, then that gets reported to the controller objects. The boundary initial request is reported to the controller object and the controller object, it interfaces with the boundary object, gets the request from the user.

Once the request from the user comes the controller object, stores a logic inside. It knows that how to realize the required behavior i.e., which objects need to participate in this use case execution, it sends them specific messages, collects the information and sends to other set of objects and so on.

So, we can say that the controller objects are the one which have the business logic required for use case execution. Because at the start of a use case, the controller object is reported about the user trying to execute. Let us say one to issue a book then the controller object knows the business logic. It would check for example, whether the book is reserved before the book is issued to check, whether the member can actually issue, whether he has exceeded his quota and so on. Finally, it will issue it and store the information in a book register and finally print the issue slip.

So, the controller object is an important class object for every use case. The corresponding controller object gets the control and it determines what are the steps

through which the use case will be executed and which objects need to interact and it request those objects to do their specific part of the execution.

(Refer Slide Time: 20:05)

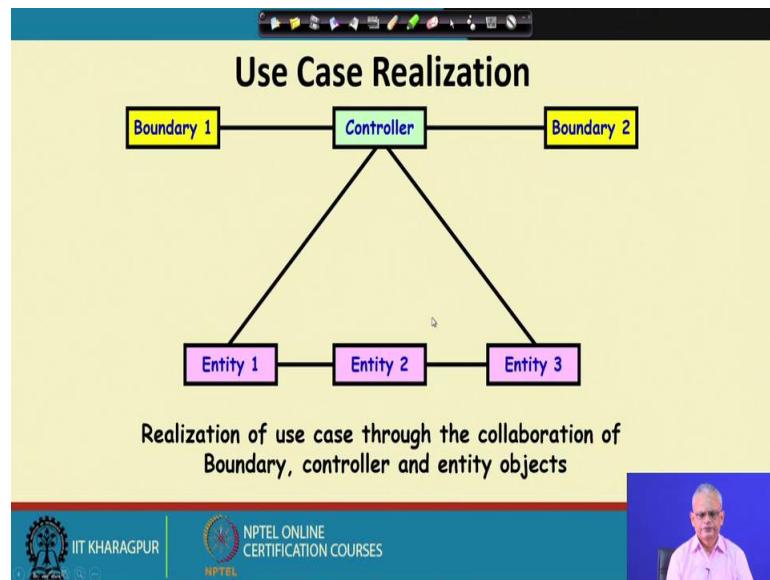
Controller Classes

- Controller classes coordinate, sequence, transact, and otherwise control other objects...
- In Smalltalk MVC mechanism:
 - These are called controllers

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

We can say that the controller objects coordinate, sequence, transact and control other objects. So, these are the ones which are the brain behind the application. They coordinate the boundary objects and the entity objects and get the work done. The work that is required as part of a use case which are done by the controller object and the term possibly comes from the small talk model view controller mechanism and from there possibly in the name which is term as the controller classes.

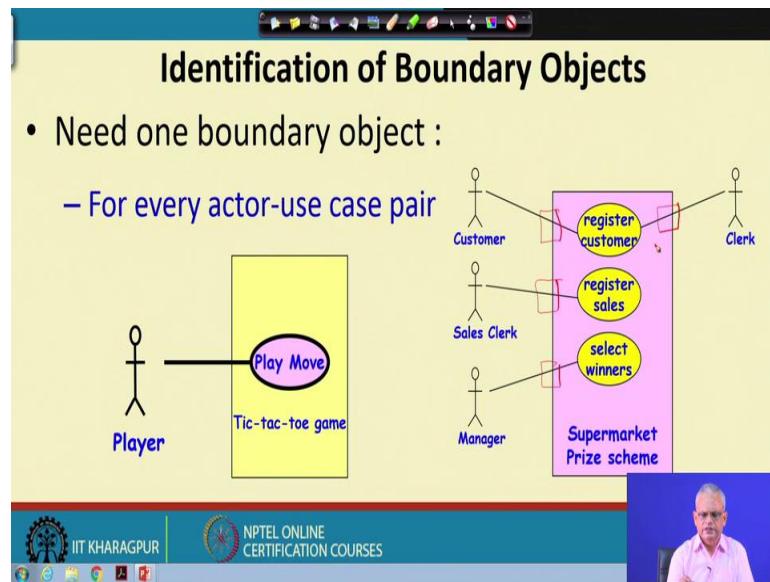
(Refer Slide Time: 20:50)



If we see how the use case execution proceeds, the boundary gets the request and it passes on the request to the controller. The controller knows the business logic that is which entity classes to request to get the work done. It sends the request entity 1 and the entity 1 may collaborate with entity 2, entity 3 and return to the controller the controller might request some other entities and so on.

So, depending on the complexity of the use case the business logic that is stored in the controller can be rather trivial or may be very complex. It takes the control and coordinates the actions of other classes and finally returns the result to the user.

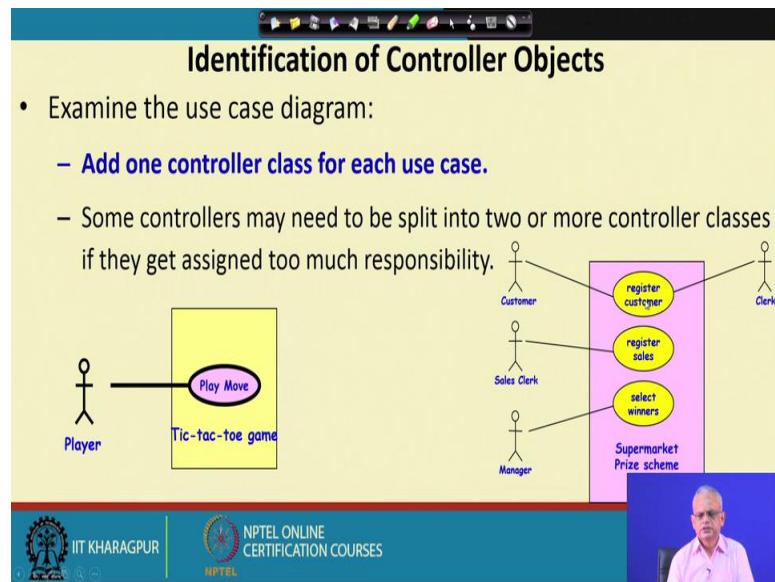
(Refer Slide Time: 22:04)



Now, let us see that how to identify these 3 types of classes. Given a problem description, we try to solve a problem, how do we go about to identify the 3 classes of objects? The simplest are the boundary objects for this we need to just examine the use case diagram. For every actor and use case we draw one boundary object that indicates the overall interface required between the use case and the actor. In this simplest use case diagram we have only one actor and one use case and therefore, we have only one boundary object.

But for example, we have 3 use cases and 4 users and we need a boundary between every user and every use case. So, we need 4 boundary objects here; so, we can say that the number of boundary objects we look at how many use case actor pair exist use case actor 2 use case actor pair 1 ones a total 4. So, identifying the boundary objects is a rather straight forward if we have the use case diagram available to us. We just look at every boundary here to the use case actor and have one boundary object for that.

(Refer Slide Time: 24:04)



Now, let's see the controller objects. This is the second class of objects. Identifying the controller object is also straight forward. We know that each controller class realizes behavior of one use case and therefore by looking at the use case diagram, we just count the number of use cases and we need that many controller classes. Each controller class has the business logic to realize the behavior for that use case. For the first simple diagram above, we have only one controller required because there is just one use case.

For the second diagram above, we need 3 controller classes; the register customer controller, register sales controller and select winner controller. To start with we just have one controller class for each of the use cases for any problem. As we proceed with the design, we might see that for some use cases the controller class becomes extremely large with very sophisticated business logic. Then we need to split those controller classes to simpler multiple controller classes.

(Refer Slide Time: 25:33)

Identification of Entity Objects

- Usually appear as nouns in the problem description.
- From the list of nouns, need to exclude:
 - **Users (e.g. accountant, librarian, etc)**
 - **Passive verbs (e.g. Acknowledgment)**
 - **Those with which you can not associate any data to store**
 - **Those with which you can not associate any methods**
- Surrogate users may exist as classes:
 - **Library member**

Now, that brings us to the third category of classes that is the entity objects. Actually these are the hardest. The boundary classes and controller classes are easily identified just by inspecting the use case diagram. We can very quickly identify the boundary classes and controller classes.

But entity classes, these require more experience and thought. We need practice identifying the entity classes, but some guidelines exists. Let us look at how to go about because this is the most crucial class. Others to identify boundary and controller classes are rather straight forward. Let's see how to go about identify the entity objects.

Typically the entity objects are identified as a noun analysis because the entity object occur in the noun. Initially when we are new to the object oriented design, we need to underline the nouns and then see which noun satisfy or they can be considered as the entity classes. But as you become more experienced, we do not have to really identify each of the noun just by reading the problem description mentally, we can determine which the entity classes are.

But to start with, we underline the nouns in the problem description and then all nouns are not entity objects, we exclude several types of noun. For example, if your nouns like accountant, library and etc. which are the users, we exclude them. But of course, sometimes we need to have some of the users also as classes for example, a library member is a class but many of this are not classes. The passive verbs also appear like

noun, but they are not really entity objects. we will look at the nouns first, eliminate the users and passive verbs and then will look at the nouns and try to associate some data and methods.

If we see that we cannot really have meaningful set of data associated with the noun and meaningful set of methods then also we eliminate that. Even though most of the users we eliminate, but sometimes we need to have some users of classes. As we were saying that library members are classes because we need to store information that who are the members, what books they borrowed and so on. These are called as surrogate user classes.

We are almost at the end of this lecture. We will stop here and we will see how to identify the entity classes for specific problems. Then we will take up few problems and then we will walk through the design process and see that how we develop the domain model, and then the interaction diagram. Finally, the class model that we will do in the next lecture. We will stop now.

Thank you.

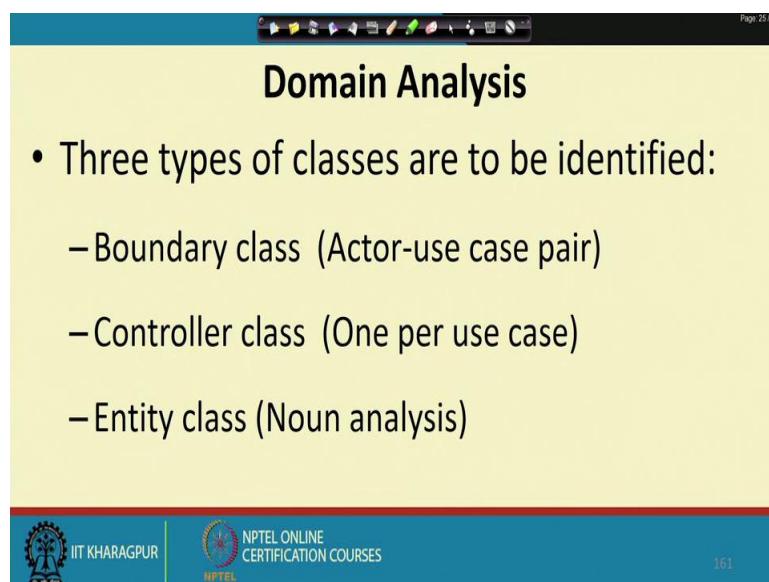
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 41
Domain Analysis

Welcome to this lecture. In the last lecture, we were discussing the topics in object oriented software design. We had seen that given a problem description, the first thing we need to do is to draw the use case diagram. The use case model is the central model and it portrays the customer's perspective of a problem.

Once the use case diagram has been successfully completed, that sets the ground floor in doing the Domain Analysis. In the domain analysis, we try to develop the first cut class diagram. We were discussing how to go about doing the domain analysis and we said that from the problem description and the use case model we can identify 3 types of objects. From the use case model, we can identify the boundary objects and the controller objects and from the problem description we can identify the entity objects. So, now with that background let us proceed in this lecture.

(Refer Slide Time: 01:35)



The slide is titled "Domain Analysis". It lists three types of classes to be identified:

- Three types of classes are to be identified:
 - Boundary class (Actor-use case pair)
 - Controller class (One per use case)
 - Entity class (Noun analysis)

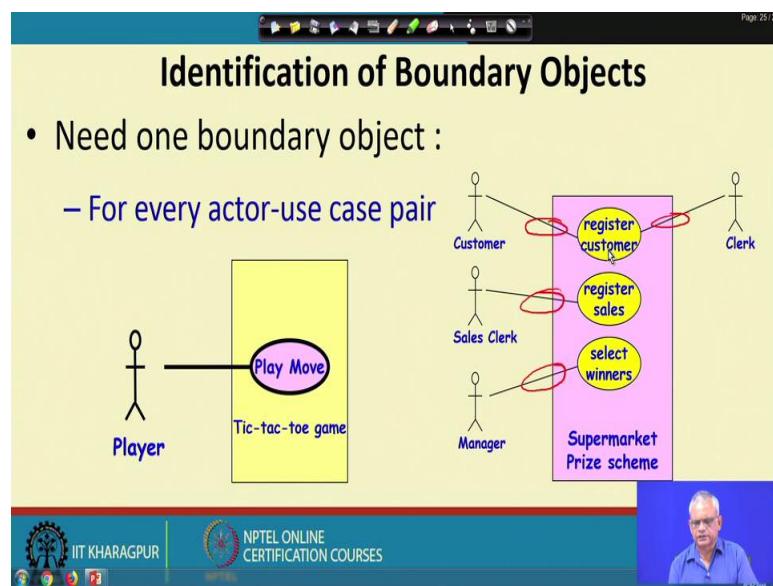
At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with the text "NPTEL ONLINE CERTIFICATION COURSES".

The aim of the domain analysis is to develop the first cut class diagram and in the subsequent steps in the object oriented design, we will refine this diagram. In domain analysis we are interested to identify 3 types of objects the boundary class that we

identify from the use case diagram where every actor use case pair becomes a boundary class.

The controller class identification is also straightforward. We identify one per use case. We look at the use case diagram, we need as many controller class as the use cases. Finally, the entity classes which we identify by doing a noun analysis on the problem description.

(Refer Slide Time: 02:41)



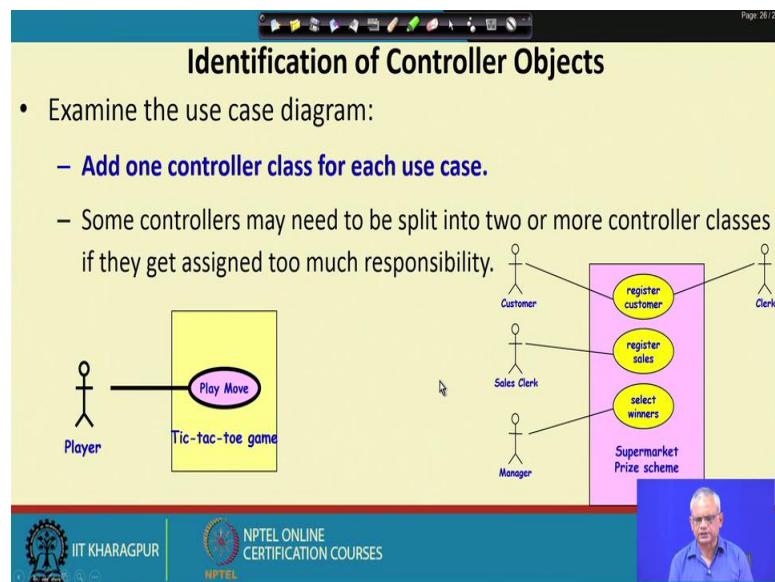
So, as you can see the boundary class and the controller class are rather straightforward to identify just from an inspection of the use case diagram. whereas, the entity classes require bit more experience and practice to be able to successfully identify the entity classes.

Now, let's try to identify the boundary classes for the use case diagrams, which you had already developed. Let's look at the Tic-tac-toe game use case as drawn in the above figure. This is a simple problem and we have only one use case i.e., play move. Therefore, we have just one boundary here and we need only one boundary class. On the other hand for the supermarket prize scheme, we have 3 use cases and we have 4 actors. Therefore, there are 4 actor use case pair, one boundary between the register customer and customer another boundary between the register customer and clerk, a boundary between register sales and sales clerk and a boundary between the manager and the select winner. So, we need 4 boundary classes. We can name this boundary classes as customer

register-customer boundary, sales clerk register-sales boundary, manager-select winners boundary and register customer-clerk boundary.

There are 3 controller classes, register customer controller, register sales controller, and select winner controller. As we had discussed in the last lecture, the boundary is for the user interface i.e., for the user input and output to the user taken care by the boundary class. Whereas, the controller class has the overall control of executing the use case. Once the user initiates the use case the controller class has business logic. It coordinates several other classes including entity classes to finally provide the result for the use case.

(Refer Slide Time: 05:39)



The controller classes are also easily identified from the use case diagram. For every use case we need a controller which has the business logic and is an overall control of realising the behaviour of the use case by coordinating the actions of several other classes. But, some of the controllers may become extremely complex. They might have to coordinate the actions of dozens of other objects. In that case to simplify the design, later in the design steps we might split them into two or more controller classes. But, you might ask the question that how do we know that a controller class has become extremely complex? The answer to that question is that, as we go to the next step of the design i.e., developing the sequence diagram we will find that for those classes which are very complex behaviour, the sequence diagram becomes extremely complex. There large number of message exchanges with many objects and becomes easy by just inspecting

the sequence diagram for use case to see the controller class become very complex . We will examine that in the next step. Similarly for this example there are 3 controller classes that are required.

(Refer Slide Time: 07:28)

Identification of Entity Objects by Noun Analysis

- Entity objects usually appear as nouns in the problem description.
- From the list of nouns, need to exclude:
 - Users (e.g. accountant, librarian, etc)
 - Passive verbs (e.g. Acknowledgment)
 - Those with which you can not associate any data to store
 - Those with which you can not associate any methods
- Surrogate users may need to exist as classes:
 - Library member

Now, let's see how to go about identifying the entity object by noun analysis?

(Refer Slide Time: 07:37)

Identifying Classes by Noun Analysis

- A partial requirements document:

A trading house maintains names and addresses of its regular customers. Each customer is assigned a unique customer identification number (CIN). As per current practice, when a customer places order, The accounts department first checks the credit-worthiness of the customer.

- Not all nouns correspond to a class in the domain model

To perform the noun analysis, we examine the problem description and then identify the noun. The entity objects are usually nouns in the problem description. But, if we identify the nouns in the problem description, not all nouns correspond to entity classes. We need

to exclude several of the classes. For example, we might have to exclude the users if there is a noun like accountant, librarian etc. We need to exclude passive verb forms, the passive verb forms appear like noun like acknowledgment. But, we know that these are not really nouns and we exclude them and also from the list of nouns that we identify. We check, if we can associate some data for it to store and we can associate some methods with it. If we cannot associate any data or methods with it, then this cannot be classes because every class after all must have some data and methods.

This is the general guideline about the nouns, We need to exclude some of the nouns during the noun analysis, but one thing we must mention that some of the users can be actually entity classes. This occurs when we have to remember some aspects of the users. For example, library member in a library automation system needs to be a class, because we need to remember how many books a library member has taken, when he returns, are their fines outstanding and so on. So, sometimes the users may become classes and we call them as surrogate classes for users.

(Refer Slide Time: 10:28)

• Remember that a class represents a group (classification) of objects with the same behavior.
– **We should therefore look for existence of similar objects during noun analysis**

• Even then, class names should be singular nouns:
– Examples: **Book, Student, Member**

**Identifying
Classes**

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** | **NPTEL**

Another hint is that entity classes normally appear as groups, there are several objects which are very similar. For example, book in a library, thousands of books. So, we have thousands of objects which are similar and let's say member, thousands or hundreds of members where all are similar. That is a characteristic of entity classes that normally they occur in groups or a large number of them.

But even though we know that there are many class corresponds to many objects, we normally name a class as a singular. For example, we name even though it represents the objects are hundreds or thousands of objects we name the class as book. Similarly there may be tens of thousands of students, but then the name of the class is student, name of the member class is member. So, the class name is normally singular even though we create many instances from the classes.

(Refer Slide Time: 12:10)

The slide title is "Noun Analysis: Example". The main text is:
A ~~trading house~~ maintains names and ~~addresses~~ of its ~~regular customers~~. Each ~~customer~~ is assigned a unique customer identification number (CIN). As per current practice, when a ~~customer~~ places ~~order~~, the accounts department first checks the credit-worthiness of the ~~customer~~.

The footer bar includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man speaking.

Now, let's take an example to see how the noun analysis is performed. Let me just tell that even though here we really do the noun analysis, as you gain some experience of solving a few dozens of problem or at least a dozen of problem, you do not have to really do the noun analysis to identify all nouns and then eliminate the ones which are not classes, but with experience you can read through the text and mentally identify, which are entity classes.

But for the first few problem description, we actually need to do the noun analysis and then we can get the hang of it that is now which type of noun we are looking for and we can mentally note down those as we read a problem description. Now for this simple example above in the figure, let us do noun analysis. This is the example for which we had done the use case diagram. Now, let's do the noun analysis for this part of the problem description, the first is to identify all nouns.

So, we identify trading house is a noun. To identify names, addresses, regular customer, customer, unique customer identification number, let me just do customer identification number. Unique is a qualifier to this, practice, customer, order, accounts department, credit worthiness and customer. So, these are the nouns in this problem description. We know that many of these nouns are not really corresponding to entity classes and we need to eliminate those. Only with those entity classes which represent a group of objects or you can associate some data and method are the entity classes.

Now, trading house is not really an entity class, because it is the name of the problem. Names and addresses are actually attribute, we cannot associate any further methods or data with names and addresses. So, these are attributes rather than classes. Regular customer and customer they are actually synonyms, we just retain here customer i.e., they are actually the regular customers. So, we will just delete one of that and we will just retain the customer. This is a possible entity class because there are many occurrences of a customer and each customer we need to store the details. For example, what is the name, address, credit worthiness and so on.

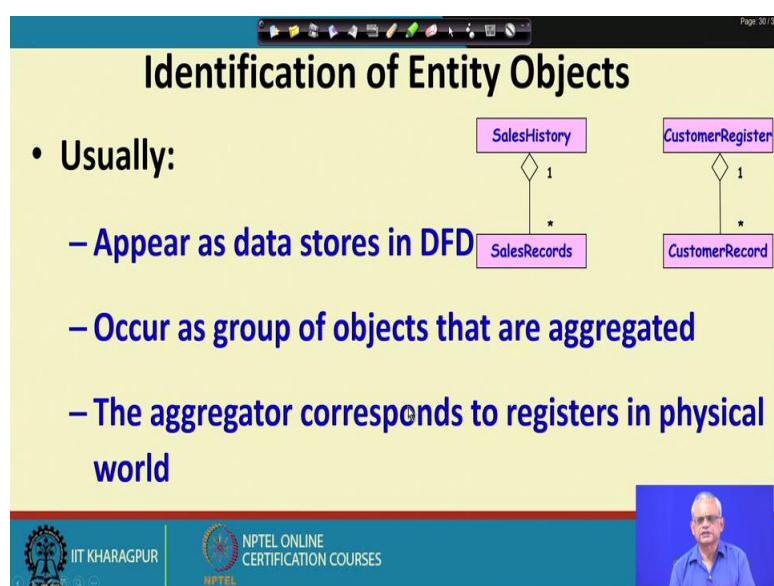
Customer identification number is just a part is attribute of the customer rather than being a class. So, I will cross it. Practice is a noun form of a verb, so I will just cross it. Customer have already identified. Order, there are many orders and we can associate some data and methods with order and this is a possibility. Department, accounts department is not the name of a department and it is actually user, credit worthiness is an attribute of the customer. So, after eliminating all that we see that we are left with only the customer and this is again the same customer. So, we have the customer and order these are the only two that I have been identified.

This is the same problem description just repeating, what I said first need to do the noun analysis need to identify all the nouns. So, trading house names, addresses, regular customer, customer, customer identification number, practice, customer may be I just left that. We should have the customer identified again here, but, we ahve already identified customer once. So, this is just a duplicate accounts department credit worthiness and again customer.

Now, we need to ignore or delete those which are either users or with which you cannot assign any data or methods. So, trading house can eliminate this is just the name of the

software. These are the attributes where we cannot assign any further methods data with. So this can eliminate that regular customer and customer are synonyms. We just need to keep one of that and delete another. Customer identification number is an attribute but not a class. We cannot assign any methods or further data with customer identification number. Therefore, delete it. Practice is a noun form of a verb, so delete it. Order, accounts department we delete it, it is a user who checks the credit worthiness and credit worthiness is an attribute and customer already selected and therefore, we delete it here also. So, finally, you are left with customer and order.

(Refer Slide Time: 19:49)



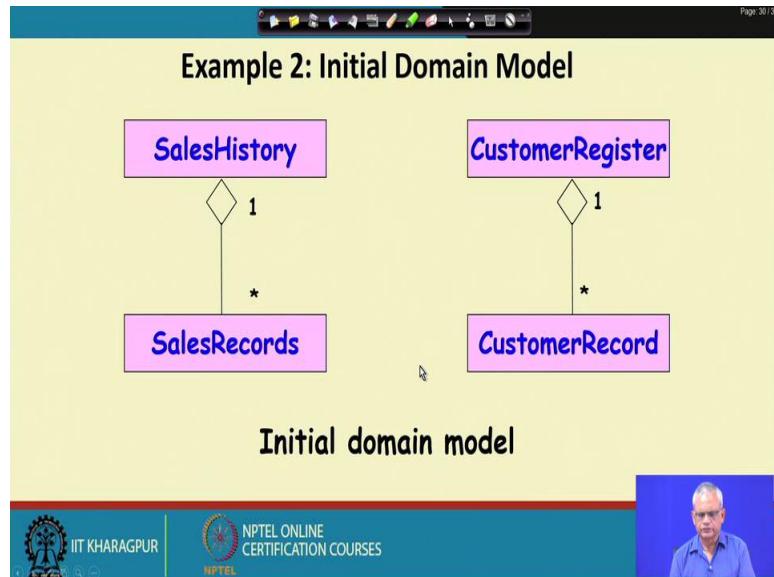
Normally, the entity objects occur in a group. Many of them like sales record or customer record, there are many objects that are get created. We normally have another class which aggregates all this objects and it, in fact, creates all these objects.

So, for sales records we have an aggregator here sales history. Similarly for the customer record there are many customer records and the aggregator is the customer register. The customer register keeps track of customer record. It in fact, creates the customer record. Almost every entity class have this form that is aggregated and multiple objects created for the sales record.

If we can look at the DFD representation of the problem, then the entity objects appear as the data stores in a DFD. These are group of objects that are aggregated and the

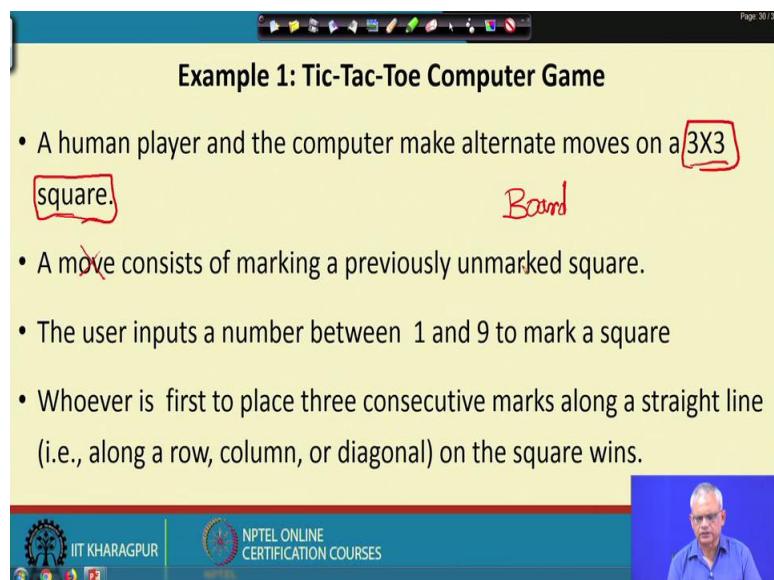
aggregator typically corresponds to a register in the physical world which keeps track of this large number of objects that are created.

(Refer Slide Time: 21:27)



So, once we create the entity classes, we call it the initial domain model. Then we add the boundary and the controller classes and then we go for the next step that is the sequence diagram development from the domain model

(Refer Slide Time: 21:48)



Now, let's do few practice few and then see whether we can do the domain model the Tic-Tac-Toe Computer Game already done the use case diagram, but just to refresh your

memory just quickly read it. A human player and the computer make alternate moves on a 3x3 square. Move consists of marking a previously unmarked square, the user inputs a number between 1 and 9 to mark a square and whoever is first to place three consecutive marks along a straight line that may be along a row column or diagonal wins.

(Refer Slide Time: 22:35)

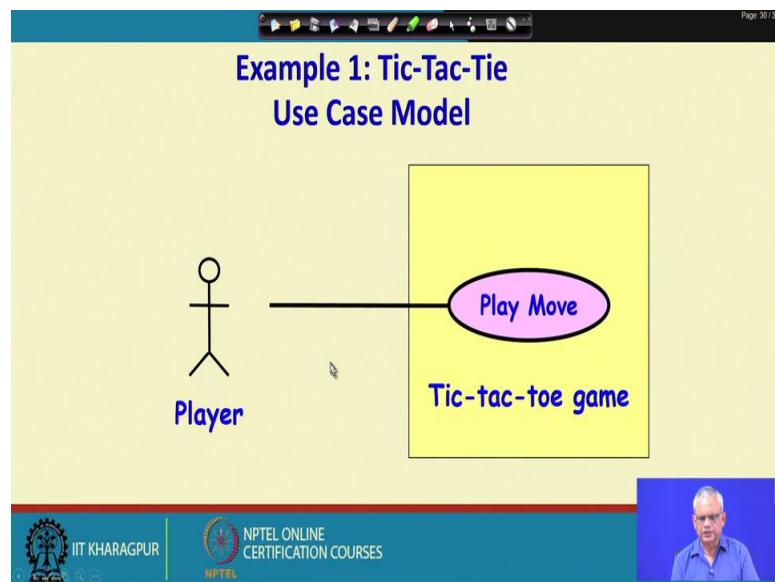
The slide has a yellow background with a black header bar. The title 'Example 1: Tic-Tac-Toe Computer Game cont...' is centered in the header. Below the title is a bulleted list of rules:

- As soon as either of the human player or the computer wins,
 - A message announcing the winner should be displayed.
- If neither player manages to get three consecutive marks along a straight line,
 - And all the squares on the board are filled up,
 - Then the game is drawn.
- The computer always tries to win a game.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', and the NPTEL logo with the text 'NPTEL ONLINE CERTIFICATION COURSES'. To the right of the footer is a small video window showing a man speaking.

Whenever human player or computer wins, message announcing the winner is displayed, but it may so happen that all the squares are adjusted and no player manages to get three consecutive marks along a straight line. In this case, the game is drawn. The computer always tries to win a game. Now, let us see how to go about developing the domain model we have drawn the use case diagram for this problem.

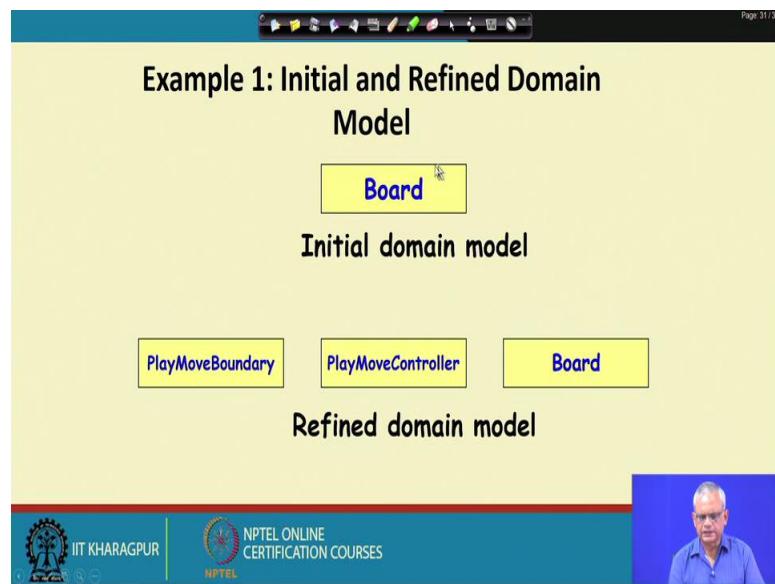
(Refer Slide Time: 23:13)



The use case model appeared like this i.e., only one use case play move, we write the name of the software the use case and then the actor is the player. Now, the next step is to develop the domain model. For developing the domain model, identifying the controller class and the boundary class is straightforward, we just have one boundary and one controller. But, the major problem that we need to address is how to identify the entity classes. Let's revisit the problem again, we need to do the noun analysis of this. We see here, we can eliminate mentally human player computer move and then you can see the 3/3 square possible move eliminate move also.

Marking previously unmarked square, unmarked is the attribute of a square. The user inputs number between 1 and 9 to mark a square and whoever is the first to place three consecutive marks along a straight line of the square wins. So, based on this, we can identify this 3/3 square to be the entity class and we will call it as the board. So, the board consists of 3/3 square and each square has the attribute whether it is marked or unmarked.

(Refer Slide Time: 25:41)



Once we identify these becomes straightforward to draw the entity diagram only one entity class that is the board. Then we refine it, we add the boundary and the controller classes. So, finally we have 3 classes that have been identified as the part of the domain model and in the next step we go for the sequence diagram from the domain model. By drawing the sequence diagram, we will be able to not only identify the methods each class will have but also identify any class relations between them and also we will identify the attributes of each of these classes.

We will take one more example just to have more practice on identifying the domain model, but now we almost at the end of the lecture we will stop now and continue in the next lecture.

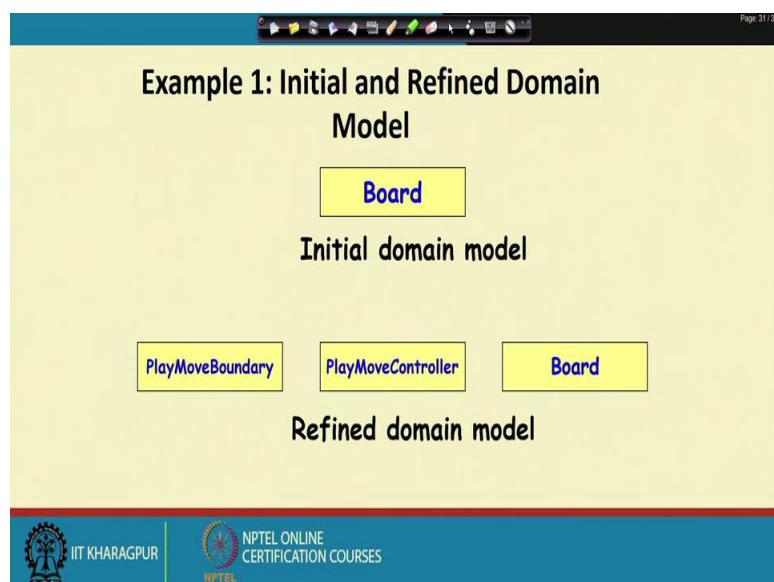
Thank you.

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

Lecture – 42
Examples of object-oriented design

Welcome to this lecture. In the last lecture we were doing some practice. Initially we discussed about how to do the domain analysis. We took some examples and we try to develop the domain model. We took the example of a Tic-Tac-Toe computer game. We identified the entity classes there and that was our initial domain model. Then we had a refined domain model, where we added the boundary and the controller classes.

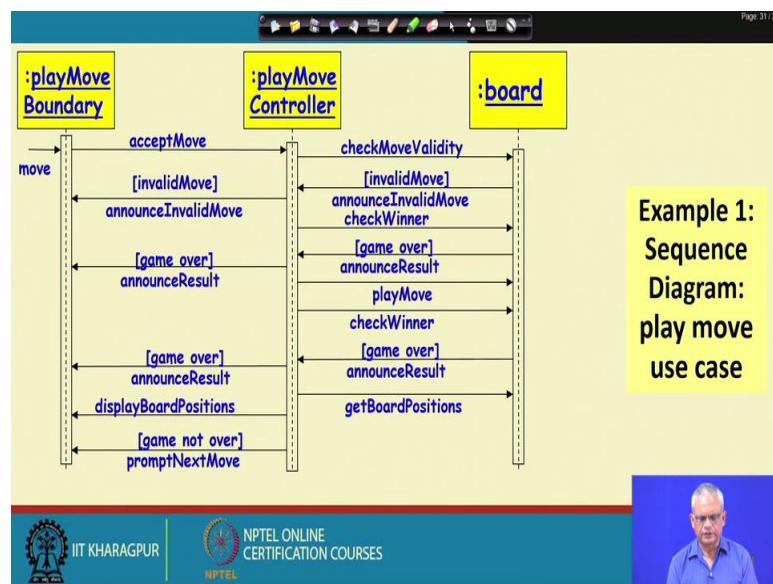
(Refer Slide Time: 00:52)



Now, let's proceed from there. So, in the last class we had identified the board class from a noun analysis. We did not really get down to do noun analysis, but mentally we rejected all those nouns which cannot be entity classes. As we get more experienced, we do not even have to do that, we just read through the problem and we can identify what the entity classes are. Once we identify the entity classes that forms our initial domain model, the boundary and the controller classes are easily identified almost mechanically from the use case diagram and, then we add them here which forms our refined domain model. From here the next step would be to identify the sequence diagram. We know that we have to develop one sequence diagram for every use case. For this problem of Tic-

Tac-Toe, we have only one use case that is play move. Therefore, we need to develop only one sequence diagram. In the sequence diagram, when the user starts a play, how do these classes interact, and finally produce the result we display that graphically in the sequence diagram.

(Refer Slide Time: 02:26)



For the Tic-Tac-Toe problem to develop the sequence diagram, we need to understand the business logic well by reading the problem many time and understanding what really happen. Then we can draw the sequence diagram easily. Let's see here, as the user makes a move that is detected in the play move boundary. So, these are the 3 classes i.e., the playMove Boundary, the playMove Controller and the board which interact to realise the play move. Once the move in initiated by the user that is conveyed to the controller.

The controller is the one which actually coordinates all other objects. The boundary is only for getting the data and reporting it to the controller or taking data from the controller and displaying to the user. So, the accept move as the move is made, that is conveyed to the controller. The controller knows what to do, knows that whether need to check the validity or the user has marked on a square which is already occupied or it was an ambiguous move.

If it is given graphically and it checks the move validity, but how will it check? It needs to check through what the current marked squares on the board are and that data is there with the board. So, it requests the board to check the validity. If the board conveys

invalid move, then it displays invalid move and prompts to give a new move and the use case would end here. If it is a valid move, then it is registered on the board and the controller knows that once the move is valid, needs to check if there is a winner. Therefore, it invokes the check winner in the board. If there is a result either the game is over or there is a winner i.e., game is drawn or there is a winner it will be announced.

But, the controller again knows that if there is no winner or game is not drawn the computer has to play the next move. It requests the board to make the next move because board class knows all the squares which are marked and which are available. This is the best class that can play the move, requests the play move and then once it has made the move it again checks for winner. If there is a result, then announces result otherwise, it gets the board position and displays the board position and if the game is not over prompts for the next move.

So, as you can see that the controller actually implements the business logic, it coordinates the actions of other objects and then realizes the use case.

(Refer Slide Time: 06:29)

The screenshot shows a presentation slide titled "CRC Card". The slide content is as follows:

- Used to assign responsibilities (methods) to classes.
- Complex use cases:
 - Realized through collaborative actions of dozens of classes.
 - Without CRC cards, it becomes difficult to determine which class should have what responsibility.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "IIT KHARAGPUR", the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". To the right of the footer, there is a small video window showing a man speaking.

But, see that the sequence diagram can become complex.

(Refer Slide Time: 06:40)

The slide is titled "CRC Cards Cont..." and contains the following bullet points:

- Systematize development of interaction diagram for complex use cases.
- Team members participate to determine:
 - The responsibility of classes involved during a use case execution

At the bottom left, there are logos for IIT Kharagpur and NPTEL. On the right, there is a video player showing a man speaking.

In our previous example we had only 3 classes, but the message exchanges there are many exchanges. Imagine if we have the dozen classes and there are large numbers of message exchanges among them, it becomes very difficult for a person to keep this in mind and draw the sequence diagram. For that the CRC Card has been developed.

The CRC Card stands for class responsibility collaborator card. It is used to assign responsibilities to classes. Responsibilities are actually the method of the classes, that is which class should have which method. The CRC card simplifies the development of sequence diagram, because here by using the CRC card we can easily identify which class would have which responsibility.

(Refer Slide Time: 07:56)

The slide has a title 'Class-Responsibility-Collaborator(CRC) Cards'. Below the title is a bulleted list:

- Pioneered by Ward Cunningham and Kent Beck.
- Index cards prepared one each per class.
- Contains columns for:
 - Class responsibility
 - Collaborating objects

To the right of the list is a template for a CRC card, represented as a table:

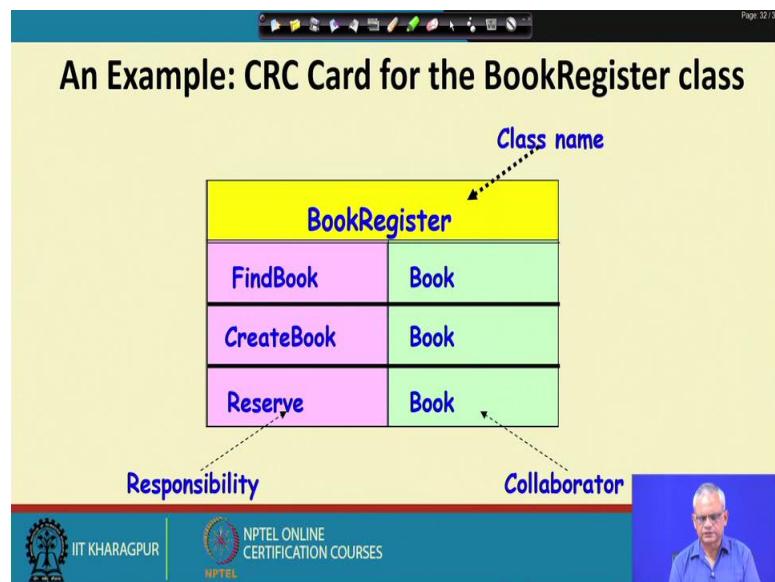
Class name		
Responsibility	Collaborator	

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, along with a video player showing a person speaking.

And then we can develop the sequence diagram. CRC card stands for class responsibility collaborator, the pioneered developed by Cunningham and Kent Beck. These cards are like playing cards i.e., small cards may be 4 inch by 6 inch. Here at the top of the card, the class name is written and then there is a small table here. So, the classes are already identified during domain analysis. The entity classes, boundary classes and controller classes. For every class identified during domain analysis a small card like this is made. The name of the class is written here and then on these rows, we write the responsibility and collaborator.

The class responsibility is that, what exactly it should do during a use case execution, it can take help of which class, what responsibility it will perform and for performing this responsibility, it will take help from which class etc. So, that is a collaborator, the class responsibility collaborator diagram. If we develop for every use case then we will have the responsibility here filled, but we do it for one use case by another and then we draw the corresponding sequence diagram.

(Refer Slide Time: 09:59)



If we think of it, the CRC cards help us systematize the development of the sequence diagram for complex problems. For simple problems, just by reading, it we can do it. For example, for the Tic-Tac-Toe game, we could do it without developing a CRC card, but for more complex problems we need to develop the CRC card. The responsibility here are the methods that will be supported and collaborator is the class whose services will be invoked by this class.

So, this is an example of a CRC card in the above figure. Name of the class is BookRegiter and then FindBook is the method that needs to support find book. And to be able to find book, it needs to invoke the book class method. It needs to support a CreateBook method and for creating a book, needs to invoke the book class method. Similarly for ReserveBook, it supports a Reserve method and then as part of the reserve method, it invokes a method of the book class.

(Refer Slide Time: 11:09)

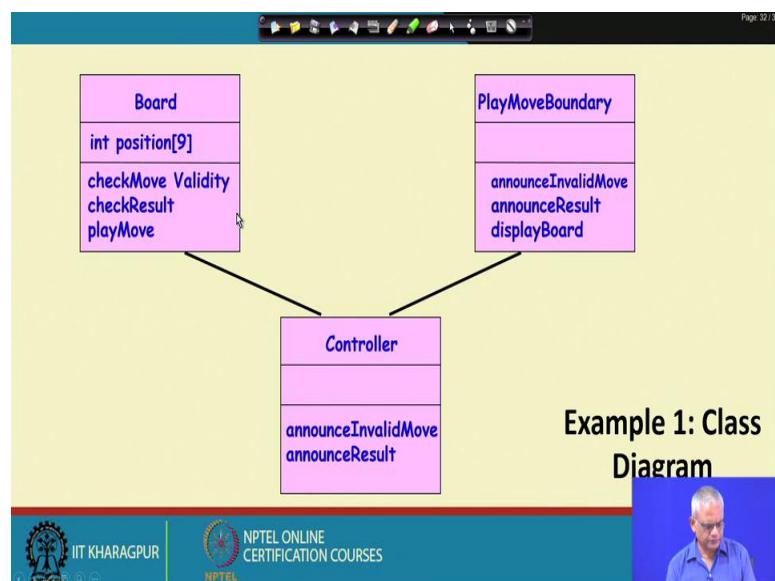
Using CRC Cards

- After developing a set of CRC cards:
 - Run structured walkthrough scenarios
- Walkthrough of a scenario :
 - A class is responsible to perform some responsibilities
 - It may then pass control to a collaborator -- another class
 - You may discover missing responsibilities and classes

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

But, how does we go about developing the CRC card? Once having the CRC card, identify a use case and then determine roughly how many classes would be required to collaborate for this use case. Then we get some team members may be 3 or 4. Each team member is distributed some of the card like 2 or 3 card. Then the use case text description is read out. As the use case text description is read out, the team members identify if their class need to do anything.

(Refer Slide Time: 12:04)



This is called as a structured walkthrough as the use case scenario is read out. Let say book to be issued to a member, then initially the member identifies the book. Then the member's lists of issued books are checked. Once it is read out that the members list of issued books have to be checked then the team member holding the member CRC card for the member class. We will say that for checking if the books, how many books have been issued to the member? The member class can check it and then he writes the responsibility as check. How many books issued? Check number of books issued and so on as the text description is read out each of the team member identifies if the class they are holding the CRC card for the class they are holding, whether it needs to take some action and that will appear as the responsibility.

Now, once the sequence diagram is done, the class diagram is automatically developed if we were using a case tool. Otherwise, we need to check the sequence diagram, identify all the messages that are sent to this class and put them here. These are the responsibilities of this class. For every class, if we go through the sequence diagram we, will see that these are the messages that are sent to this class. We just list them out here as the methods to be supported and based on the method to be supported, we write the attributes and that forms our class diagram.

(Refer Slide Time: 14:31)

Example 2: Supermarket Prize Scheme

- Supermarket needs to develop software to encourage regular customers.
- Customer needs to supply his:
 - Residence address, telephone number, and the driving licence number.
- Each customer who registers is:
 - Assigned a unique customer number (CN) by the computer.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's take the next problem, which is the supermarket prize scheme. We had also developed the use case diagram for this problem, but just to refresh your memory I will

just quickly read through that description of the problem. Since the use case diagram was already developed, we need to develop the domain model. For that, first we need to identify the entity class. Once we identify the entity classes, we can look at the use case diagram and easily identify the boundary and the controller classes.

Now, you will have to identify the entity classes and with that perspective, while reading the problem, identify what can be the entity classes. Supermarket needs to develop a software to encourage regular customers. So, if you are looking for entity class, the customer is a noun and there are many customers and the supermarket needs to remember the residence address, telephone number, and driving licence of the customer. From this description, we can identify that the customer is an entity class. We need to aggregate the customer using a customer register. The customer register class will aggregate the customer class. There will be many customer records that will be created and the customer record here itself you can find that the attributes are residence address, telephone number and driving licence. We can write the attributes and each customer is also assigned customer number. So, customer number also becomes attributes. From this part of the description we can identify customer with entity class.

(Refer Slide Time: 17:31)

Page 33 / 33

Example 2: Supermarket Prize Scheme

- A customer can present his CN to the staff when he makes any purchase.
- The value of his purchase is credited against his CN.
- At the end of each year:
 - The supermarket awards surprise gifts to ten customers who make highest purchase.

A customer can present the CN to the staff, when he makes any purchase. The value of his purchase is credited against his CN. So, all his purchase needs to be remembered. These will form the purchase record. At the end of the year, the supermarket awards

surprise gift to 10 customers who make highest purchase. So, all the purchase has to be remembered, because at the end we need to look at the computer i.e., needs look at the purchase records and identify the customer who has made the highest purchase. So, based on this description we will write purchase register aggregates the purchase records and two entity classes we have so far identified.

(Refer Slide Time: 18:50)

Page: 34 / 34

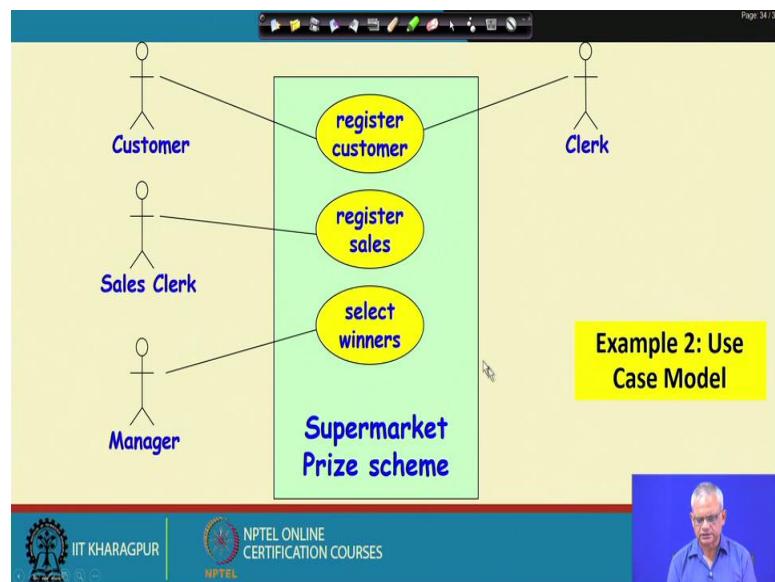
Example 2: Supermarket Prize Scheme

- It also, awards a 22 carat gold coin to every customer:
 - Whose purchases exceed Rs. 10,000.
- The entries against the CN are reset:
 - On the last day of every year after the prize winner's lists are generated.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

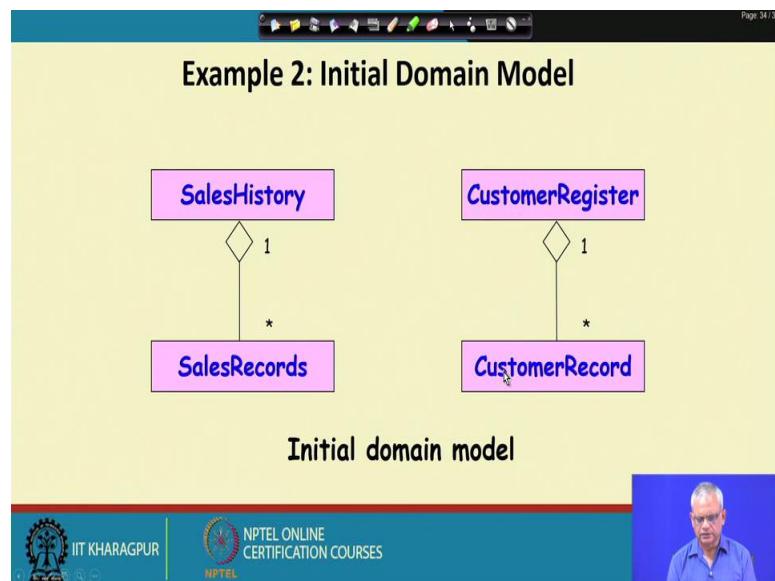
It also awards a 22 carat gold coin to every customer whose purchase exceeds Rs 10,000. So, no entity class is here. The entries against the CN are reset on the last day of every year after prize winners list are generated so, here also no entity classes. So, based on the noun analysis we had identified two entity classes.

(Refer Slide Time: 19:11)



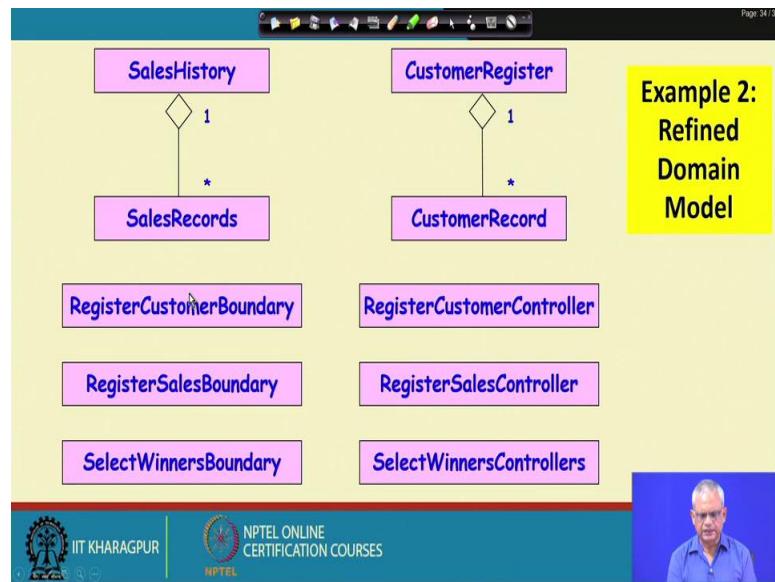
And we can then look at the use case model and identify the boundary and controller classes, 3 controller classes and 4 boundary classes.

(Refer Slide Time: 19:34)



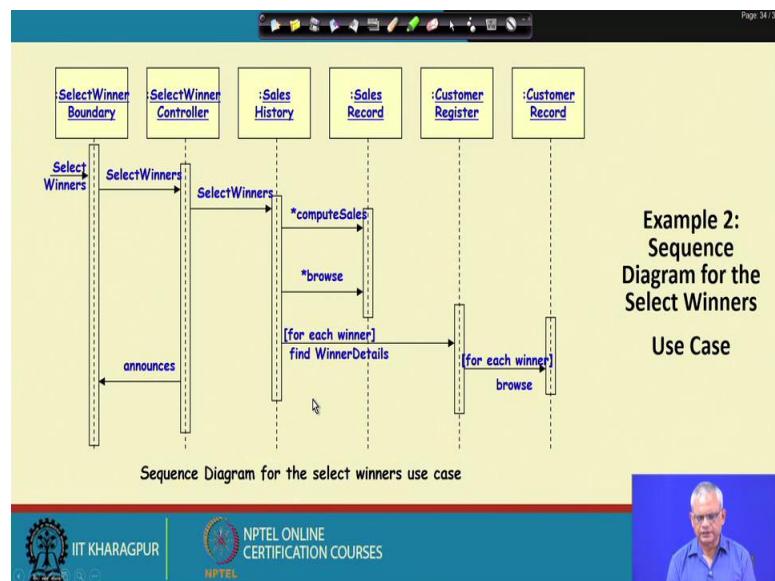
Now, this is our domain model, the customer register and customer record. We have renamed the purchase record, because these are actually the sales by the company and the purchase is from the customer perspective and these are internal data. So, we keep it as sales record and sales history. So, these are the 2 important entity classes.

(Refer Slide Time: 20:08)



And then we can identify and add the controller classes and the boundary classes, 3 controllers and 4 boundary classes.

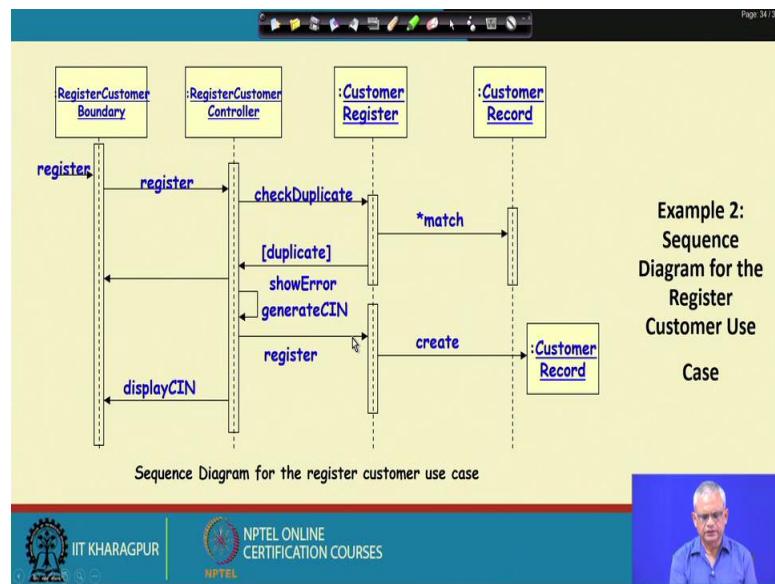
(Refer Slide Time: 20:18)



Then we develop the sequence diagram. First for the select winner this is possibly the simplest sequence diagram, we need to look at the sales history. First the controller gets charge, the controller has the business logic and it knows that the select winner can be obtained from the sales history. The sales history in terms computes the total sales and then reports to the controller. The controller finds the top winner details, ask it to finds

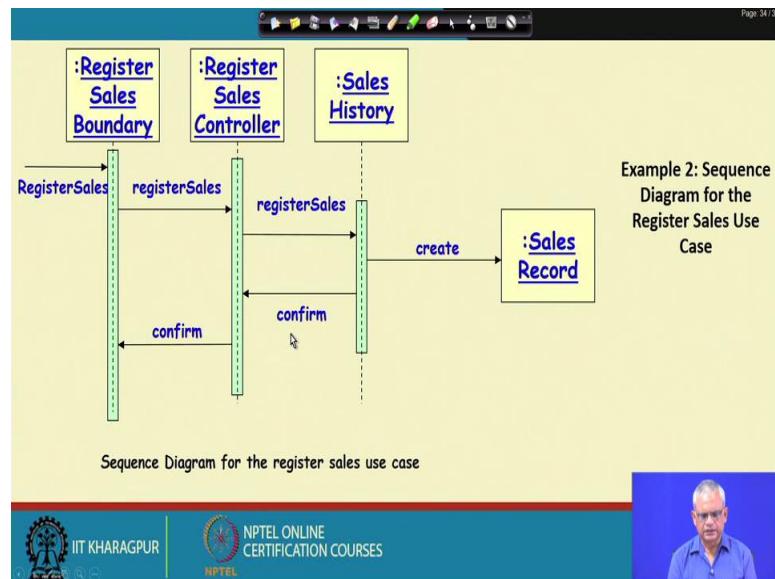
the top winner details. And then, for each winner detail identifies the address from the customer register. Then the controller asks the boundary to announce the winners similarly, register customer.

(Refer Slide Time: 21:20)



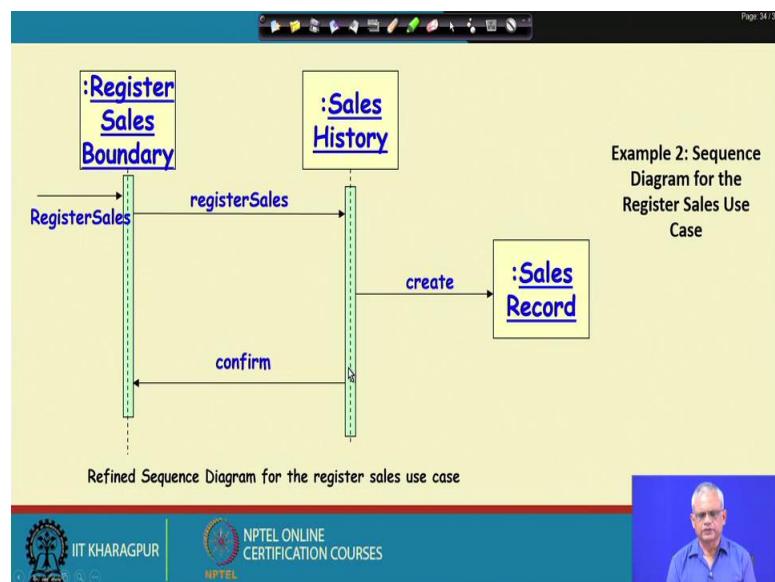
Once the register customer boundary, we have the register request coming to the controller, the controller first checks if it is already registered. For that it requests the customer register to match. If there is a duplicate, it declines to register and shows error. Otherwise, it generates a CN and request the customer registers to create a customer record and the customer record is created here and then displays the CN.

(Refer Slide Time: 22:03)



Similarly, register sales is reported the controller, the controller requests the sales history to create a sales record create and it is confirmed. So, we have so far looked at domain analysis and development of the sequence diagram. For simple examples, we will not spend more time on the object oriented design part.

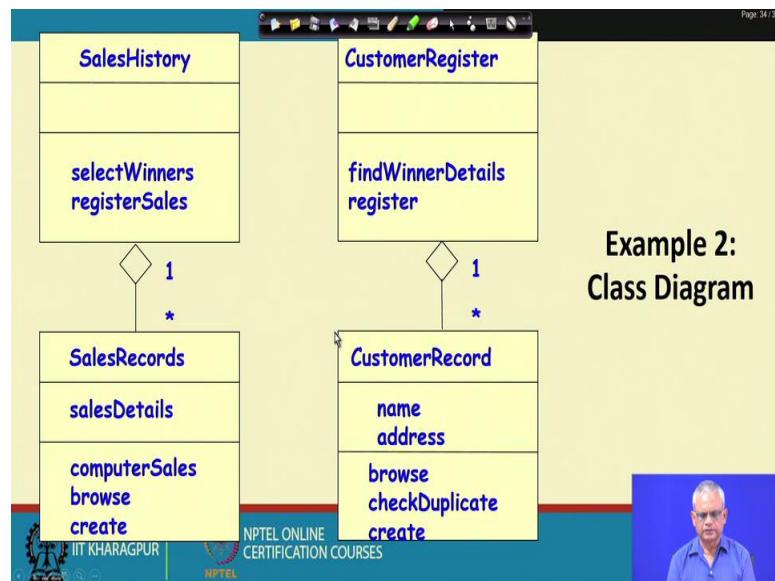
(Refer Slide Time: 22:40)



We will now, look at the testing part. This is also for the previous diagram; we have a small refinement here. The controller class actually has very little role here. It just gets the message and transmits it to sales history and then it just gets confirmation here and transmit. So, the business logic here is very simple. Therefore, we can eliminate the controller class and we can redraw the small business logic and be part of the sales

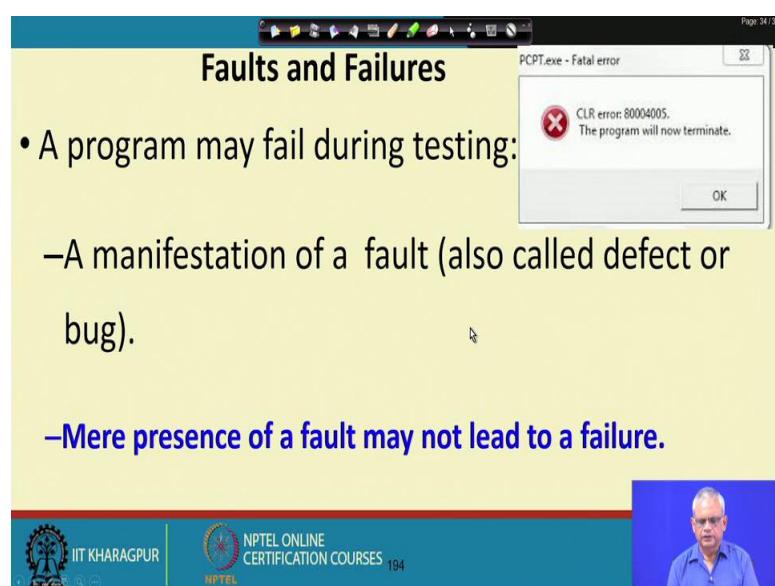
history. So, it is a very trivial business logic. We have eliminated the controller class. Similarly, when the controller class has too much of business logic, we might split the controller into 2.

(Refer Slide Time: 23:31)



Based on that we develop our full class diagram, where we write the methods and looking at the sequence diagram a case tool will automatically do it. Then we write any attribute that we identify.

(Refer Slide Time: 23:58)



Now, let's look at software testing. Once we write the program, while testing it may fail. We may see a failure, a symptom of the failure: it says that fatal error program will now terminate, but then we observe that the program has failed, but we do not see the bug, this is the symptom of the failure. By testing we identify the failure. If there is a failure, but we do not really see the bug or the error which had caused the failure, for that we need to do a debugging to identify the bug.

So, it is important to see here that when we run test cases, we see the failure, we do not see the bug. A failure is a manifestation of a bug or a fault, there was a bug in the code and while running that manifested itself in the form of a failure. We do not see the bug, but we see the manifestation of it, but then we must also remember that even if there may be a bug it may not lead to a failure. For example, the data that we give does not lead to the error to express it itself.

(Refer Slide Time: 25:53)

The slide has a yellow background and a dark blue header bar. The title 'Errors, Faults, Failures' is centered in a bold, black font. Below the title is a bulleted list:

- Programming is human effort-intensive:
 - Therefore, inherently error prone.
- IEEE std 1044, 1993 defined errors and faults as synonyms :
- IEEE Revision of std 1044 in 2010 introduced finer distinctions:
 - To support more expressive communications, it distinguished between Errors and Faults

At the bottom left, there is a logo for IIT Kharagpur and NPTEL, with the text 'NPTEL ONLINE CERTIFICATION COURSES 195'. On the right side, there is a small video frame showing a man speaking.

Let's now try to distinguish between error, fault and failure. Programming is effort intensive, we mainly manually write the code and therefore, inherently error prone. In 1993 the IEEE standard defined errors and faults as synonyms. Then a revision of that in 2010 introduced the distinction that errors and faults are actually different. Errors are the mistakes committed by the programmer and faults are what happens due to that mistake. So, there is a bug or a defect. So, error and mistake are synonyms and faults, bugs and

defects are synonyms, we are almost at the end of the time here. We will stop at this point and continue on this topic in the next lecture.

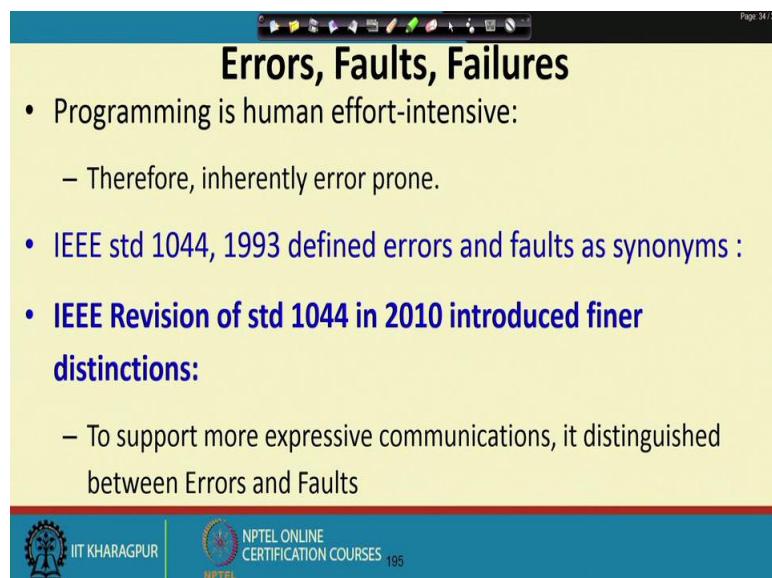
Thank you.

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

Lecture – 43
Basic concepts in Testing-I

Welcome to this lecture. In the last lecture, we started looking at the testing issues. We are trying to identify some very basic issues in testing. We had said that, when testing we observe the failures. We do not observe the bugs; we just observe a manifestation of the bug that is failure. And we were trying to understand the terms error, fault and failure; failure is caused by faults.

(Refer Slide Time: 00:57)



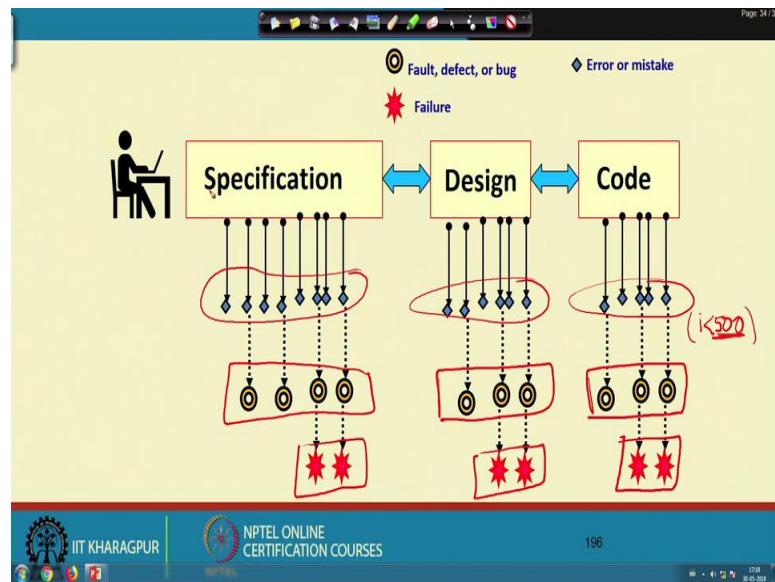
The slide has a title 'Errors, Faults, Failures' and contains the following bullet points:

- Programming is human effort-intensive:
 - Therefore, inherently error prone.
- IEEE std 1044, 1993 defined errors and faults as synonyms :
- IEEE Revision of std 1044 in 2010 introduced finer distinctions:
 - To support more expressive communications, it distinguished between Errors and Faults

At the bottom, there are logos for IIT Kharagpur and NPTEL, along with the text 'NPTEL ONLINE CERTIFICATION COURSES 195'.

We were saying that initially errors and faults were synonyms, but then later the IEEE standard said that there is a difference between error and faults. That will help us to express the ideas better. The faults or bugs are caused by mistakes or errors on the part of the programmer. These create faults, bugs or defects and this in turn may cause failures.

(Refer Slide Time: 01:41)



So, I represented that in this diagram above, the programmer here manually does many activities, does specification, design, code, and the programmer can commit mistakes or errors. They can commit many mistakes or errors during specification, design, code, but not all mistakes become faults or defects. For example, let say the programmer was writing a code $i < 50$ and by mistake or the error wrote $i < 500$, but then in the program there is no way that I can get a value more than 50. Therefore, even though he made a mistake or an error, this does not become a fault or a bug.

So, only some of the errors become faults or defects or bugs, but then some all the bugs, they do not result in failures. May be the test data that is normally given do not cause this bugs to express and some of the bugs they cause failures. So, can say that, the mistakes or the errors on the part of the programmer, some of them cause bugs or faults and some of the faults cause failures.

(Refer Slide Time: 03:59)

A Few Error Facts

- Even experienced programmers make many errors:
 - Avg. 50 bugs per 1000 lines of source code
- Extensively tested software contains:
 - About 1 bug per 1000 lines of source code.
- Bug distribution:
 - 60% spec/design, 40% implementation.

Bug Source

Spec and Design
Code

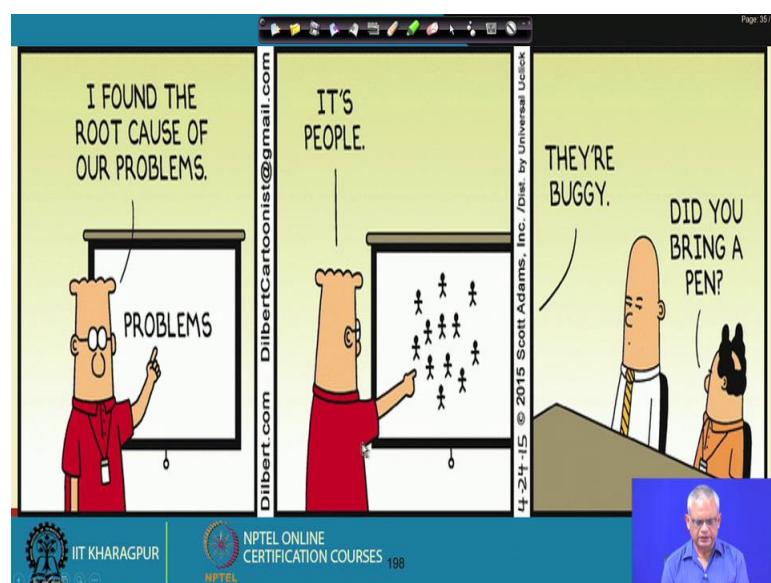
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

Page 35 / 35

Now, let's look at some facts about errors. Even the most experienced programmers do mistakes or errors. A typical industry average is 50 bugs per 1000 lines of source code is what good programmers make. Testing reduces the number of bugs, after thorough testing about a 1 bug per 1000 lines of source code still remain. Even if we have tested a program very well, still there may be 1 bug for thousand lines of source code, but what are the origin of all these different bugs, 60 percent can be traced to specification and design and about 40 percent from code is the average figure.

(Refer Slide Time: 05:07)



This is just a lighter moment a cartoon that holds a message here. This is a Dilbert cartoon from dilbert.com. So, this is an engineer, he is Dilbert he is giving a presentation says I found the root cause of our problems, it is a people. They may make mistake; they are buggy and just see here the manager says did you bring a pen. So, he has already forgotten to bring a pen. So, he made a mistake. So, people forget and they make mistakes, even very experienced people they do mistakes and that is the reason why there are bugs in the code.

(Refer Slide Time: 06:14)

The slide has a yellow background and a blue header bar. The title 'How to Reduce Bugs?' is in bold black font. Below the title is a bulleted list of five items. At the bottom, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and text for NPTEL Online Certification Courses 199.

- Review
- **Testing**
- Formal specification and verification
- Use of development process

Page 35 / 35

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 199

Assuming that the best programmers make mistakes and there lead to bugs the code. How do we reduce bugs? Because finally, we have to give a good software to the customers. There are many techniques to reduce bugs. One is to do review, we can do a specification review, design review, code review and that is a very effective way to reduce the bugs.

Testing is widely practised and acknowledged to be a good technique to reduce bugs. Formal specification and verification are not used for all the part of the code. These are expensive, difficult to use, cannot handle large programs and so on. Therefore, their use is testified use of a proper development process. This a defensive mechanism, it reduces the number of bugs.

(Refer Slide Time: 07:40)

Page 35 / 55

How to Test?

- Input test data to the program.
- Observe the output:
 - Check if the program behaved as expected.

Page 35 / 55

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 200

The first question that needs to ask is that how do you test a program? Somebody may even ask you this question that how do you test the program? We test a program by giving some inputs to it called as the test inputs and then observe the output. If the output bit matches our expectation, we say that test cases passed, but if it has failed we note down for which data it failed and under what conditions. So, input data to the program observe the output and check if the program behaved as expected. Give inputs to the system, observe the output and see if it is exactly as per our expectation, then that is passed, but there is a discrepancy if there is a failure. And we note down the conditions that is what input we gave under what conditions.

(Refer Slide Time: 08:50)

Page 35 / 55

Examine Test Result...

- If the program does not behave as expected:
 - Note the conditions under which it failed (Test report).
 - Later debug and correct.

Page 35 / 55

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 201



If the program does not behave as expected, we note the conditions under which it failed and this we call as the test report. This test report, based on the test report debugging done to identify the exact faults or the bugs and then these are corrected.

(Refer Slide Time: 09:17)

The slide has a yellow background and a dark blue header bar. The title 'Testing Facts' is centered in bold black font. Below the title is a bulleted list in blue font. The footer features the IIT Kharagpur logo, the NPTEL logo with 'NPTEL ONLINE CERTIFICATION COURSES 202', and a small video frame showing a man speaking.

- Consumes the largest effort among all development activities:
 - Largest manpower among all roles
 - Implies more job opportunities
- About 50% development effort
 - But 10% of development time?
 - How?

For a typical program in an industry scenario, testing consumes the largest effort among all development phases. It also has the largest manpower. So, if you walk into any organisation you will find that there are more number of testers than that of designers or coders or those who does specification or the analysts.

Since, the industry needs a large number of testers implies more job opportunity. Typical estimate is about 50 percent of the development effort is spent on testing. Because, finally we need to give customer a very reliable software. If we deliver a software which is buggy, the company will get a bad name and the company cannot progress.

So, most companies are careful about testing. They spend about 50 percent of the development effort in testing. But testing is done towards the end of the development life cycle. Only 10 percent of the development time is typically taken to carry out the testing. But how is 50 percent of the development effort spent in 10 percent of the development time? The answer is that there is lot of parallelism in testing, many testers can carry out the work the same time, different parts, different test, they can execute different test cases and so on. Therefore, using a large manpower, the testing time is reduced, but there is less parallelism in specification or design, because their work is dependent on each

other. We cannot really parallelly deploy 100 designers or 100 analysts doing the specification.

(Refer Slide Time: 12:01)

The slide has a yellow background. At the top right, it says 'Page 35 / 58'. The title 'Testing Facts' is centered in bold black font. Below the title is a bulleted list in blue font:

- Testing is getting more complex and sophisticated every year.
 - Larger and more complex programs
 - Newer programming paradigms
 - Newer testing techniques
 - Test automation

At the bottom left, there are logos for IIT Kharagpur and NPTEL, with the text 'NPTEL ONLINE CERTIFICATION COURSES 203'. On the right side, there is a small video window showing a man speaking.

Over the years testing is getting complex and sophisticated. The reasons are that the programs themselves are becoming larger and more complex, newer programming paradigms, test automation, there is sophistication you must know how to use the test tool and also newer testing techniques. Many new testing techniques have been developed recently and testers must know these testing techniques.

(Refer Slide Time: 12:43)

The slide has a yellow background. At the top right, it says 'Page 35 / 58'. The title 'Testing Perception' is centered in bold black font. Below the title is a bulleted list in blue font:

- Testing is often viewed as not very challenging --- less preferred by novices, but:
 - Over the years testing has taken a center stage in all types of software development.
 - “**Monkey testing is passe**” --- Large number of innovations have taken place in testing area --- requiring tester to have good knowledge of test techniques.
 - Challenges of test automation

At the bottom left, there are logos for IIT Kharagpur and NPTEL, with the text 'NPTEL ONLINE CERTIFICATION COURSES 204'. On the right side, there is a small video window showing a man speaking.

Those who do not know this, they think that testing is not challenging, but over the years testing has taken a centre stage, and it has become much more challenging than even coding, designing or specification. The reason why these perceptions exist is that in the early years of testing, testing was done by inputting random test values, which is called as monkey testing. Because of the large number of testing techniques and test related innovations tools etc. that have come in to picture, monkey testing is no more used. And the testers have their own domain knowledge and not everybody can do the testing. And they must also be convergent with the test tools.

(Refer Slide Time: 13:51)

Monkey Testing is Passe...

Two types of monkeys:

- Dumb monkey
- Smart monkey

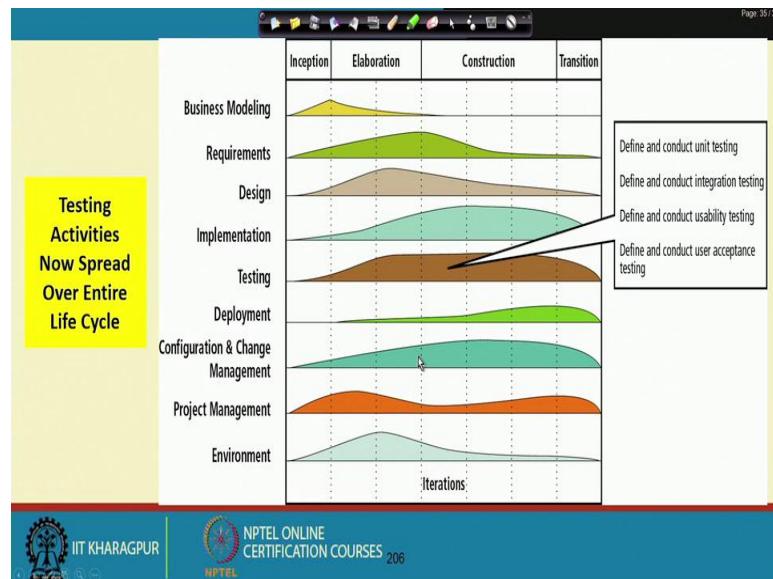
- Testing through random inputs.
- Problems:**
 - Many program parts may not get tested.
 - Risky areas of a program may not get tested.
 - The tester may not be able to reproduce the failure.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

In the initial years of testing used to be called as monkey testing. Basically give input data and observe it anything happens. There were 2 types of monkeys; initially the dumb monkeys and smart monkey. The dumb monkeys they understood very little and just kept on typing, giving data. Whereas, the smart monkeys they knew that how the software works, What are the menu choices, What it is menu choice needs to do etc. They can execute specific scenarios and so on. So, the dumb monkeys could only crash the system, that is the only thing that they can notice whereas the smart monkey knows what data is expected and so on. Therefore, is a much more effective tester, but then both the monkeys they give random inputs. The problem with monkey testing is that if random data is given, many parts of the program do not get tested, the risky areas of the programmer not tested well, because they just gave random input and not identified the risky areas and tried to test those. And also the worst thing is that many times they just

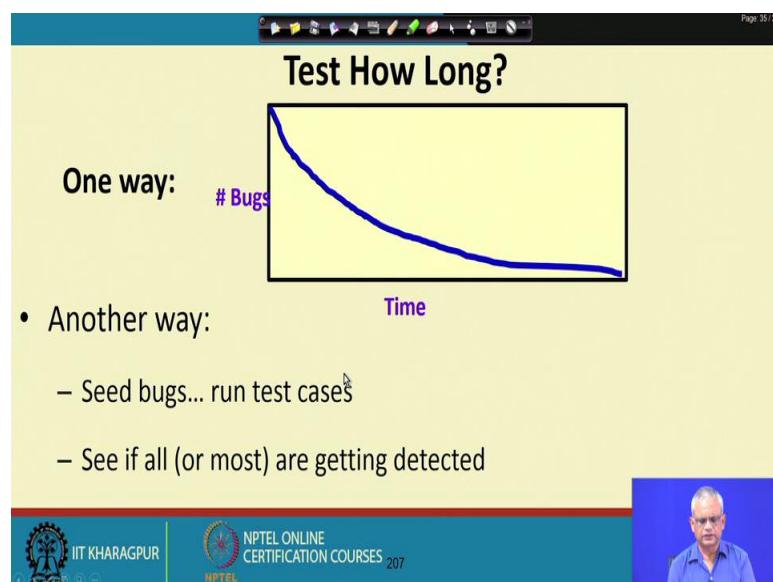
say that the program failed, but they cannot identify why, what they have been doing. They cannot reproduce the failure.

(Refer Slide Time: 15:43)



This is another basic concept that nowadays the testing effort is spread over the entire lifetime.

(Refer Slide Time: 15:48)



In the water fall model the testing was done towards the end of the life cycle, but now it is spread over the entire lifecycle. Define and conduct unit testing, define and conduct integrate testing, usability testing, user acceptance testing and so on. But, another basic

issue that we must understand before we look at the test methodology then so on is to test how long? Started to test as we test more and more find some failures and So on, but then when do we know when to stop testing i.e., the stopping criterion.

One way is that as the failure reduces, we test, let's say that we test for 2 days and if we do not find a failure we will stop there. So, that is one way. The other way is that, the manager can seed some bugs and then as the testing proceeds, you will identify whether all those seeded bugs have been found out. For all those bugs have been found out and we will say that possibly the other bugs could have also been found out and that is the time to stop testing.

(Refer Slide Time: 17:26)

Verification versus Validation

- Verification is the process of determining:
 - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
 - Whether a fully developed system conforms to its SRS document.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 208

This is another very basic concept i.e., verification versus validation. Verification is the process of determining whether the output of one phase conforms to the previous phase. Whereas, validation is the process of determining whether a fully developed system conforms to its SRS document.

(Refer Slide Time: 17:51)

The slide has a teal header bar with standard window controls. The main title 'Verification versus Validation' is centered in bold black font. Below it is a bulleted list:

- Verification is concerned with phase containment of errors:
 - Whereas, the aim of validation is that the final product is error free.

At the bottom, there's a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 209', and a video player showing a man in a blue shirt.

The verification is done after every phase just to check, whether it conforms to the previous phase. We can think of verification as the technique for phase containment of errors. Whereas, validation is for the fully developed software, we check if the final product is error free.

(Refer Slide Time: 18:17)

The slide has a teal header bar with standard window controls. The main title 'Verification and Validation Techniques' is centered in bold black font. Below it is a bulleted list:

- Review
- Simulation
- Unit testing
- Integration testing
- System testing

At the bottom, there's a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 210', and a video player showing a man in a blue shirt.

What are some of the verification techniques? These are review, simulation, unit testing, and integration testing. Just observe here that unit testing and integration testing are verification techniques, because these are not done on the fully developed system. The system testing is done fully on the fully developed system. Therefore, the system testing is a validation technique whereas unit and integration testing are verification techniques.

(Refer Slide Time: 18:54)

Verification	Validation
Are you building it right?	Have you built the right thing?
Checks whether an artifact conforms to its previous artifact.	Checks the final product against the specification.
Done by developers.	Done by Testers.
Static and dynamic activities: reviews, unit testing.	Dynamic activities: Execute software and check against requirements.



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES 211



We can make further distinction between verification and validation. Verification, we check whether we are building the system right that is not committing any errors, try to detect the errors as quickly as possible. Whereas, validation we check that after the final thing we have built, whether we have build the correct thing. Here in verification we check the artefacts that are developed after a phase if it conforms to the previous artefact. Whereas, validation checks the final product against the specification the verification activities are done by the developers. Whereas, the validation by the testers. Verification can be static or dynamic activity in static activity you do not need to execute the program. For example, review is a static activity or we might have to actually run the program and that is a dynamic activity unit testing is a dynamic activity. Whereas, validation is always a dynamic activity, we execute the software and check against the requirements.

(Refer Slide Time: 20:21)

The slide has a yellow background. At the top center is the title '4 Testing Levels'. Below it is a bulleted list:

- Software tested at 4 levels:
 - Unit testing
 - Integration testing
 - System testing
 - Regression testing

At the bottom left is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. To its right is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES 213'. On the far right is a small video window showing a man in a blue shirt.

Now, let's look at the testing levels. Software is tested at 4 levels; unit testing, integration testing, system testing and regression testing. Unit testing; each unit is tested, i.e., each module or function, integration testing; a set of modules is integrated and tested, the system testing; all the modules after integration that is complete system is ready tested. In regression testing, it is done during maintenance any bugs that are fixed need to do the regression testing.

(Refer Slide Time: 21:05)

The slide has a yellow background. At the top center is the title 'Test Levels'. Below it is a bulleted list:

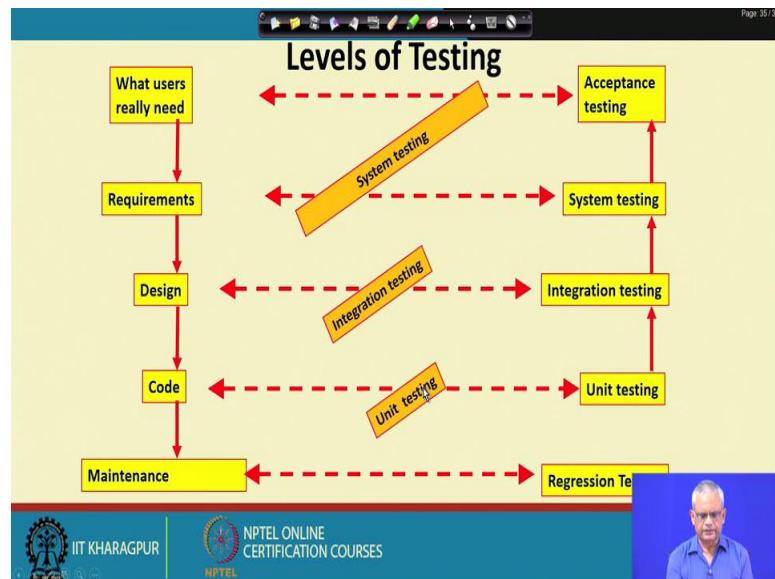
- Unit testing
 - Test each module (unit, or component) independently
 - Mostly done by developers of the modules
- Integration and system testing
 - Test the system as a whole
 - Often done by separate testing or QA team
- Acceptance testing
 - Validation of system functions by the customer

At the bottom left is the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. To its right is the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES 214'. On the far right is a small video window showing a man in a blue shirt.

Unit testing each module unit that is a function or component is independently tested, mostly done by the developers of the module. Whereas, integration and system testing

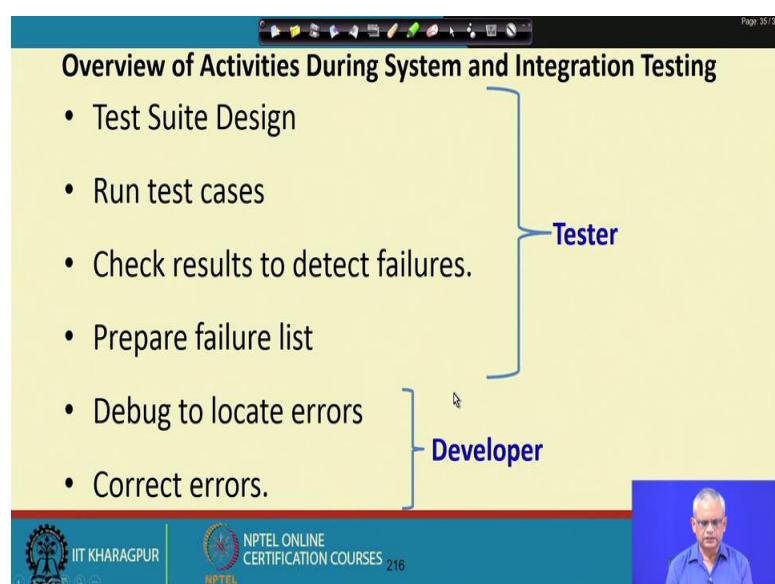
are done by the testing or a quality assurance team and acceptance testing is a validation testing done by the customer.

(Refer Slide Time: 21:38)



From this diagram, we can see here that on the development side, we find what the user needs, get the requirements, then design and code finally, maintenance and the testing side, unit testing on the code, integration testing on the design and system testing based on the requirements and regression testing for the maintenance work.

(Refer Slide Time: 22:14)



If we look at the testing activities that are done during testing; one is test suite design, running test cases, checking results to detect failures and to prepare the failure list or the test report is done by the tester whereas, the developer debugs and corrects the error.

(Refer Slide Time: 22:43)

Quiz 1

- As testing proceeds more and more bugs are discovered.
 - How to know when to stop testing?
- Give examples of the types of bugs detected during:
 - Unit testing?
 - Integration testing?
 - System testing?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 217

Now, let's have a few quiz as testing proceeds more and more bugs are discovered. So, how do we know when to stop testing? This well said there are 2 techniques; one is to check that few days of testing does not discover any new failures. And the second is by seeding bugs.

Give examples of the types of bugs detected during unit testing, integration testing, and system testing. In unit testing, we can identify the bugs in a unit for example, logical errors. Integration testing, we identify the bugs that are at the interface of 2 modules. So, here we identify the interface bugs, system testing here we identify bugs for example, performance related bugs, which are not identified during unit or integration testing.

(Refer Slide Time: 24:00)

Page 35 / 35

Unit testing

- During unit testing, functions (or modules) are tested in isolation:
 - What if all modules were to be tested together (i.e. system testing)?
 - It would become difficult to determine which module has the error.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 218



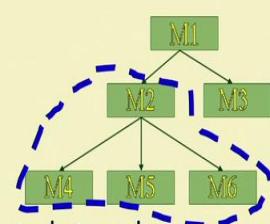
Now, let's first look at the unit testing. Here the units are tested in isolation. The units can be functions or modules, but the question is that why not test all the modules together? Why do we do unit testing? The reason is that if we do not do unit testing, it would be difficult to determine which module has the error in unit testing. We are testing only a small unit and we know that the bug is there, we quickly debug and correct. But if we do not do unit testing for each bug we have to trace out which unit has the problem and debug it and correct it.

(Refer Slide Time: 24:53)

Page 36 / 36

Integration Testing

- After modules of a system have been coded and unit tested:
 - Modules are integrated in steps according to an integration plan
 - The partially integrated system is tested at each integration step.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 219



So, it becomes much more expensive, if we do not do unit testing. In integration testing we integrate few modules together and then check their interfaces, if there are any problems.

(Refer Slide Time: 25:05)

The screenshot shows a presentation slide with a yellow background. At the top, the title 'Integration and System Testing' is centered. Below the title, there is a bulleted list of test types:

- **Integration test evaluates a group of functions or classes:**
 - Identifies interface compatibility, unexpected parameter values or state interactions, and run-time exceptions
 - **System test tests working of the entire system**
- **Smoke test:**
 - System test performed daily or several times a week after every build.

At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video thumbnail of a man speaking.

In integration testing, we integrate a group of functions or classes and then find if they are compatible, if there are unexpected parameter values. On the other hand system testing, we look at the entire system. There is another term here smoke test. Before, we start to do a test normally do a smoke test, just imagine that a plumber has put a pipeline. And before he actually puts real water in that and tests we would like to see there are any leakage. And they do a smoke test, see if there are leakage, they put smoke. Similar thing here in software in smoke test we just check if some basic functionalities are working and it does become test worthy. We have looked at some very basic concepts in testing we are at the end of this lecture we will stop here and continue in the next lecture.

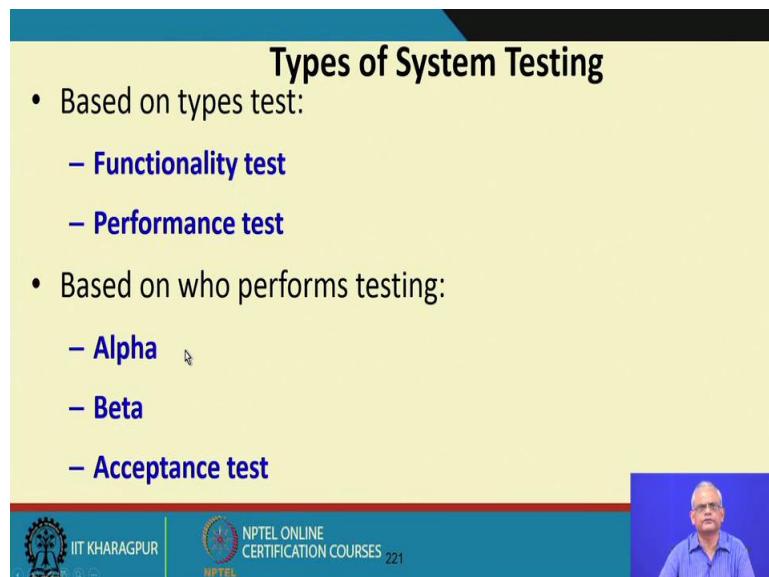
Thank you.

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

Lecture – 44
Basic concepts in Testing-II

Welcome to this lecture. In the last lecture, we had started discussing the testing issues, looked at some very basic concepts. Today let us build on the aspects that we had discussed last time. If you remember in the last lecture, we had discussed about the types of testing. Unit, integration, system these are the 3 levels of testing, then we have the regression testing, which is a different dimension of testing. We had looked at some very basic concepts of unit testing and integration testing and the system testing. We just started to discuss last time.

(Refer Slide Time: 01:15)



Types of System Testing

- Based on types test:
 - **Functionality test**
 - **Performance test**
- Based on who performs testing:
 - **Alpha**
 - **Beta**
 - **Acceptance test**

Now, let's proceed further. We can consider system testing to be 2 types, either a functionality test or performance test. In other words, if we have to do a system testing for some software, we have to write 2 types of test cases. One is the functionality test cases and the other is the performance test cases. We will see what the functionality tests need to test which aspects and which aspects will be tested by the performance test.

We can also consider the system testing to be of 3 types. Based on who performs testing, there are 3 different types of system testing. The alpha testing is done by the developing

the developing organization i.e., the testers in the developing organization. This is a type of system testing, which is first performed and after the system passes the alpha testing, beta testing is performed. Beta testing is performed by a set of friendly customers or users.

(Refer Slide Time: 02:59)

Performance test

- Determines whether a system or subsystem meets its non-functional requirements:
 - Response times
 - Throughput
 - Usability
 - Stress
 - Recovery
 - Configuration, etc.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 222

NPTEL

And after all the issues that arise during beta testing are addressed, the acceptance testing is performed and acceptance testing is performed by the customer. Once the system is delivered to the customer perform the acceptance testing and they accept the software if it passes the acceptance otherwise, they return it, they do not accept it.

Now, as we proceed we will look at the functionality test. The functionality test is basically all the functions that are documented in the SRS document. This tests whether all those functions are working fine. The second category of tests that have to be developed the test cases and to be executed are the performance tests. The performance tests determine, whether a system or subsystem satisfies the non-functional requirements.

So, both the functionality and the performance test in system testing are designed based on the SRS document. The functionality tests are designed based on the functional requirements and the performance tests are designed based on the non-functional requirements. There are several types of non-functional tests. The response times, whether all the response times are met throughput, usability, stress, recovery, configuration, etc. There are many types of performance tests. Later in this lecture series

we will see what are the aspects that are tested in this different types of performance tests i.e., throughput, usability, stress, recovery, configuration etc.

(Refer Slide Time: 05:11)

User Acceptance Testing

- User determines whether the system fulfills his requirements
 - **Accepts or rejects delivered system based on the test results.**



Just to have some more basic ideas on testing. Once the software is delivered to the customer, the customer performs what is known as the user acceptance testing or acceptance testing. They just check whether the system fulfils all their requirements and based on their test they either accept or reject the software.

(Refer Slide Time: 05:44)

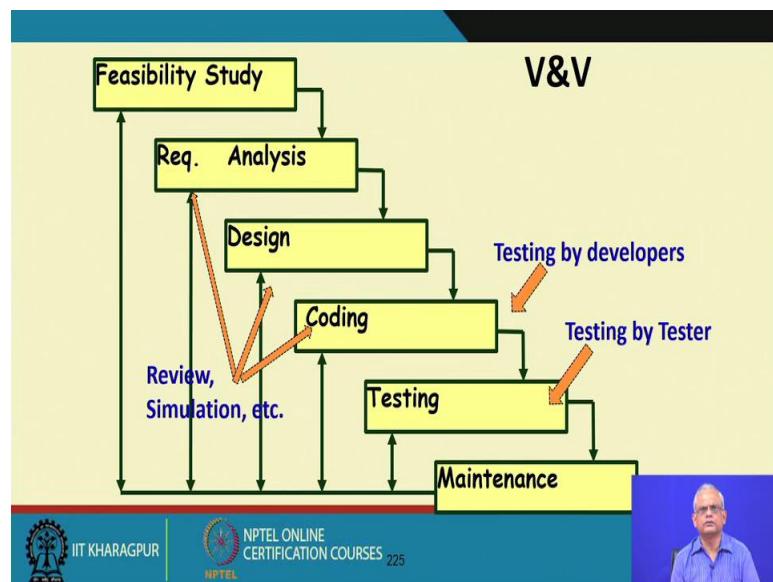
Who Tests Software?

- **Programmers:**
 - Unit testing
 - Test their own or other's programmer's code
- **Users:**
 - Usability and acceptance testing
 - Volunteers are frequently used to test beta versions
- **Test team:**
 - All types of testing except unit, beta, and acceptance
 - Develop test plans and strategy



Now, based on our discussion so far, we can see that a software is tested by different types of personnel. One is the developing team in the developing organization, the programmers who wrote the software, they perform the unit testing. They may also test the other programmer's code if necessary. So, the programmers themselves are one type of tester. The users they perform the usability and the acceptance testing and some of the friendly users they may do the beta testing. Then often the organizations have a separate testing team and they do all testing i.e., integration, alpha testing and so on. So, they do most of the testing accepting the unit, beta and the acceptance testing. And also they develop the test strategy that what kind of tests need to be performed, and they also develop the test plan when in what sequence the test need to be applied.

(Refer Slide Time: 07:12)



If we look at the broader picture, in the development lifecycle whether shown an iterative waterfall model. In the initial stages of the waterfall model, you can see that the review is done after the end of every stage. The SRS document is reviewed during the requirement analysis and specification during design, design document is reviewed, the code is reviewed. And there may be necessary to simulate this and these are all done by the developers.

So, in the initial stages, the testing is done by developers, but during the testing stage where the integration and system testing is carried out the testing is done by the tester.

(Refer Slide Time: 08:14)

Pesticide Effect

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

226

Let's look at another very important fundamental concept of testing. This concept goes by the name pesticide effect. This concept is given the name pesticide effect because of its analogy to the use of pesticide in a crop field. Let's assume that we have a cotton crop and then the farmer found that it is infested with pests. So, what does the farmer do? These are the pests who have infested the crop and the farmer would apply some pesticide and then most of the pests get killed, but then few of them are survived.

Now, in the next season, when the farmer plants the crop, again those which are survived they multiply and also new types of pests come. Again the farmer applies pesticide, but then if the farmer applies the same pesticide let us say DDT, those which have survived DDT, again applying DDT will do nothing to them, because, they have their resistance to DDT. So, the farmer needs to apply a new type of pesticide; let say malathion and then many of the pests get killed, but then those which survived and again they appear in the next crop multiply and appear and new types of pests appear. But again applying malathion, we will do nothing to these pests, because they had survived malathion and malathion does not kill them. So, the farmer needs to apply some other pesticide.

In testing, it is the same thing, there is a very strong analogy with the pesticide effect in crops. We can state by saying that errors that escape a fault detection technique cannot be detected by further applications of that technique. What it means is that we have many types of testing techniques, dozens of testing techniques we will see some of those may be

we will look at a dozen, but there are several dozens of testing techniques. Each time we apply a testing technique, some of those faults are detected, but then some of the faults escape and if we again and again apply the same testing technique. We will be not detecting those which had already escaped, because this testing technique does not detect those kind of faults.

We need to apply a new type of testing technique or a different type of testing technique and we catch some of the faults and again apply another testing technique. So, the bugs which escape a testing technique, further applications of that same testing technique we will yield very little result. That is the reason why in a software development organisation, several types of testing techniques are used one after other and this is called as the pesticide effect.

(Refer Slide Time: 12:46)

Capers Jones Rule of Thumb

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

In IEEE Computer, 1996

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 227

Capers Jones who is a well-known researcher in various testing, had written an article in the IEEE computer in 1996. He had proposed a rule of thumb in that article. It states each of software review, inspection and tested will find 30 percent of the bugs present. So, what he says is that each of this can be considered as a bug filter. In review, we get some bugs exposed. During inspection, we will get some more bugs exposed and then there are various types of testing and in each of those we get only 30 percent of the bugs present.

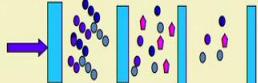
So, initially if the code had 1000 bugs review will catch 300 of them, out of the remaining 700, 30 percent is 210, 210 will be obtained by inspection and then we have

490. And out of that 490 apply unit testing we will get 30 percent of that, which is about 182 or something and so on. So, that is the one he had experimentally observed, that each of the software inspection and testing technique can be considered as a bug filter. And this bug filter can find 30 percent of the bugs present.

(Refer Slide Time: 15:09)

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?
 - $1000 * (0.3)^4 = 81 \text{ bugs}$



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 228

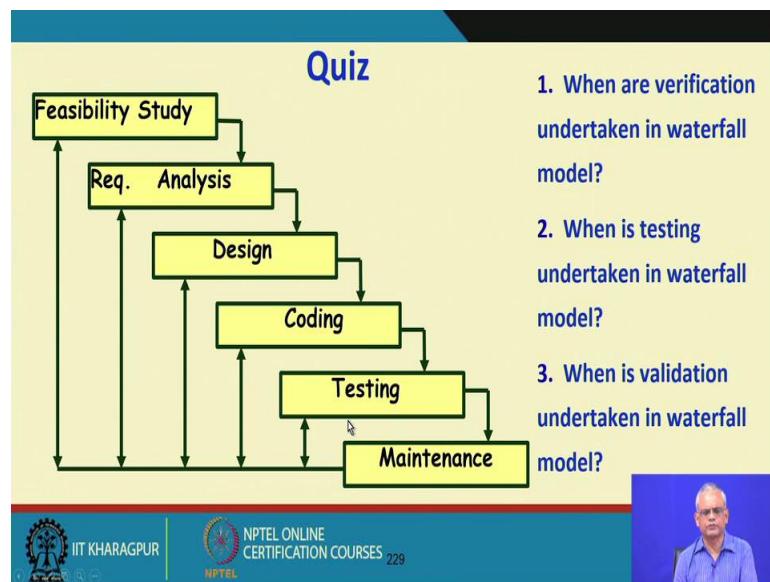


Now, let us do a small arithmetic, small problem based on this pesticide effect. Assumed that we had thousand bugs. And we deployed 4 fault detection techniques and each technique is highly successful unlike, what Capers Jones had written. Our techniques could detect 70 percent of the bugs existing at that time.

So, after the application of 4 detection techniques, how many bugs should remain at the end? So, we can do it initially after the first technique 300 will survive, 70 percent will get detected out of 1000. So, that is 700 detected in 300 existing and then again 30 percent of that will survive and so on. We can do it quickly when noting that it is equal to 1000 into 0.3 to the power 4, because each time only 0.3 survive.

So, the total surviving bug out of the application of these 4 techniques is 81 bugs in this program. This is a big number of bugs 81 bugs, when the user's encounter this bugs in failure. They would be extremely unhappy and they will call it as a poor quality software. Therefore, we need to have many more bug filters such that the bugs reduce to very small level.

(Refer Slide Time: 17:05)



Now, let us have a small quiz; given that this is the iterative waterfall model, when exactly are the verification tasks undertaken in this model. In which stages are the verification work undertaken. The verification, the form of review, inspection, simulation etc. are done during requirements analysis design and coding that is answer, when is testing undertaken in waterfall model, to answer this you can say that the unit testing is done during the coding stage. And the integration and system testing is done during the testing stage, when is the validation undertaken in waterfall model. Validation, if we remember from our last lecture discussions, validation is basically system testing and system testing is undertaken during the testing stage. Now, let's look at some more basic concepts in testing.

(Refer Slide Time: 18:33)

- 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...▲**

After Testing is Complete,
How Many Latent Errors?



Let's assume that you have tested one software very thoroughly or let's say developing organization, developed a large software and then tested it very thoroughly using the available techniques. How many errors will survive, after all the testing activities are over and success report for all the test have been obtained. Several studies have been undertaken on this issues. Jones, Schroeder etc., but then they conclude that a typical testing process removes 85 percent of the error and 15 percent continue to stay and those are called as a latent errors.

But, why cannot we remove more errors, can not the testing techniques remove 99.9 or 100 percent of the errors, why they remove only 85 percent of the errors. The answer to this question is that all the testing techniques that we will discuss are basically heuristic technique. And these do not guarantee that they will detect all errors. So, all the testing technique are heuristics they help to reduce the bugs, but provide no guarantee of complete removal of bugs. This is an important observation, that all testing techniques which you discuss are heuristics. They help only to reduce bugs and do not provide any guarantee concerning complete bug removal.

(Refer Slide Time: 20:40)

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.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 232



What do you mean by test cases? A test case, we execute to establish their some functionalities working correctly. When we execute a test, case some of the program elements are executed. Therefore, we can consider the effectiveness of a test case based on how much coverage is achieved by that test case. So, the coverage measurement of test cases is an important metric for test cases, but then we also have another one, many test cases we design them or detecting specific types of faults. For example, arithmetic faults and so on. So, we call them as fault based test cases. So, the test cases are 2 types; one we measure the effectiveness of the test cases by the coverage achieved and the other is how many faults they are targeting.

(Refer Slide Time: 22:08)

• **Test data:** Test data versus test cases

- Inputs used to test the system

• **Test cases:**

- Inputs to test the system,
- State of the software, and
- The predicted outputs from the inputs



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 233



Now, let's look at a finer distinction between a test data and test case. Test data as the name says, it is just the test inputs that are given to the system, these are the basic data to execute. Whereas the test cases not only contain the input data to the system, not only describe the input data to be given to the system, but also describe at which state of the software this data needs to be input and also the predicted output that should be obtained from the input. So, test data is just a component of the test cases and in addition the test cases contain the state at which the input is to be given and also what is the predicted output corresponding to the inputs.

(Refer Slide Time: 23:13)

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.



Based on that we can say that a test case has 3 components I, S and O, I is the input data or test data to the system, S is the state of the system at which the data will be input, and O is the output that should be expected. Just to give an example of the state at which the data to be input, let's consider a functionality in a library system; like let say renew book. For a renew book test case to be executed, we must have the same book already issued out to member, the book should have been created that should exist in the library and then it must have been issued to a member and only then that member can renew it. So, we call that as the state of the system i.e., unless the book has been created and the book has been issued out, this test case of renew book will not work.

(Refer Slide Time: 24:40)

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**.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 235

Every, software is tested using a set of test cases, which are carefully designed and all this test cases, which have been designed to test the software is called as a test suite.

(Refer Slide Time: 24:53)

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 but politely display the message: "**incorrect data type, please enter a number...**"



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 236

Now, let's see another basic concept, which is about negative test cases. We have been so far discussing about positive test cases which detect bugs in the code, that is the software does not do anything it should have been doing, but negative test cases this provide unexpected are invalid inputs and then see how the system behaves. The system should behave gracefully, it should not crash.

For example, if it wants the number of books to be entered, number of books to be issued is to be entered, and the librarian entered let say instead of a number just entered something like an alphabet like B or something by mistake, and then the system should not crash.

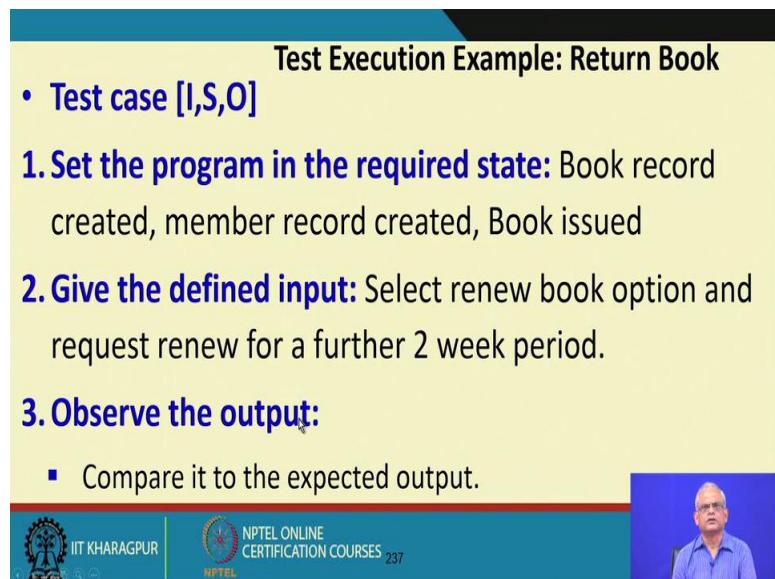
So, the aim of the negative test case is to give some input, which is not really correct input to be given as per the SRS document, these are wrong inputs, but just to check that the system behaves acceptably and does not crash. For example, in a numeric field a user types an alphabet and then it is to be observed that the system does not crash and displays a polite message, that data type entered is incorrect please enter a number.

(Refer Slide Time: 26:57)

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 week period.
- 3. Observe the output:**
 - Compare it to the expected output.



But, as the test execution is done the test cases have been designed, how is the test execution done? All the test cases have been designed and now the test execution is to be done, test have to be executed. So, the program has to be put in the required state, the input has to be given and the output has to be observed. We are almost at the end of this lecture and with this set of basic concepts on testing, we just stop here and we will continue in the next lecture.

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

Lecture – 45
Basic concepts in Testing-III

In the last lecture we had discussed some very basic concept on Testing. We were discussing the test cases, test data etc. We had said that for every software, a set of test cases are designed carefully and this is known as the test suite.

(Refer Slide Time: 00:52)

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 week period.
3. Observe the output:
 - Compare it to the expected output.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 237

For each test case the system we execute the software using the test data. And each test case typically contains 3 parts. The test data, the state at which the system should be there when the test input is given and expected output. So, to execute the software with a test case, we first put the software in the required state S.

For example, some book record might have to be created, some member record might have to be created, books issued for a library software. Once it is there in the required state, they will give the input and then observe the output. We note, this the observed we observe the output and note our conclusion on a test report let see how a test report is organized.

(Refer Slide Time: 02:02)

Test Case number Sample: Recording of Test Case & Results

Test Case author

Test purpose

Pre-condition:

Test inputs:

Expected outputs (if any):

Post-condition:

Test Execution history:

- Test execution date
- Person executing Test
- Test execution result (s) : Pass/Fail
- If failed : Failure information and fix status

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 238

Typically, each test case has a number, it becomes easier to see that which test case got executed, because once the system fails the developers need to reproduce the bug. So, test case can uniquely identify, which test case they should execute, who had written the test case, the author. The purpose of the test case like, which functionality or performance etc. it is designed to test, the test data or the test input, the expected output. Pre-condition is the state at which the system should be there, test input, expected output and the state at which the system should be left, after the test execution completes.

Once the test output is produced, the testers observe the output and then they write their analysis of the observed output. They write the test, execution date, person executing the test, they observe the pass/fail, whether it has passed the test case or it failed the test case. If failed, they write what exactly happened, what was observed during the failure, did it crash, they did it produce a wrong result; did it go into an infinite loop and so on. Then there is another field because the developers need to fix each of the failure cases of test cases. So, there is a fixed status field where the developers can record, whether they are fixed it or not yet fixed.

(Refer Slide Time: 04:09)

The slide content is as follows:

- **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

Test Team ---
Required Human
Resources

Page 111

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 239

Now, let's see the composition of the testing i.e., when an organization develops software, what types of testers does it need? You saw that there are many activities during testing, the different activities of testing are test planning, test scenario and test case design, test scenario design, test case design we will see, what is the difference between a test scenario and test case? But, roughly speaking a test scenario is an abstract description of a test case. For example, we might say that all path should be covered or a specific path needs to be covered so that test scenario that has specific path in the program needs to be executed. Whereas, the test case is more concrete than the test scenario in the sense that we provide the test data, input data for which that specific path will be covered test execution, test result analysis, test tool uses and so on. So, there are many activities during testing and let see what are the kind of human resource we need for the testing, what test planning? This is done by experienced people, test scenario design and test case design; this is also done by experienced qualified people.

Test execution can be done by a semi experienced to inexperienced persons, depending on the software that is being tested. Test result analysis this is the experienced people and test tool users the experienced people and also testing several other people participate like users, industry experts and so on. So, the composition of the testing is of various types of personnel, because different activities in testing require different levels of expertise.

(Refer Slide Time: 06:47)

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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 240

NPTEL

Page 11

But, let's now answer a very fundamental question, why do you need to design test cases? Can't we just test it with all possible inputs, because if we could test with all possible inputs, then we can be sure that the software is working fine, but why we go about only designing few test cases and then execute those. The answer to this question is that, even for a very small trivial almost trivial software, if we want to test it exhaustively, it is extremely large number of test cases trillions of test cases can be designed for exhaustive testing and that input data domain is extremely large and it will take thousands or millions of years to even test a simple program.

We will see through an example that why this is so? We will take a simple program and see how many test cases are required to test it for all possible test inputs. Since, it is impractical to test with all possible inputs; we need to develop an optimal test suite. This optimal test suite should be of reasonable size and it should be able to uncover the largest or a maximal number of errors.

(Refer Slide Time: 08:46)

The slide has a yellow header bar with the title 'Design of Test Cases' and a blue footer bar with logos for IIT Kharagpur and NPTEL.

- 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

Also, we cannot give random data, because random data has a problem that we might give the same data again or we might give a data which detects similar problem, similar bug, then the test cases that were given earlier. We will see through an example. So, even though we might test a software using 1000 test cases, a very small function let say we tested with 1000 test cases, but still if we are doing it in random testing, we cannot be sure that it has been tested well.

Because most of the test cases that have been generated randomly, may be detecting the already detected bugs in that test suite. So, if the test cases are generated randomly, then claiming that we tested it using a large number of test cases, does not indicate truly the effectiveness of testing. We might have tested with million test cases, but if it just random test cases, then it does not mean that most of the bugs have been eliminated, might just have detected very small number of bugs if at all.

(Refer Slide Time: 10:25)

Page 111

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 function: `find-max(int x,int y)`
 - Find the maximum of two integers x and y.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 242



Now, let's see how to design test cases. Testing using random test cases does not mean that the system is tested well; most of the problems have been uncovered. Let's convince ourselves with small example. Let's say we have a function, name of the function is find-max and the parameters are two integer parameter x and y and the function let say is a trivial function, which just written the larger of the two, the code is just 2 lines.

(Refer Slide Time: 11:19)

Page 112

Design of Test Cases

- The code has a simple programming error:
- `If (x>y) max = x;`
`else max = x; // should be 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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 243



But, then let's say in this 2 line code we have made a mistake. If x greater than y, max equal to x, else max equal to it should have been y, but by mistake programmer has

written max equal to x. Now, if we give 2 test cases for the test cases only represented the test data omitted the other for conciseness and understandability. Now, the test should where we test it with x is equal to 3, y equal to 2 that is the first statement is executed. And x is equal to 2, y equal to 3 that is a second statement is executed we can detect the bug. Because, if x greater than y then we should report that the max is x. Otherwise we should report max is equal to y, but then here the bug will get exposed, but let say we have a larger test suite where only the first statement is always executed. Even, if we have a thousand test cases like this, where x is larger than y and only the first statement is executed and the bug that is there in the second statement will not be detected.

(Refer Slide Time: 13:01)

• 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

Test Plan

The slide has a dark blue header bar with various icons. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 244', and a small video window showing a person speaking.

So, that example should convince us that if we have a large number of random test cases that does not mean that we have tested the software well. Now, let see what exactly the test plan, because before testing starts, the test plan is prepared. The contents of the test plan are that for a certain testing to be carried out, what are the features to be tested?

That is what features have been already completed and those need to be tested now, what features not to be tested? May be those features are still undergoing development and need not be tested, the test strategies that are deployed we will see that there are many strategies. The black box and white box and also there are several strategies of black box testing and several strategies for white box testing. Test suspension criteria, that is if a

fatal error let say crash is observed then we may not be able to run the other test cases. And then we must suspend the testing and then ask the developers to fix the problem before the other testing can be done. The test suspension criteria says that when, under which situation, the test cases execution can be suspended and the developers ask to fix the bugs before a testing is resumed. Stopping criteria; this says that as the testing goes on progressively less bugs are detected and how long to test? The test effort is what is the plan, test effort that is required, how many testers, for how many days and so on? The test schedule is about when, what type of testing will be done?

(Refer Slide Time: 15:30)

Design of Test Cases

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 245

NPTEL

Now, let see how to go about designing test cases? One very important concept here is that when we design test cases, each of our test case should target to detect different types of faults or bugs. No two test cases should actually target the same bug, because that will only increase the number of test cases without reducing the bugs in the code.

(Refer Slide Time: 16:08)

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

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 246 | 

Now, let's see the test strategy. So, far we have been saying that there are many test strategy and these can be considered as bug filters and so on. Now, let's see at very overview level, what are the test strategies?

(Refer Slide Time: 16:48)

Quiz: How would you use this for planning unit test effort?

Problems Detected

Test Technique / Source	Percentage
test Technique 1	50%
test Technique 2	30%
test Technique 3	10%
customer reported	10%

Consider Past Bug Detection Data...

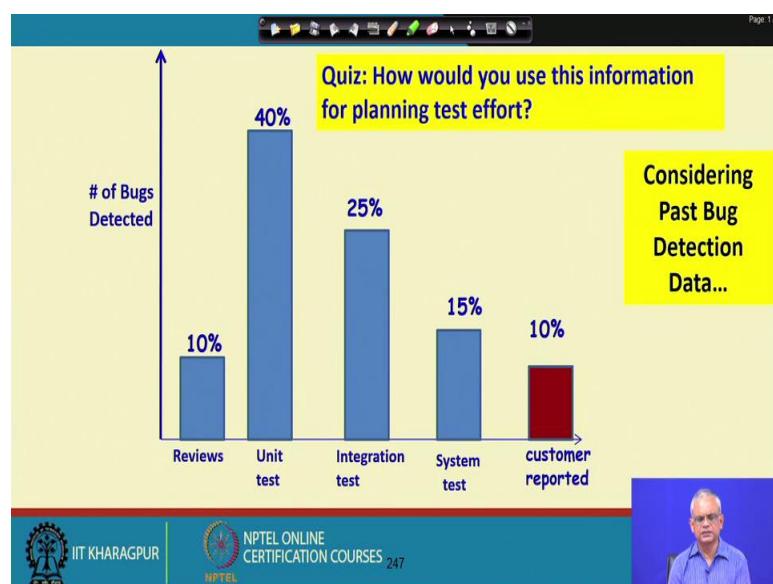
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 248 | 

The test strategy is basically type of test to be deployed; there are many types of test strategy as we will see subsequently. The white box and black box is a graph characterization of the test strategies, but then there are many types of white box strategies and many types of black box strategy, but given that there are dozens of

strategies to be deployed for testing. The testers have to plan how much time should be devoted for which type of testing? We will see how to plan for the time. For the 2 types of testing; the black box, these are also called as the usage based, because these are designed based on the customers usage data. The white box these are guided by the results in the black box, because the black box tests are performed first.

So, if we find that some problems or many problems are being detected in some part of the software, we need to do more thorough white box testing on that part. And that is the reason we can say that the white box testing results are normally guided by the black box testing results.

(Refer Slide Time: 18:07)



Now, let's see that this is the past bug data, for some software. Now, let's say we are planning to develop some additional parts for this software. Now, how much testing we should do? Let say from the previous one we had seen that reviews get 10 percent of the bugs, unit testing gets about 40 percent of the bugs, integration testing does about 25 percent, system testing gets about 15 percent, and the customer reported is about 10 percent.

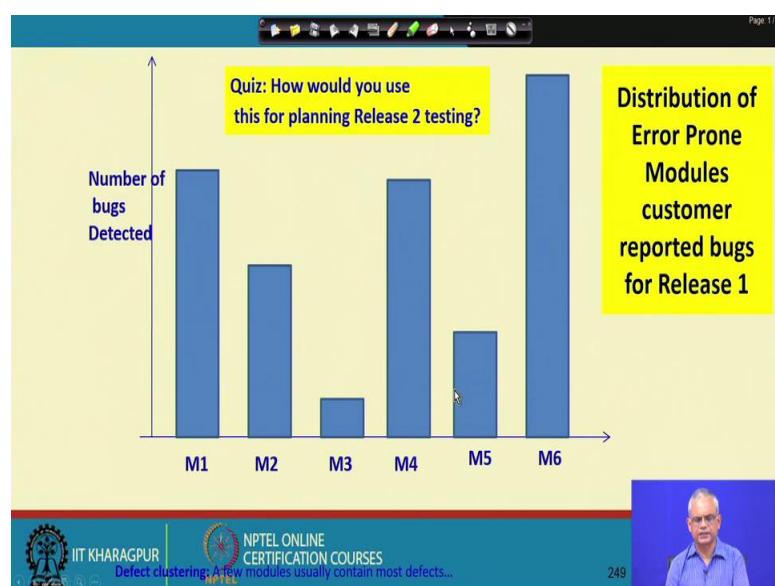
Now, we want to do a new release where we have developed some parts of the software. So, how much we want to release it within a week ? And total test hours that is available to us, because there may be 10 testers working, may be let say 200 hours of testing. Then for which type of testing, review, unit testing, integration, system and customer reported,

how much time should be there? How much test time should be allocated for unit testing, for integration testing?

It's common-sense and nothing very profound about this? Since, unit testing is detecting the largest percentage of bugs that is 40 percent; we need to spend more time on unit testing. And to be fair we should spend 40 percent of time, that we have let say 7 days and let say 100 main hours, we should spend 40 percent of that doing unit test, 10 percent of review, 25 percent on integration testing and 15 percent of the system testing.

Now, let say that we have past bug data available to us and we had deployed various types of test strategies, now let say we have written how many bugs were detected by different test strategies. Let's say strategy1 detected 50 percent of the bugs, strategy2 30 percent, strategy3 10 percent, and twist were customer reported. So, if we want to do a new release for the software, how much time do we plan to do the different types of testing? Test technique1 detected 50 percent, test technique2 30 percent, test technique3 10 percent. Again commonsense, that the test technique1 is very effective detecting nearly half of the bug. And therefore, we must spend 50 percent of the time doing this very thoroughly, 30 percent of the time for test technique2; test technique3 is not that effective, detected only 10 percent of the problem so, 10 percent of time to that.

(Refer Slide Time: 21:44)



Now, let say we have another set of data let say based on our unit testing, we found that the module 6 has the maximum number of bugs, module next is module 1 and module 4

and the lowest is module 3. Now, we want to have one more release of this, where we have made changes different parts of the software. So, where do we spend how much time? Again common sense is that module 6 is the one where maximum numbers of problems are detected so, may be more problems are there. And therefore, we need to spend proportionately much larger time on M 6 and then M 1, M 4 and M 3 you need to spend the least amount of time.

(Refer Slide Time: 22:52)

The slide has a title 'When and Why of Unit Testing?' and two bullet points under the heading '• Unit testing carried out':

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

At the bottom left, there are logos for IIT Kharagpur and NPTEL, along with text 'NPTEL ONLINE CERTIFICATION COURSES 251'. At the bottom right, there is a small video frame showing a person speaking.

Now, let see some details of unit testing. So, far we have been looking at some very basic concepts. Now, let's look at more details on unit testing, but before we start discussing detailed way of designing test cases for unit testing. Let's be clear about one aspect one question, that we might have that can we just eliminate unit testing and do a thorough integration and system testing?

(Refer Slide Time: 23:44)

The slide is titled "Why unit test?". It contains a bulleted list under the heading "Without unit test:":

- Errors become difficult to track down.
- Debugging cost increases substantially...

On the right, there is a diagram of a window divided into a grid of smaller panes. Some panes are green, some are blue, and one is red. A red arrow points from the bottom-left pane to the word "Failure" in a blue box. Below the slide, there is a footer with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video player showing a man speaking.

Unit testing is carried out after coding is complete and the code has been successfully compiled and that situation we do the unit testing. And we cannot really eliminate unit testing, because it is a very effective technique as it reduces the debugging effort. If we don't do unit testing, we do a thorough integration and system testing. Then the debugging effort will be much more, let's understand why? Let's say these are all different units that are shown here in the figure above and we just did not do a unit testing and there was let say assume a bug was there. Then we run several test cases on some of the test cases they failed, but then to debug we need to check each and every of this unit or may be at least those units, which got executed during the execution of that failed test case. But, in unit testing, when we do unit testing, we know that the bug has to be within that unit, we do not have to look at the other units. And that is the reason, why debugging effort for problem reported during unit testing is much less compare to when unit testing is not done and we directly go for system testing.

In system testing the errors are much more difficult to track down, that is localized which where is the bug. And therefore, the debugging cost increases substantially. This is another basic concept in unit testing.

(Refer Slide Time: 25:44)

• Testing of individual methods, modules, classes, or components in isolation: **Unit Testing**

- Carried out before integrating with other parts of the software being developed.
- Following support required for Unit testing:
 - **Driver**
 - Simulates the behavior of a function that calls and possibly supplies some data to the function being tested.
 - **Stub**
 - Simulates the behavior of a function that has not yet been written.

Driver **Unit** **Stub**

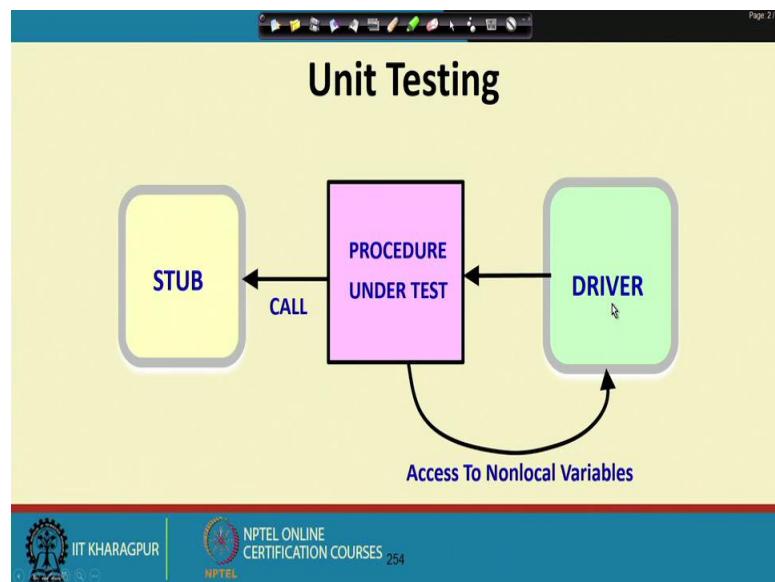
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 253 | NPTEL

We use the term unit testing when we either execute an individual method and test it, unit testing may be applied at the level of modules, we just test unit test a module, we might unit test a class in object oriented code or may unit test a component in a component based software.

But, the problem here is that if we take up a unit and we want to execute it, but then there must be some software which supplies it data invokes that unit. And similarly the unit that we are testing it might need to invoke other units. There might be call to other functions and so on. Since we are testing the unit in isolation, we need to have this 2 small code written the driver is the one which will call the unit and supply the necessary data. And the stubs are the functions some dummy functions which the unit will call so, the driver and stub software written during unit testing.

The driver and stub are two, small software so the tester has to write some code here, the driver simulates the behaviour of the function that should call the unit and also supplied the necessary data the test data. They stop on the other hand these are the functions that need to be called by the unit.

(Refer Slide Time: 27:55)



We can represent the same thing in this form that in order to do unit testing for some unit, the tester has to write 2 small software one is the driver, which calls the unit under test and also supplies it the necessary data and also if the unit needs to excess some global data, then it provides the global variables or global data to the unit. And also needs to write the stub, which the procedure under test that is the unit needs to call. So, for unit testing we need the drivers and stubs to be written before the unit testing can be carried out.

(Refer Slide Time: 28:54)

- Unit testing can be considered as which one of the following types of activities?
 - Verification
 - Validation

Now, let's have a small quiz. So, far we looked at unit testing in this lecture. And we saw that in unit testing we just isolate one unit and then test it, it is adding some test data and we write the driver and stub for that unit, but then unit testing can be considered as which one of the following types of activity, is it a verification activity or a validation activity.

The answer is that it is a verification activity, unit testing is a verification activity, because by definition validation is checking the full developed software against its requirements, in unit testing we do not do that we just take out a small unit and then test it against its functional requirement for that unit. So, unit testing is not validation testing, it is a verification testing, we are almost at the end of this lecture we will stop at this point and then continue with the next lecture.

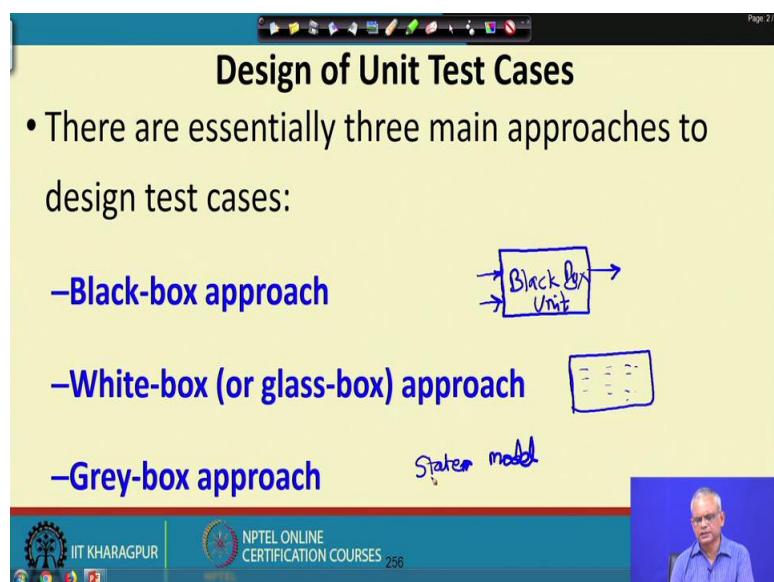
Thank you.

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

Lecture - 46
Unit testing strategies-I

In the last lecture we had started to discuss about unit testing. We looked at some aspects of unit testing the stubs and the drivers. The stub and driver are small software that needs to be written by the testers before they can start carrying out the unit testing and of course, the unit testing is done by the developer themselves. Therefore, these are the developers who do unit testing and write the stubs and drivers. The independent testers do the integration and system testing who are part of the test team.

(Refer Slide Time: 01:12)



Design of Unit Test Cases

- There are essentially three main approaches to design test cases:
 - Black-box approach**
 - White-box (or glass-box) approach**
 - Grey-box approach**

The unit testing is performed by the developers themselves and before they can unit test unit, they need to write the drivers and stubs and then carry out the unit testing. Now let's look at more detail into unit testing. How does one design unit test cases? Let's say we want to do unit testing for a function as I was mentioning that unit testing can also be done for a module or for a class or for a component. Let us now assume that we are interested to do unit testing for a function.

Now, what are the main approaches to design unit test cases? One is called as the black box approach, the other is white box approach and the third is grey box approach. For a

function unit testing is basically a black box and white box. We will see how to design those test cases and why these are called as black box and white box, but for class or a module or a component we might also have to do a grey box approach. In black box approach, we do not need to know the entire detail of the software we just view the unit as a black box. This is our unit we just know that if we give input to the black box unit, it should behave in some way in the sense that it will produce some output. So, the black box approach we design the test cases by observing the inputs that should be given and output that should be produced.

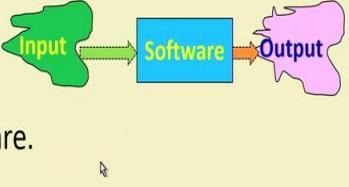
On the other hand in the white box or glass box approach we know the details of the code that is there. Therefore, we can create a model of the code for example, a control flow diagram or something. Based on the internal knowledge of the code, we might design some test cases. So, this is called as a glass box, because we need to check the internal code here we do not even bother about what is the code we just look at it is specification, it is what is the input and what output should be produced and based on that we design the test case, on the other hand in the grey box approach we have something in between a black box and a white box.

We use the design of the unit may be a class diagram, may be a state model, a class diagram or may be a call relation between modules and so on. So, these are the design models which are intermediate between a black box and white box, do not really need to look at the code and neither is it a black box, we have knowledge which is intermediate between a black box and a white box and that is why it is called as a grey box. If it is a pure function that we are testing we need to test only black box and white box, but for classes, components and so on we might have to do a grey box testing as well.

(Refer Slide Time: 05:38)

Page 3/3

Black-Box Testing

- Test cases are designed using only **functional specification** of the software:
 - Without any knowledge of the internal structure of the software.
- Black-box testing is also known as **functional testing**.


IIT Kharagpur
NPTEL ONLINE CERTIFICATION COURSES 257




Now, first let's do black box unit testing, the black box unit testing is done based on the functional specification of the software or functional specification of the unit. So, given the unit, we do not need to look at the code, we just see what is its input output behavior and based on that we design the test cases. We should not look at the internal of the software but only the input output behavior. The black box testing is also called as functional testing, because based on the knowledge of the functionality without looking at the code we test it so, it is also called as a functional testing.

(Refer Slide Time: 06:38)

Page 3/3

What is Hard about BB Testing

- Data domain is large
- A function may take multiple parameters:
 - We need to consider the combinations of the values of the different parameters.
$$f(p_1, p_2)$$


IIT Kharagpur
NPTEL ONLINE CERTIFICATION COURSES 258




Now, let's try to appreciate why black box testing can be hard. Here we just look at the input data and the output behavior that is what it should output and then design test

cases. The main complexity here is that the data domain for any typical software is large and therefore, we need to decide which test cases to apply because we need to have a optimal number of test cases and we have to select that out of the billions and trillions of test data that are possible. Not only that a function may take multiple parameters so, we need to fix the values of those parameters.

So, if a function takes just one parameter we need to just test with respect to that parameter, but if it takes 2 parameters and there are number of values that are possible for this parameter and for this parameter, this parameter 1 and parameter 2 for a function. Then we have to possibly test with various combinations of values may be fix this as one this and test with respect to this, may be fix this and test with respect to this and so on. So, we need to try out various combinations and that makes testing much more challenging and complex.

(Refer Slide Time: 08:43)

What's So Hard About Testing?

- Consider `int check-equal(int x, int y)`
- Assuming a 64 bit computer

$$2^{64} \times 2^{64} = 2^{128}$$
 - Input space = 2^{128}
- Assuming it takes 10secs to key-in an integer pair:
 - It would take about a billion years to enter all possible values!
 - Automatic testing has its own problems!

First let's convince our self that, even a very small function just 2 line function can become extremely hard to test if we want to do the black box testing thoroughly that is a exhaustive black box testing with all possible input values. Let's consider a very trivial function name of the function is check equal, takes 2 integer parameters and then checks if these 2 parameters are the same.

Basically just one line is here, if x equal to y return 1 else return 0, but then we are not looking at the internals we are just doing a black box testing. So, what we will do is, we

will check whether this function works well for all possible combinations of input data. Now let's say we are using a 64 bit computer, in a 64 bit computer each integer is represented using 64 bit and for simplicity let's assume because integer representation we do not want to go that level, but we just say that 64 bits are used and therefore, 2^{64} inputs are possible for the first parameter. Similarly for the second parameter 2^{64} different inputs are possible and therefore, the total number of inputs with which the function needs to be checked is all possible combinations of the first parameter and the second parameter which is $2 * 2^{64} = 2^{128}$.

This is a huge number, we can read 2^{10} , 2^{20} , 2^{30} , but I cannot even read what is 2^{128} . It is extremely large 2^{10} is kilo so, 20 is million, 30 is billion, trillion and so, on. I do not even know how to read, it is extremely large number and let's assume that we are testing manually this 2^{128} possible values with which we need to test and let us say that we are expert in typing we can type very fast and just take 10 seconds to enter each test data. If we compute that the number of hours that will come here is very large and we can easily compute how many hours and then we might take a billion year to enter all possible values and that is just to test this small program. Therefore, the black box testing is hard if we consider all possible inputs. But our objective now is to find out black box test strategies which will drastically reduce the number of test cases may be 3, 4 or something, but they should be almost as effective as the exhaustive testing. So, that is our objective. With that objective let us see what are the test strategies that we have.

(Refer Slide Time: 12:48)

The slide has a dark blue header bar with various icons. The main title 'Solution' is centered in a large, bold, black font. Below the title is a bulleted list:

- Construct model of the data domain:
 - Called Domain based testing
 - Select data based on the domain model

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, 'NPTEL ONLINE CERTIFICATION COURSES 260', and a small video window showing a person speaking.

In all these test strategies that we discussed, we will create a model of the data domain because we know the input data, the output data, we will create a data model and based on the data model we will design the test cases.

(Refer Slide Time: 13:20)

• Software considered as a black box:
– Test data derived from the specification
• Also known as:
– Data-driven or
– Input/output driven testing
• The goal is to achieve the thoroughness of exhaustive input testing:
– With much less effort!!!!

Black Box Testing

System

Input Output

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 262

NPTEL

(Refer Slide Time: 13:24)

• To design test cases:
– Knowledge of internal structure of software necessary.
– White-box testing is also called structural testing.

White-box Testing

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 261

NPTEL

On the other hand in the white box testing, we know the internal structure that is the code. We can check and therefore, it is also called as structural testing because we know the structure of the software.

Now, let us look at a black box testing and after that we will look at white box testing. In black box testing, we look at the system, the black box and then we know the input output behavior. It is also called as data driven testing or input output testing. We look at this first and we have already seen that exhaustive testing is very difficult. We need to have some test strategies which are almost comparable with respect to the effectiveness or thoroughness of testing of the exhaustive testing, but the number of test cases should be much less and should take very less effort compared to exhaustive testing.

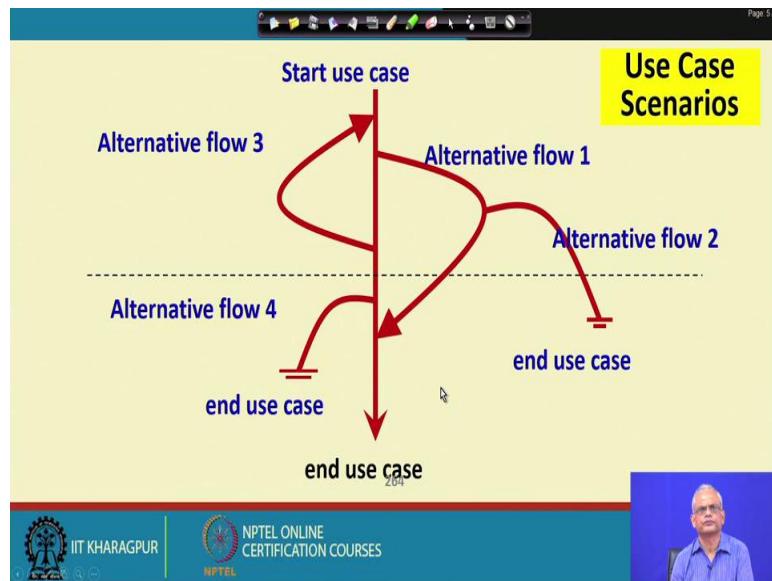
(Refer Slide Time: 14:34)

The slide has a yellow background with a blue header bar. The title 'Black-Box Testing' is in the top right corner of the yellow area. Below the title, there is a bulleted list of six test strategies. At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 263', and a small video window showing a person speaking.

- Scenario coverage
- Equivalence class partitioning
- Boundary value testing
- Cause-effect (Decision Table) testing
- Combinatorial testing
- Orthogonal array testing

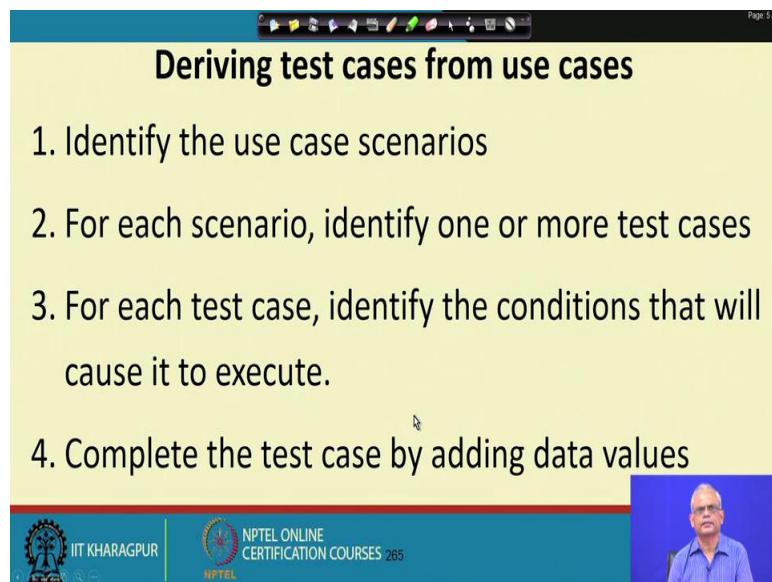
There are a large number of black box test strategies that have been reported. We will look at only few of them, the scenario coverage, equivalence class partitioning, boundary value testing, cause effect testing, combinatorial and orthogonal array testing.

(Refer Slide Time: 14:58)



First let's look at the scenario based testing. From our discussion on scenarios which we had during the object oriented discussion and object oriented design, we know that a use case consists of many scenarios, one is called as the main scenario and then we have the alternate scenarios. At some point in the main scenario, the alternate flows occur. These may actually end or they might just have a separate set of actions and then have similar action as the main line. Now if this is a kind of description of a use case with different scenarios then how do we design the scenario based testing.

(Refer Slide Time: 16:00)



The first thing is to identify the use case scenarios and typically for requirements document is well written. We will have identified the scenarios there in the form a text

description. For each scenario, we will identify one or more test cases and these test cases are actually will execute the scenarios, and then we identify the test data and then also identify the state at which the system should be there for the test data to be entered and that we say that, identify the conditions that will cause this to execute and then complete the test cases by adding the data values.

(Refer Slide Time: 16:56)

Scenario number	Originating flow	Alternative flow	Next alternative	Next alternative
1	Basic flow			
2	Basic flow	Alt. flow 1		
3	Basic flow	Alt. flow 1	Alt. flow 2	
4	Basic flow	Alt. flow 3		
5	Basic flow	Alt. flow 3	Alt. flow 1	
6	Basic flow	Alt. flow 3	Alt. flow 1	Alt. flow 2
7	Basic flow	Alt. flow 4		
8	Basic flow	Alt. flow 3	Alt. flow 4	

Identify use case scenarios: Example

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES



If we represent our scenarios in this form that first is basic flow, then basic flow with alternate flow 1, alternate flow 1 with alternate flow 2 etc. We develop the table and then each of this are a scenario and we need to have that many test cases.

(Refer Slide Time: 17:25)

The slide has a yellow header bar with the text "Identify the test cases". Below this, a bulleted list details the parameters of any test case:

- Parameters of any test case:
 - Conditions
 - Input (data values)
 - Expected result
 - Actual result

Test case ID	Scenario/ condition	Data value 1	Data value 2	Data value N	Exp. results	Actual results
1	Scenario 1					
2	Scenario 2					
3	Scenario 3					

At the bottom, there are logos for IIT Kharagpur and NPTEL, along with a video player showing a speaker.

For each scenario we need to identify the conditions or the state at which the input to be given, the data input values there may be many inputs to be given, the expected result for each of the input and the actual result that was observed.

Let's look at the next black box testing; the scenario coverage was rather intuitive even straight forward, just identify all the scenarios by looking at the SRS document and then we just list out all the scenarios and then the condition under which these will execute and then the test data and expected output.

(Refer Slide Time: 18:22)

The slide has a yellow header bar with the title "Equivalence Class Partitioning". Below this, a bulleted list describes the input values to a program:

- The input values to a program:
 - Partitioned into equivalence classes
- Partitioning is done such that:
 - Program behaves in similar ways to every input value belonging to an equivalence class.

On the right, there is a diagram of four boxes labeled 1, 2, 3, 4. Box 1 contains 'x', box 2 contains 'y', box 3 contains 'z', and box 4 is empty. Below the boxes is the text "Diff Set".

At the bottom, there are logos for IIT Kharagpur and NPTEL, along with a video player showing a speaker.

Now let us look at the equivalence class testing. This is another black box testing. In this approach we look at the input data domain and then we partition it into equivalence class. The main idea behind partitioning the input data into equivalence class is that the program will behave similar way for every input belonging to an input equivalence class. So, if this is the set of input to the unit, we identify equivalence classes in the input and the main idea here is that for this let's say equivalence class 1. So, this is the set of data, data set or the data domain of the function and then we have partitioned it into equivalence classes 1, 2, 3, 4 etc. For each equivalence class, all input should execute the software in similar way, that is the same set of statements may get executed of course, here we do not have a knowledge of the internal of the software, but just by observing the behavior we can guess that these are executed in similar manner.

(Refer Slide Time: 20:12)

Equivalence Class Partitioning

- The input values to a program:
 - Partitioned into equivalence classes
- Partitioning is done such that:
 - Program behaves in similar ways to every input value belonging to an equivalence class.
 - At the least, there should be as many equivalence classes as scenarios.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 269 | NPTEL

From each equivalence class, we select one value with which to test and if the scenarios are extremely simple then each scenario need one equivalence class, but then it is much more complicated. Normally every scenario might need hundreds of equivalence classes or thousands of equivalence classes. So, we need to pick a value from each equivalence class, one of the main assumption in equivalence class is that we want to reduce the number of test cases with which the system needs to be tested and if these are all the set of test cases that are possible in exhaustive testing we have partitioned it into equivalence classes E1, E2 and E3.

(Refer Slide Time: 20:49)

The slide has a title 'Why Define Equivalence Classes?' and a bullet point '• Premise:' followed by two points:

- Testing code with any one representative value from a equivalence class:
- As good as testing using any other values from the equivalence class.

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a speaker.

The way we have partitioned into E1, E2 and E3 is that each element in equivalence class E1 they behave similarly, the system execute similarly for each of the inputs. So, if we execute one of the data here then we have tested that behavior of the unit. So, our work now becomes to test with one representative value from each equivalence class and defining the equivalence class really helps, because testing using one value is equivalent to testing any other value in that equivalence class. Each equivalence class, typically may contain thousands or millions of data points, we are assuming that executing using one of these points. Since the behavior is similar it is same as executing all of them and therefore, the equivalence class testing drastically reduces the number of test cases.

(Refer Slide Time: 22:33)

Page 77

Equivalence Class Partitioning

- How do you identify equivalence classes?
 - Identify scenarios
 - Examine the input data.
 - Examine output
- Few guidelines for determining the equivalence classes can be given...


IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES 271

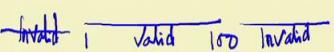



Even though it is a very good, effective technique and reduces the number of test cases drastically, but the question that we are confronted is that how does one go about identifying the equivalence classes.

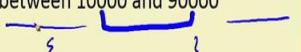
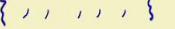
One is we need to identify the scenarios, we need to examine the input data, we need to examine the output data, and then based on these we need to design the equivalence classes, but then that is a bit of a vague statement can we tell something more concurrent? We will give that in the form of a few guidelines.

(Refer Slide Time: 23:20)

Page 77

- If an input condition specifies a range, one valid and two invalid equivalence class are defined. E.g. 1 to 100
 
- If an input condition specifies a member of a set, one valid and one invalid equivalence classes are defined. E.g. {a,b,c}
- If an input condition is Boolean, one valid and one invalid classes are defined.

Example:

- Area code: range --- value defined between 10000 and 90000
 
- Password: set - six character string.
 

Guidelines to Identify Equivalence Classes


IIT KHARAGPUR
Jerry Gao Ph.D., NPTEL ONLINE CERTIFICATION COURSES 272
NPTEL



The guideline is that if the input data is a range, we will have 2 immediate equivalence classes one is a valid and two invalid. So, the range is 1 to 100 anything between 1 to 100 is a valid data and anything more than 100 is invalid and anything less than 1 is invalid. So, we have one valid equivalence class and one invalid equivalence class which has contents values less than the minimum value that is acceptable and one which has values higher than highest value that is acceptable. So, there are two invalid equivalence classes and one valid equivalence class.

If the input is the set of data like let's say a, b, c. These are all valid data the system will respond to this. But then we have invalid data which are not part of this set. Anything that is not part of this set is an invalid equivalence class. So, for a set of data, we have two equivalence classes, one valid and one invalid, valid belongs to the set, invalid is anything other than that.

If the input condition is a boolean, again we have one valid and one invalid i.e., we give a boolean value or do not give a boolean value. Just to give some example, if we want to enter an area code let say the area code is between 10000 and 90000. So, the valid values are between these two and then anything more than 90000 are invalid, anything less than 10000 are invalid.

So, one valid and two invalid, what about a password which contains 6 characters, a string of 6 characters, containing 6 characters in a string, we can consider it as a set of values. So, all those strings which contain 6 characters, they are the set of values. So, we have one valid password which is any of these members of the set and invalid password which is not part of this, may contain 5 characters or something or may be special characters.

(Refer Slide Time: 26:27)

Page 8/8

Equivalent class partition: Example

- Given three sides, determine the type of the triangle:
 - Isosceles
 - Scalene
 - Equilateral, etc.
- Hint: scenarios expressed in output in this case.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 273

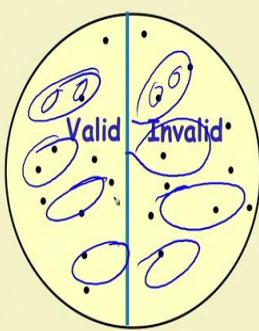
Now, let's look at an example how we do equivalence class partition. Let us say we have given 3 sides of a triangle and we want to decide whether it is an Isosceles, Scalene, Equilateral etc. So, this is a simple function, we take argument 3 sides of a triangle and displays what is the type of the triangle and we want to design equivalence class partition for this. Looking at the output here gives us the hint about the equivalence classes. So, all the data sets for which it will produce isosceles is equivalence class, scalene is another equivalence class, equilateral another class.

(Refer Slide Time: 27:31)

Page 8/8

Equivalence Partitioning

- First-level partitioning:
 - Valid vs. Invalid test cases



 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 274

So, here the equivalence classes are easily identifiable from the output, but for any input data, we have always 2 equivalence classes, one valid and one invalid or there may be several invalid. So, to start with, there is one valid equivalence class and another invalid equivalence class and we might have several invalid equivalence classes here which we might later identify and also there may be several valid equivalence classes out of this.

So, given a problem to design the equivalence tests, we first define 2 equivalence classes the valid and invalid and then look at further division, the valid into set of equivalence classes and each of this may again contain equivalence classes. We will just look at that. In the next lecture right now we are we almost at the end of this lecture, we will stop here.

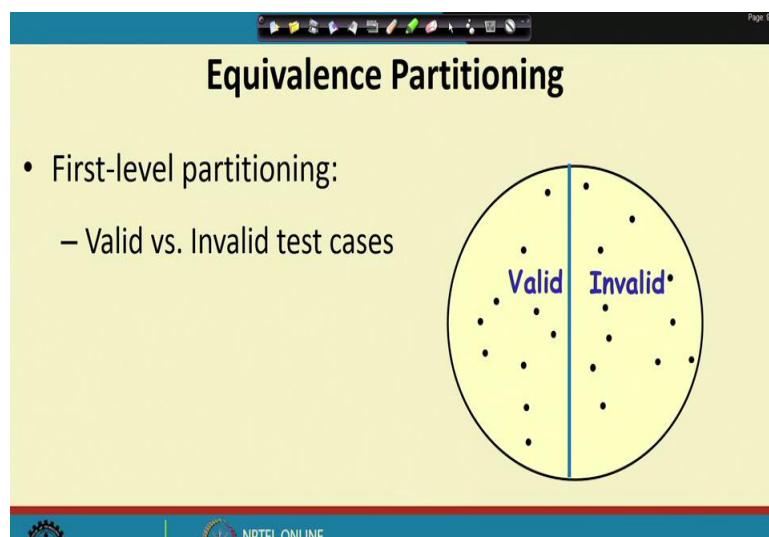
Thank you.

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

Lecture - 47
Unit testing strategies-II

Welcome to this lecture. In the last lecture we were discussing about the equivalence class partitioning based on Unit Testing. We had said that the equivalence class based unit testing. By observing the input and output data, we analyze the input and output data and determine the equivalence classes. Once we identify the equivalence classes, we just take one data from each equivalence class and that forms our test suite. But, the main question that we were trying to address is that how do we go about identifying the equivalence classes.

(Refer Slide Time: 01:10)



The slide is titled "Equivalence Partitioning". It contains a bulleted list: "First-level partitioning:" followed by "- Valid vs. Invalid test cases". Below the list is a diagram of a circle divided into two equal halves by a vertical line. The left half is labeled "Valid" and the right half is labeled "Invalid". Numerous small dots are scattered throughout the circle, with a higher density in the "Valid" half. The slide has a blue header bar with icons and a blue footer bar containing the IIT Kharagpur logo and NPTEL Online Certification Courses 274.

Now, let's continue our discussion further to identify the equivalence classes. We always can say that there are 2 types of equivalence classes; one is the valid set of equivalence classes and the other is invalid set. The first task is to identify what is the valid equivalence class and what are the invalid equivalence classes. Once we identify these 2 equivalence classes, then we can identify further equivalence classes which are valid and further equivalence classes which are invalid.

(Refer Slide Time: 01:50)

Page 9/9

Equivalence Partitioning

- Further partition valid and invalid test cases into equivalence classes

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 275

NPTEL

A video player window shows a man speaking.

We have just represented here that once we identify the valid and invalid equivalence classes, we can identify further multiple valid equivalence classes and multiple invalid equivalence classes.

(Refer Slide Time: 02:13)

Page 9/9

Equivalence Partitioning

- Create a test case for at least one value from each equivalence class

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 276

NPTEL

A video player window shows a man speaking.

Once we do that, we can pick any value from each of this and that will form our test cases. But then the question is that how does one be sure that he has identified all the valid and all the invalid equivalence classes. It is more to do with practice. Identifying equivalence classes is not really trivial that you just look at the problem and then say that

these are the valid and invalid. You need to analyze the problem and based on the experience of solving many problems, we can say that which are the valid and invalid equivalence classes, but unless we have enough practice, it is likely that we might miss out some of the equivalence classes and therefore, the equivalence class testing may not be done very thoroughly.

(Refer Slide Time: 03:29)

Page 9/9

Equivalence Class Partitioning

- If the input data to the program is specified by a range of values:
 - e.g. numbers between 1 to 5000.
 - One valid and two invalid equivalence classes are defined.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 277

NPTEL



Let's take start with some very simple examples.

(Refer Slide Time: 03:35)

Page 9/9

Equivalence Class Partitioning

- If input is an enumerated set of values, e.g. :
 - {a,b,c}
- Define:
 - One equivalence class for valid input values.
 - Another equivalence class for invalid input values..

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 278

NPTEL



See how to go about designing the equivalence class tests. The simplest example is only 1 input data, let say an integer number between 1 to 5000. So, this is the range of values. Therefore, we have 1 valid equivalence class that is within 1 and 5000 and there are 2 invalid equivalence class, 1 is greater than 5000 and the other is less than 1.

Now let's say a function takes a set of values a, b, c. These letters are valid and anything else is invalid. So, we have 2 equivalence classes, one is valid equivalence class, where we give a value a, b or c and then there is invalid equivalence class, which gives a value other than a, b, c, may be d or something.

(Refer Slide Time: 04:56)

The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the word "Example" is centered in a bold black font. The main content of the slide is a bulleted list:

- A program reads an input value in the range of 1 and 5000:
 - Computes the square root of the input number

Below the list is a diagram showing an input value "i" entering a yellow circle labeled "SQRT". The output is the square root symbol \sqrt{i} . At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES 279", and a small video thumbnail of a man speaking.

Let's take another example, where we have a function named as square root, which takes a value input value in the range 1 to 5000 and produces the square root. So, the input value by observing we find that there is 1 valid equivalence class and 2 invalid equivalence classes.

(Refer Slide Time: 05:23)

The slide shows a horizontal number line from 1 to 5000. The value 1 is labeled 'Invalid'. The range between 1 and 5000 is labeled 'valid'. The value 5000 is labeled 'Invalid'. A blue arrow points from the 'invalid' label at 1 towards the 'valid' range, and another blue arrow points from the 'valid' range towards the 'invalid' label at 5000.

Page: 10 / 10

Example (cont.)

- Three equivalence classes:
 - The set of negative integers,
 - Set of integers in the range of 1 and 5000,
 - Integers larger than 5000.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 280

NPTEL



The valid equivalence class is anything between 1 to 5000. 1 invalid equivalence class is anything more than 5000 and then another invalid equivalence class is the set of negative numbers.

(Refer Slide Time: 05:41)

The slide shows a horizontal number line from -5 to 6000. The value -5 is labeled 'Invalid'. The value 1 is labeled 'Invalid'. The range between 1 and 5000 is labeled 'valid'. The value 5000 is labeled 'Invalid'. A blue arrow points from the 'invalid' label at -5 towards the 'valid' range, and another blue arrow points from the 'valid' range towards the 'invalid' label at 5000.

Page: 10 / 10

Example (cont.)

- The test suite must include:
 - Representatives from each of the three equivalence classes:
 - A possible test suite can be:
 $\{-5, 500, 6000\}$.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 281

NPTEL



Then we take 1 representative value from each equivalence class. So, 1 negative value, 1 value which is valid and 1 value which is more than the valid value which is 6000 and the largest valid value is 5000. So, these 3 values form the equivalence class test for this simple problem.

(Refer Slide Time: 06:13)

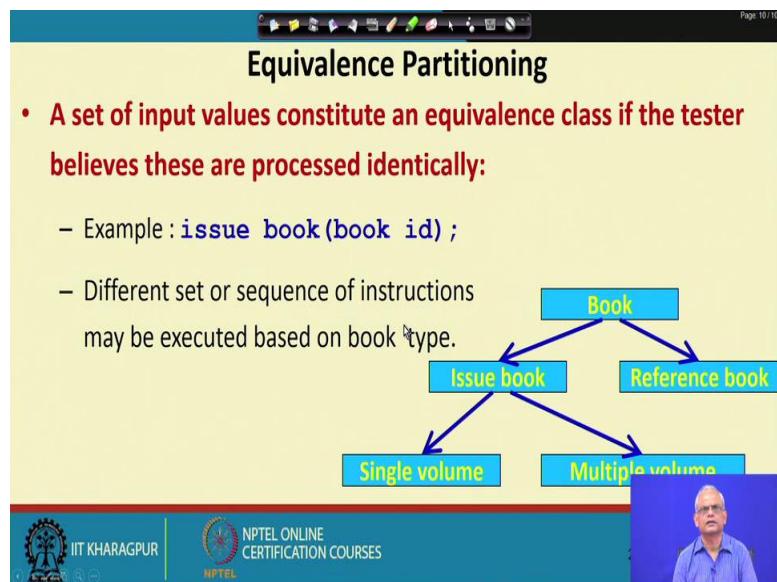
Page 10 / 10

Equivalence Partitioning

- A set of input values constitute an equivalence class if the tester believes these are processed identically:
 - Example : **issue book (book id)** ;
 - Different set or sequence of instructions may be executed based on book type.

```
graph TD; Book[Book] --> IssueBook[Issue book]; Book --> ReferenceBook[Reference book]; IssueBook --> SingleVolume[Single volume]; IssueBook --> MultipleVolume[Multiple volume]
```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES



Now, consider another example. Let's say we have a function, name of the function is issue book and it takes the id of a book may be the I S B N number or something as parameter and then it issues the book.

Can we identify the equivalence class for this function? We intuitively know how the issue book works, just takes a book and then it issues the book, the member of course. We have the valid and invalid set for this, a valid book id and an invalid book id, but then among the valid book id, what are the equivalence classes? One is that it is an issuable book and it is issued out, if it is a reference book then the behavior is different, will say that reference book cannot be issued out. Among the issuable books it may be a single volume book or a multiple volume book. If it is a single volume book may say it may just issue out, if it is a multiple volume book it may say that you have to take all the volumes together.

So, the valid equivalence classes we might say that there are 3 of this. We need to pick 1 for single volume book, 1 data item input data for a multiple volume book and another for reference book. So, for this example there are 3 valid equivalence classes, reference book, multiple volume, single volume, and there will be invalid class.

(Refer Slide Time: 08:50)

Page: 11 / 11

Equivalence Partitioning: Example 1

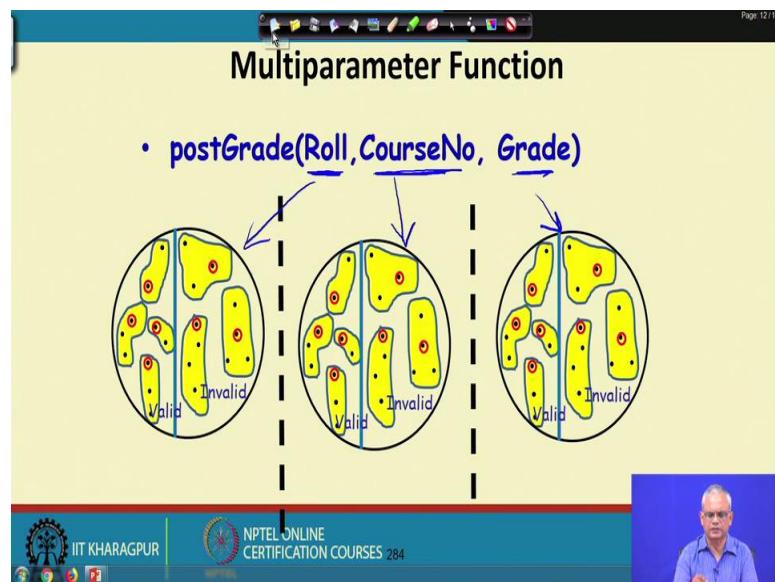
- Example: **Image Fetch-image(URL)**
 - **Equivalence Definition 1:** Partition based on URL protocol (“http”, “https”, “ftp”, “file” etc.)
 - **Equivalence Definition 2:** Partition based on type of file being retrieved (HTML, GIF, JPEG, Plain Text, etc.)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's look at another example. Let's say we have a function name of the function is fetch-image. We give a URL, that's a website address and the function fetches the image and returns the image. So, based on this simple description, what are the equivalence classes? One is of course the valid set and the invalid set, but I am just asking, what are the set of valid equivalence classes for this fetch image?

One equivalence class may be that, whether the URL is given in terms using “http” “https” “ftp” or it's a local “file”. So, we need to these are the set of equivalence classes, based on how the URL is defined. Is the URL given for local file, is an ftp, https or http? But then for the same problem we might have another definition of equivalence classes i.e., what is the type of the image, is it a HTML image, is it a GIF image, is it a JPEG image, is it a Plain Text image? So, observe here that for the same one parameter, we have 2 equivalence classes that can be defined. So, for the valid set of equivalence classes, one is based on the URL protocol and the second is based on the file type and for the URL we have http, https, ftp and file and based on the image type, we have HTML, GIF, JPEG, Plain Text etc. This forms our set of valid equivalence classes. We need to pick one value from each of the equivalence class. The point that I want to emphasize here is that there may be different ways in which we might define the valid equivalence class, we need to look at all the characteristics of the input data just like here one characteristic was the type of the protocol used, the second is the type of the image file that is used.

(Refer Slide Time: 12:25)



But then so far we have been looking at the simple case of a function taking a unit or taking just one parameter. But what happens if a unit takes multiple parameters? Let's say we have a function called as `postGrade`. That is the teacher wants to post the grade, and the `postGrade` function takes 3 parameters; one is the Roll number, the second parameter is the Course Number, and the third one is the Grade obtained in that.

In this case, we can define equivalence classes for each of the 3 parameters here for the unit. But then, while giving the parameter 3 values of parameter do we test it for all possible combinations of these or do we select 1 and for a specific value of this? Now let us answer this question.

(Refer Slide Time: 13:59)

int Normalization Factor;
postGrade(Roll, CourseNo, Grade)
{ Grade=Grade*NormalizationFactor
-----}
Multiparameter Function Accessing Global Variables

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 285

NPTEL

A small video window in the bottom right corner shows a man speaking.

But, not only the number of parameters, but also a function might use some global variables. Look at the function postGrade, let's say it uses a normalization factor while computing the grade while posting the grade not only the teacher defines the roll course number and grade, but internally it does a normalization by using a normalization factor. In that case we need to consider all possible equivalence classes that may be defined for the normalization factor variable. Therefore, we will have 4 sets of equivalence classes here for this function. We need to consider combinations of values for various equivalence classes, but then again the question remains that how do we define the combinations of values that we have chosen for each equivalence class? Do we consider all possible combinations adjustably for each parameter or what do we do? Let's try to address that question.

(Refer Slide Time: 15:27)

Input	Valid Equivalence Classes	Invalid Equivalence Classes
An integer N such that: -99 <= N <= 99	?	?
Phone Number Area code: [11,..., 999] Suffix: Any 6 digits	?	?

Let's see a multi parameter equivalence class example, the input to a certain function is 2 parameters. One is an integer value such that the integer lies between -99 to +99. It also takes another parameter, which is a phone number; the phone number contains an area code of any 6-digit number. We know that we have to define valid equivalence class and invalid equivalence class. So, for each of these, let's try to first define each of these 2 parameters, what the valid and invalid set of equivalence classes are and then based on that we will try to answer that how to consider the combinations of these two different parameters.

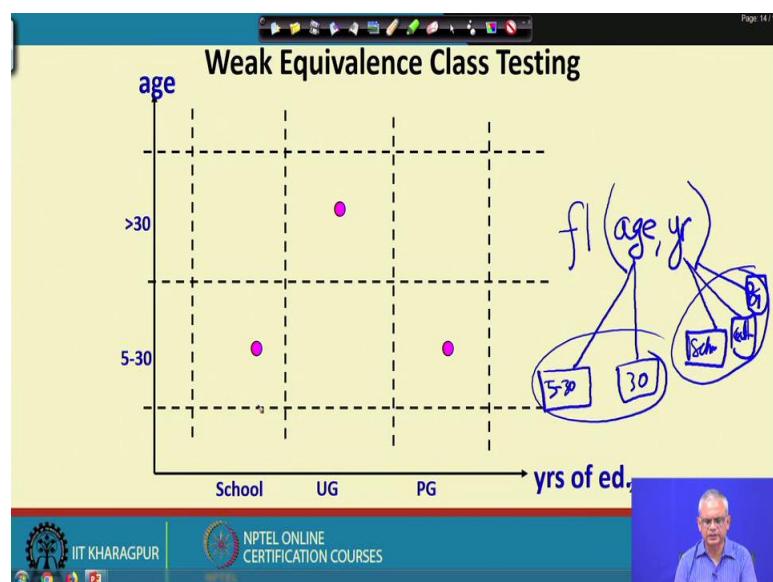
(Refer Slide Time: 16:30)

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, 99]	<-99 >99 Malformed numbers {12, 1-2-3, ...} Non-numeric strings {junk, 1E2, \$13} Empty value
Phone Number Prefix: [11, 999] Suffix: Any 6 digits	[11,999][000000, 999999]	Invalid format 5555555, Area code < 11 or > 999 Area code with non-numeric characters

For the first parameter, we have a valid equivalence class, where the number lies between -99 to +99. Then we have invalid equivalence classes like less than -99 and greater than +99, numbers which are malformed like 1 - 2 - 3 etc., non-numeric strings, empty value- all these are invalid equivalence classes.

For the second parameter phone number, it consists of 2 ranges. So, we might consider that the area code is between these 11 and 999 and any of these are in these correct range, but there can be invalid due to being out of range for the first argument, first part, second part. For example, we might have invalid format area code is less than 11 or 999. Area code with non-numeric characters these are the invalid equivalence classes. Now the question remains that, how do we combine these 2 parameters?

(Refer Slide Time: 18:05)

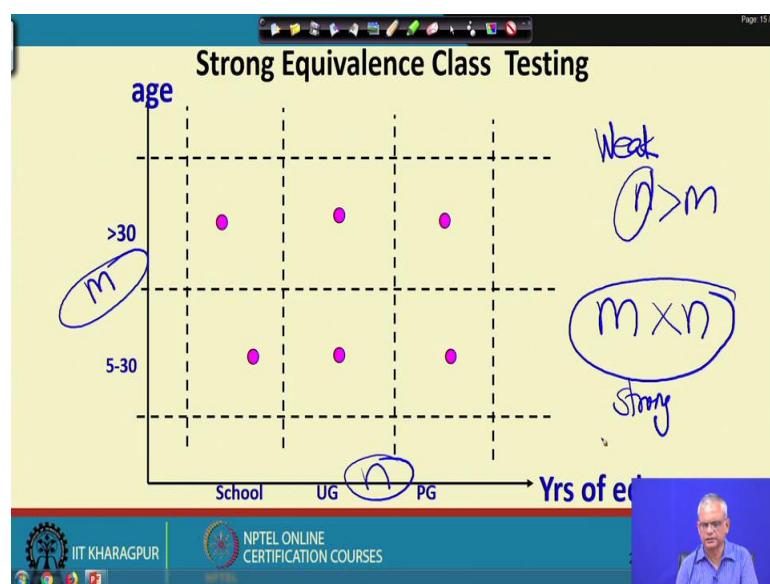


One way is to combine two different parameters called as weak equivalence class testing. Let's say that a certain function takes 2 parameters, one is a call that function as f_1 or something and it takes 2 parameters, one is age and the other is years of education. Age is an integer and for age we have identified the equivalence classes 5 to 30, greater than 30, number of years of education based on that we have identified school, education, college, under graduate college or post graduate. So, the first parameter has 2 equivalence classes, the second has 3 equivalence classes.

We plot these 2 parameters here in the above diagram. We try to cover all of these that is our test cases should be able to test both the age equivalence classes, 1 value from the

age equivalence class and also 1 value of the year of education. In that case using just 3 test cases will be able to cover both the valid age equivalence and the valid year of education. Similarly, if there are 3 or 4 parameters, we can try to cover all of them and that we call them as the weak equivalence testing.

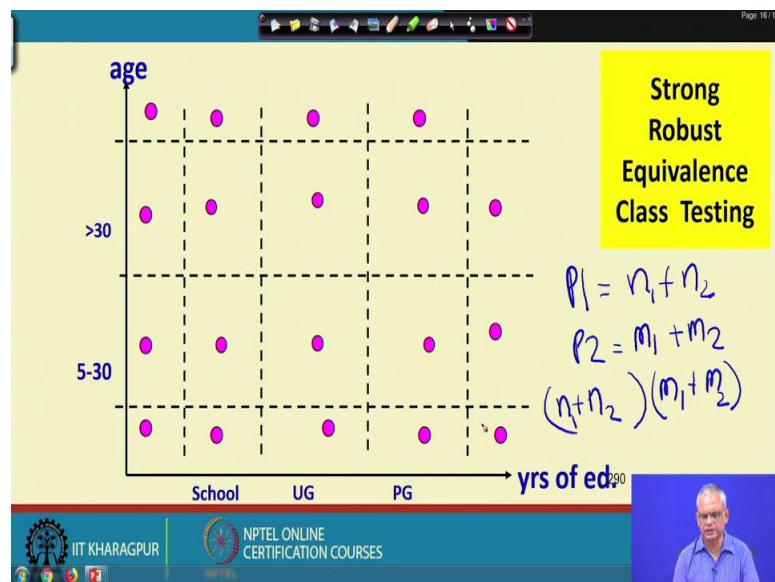
(Refer Slide Time: 20:30)



We can also define a strong equivalence class testing here obviously, the number of test cases is much more. Here, not only that we want to cover all values of the 2 parameters, but also we ensure all possible combinations of the 2 parameters. So, 5-30 is combined with all the 3 of this, 3 valid of the years of education similarly greater than 30 is combined with all the 3 or if we look at another way. For the second parameter year of education, let's say school is combined with both the valid age, UG is combined with both the valid age equivalence classes and so on.

So, the number of test cases required here, if the first parameter has m number of equivalence classes, and the second has n number of equivalence classes. Obviously, we need $m \times n$ number of test cases to achieve strong equivalence class testing. But what about weak equivalence class testing, if a 2 parameters with m and n and let's say n is greater than m , what is the number of test cases that we need ? We may just think about it, but then that is equal to n . So, this is the number of test cases required for weak equivalence class testing and this is the number of test cases required for strong equivalence class testing.

(Refer Slide Time: 22:37)



But, we might also like to consider the invalid values for both the parameters, then we call it as robust equivalence class testing. Here the number of test cases required is total number of valid equivalence classes for 1 parameter plus the number of invalid equivalence classes. Similarly, the number of valid and invalid equivalence class for the other parameter. Let's say the parameter 1 there are n_1 valid equivalence classes and n_2 invalid equivalence classes and for parameter 2, we have m_1 invalid equivalence classes, and m_2 valid equivalence classes. Then for robust equivalence class testing we need $(n_1+n_2)*(m_1+m_2)$ number of test cases and obviously, the robust equivalence class testing will expose many more bugs than a weak testing or just strong testing.

(Refer Slide Time: 24:20)

The image shows a computer monitor displaying a presentation slide. The title 'Quiz 1' is in the top right corner. The slide content is a list of bullet points:

- Design Equivalence class test cases:
- A bank pays different rates of interest on a deposit depending on the deposit period.
 - 3% for deposit up to 15 days
 - 4% for deposit over 15 days and up to 180 days
 - 6% for deposit over 180 days upto 1 year
 - 7% for deposit over 1 year but less than 3 years
 - 8% for deposit 3 years and above

At the bottom of the slide, there is a navigation bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 291', and a small video thumbnail of a person speaking.

Now, based on what we discussed, let's try to solve one more example. Let's say we have a problem, we have a function that computes the interest for a certain principal and there are different rates of interest on a deposit depending on the deposit period. So, we have this function `computeInterest`, it takes deposit amount and a period as parameter. We know that the interest computed depends on the period, it should be 3 percent for deposit up to 15 days, 4 percent for 15 days to 180 days, 6 percent for 180 days to 1 year, 7 percent for deposit over 1 year, but less than 3 years and 8 percent for deposit above 3 years. Now, observe here that even though there are 2 parameters, the behavior of the function does not matter much and the amount. So, amount there is only 1 equivalence class, 1 valid equivalence class and then we have invalid equivalence class. Whereas, the period there are 1, 2, 3, 4, 5 so, there are 5 equivalence classes for the period, because depending on the period, it provides different behavior that is different interest rate. Now, the number of test cases required here is for strong equivalence testing is equal to 1, 2, 3, 4, 5. For weak equivalence class testing, it is also similar because the first one is one.

So, both weak and strong equivalence class testing will require 5 test cases, but for robust testing we have 1 valid and 1 invalid for this so, there are 2 classes for the first parameter. The second parameter has 1 invalid class. So, we have 6 here. So, the number of test cases becomes 12 for robust testing.

We are almost at the end of time for this lecture. We will stop here and we will see that there are many more strategies for black box testing, we have just examined 2 simple strategies. One is the scenario based testing, which is the possibly one of the simplest black box testing strategies and we also looked at the equivalence class partition. We will look at few more strategies for black box testing in the next lecture.

Thank you.

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

Lecture - 48
Equivalence Class Testing - 1

Welcome! We will continue our discussion on Equivalence Class Testing. Last time we had just discussed little bit about equivalence class testing and will continue from there.

(Refer Slide Time: 00:30)

The slide is titled "Equivalence Class Partitioning". It contains the following text:

- The input values to a program:
 - Partitioned into equivalence classes
- Partitioning is done such that:
 - Program behaves in similar ways to every input value belonging to an equivalence class.
 - At the least, there should be as many equivalence classes as scenarios.

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a professor.

The main idea behind equivalence class testing is that, we partition the data value. The data that is input to a function, we model it and partition it into equivalence classes. Now, let me just tell little bit about, why we want to partition it into equivalence classes and how does it help in designing test cases. We partition it into equivalence classes, because each data point in an equivalence class exhibits similar behavior. If we think of this as a representation of the control flow in a program, we can see that there are many ways the control can flow and each way the control flows represents a behavior.

So, if we partition the data that is input to a program. If this is the set of data that is to be input to the program and we know that these are the data for which it follows this path and these are the data for which it follows this path. So, these represent different behavior, different output, different type of processing and so on and may be this one corresponds to a behavior exhibited by this and so on.

Now all the data points here correspond to the behavior expressed by this processing. So, the idea here between equivalence class partitioning is that, if we can identify the equivalence classes for which the behavior, for any class is similar for all the data points, then we can just test it with one of the data point here. So, that we know that these behavior is ok. We will also select a data point that will drive it through a different behavior and so on.

That is the main idea here that a program behaves in similar ways to every input value belonging to an equivalence class. So, this is an equivalence class, then every input will exhibit similar behavior, because the set of statements that are getting executed are similar and therefore, if it works for one point where it followed this set of statement execution, then it is reasonable to expect that it will also work satisfactorily for all other points in that same equivalence class. But we also need to check about the behavior when it follows this path and so on this set of statements. But one thing that we must remember is that from a black box description of a functionality that is a use case.

We know that there are alternate scenarios. In the alternate scenarios, this the main line scenario and then there are many alternate scenarios. We can think of that in the main line scenario, certain set of statements get executed and for alternate scenarios, different set of statements get executed. Therefore, if we look at the use case description and look at the main line scenario and the alternate scenarios, we can associate equivalence classes to each of this.

In main line scenario, the data that makes the system execute the main line scenario corresponds to one equivalence class and for every alternate path, we have a different equivalence class. This is one way to identify the equivalence classes, but will see that even when it follows the main line scenario, there may be further equivalence classes in that we will just explore that.

(Refer Slide Time: 05:52)

The slide has a yellow header with the title "Why Define Equivalence Classes?". Below the title is a diagram consisting of three horizontal boxes labeled E1, E2, and E3 from left to right. Box E1 contains red dots, box E2 contains yellow dots, and box E3 contains green dots. To the left of the diagram is a bulleted list: "• Premise:" followed by two points: "–Testing code with any one representative value from a equivalence class:" and "–As good as testing using any other values from the equivalence class." At the bottom of the slide is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "78". On the right side of the footer bar, there is a video player showing a man speaking.

So, let's just repeat here, that the total data input data domain. The input data domain we model it and then we split it into equivalence classes such that, the tester or we if we are doing the testing, the tester feels that for every data point here, the system exhibit similar behavior. Since all of these make the system execute certain set of statements, it is reasonable to execute using any one of these points. We can select any one of these points and similarly with other equivalence classes and that will form our equivalence test cases equivalence class test cases.

(Refer Slide Time: 07:07)

The slide has a yellow header with the title "Equivalence Class Partitioning". Below the title is a bulleted list: "• How do you identify equivalence classes?" followed by four points: "–Identify scenarios", "–Examine the input data.", "–Examine output", and "• Few guidelines for determining the equivalence classes can be given...". At the bottom of the slide is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the number "79". On the right side of the footer bar, there is a video player showing a man speaking.

One way of identifying the equivalence classes is by examining the use case description. The main line scenario and alternate scenarios correspond to different equivalence classes, but beyond that, we said that every equivalence class there may also consist of other equivalence classes, how do we identify that?

One is based on the use case diagram and use case description. We identify the scenarios, we examine the input data which make it execute a specific scenario and those set of input data constitute the equivalence class. But, there are other guidelines for identifying equivalence classes, because this allows us to identify a broad set of equivalence classes, but let's see, what are the other guidelines.

(Refer Slide Time: 08:13)

If an input is a range, one valid and two invalid equivalence classes are defined.
Example: 1 to 100

If an input is a set, one valid and one invalid equivalence classes are defined. Example:
 $\{a,b,c\}$

If an input is a Boolean value, one valid and one invalid class are defined.

Example:

- Area code: input value defined between 10000 and 90000---
- Password: string of six characters ---

Guidelines to Identify Equivalence Classes

The other guidelines are that we look at the input values. If the input is a range of values as we find the description of the functional requirements or the use case. If the input corresponds to a range of values, let's say 1 to 100. If these are the set of valid values, then we have 3 equivalence classes; one is the valid set of equivalence classes and there are 2 invalid equivalence classes on the both ends of the valid equivalence class as shown in the diagram above.

Now, what if the input is a set of values? Let's say only a, b, c, let's say our system or software takes only a character and based on the character, it does certain functions. These are the menu choices let's say a, b, c and these are only the valid menu choices.

You have to input any one of these. In that case, this defines a valid set of equivalence class and anything other than a, b, c is the invalid set of equivalence class.

So, if the input is a set, we need to identify a valid set of equivalence class and invalid set of equivalence class. What if the input is a Boolean value? Again here, we have one valid that is the Boolean value and any other value such as a character, control characters and so on. These will constitute the invalid class.

Now, let's check our understanding for two example problems. Let's say a function takes an area code i.e., the pin code and the pin code has value between 10000 and 90000. So, what are the equivalence classes here? We can see here that this defines a range 10000 to 90000. Therefore, there will be 3 equivalence classes; one is the valid equivalence class between 10000 and 90000 and another less than 10000 and more than 90000. So, the crucial thing to recognize that for the area code or the pin code, it denotes a range of values and as soon as we know that it is a range of values, we can identify 3 equivalence classes, which is valid and 2 invalid set of equivalence classes.

(Refer Slide Time: 11:24)

• If an input is a range, one valid and two invalid equivalence classes are defined.
Example: 1 to 100

• If an input is a set, one valid and one invalid equivalence classes are defined. Example:
{a,b,c}

• If an input is a Boolean value, one valid and one invalid class are defined.

Example:

- Area code: input value defined between 10000 and 90000--- **range**
- Password: string of six characters --- **set**

Guidelines to Identify Equivalence Classes

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 80

Now, let's look at another one, a function that takes a password. Let's say the function is log in and it takes a password and the password consists of a string of 6 characters. Now, what will be the equivalence classes here? It is crucial to recognize that string of 6 characters is actually a set; it is a finite set of all possible string of 6 characters. Therefore, as soon as we recognize that it is a set, will identify 2 equivalence classes in

this case, that is one is a valid equivalence class, which is a member of the set and another is invalid equivalence class, which is less than 6 characters, more than 6 character etc. So, those are the invalid. So, there are 2 equivalence classes here, one is the valid equivalence class, which is a member of the set and invalid equivalence class all the data points which are not the member of the set.

(Refer Slide Time: 12:57)

Equivalent class partition: Example

- Given three sides, determine the type of the triangle:

- Isosceles
- Scalene
- Equilateral, etc.

determineTriangle(int side1, int side2, int side3)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

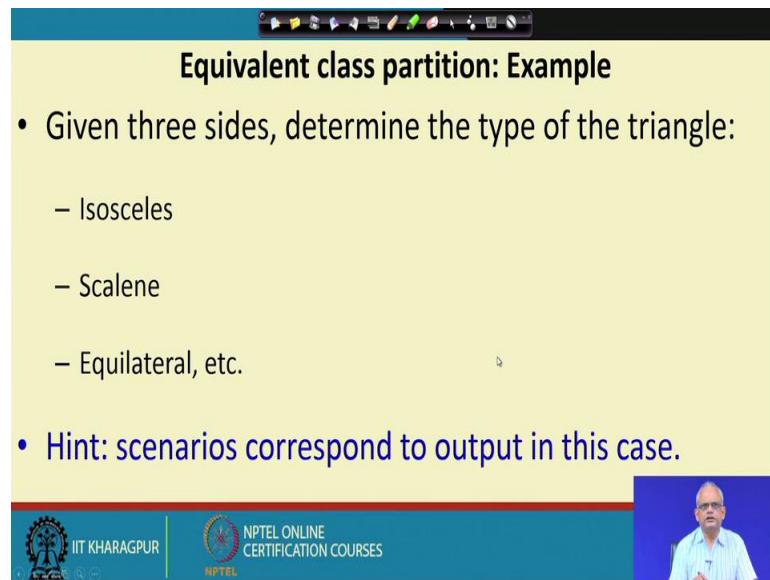
Now, we try to solve some problems. Let's say we have a function which takes 3 integers and determines what is the type of the triangle. Than it forms, may be the name of the function is to determine triangle type. It takes 3 integers side1, side2, and side3. Now, based on whether the triangle defined by this 3 sides, they form an isosceles, scalene, equilateral etc. It defines the type of the triangle, otherwise if it is not a valid triangle it will display invalid triangle.

How do we identify the equivalence classes here? By observing the output here, the display whether it is isosceles, scalene or equilateral, we know that these correspond to different scenarios. If we give set of 3 values, which forms a triangle, then this corresponds to isosceles, scalene, equilateral etc. Therefore, by observing the output, we can get an idea of the scenarios and equivalence classes that these will define.

(Refer Slide Time: 15:01)

Equivalent class partition: Example

- Given three sides, determine the type of the triangle:
 - Isosceles
 - Scalene
 - Equilateral, etc.
- Hint: scenarios correspond to output in this case.

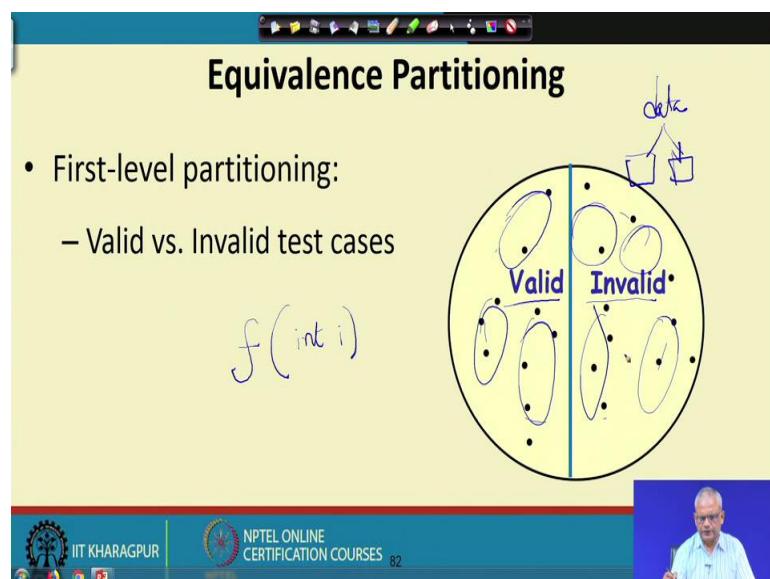


So, for this problem, the initial set of equivalence classes are the scenarios, which correspond to the output in this case.

(Refer Slide Time: 15:15)

Equivalence Partitioning

- First-level partitioning:
 - Valid vs. Invalid test cases



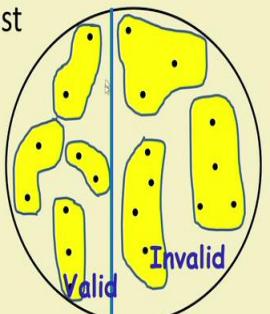
But, let's see if we have any general problem, a function f , where the function f takes a parameter, may be integer, float or something character. Let me just write integer i , f is a function, which takes an integer or float or something, then we can say that to start with will have 2 equivalence classes. The set of data can be partitioned into 2 equivalence classes.

So, these are all the set of data and we can partition the data into 2 equivalence classes; one is the valid set of data which is the valid equivalence class and the invalid set of data which is the invalid equivalence class. So, the 2 equivalence class for every problem we should be able to identify to start with and then as we proceed to find further equivalence classes, will find further valid set of equivalence classes and further invalid set of equivalence classes. Let's see, how we go about doing that.

(Refer Slide Time: 16:46)

Equivalence Partitioning

- Further partition valid and invalid test cases into equivalence classes



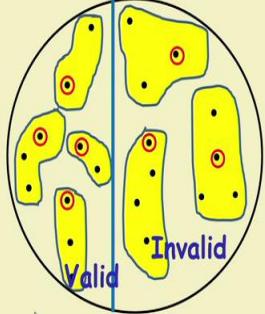
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 83 | 

So, this says that to start with the input data, we will partition it into valid and invalid set of equivalence classes. In the next step, we will further identify equivalence classes within the valid set and also equivalence classes within the invalid set.

(Refer Slide Time: 17:18)

Equivalence Partitioning

- Create a test case using at least one value from each equivalence class



IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 84

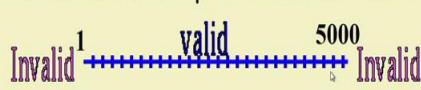
A slide titled "Equivalence Partitioning". It contains a bulleted list: "Create a test case using at least one value from each equivalence class". To the right is a diagram of a large circle divided into several yellow regions, some containing small red circles, representing valid equivalence classes. A single blue vertical band on the right side of the circle is labeled "Invalid". At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a speaker.

Once we have done that within the equivalence classes, we might for some problems find further equivalence classes here, but once we have ultimately found the equivalence classes. Then we just pick one data value from each equivalence class and that forms our equivalence class test cases.

(Refer Slide Time: 17:39)

Equivalence Class Partitioning

- If the input data to the program is specified by a range of values:
 - e.g. numbers between 1 to 5000.
 - One valid and two invalid equivalence classes are defined.



IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 85

A slide titled "Equivalence Class Partitioning". It contains a bulleted list: "If the input data to the program is specified by a range of values: –e.g. numbers between 1 to 5000. –One valid and two invalid equivalence classes are defined.". Below the list is a diagram of a number line with points labeled "1" and "5000". Blue arrows point from these labels to the words "valid" and "invalid" respectively. At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video window showing a speaker.

Now, let's look at some problems and see how to identify the equivalence class test cases? Let's say our function takes an integer between 1 to 5000. These are the valid set of data. Any data, any integer value that is between 1 to 5000 is a valid input. So, this

corresponds to a range of input and we can clearly identify that there is a valid class and there are 2 invalid classes less than 1 and more than 50000.

(Refer Slide Time: 18:28)

The slide has a yellow background. At the top, the title 'Equivalence Class Partitioning' is centered. Below the title, there are two main bullet points:

- If input is an enumerated set of values, e.g. :
 - {a,b,c}
- Define:
 - One equivalence class for valid input values.
 - Another equivalence class for invalid input values..

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 86', and a video player showing a speaker.

Similarly, if we have a set of values like a, b, c, we find one equivalence class which is valid, and another which is invalid and then we will pick up one value here like b or c or something, which is not part of here, may be f or something.

(Refer Slide Time: 18:51)

The slide has a yellow background. At the top, the title 'Example' is centered. Below the title, there is a single bullet point:

- A program reads an input value in the range of 1 and 5000:
 - Computes the square root of the input number

Below the text, there is a diagram showing an input value 'i' entering a yellow circle labeled 'SQRT', which then outputs the square root symbol ' \sqrt{i} '.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 87', and a video player showing a speaker.

Now, let's look at another example here the program reads an input value in the range of 1 to 5000 and computes the square root of the input number. The input is a range here

can easily identify. Therefore, to start with there are 3 classes, 1 valid class and 2 invalid classes, 1 invalid class less than 1 another invalid class more than 5000.

(Refer Slide Time: 19:23)

The slide has a yellow background. At the top, it says "Example (cont.)". Below that is a bulleted list:

- Three equivalence classes:
 - The set of negative integers,
 - Set of integers in the range of 1 and 5000,
 - Integers larger than 5000.

Below the list is a horizontal number line with three points labeled: "Invalid" at the left end, "1" at the center, and "5000" at the right end. The word "valid" is written above the number line between 1 and 5000. There are blue dashed arrows pointing from "Invalid" to "1" and from "5000" to "Invalid".

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video thumbnail of a professor.

So, the set of negative integers, the set of integers between 1 and 5000 and integers larger than 5000, these are the 3 equivalence classes.

(Refer Slide Time: 19:40)

The slide has a yellow background. At the top, it says "Example (cont.)". Below that is a bulleted list:

- The test suite must include:
 - Representatives from each of the three equivalence classes:
 - A possible test suite can be:
 $\{-5, 500, 6000\}$.

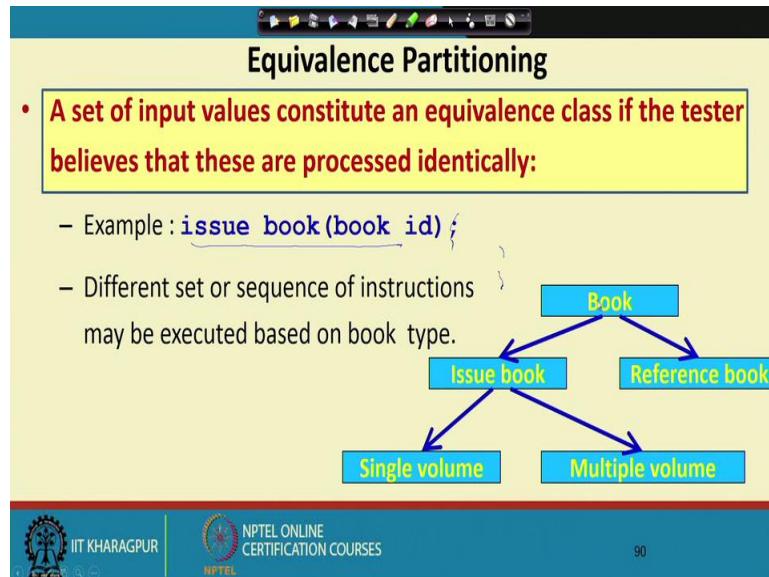
Below the list is a horizontal number line with three points labeled: "Invalid" at the left end, "1" at the center, and "5000" at the right end. The word "valid" is written above the number line between 1 and 5000. There are blue dashed arrows pointing from "Invalid" to "1" and from "5000" to "Invalid".

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a small video thumbnail of a professor.

Then we can pick representative values from each, we may choose -5, 500, 6000 or you may choose any other value, but then once we form the test cases we assume that if it

passes value, a valid value like 500, then any other value it will also pass. There is the assumption behind equivalence class testing.

(Refer Slide Time: 20:08)



So, we can say that a set of input values constitute an equivalence class. If the tester believes that these are processed identically by the program. Let's do some more problems to get a clear idea about how to proceed to identify the equivalence classes. Let's assume that, we have a function called issue book for a library software and then it takes the book id and then issues that book.

But, then we can imagine that the code that is there in the issue book, depending on the type of the book, it will execute different set of statements. For example, if the book id is a reserved book, then it will execute some set of statements and it may output that, the book is reserved and cannot be issued out.

Similarly, if the book is a reference book, then it will also execute slightly different set of instructions and then it will say that it is a reserved book and cannot be issued. If it is a book that can be issued, it will issue it. If it is a multi-volume book i.e., a set of books, then it will say that these many number of books are issued out. So, based on the input, we can identify the equivalence classes.

The first level, we can have the issue book, first is the valid and invalid data something, which is not a book id may be a floating point number or something. So, valid and

invalid and then for the valid book, we can have an issuable book and a reference book and for the issuable book we can have single volume and multiple volume.

(Refer Slide Time: 22:52)

Equivalence Partitioning: Example 1

- Example: **Image Fetch-image(URL)**
 - **Equivalence Definition 1:** Partition based on URL protocol (“http”, “https”, “ftp”, “file”, etc.)
 - **Equivalence Definition 2:** Partition based on type of file being retrieved (HTML, GIF, JPEG, Plain Text, etc.)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

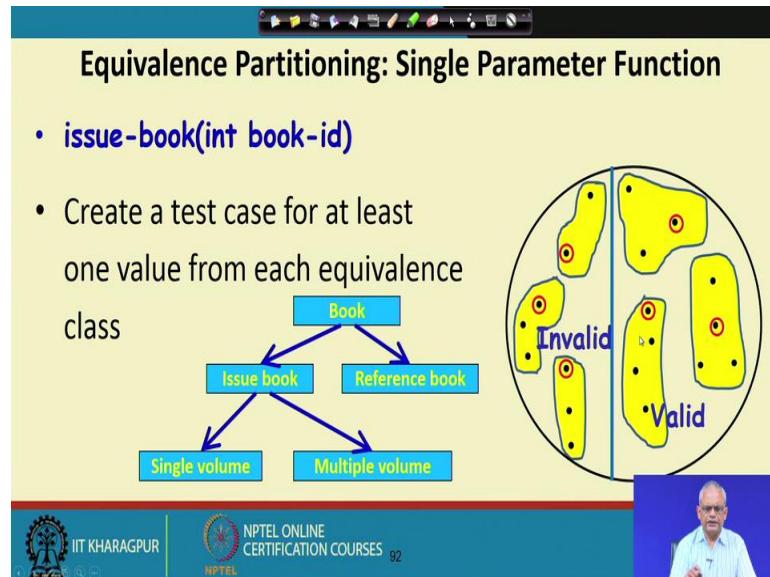
Let's do some more examples, let's say we have a function, which is Fetch-image. It takes a URL and then fetches the image and returns the image. So, the function, it takes 1 parameter which is the URL and depending on the image present there, it retrieves the image and returns the image. Now, how do we define the equivalence class here? One is that based on the type of URL. For example, it may be using http protocol, https protocol ftp file. So, this is the protocol used in the URL based on that we can partition into a set of equivalence classes.

So, that I have written the set of data valid data and then based on the URL specified, we can have some equivalence classes like http, ftp, file, https etc, but again the specific URL may point to different types of images. So, depending on the image type, we again can define an equivalence class, depending on whether it is a HTML image, GIF image, JPEG, Plain Text etc.

So, we have another set of equivalence class here. Just want to make this point that based on the input, it is not that there is a single hierarchy, we can have multiple hierarchy based on different characteristics of the input the URL, the protocol of the URL, the file type and so on. So, even though the concept is straight forward, but sometimes

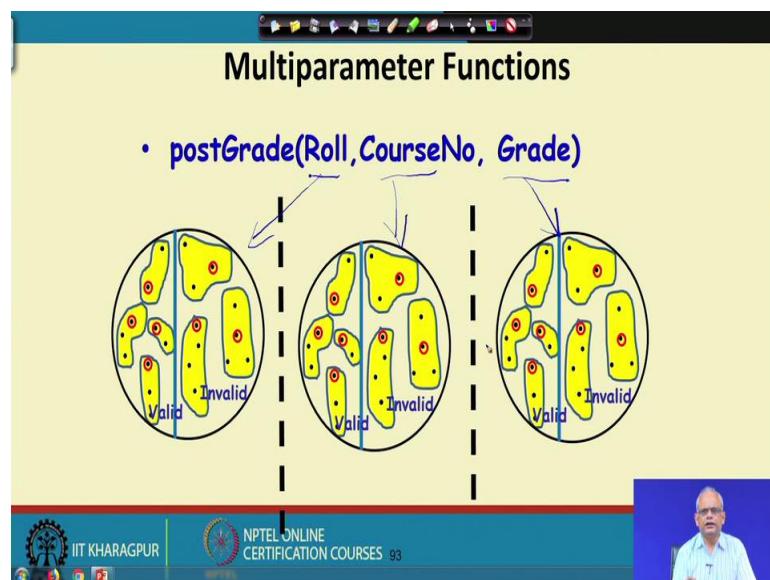
identifying the equivalence classes can be requiring a some thought on the input data value.

(Refer Slide Time: 25:31)



Once we identify the equivalence classes, we just select one data point corresponding to the valid equivalence classes. So that is single volume, multiple volume and reference. Similarly, we select one data point for the invalid equivalence classes.

(Refer Slide Time: 25:59)



So, far we just looked at how to identify the equivalence classes when a function takes one parameter, but we can have a function which takes multiple parameters. Now, how

do we identify the equivalence classes here? Of course, it is rather straight forward of the discussion that we had so far. We will examine each input parameter and based on the input parameter, we can identify the equivalence classes for the different input parameters. But, now how do we define the test cases? We can select points from here for corresponding to this parameter, points corresponding to this parameter and so on.

But, the test for the test case one way is that we take one point from here and form the test case. Another way can be that we take one point here and corresponding to that we define test cases by considering each of the point here and each of the point here. So, that will give us large number of test cases. So, let me just repeat that once we identify the equivalence classes for each of the 3 parameters, one way is that we define the test cases such that every equivalence class is represented in our test case. The other way is that we form a (Refer Time: 27:53) combination of the input values. Now, we are almost at the end of this lecture, we will continue from this point in the next lecture.

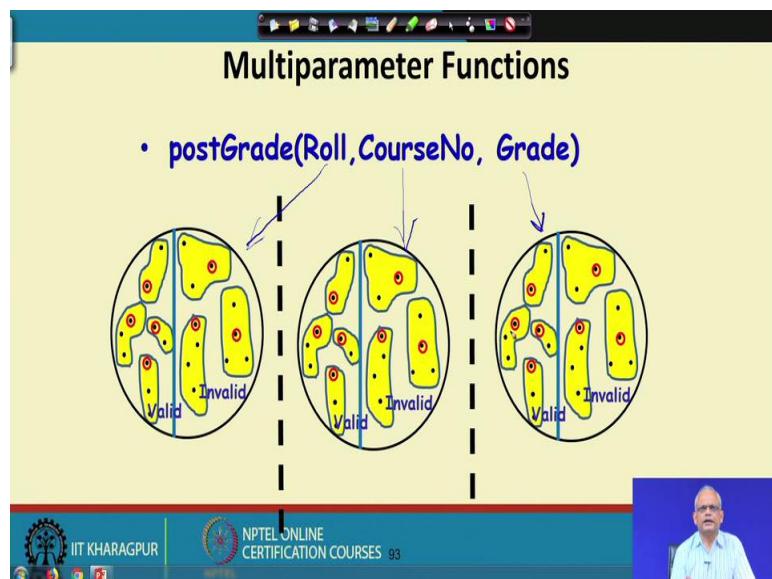
Thank you.

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

Lecture - 49
Equivalence Class Testing-II

Welcome to this lecture. In the last lecture we were discussing about the Equivalence Class Test cases. We had identified when a function or a unit takes a single parameter, we can examine the input value and then identify the set of equivalence classes, one is the valid set of value, another is the invalid set of value and then for each of the valid and invalid we can identify different sets of equivalence classes, but then we were discussing about multi parameter functions.

(Refer Slide Time: 01:02)

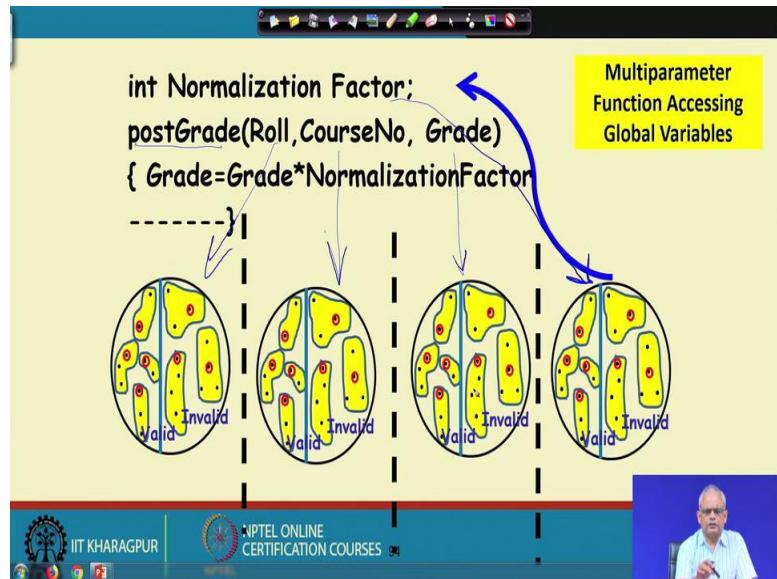


As it is quite common that a function takes more than one parameter. In that case how do we identify the equivalence class? One way is that we can identify equivalence classes corresponding to each of the parameters here. Then for defining the test case, one way is that we define the test cases such that every point here in this set of equivalence classes and a point representative from this set of equivalence classes and a representative from this set of equivalence classes is considered.

The other way is that we have an exhaustive combination of the points from these 3 equivalence classes. So, these are 2 different strategies to define the equivalence class

test cases. We will just look at it further we will develop this idea further in our discussion today.

(Refer Slide Time: 02:14)



But before that, we must be aware that not only is that a function like postGrade can take the roll number, course number and grade. So, the teacher assigns a grade for a specific course to a student identified by some roll number. There are 3 parameters for this function postGrade, but also there can be some global variables for example, NormalizationFactor.

Let's say we have a policy of normalizing the grades and we multiply the grade with a NormalizationFactor. Now we can think of as an additional factor which influences the outcome here. Therefore, we can form equivalence class not only corresponding to the 3 input parameters, but also corresponding to the global variable. Just extension of our discussion of multi parameter function is that a global variable accessed by a function is similar to a parameter to the function. Therefore, we must identify the equivalence classes corresponding to that parameter and then we combine the points from these different equivalence class.

(Refer Slide Time: 04:07)

Input Parameter	Valid Equivalence Classes	Invalid Equivalence Classes
An integer N such that: -99 <= N <= 99	?	?
Phone Number Area code: [11,..., 999] Suffix: Any 6 digits	?	?

f(int, phone)



Let's do some more practice. Let's say we have a function which takes 2 parameters one is an integer between -99 to +99. Let's say some function f takes 2 parameters; one is an integer parameter and it defines a range between -99 and +99. The first parameter takes any value the valid set of values are the -99 to +99 and the second parameter is a phone number.

The phone number is defined by an area code which is between 11 to 9999999 and then there is a actual phone number, the area code and the phone number, and the phone number is let's say 6 digit. So, the second parameter is a phone number which is again an integer, but the first 2 or 3 digits they correspond to the area code and the suffix 6 digits correspond to the phone number. Now how do we identify the equivalence classes for this? Of course, the first thing is that we identify the valid set of equivalence classes and invalid set of equivalence classes.

For the first parameter the valid set of equivalence is any number between -99 to +99 and then there are 2 invalid equivalence classes, one is less than -99 and another is greater than +99, but what about the second parameter? The second parameter again we have a valid equivalence class where we have a 6- digit integer suffix and the first 2, first the prefix that is area code is between 11 and 999 and the invalid set of equivalence may arise because the area code is wrong or may be the suffix is wrong.

(Refer Slide Time: 06:47)

Equivalence Partitioning: Example		
Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, 99] 	< -99 > 99 Malformed numbers {12, 1-2-3, ...} Non-numeric strings {junk, 1E2, \$13} Empty value
Phone Number Prefix: [11, 999] Suffix: Any 6 digits	[11,999][000000, 999999]	Invalid format 5555555, Area code < 11 or > 999 Area code with non-numeric characters

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

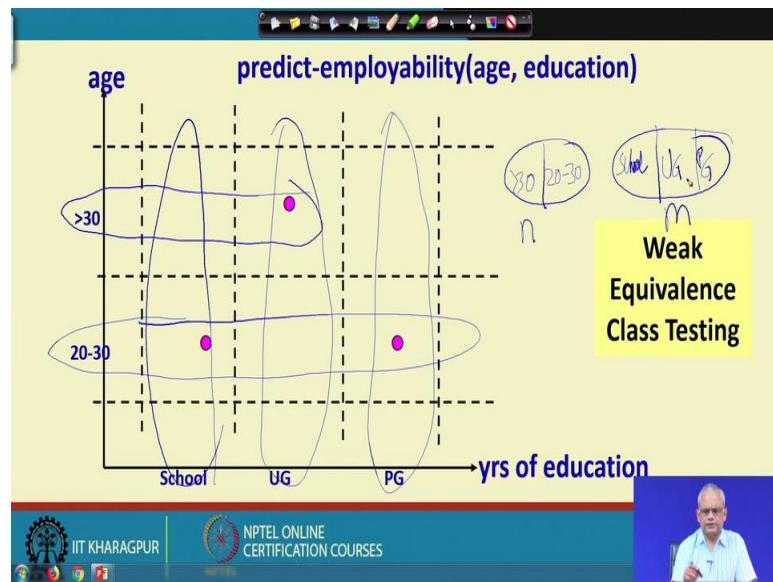


If we write down our idea we will have this, one is that we have for the first parameter is a range and then we have one equivalence class which is one is the valid equivalence class which is any value within the range, another is a invalid equivalence class which is less than minus 99, one more invalid class is greater than 99 and then we can have other equivalence classes as well for example, Malformed numbers some are characters and so on. Non-numeric strings, empty value etcetera these are the different invalid equivalence classes.

Similarly, for the second parameter we have a valid equivalence class where the area code is between 11 to 999 and the phone number is between 000000 to 999999. Now the invalid equivalence classes is the invalid format. We don't have the area code and the phone number or we might have the area code which is less than 11 or greater than 999 99. These are 2 different equivalence classes, invalid equivalence classes; area code with nonnumeric characters, phone number which is 7 digits, phone number which is character and so on.

So, we will have many invalid equivalence classes and one valid equivalence class for each of this.

(Refer Slide Time: 08:40)



Once we identify the equivalence classes for the 2 parameters, we combine representative values from these two equivalence classes corresponding to the 2 parameter to define our test cases at the simplest.

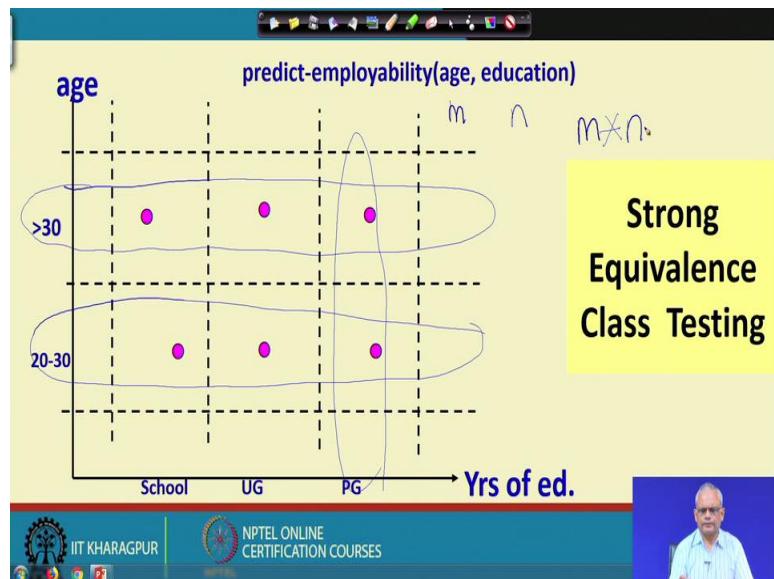
We will discuss the weak equivalence class testing where let's say we have 2 parameters age and education. Let's say for the age, we identify two equivalence classes, one is age greater than 30 and the second is the age between 20 to 30; these are the two equivalence classes. Similarly, for the second parameter education, let's say we identify 3 equivalence classes School, UG and PG. Depending on the years of education, we identify whether it is a School education, UG education or a PG education.

Now how do we combine these 2 values from these 2 parameters to form the test cases? One way which is called as the weak equivalence class testing is that we see every representative from every equivalence class is represented in the test case. So, we see that every equivalence class from the first parameter is covered. Similarly, every representative of the equivalence classes of the second parameter are also covered.

So, if we have n equivalence classes for the first parameter and m equivalence classes of the second parameter then how many test cases we need to be able to do a weak equivalence class testing for this function? Can think over. Let me just tell you that we will just take maximum of n and m . That will form the number of test cases required to

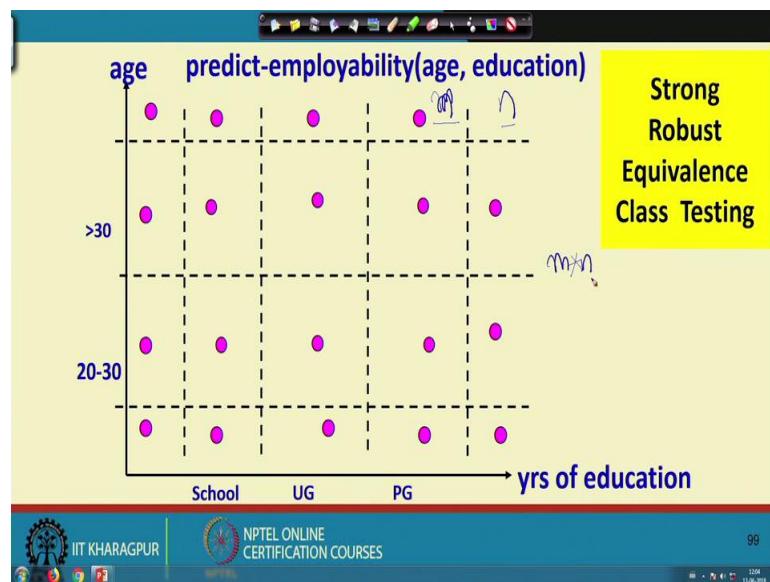
carry out weak equivalence class testing, which will have every representative from every equivalence class to be part of a test case.

(Refer Slide Time: 11:55)



The other way we can combine the representative values from different equivalence classes is called as the strong equivalence class testing. Here every value of one equivalence class of one parameter is combined with every equivalence class representative every equivalence class of the other parameter. So, for the first parameter here of age, we have a one representative from School, UG, and PG. Similarly, for the second equivalence class of the age we again combine with the 3 equivalence classes of the second parameter which is the year of education. In other word, for every value of the second parameter we combine with every other value of the first parameter and this is called a strong equivalence class testing. So, the question here is that, how many test cases will be needed if we have 2 parameters and let's say we have m equivalence classes for the first parameter and n equivalence classes for the second parameter? Since for every equivalence class here we must combine with all equivalence class of the second parameter we need $m * n$ test cases.

(Refer Slide Time: 13:47)



Based on the equivalence classes, we can also combine them in another way which is called as strong robust equivalence class testing. If you look at the 2 types of testing, we did not consider the invalid set of equivalence classes, we just considered the valid equivalence classes.

But if we consider the invalid set of equivalence classes also, then we call it as a robust testing. The idea is simple; just a small extension of the strong equivalence class testing. Here we consider the invalid values as well and this forms our set of test cases. If we use the valid plus invalid, we have m equivalence classes for first parameter, n for the second parameter. So, if these are the some of the valid equivalence classes and invalid equivalence classes, then the number of test cases needed for equivalence class testing will be $m*n$.

(Refer Slide Time: 15:04)

• Design Equivalence class test cases: **compute-interest(days)**

• A bank pays different rates of interest on a deposit depending on the deposit period.

- 3% for deposit up to 15 days
- 4% for deposit over 15 days and up to 180 days
- 6% for deposit over 180 days upto 1 year
- 7% for deposit over 1 year but less than 3 years
- 8% for deposit 3 years and above

Quiz 1

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES 100

Let's see let's try to solve some examples. Let's say for a bank application we have a function; the name of the function is compute interest and the parameter is days which is an integer and the function description is that it should compute 3 percent interest for deposits less than 15 days, 4 percent for deposit between 15 days to 180 days, 6 percent for 180 days up to 1 year, 7 percent for between 1 to 3 year and 8 percent for 3 years and above.

So, what will be the equivalence classes here, this is a rather straight forward example we have just one parameter and we examined the output. Each of these correspond to a scenario and then we consider one equivalence class corresponding to deposit between 0 to 15 days, 15 to 180 days etc. and the invalid set of inputs which is an invalid day negative day and so on.

(Refer Slide Time: 16:41)

The slide contains a list of test cases for a function `compute-interest(principal, days)`. The list includes:

- For deposits of less than or equal to Rs. 1 Lakh, rate of interest:
 - 6% for deposit upto 1 year
 - 7% for deposit over 1 year but less than 3 years
 - 8% for deposit 3 years and above
- For deposits of more than Rs. 1 Lakh, rate of interest:
 - 7% for deposit upto 1 year
 - 8% for deposit over 1 year but less than 3 years
 - 9% for deposit 3 years and above

At the bottom left, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. At the bottom right, there is a video feed of a speaker.

Now, let's try slightly more complex problem where we have the compute interest, but it takes 2 parameters principal amount deposited and the days for which the deposit is made. The function description says that, if the deposit amount i.e., the principal is more than 1 lakh, then the rate of interest is 7 percent, 8 percent, 9 percent, but if the deposit is less than 1 lakh then it is 6 percent, 7 percent, 8 percent. So, we see here that for the 2 parameters, we have 2 equivalence classes; one is more than 1 lakh, less than 1 lakh and the days is 1 year, 3 year and more than 3 years. So, we can have either a weak equivalence class testing or a strong equivalence class testing as required.

(Refer Slide Time: 17:56)

The slide contains a list of test cases for a program that checks if the second string is a substring of the first string. The list includes:

- Design equivalence class test cases.
 - Consider a program that takes 2 strings of maximum length 20 and 5 characters respectively
 - Checks if the second is a substring of the first
 - `substr(s1,s2);`

At the bottom left, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. At the bottom right, there is a video feed of a speaker. A hand-drawn diagram shows two ovals labeled `s1` and `s2`, with arrows pointing from them to a larger oval labeled `str`, with the word "Data" written above the arrows.

This is another problem, let's say we have a function called as substring takes 2 parameters s1 and s2. s1 is a string whose length is up to 20 characters and the s2 is a string whose length is less than 5 characters and the function checks whether the second string is a substring of the first string. Again we can form 2 equivalence class hierarchy; one is based on the string which is less than invalid set, more than 20 or it can be a control character and so on and the second one is valid set of strings for s1 and invalid set of s2 and then we would have another equivalence class hierarchy which is whether it is a substring of this or not a substring of this. If we represent here the input data for s1, we can have invalid and a valid set of equivalence class. Similarly, for s2 we will have valid and invalid. by looking at the s1 and s2 for valid s1 and s2, we can have another hierarchy whether it is a substring or not a substring.

(Refer Slide Time: 20:06)

Special Value Testing

- What are special values?
 - The tester has reasons to believe that execution with certain values may expose bugs:

-General risk: Example-- Boundary value testing

-Special risk: Example-- Leap year not considered

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 104

Now let's start discussing another type of black box testing called as special value testing. In the equivalence class testing we divided the input data domain into equivalence classes such that every point belonging to one equivalence class resulted in similar behavior, but here there are some values which are suspect.

The tester has reasons to believe that execution of the program with certain specific values may expose the bugs. There are 2 types of risky values, that tester can identify one is called as the general risk. The general risk this is identified by examining the

equivalence classes and the boundary values of the equivalence classes in general are suspect.

But there are special values which are of risk, which may not be boundary for example, the tester might think that for this function which takes a year and produces the number of days in the year is the leap year considered by the programmer or not. So, this tester by looking at the functionality, he guesses some values which might be prone to error and those are called as the special risk values. In addition to equivalence class testing, we need to do special value testing and one type of special value testing is the general risk an example is the boundary value testing and also the special risk data values should be used to define test cases.

(Refer Slide Time: 22:16)

• Some typical programming errors occur:

- At boundaries of equivalence classes
- Might be purely due to psychological factors.**

• Programmers often commit mistakes in the:

- Special processing at the boundaries of equivalence classes.**

Boundary Value Analysis

1 100

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 105

Now, let's look at the boundary value analysis. This is a general risk whenever we identify equivalence classes. The programmers are very prone to commit errors at the boundary of the equivalence classes. Here there is one valid set of equivalence class, valid set of values which constitute an equivalence class, one invalid set of value constituting an invalid equivalence class and another invalid equivalence class.

The programmers are likely to commit errors at the boundary. Here for example, whether one is included as in the valid or invalid. They might not have taken care of that sufficiently well. Therefore, mistakes may occur here at the boundaries. Hence, we must

include the data values at the boundary as test cases and those are called as boundary value test cases.

(Refer Slide Time: 23:38)

Boundary Value Analysis

- Programmers may improperly use $<$ instead of \leq
- Boundary value analysis:

The main reason here is that programmer sometimes get confused and instead of $<$ symbol, they might use a \leq symbol and so on and that is the reason we need to select test cases at the boundaries of different equivalence classes. If boundaries exist we had seen that if it is a set i.e., input domain is a set of values, then there is no boundary, but if it is a range of values, then there are boundaries between different equivalence classes.

(Refer Slide Time: 24:19)

Boundary Value Analysis: Guidelines

- If an input is a range, bounded by values a and b:
 - Test cases should be designed with value a and b, just above and below a and b.
- Example 1:** Integer D with input range $[-3, 10]$,
 - test values: $-3, 10, 11, -2, 0$
- Example 2:** Input in the range: $[3, 102]$
 - test values: $3, 102, -1, 200, 5$

Now, let's see if we have an input is a range, it is a range and the boundaries are a and b, then we need to select not only a, but we need to select one that is beyond a just to check whether the programmer has made a mistake and even the one which is should be a invalid or a different equivalence class is also considered as part of this equivalence class.

Similarly, we might take a value just inside and similarly for b, one on b another slightly inside and another just outside. So, if the range is defined as any value between -3 to 10 we can take -3, we can take 10. These are the two boundaries; one slightly away outside the boundary which is -2, 11 and one which is just inside the boundary this actually can be the equivalence class test.

So, we need to define 5 values for a range similarly the input is a range it is a 3 to 102 we will define one at boundary 3, one is 102 included the boundary in the test case, one beyond the boundary on the lower side which is -1, another beyond the boundary on the higher side which is 200 and one in between which is 5.

(Refer Slide Time: 26:21)

The slide title is "Boundary Value Testing Example". The first bullet point states: "Process employment applications based on a person's age." Below this is a table:

0-16	Do not hire
16-18	May hire on part time basis
18-55	May hire full time
55-99	Do not hire

The second bullet point says: "Notice the problem at the boundaries." followed by a minus sign and the text: "Age "16" is included in two different equivalence classes (as are 18 and 55)."

The footer of the slide includes the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES 108". There is also a small video window showing a man speaking.

Let's see for some specific examples, how do we define the boundary value test cases. Let's say for some problem where the function is based on the age, it recommends whether can be hired or not hired checks if the age is between 0 to 16 it displays do not hire, if it is between 16 to 18 displays can hire on part time basis, if it is between 18 to 55 displays may hire full time and between 55 to 99 do not hire.

So, there are 4 equivalence classes here corresponding to these 4 different scenarios and the boundaries are at 16, 18 and 55, but then we must realize that this description of the function itself is a bit problematic because 16 is considered as do not hire or it may hire on part time basis. Similarly, if it is exactly 18, what should be the output, similarly 55.

So, even while describing the problem we can commit mistake at the boundary as exemplified here that the boundary values we have not taken care sufficiently. It is the specification of the problem itself is a defective and not clearly identified the boundary should belong to which equivalence class. Of course, the same thing will get reflected in the program and therefore, boundary value testing is very important.

We will stop here, and we will continue in the next class.

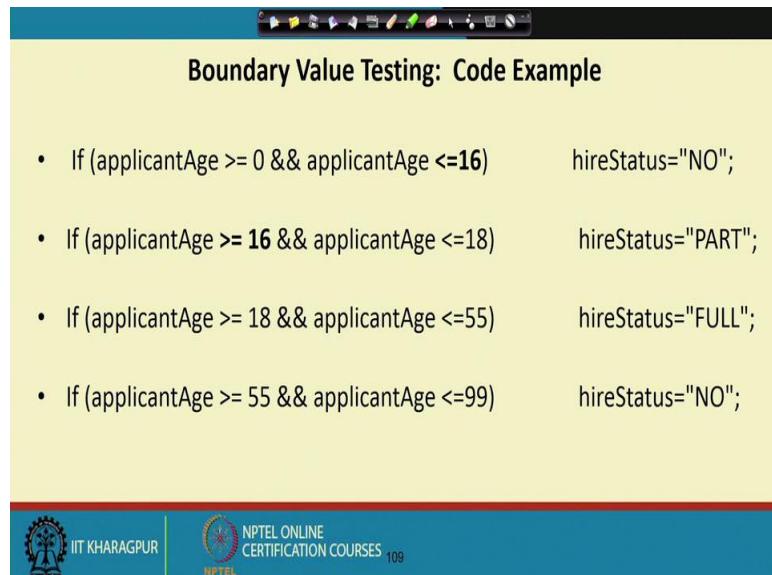
Thank you.

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

Lecture - 50
Special Value Testing

Welcome to this lecture. In the last lecture, we were discussing about the black box testing technique known as the boundary value testing. We had said that, this is a general risk; the boundaries are at general risk of errors and therefore we must define test cases based on the boundary values. Let's proceed from what we were discussing last time.

(Refer Slide Time: 00:48)



Boundary Value Testing: Code Example

- If (applicantAge >= 0 && applicantAge <=16) hireStatus="NO";
- If (applicantAge >= 16 && applicantAge <=18) hireStatus="PART";
- If (applicantAge >= 18 && applicantAge <=55) hireStatus="FULL";
- If (applicantAge >= 55 && applicantAge <=99) hireStatus="NO";

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 109

Let's say, we have a code where depending on the applicant age, we make the hire status; it is either no, do not hire under age or hire status is part time full time and over age, hire status is no. We can identify the boundaries here which is between these two equivalence classes, 16 is the boundary and if we select the boundary likely to determine the bug here that 16 is included in both the equivalence classes, similarly 18, 55 and so on.

(Refer Slide Time: 01:42)

- Corrected boundaries:

0–15	Don't hire
16–17	Can hire on a part-time basis only
18–54	Can hire as full-time employees
55–99	Don't hire

- What about ages -3 and 101?
- The requirements do not specify how these values should be treated.

And then, we can correct our code where we have the boundaries clearly specified here 0 to 15, do not hire 16 to 17, 18 to 54 and so on. Then, we can take care of values that are invalid and we need to take care of the boundary between the invalid and the valid set of equivalence classes.

(Refer Slide Time: 02:16)

```
Boundary Value Testing Example (cont)

• The code to implement the corrected rules is:
  If (applicantAge >= 0 && applicantAge <=15)           hireStatus="NO";
  If (applicantAge >= 16 && applicantAge <=17)          hireStatus="PART";
  If (applicantAge >= 18 && applicantAge <=54)          hireStatus="FULL";
  If (applicantAge >= 55 && applicantAge <=99)          hireStatus="NO";

• Special values on or near the boundaries in this example are {-1, 0, 1}, {14, 15, 16}, {17, 18, 19}, {54, 55, 56}, and {98, 99, 100}.
```

So, based on the boundary values, we have the corrected code and then what if for this code, we have let's say -1 which is not addressed here in the code and therefore, it will generate a failure. Depending on the 4 equivalence classes here, we can define the

boundary values here; for the first one -1, 0, 1, 14, 15 and 16; for the second one, 17, 18, 19; 17 at the boundary of this. So, there are basically if we look at here, we have one boundary that is less than 0, another boundary here at 15 and 16 and another here at 17 and another here at 54 and so on. We must include the boundaries here for each boundary we take a test case which is just on the boundary, one on the higher side and lower side and so on. So, these are each of this either test case.

(Refer Slide Time: 03:56)

The slide is titled "Boundary Value Analysis". It contains a bullet point: "Create test cases to test boundaries of equivalence classes". Below the text is a diagram of a circle divided into several regions by blue lines, with red circles indicating specific points or boundary values. At the bottom left, there is a logo for IIT Kharagpur and another for NPTEL. On the right side, there is a video player interface showing a person speaking.

Once we identify the boundary values, we need to define the test cases. These are the valid set of equivalence classes, invalid set of equivalence classes. And we already had selected one representative value for the equivalence class testing. Now, we will select values in the boundary for the boundary value testing.

(Refer Slide Time: 04:21)

Example 1

- For a function that computes the square root of an integer in the range of 1 and 5000:
 - Test cases must include the values:

$\{0, 1, 2, 4999, 5000, 5001\}$.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 113

A video thumbnail of a man speaking.

Let's look at some example problems, let's take the same problem that we are considering earlier for equivalence class testing that it takes one parameter only; the valid values are between 1 and 5000. If we draw this, we represent the data then we have one valid class and two invalid classes. And there are two boundaries here; one is between the valid class and the invalid on the lower side and another is between the valid class and the invalid on the upper side. We include the boundary 1 and one element, one test case which is less than 1 which is 0, 1, one element which is valid. So, it can be 2. Similarly, 5001, 5000 and let us say 4999. So, these are the set of test cases for one parameter which is a range of values between 1 and 5000.

(Refer Slide Time: 05:46)

• Consider a program that reads the “age” of employees and computes the average age.

Example 2

$\text{input (ages)} \rightarrow \boxed{\text{Program}} \rightarrow \text{output: average age}$

Assume valid age is 1 to 150

• How would you test this?

- How many test cases would you generate?
- What type of test data would you input to test this program?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 114

A video thumbnail of a man speaking.

Now, let's look at age as a parameter and the valid age range is 1 to 150. Here, we have three equivalence classes; one is a valid equivalence class which is between 1 and 150 and there are two equivalence classes. So, for boundary value testing, how many test cases do we need? Of course, need to include the boundary, the boundary value, one below the boundary value, above the boundary value and of course, we can combine these two, we can select a value anywhere here. So, there will be at least 5 test cases, at least 5 test cases for boundary value testing.

(Refer Slide Time: 07:02)

Boundaries of the inputs

The “basic” boundary value testing would include 5 test cases:

predict-longevity(age)

1. - at minimum boundary
2. - immediately above minimum
3. - between minimum and maximum (nominal)
4. - immediately below maximum
5. - at maximum boundary

$1 \leq \text{age} \leq 150$

1 ← → age → 150

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** 115

A small video player window in the bottom right corner shows a person speaking.

So, the 5 test cases: at the minimum boundary, immediately above the minimum, between the minimum and maximum, immediately below the maximum and at the maximum boundary.

(Refer Slide Time: 07:21)

The slide has a yellow header bar with the title "Test Cases for the Example". Below it is a list of test cases:

- How many test cases for the example ?
 - answer : 5
- Test input values? :
 - 1 at the minimum
 - 2 at one above minimum
 - 45 at middle
 - 149 at one below maximum
 - 150 at maximum

On the right side of the slide, there is a blue video frame showing a professor speaking. At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, and the text "NPTEL ONLINE CERTIFICATION COURSES 116".

Just an example to predict longevity and age is the parameter and age is between 100 and 150 and once we identify the 5 boundary values, you can represent them 1, 2, 45, 149 and 150 and this form our boundary value test cases.

(Refer Slide Time: 07:54)

The slide has a yellow header bar with the title "Multiple Parameters: Independent distinct Data". Below it is a list of points:

- Suppose there are 2 "distinct" inputs that are assumed to be independent of each other.
 - Input field 1: years of education (say 1 to 23)
 - Input field 2: age (1 to 150)
- If they are independent of each other, then we can start with $5 + 5 = 10$ sets.

Below the list, there are two tables showing boundary values for each parameter:

coverage of input data: yrs of ed	coverage of input data: age
$n=1$; age = whatever(37) $n=2$; age = whatever $n=12$; age = whatever $n=22$; age = whatever $n=23$; age = whatever	$m+n-1$ $n=12$; age = 1 $n=12$; age = 2 $n=12$; age = 37 $n=12$; age = 149 $n=12$; age = 150

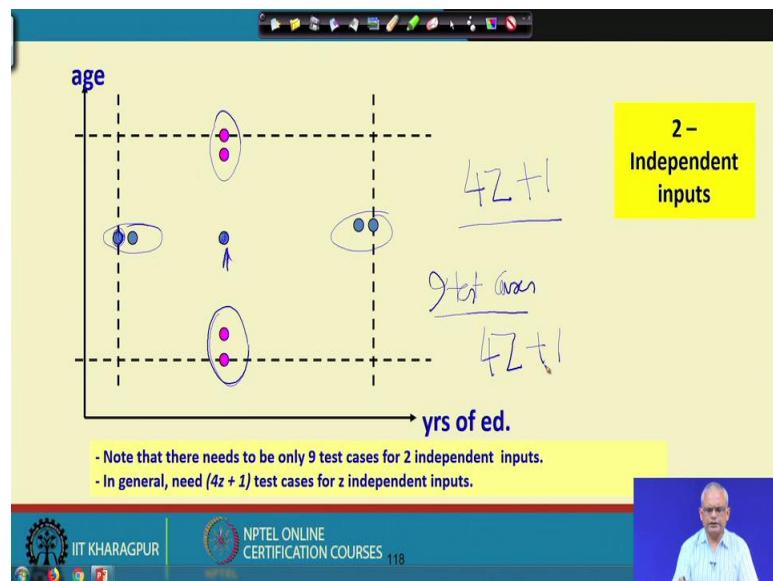
On the right side of the slide, there is a blue video frame showing a professor speaking. At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, and the text "NPTEL ONLINE CERTIFICATION COURSES 117".

But, what about multiple parameters; for each of the parameter, we might identify different boundaries and let's say for parameter years of education and let's say, age we identify different boundaries. And let's say the boundary value for the years of education is 1, 2, 12, 22, 23; these are the boundary values for years of education. Similarly, for the

age, we have 1, 2, 37, 149 and 150. These are boundary values. Now, how do we define the test cases by combining these two boundaries? One way is that we just represent each of these in the test case and take any one of the value from the other equivalence class for the other parameter. And similarly, we define test cases corresponding to each of the boundaries here and some value of the other boundaries; pick one boundary from here, the first parameter and combine with all other parameter boundaries here.

And similarly, for every value on the boundary and the first parameter, we take any value from the second parameter like 37 or something. So, what will be the number of test cases that will be required? If there are let's say m here, m boundary values for the first parameter and n boundary values for the second parameter, we will have $m + n - 1$ because 1 is common here. Therefore, we will have $m + n - 1$.

(Refer Slide Time: 10:31)



And we represent the same thing here. This is for 2 input parameters and we define the boundaries here. Let's say for the first parameter, we define there are 5 boundary values and for the second parameter also, there are 5 boundary values and this is common here. For the first parameter, these two are the boundary and this is a point just inside and these are the two points on the boundary and this is just inside. Now, if we combine these two parameters, we get 9 test cases. And we can represent it as $4z$ where z is the number of parameters + 1 because for every boundary, we have 4 and one representative value

here. And similarly for the other one, for the boundary and one representative value, and therefore, we have $4z + 1$ which is the number of test cases required.

(Refer Slide Time: 12:25)

Boundary Value Test

Given $f(x,y)$ with constraints $a \leq x \leq b$ and $c \leq y \leq d$

Boundary Value analysis focuses on the boundary of the input space to identify test cases.

Defined as input variable value at min, just above min, a nominal value, just above max, and at max.

$4 * 2 + 1 = 9$

Let's say, we have a function $f(x, y)$, the function f takes two parameters, two integer parameters x and y and x takes value between a and b and y takes value between c and d now how many test cases are needed ? If we represent the boundary here, we have a boundary at c , we have boundary at d , we have boundary at a and b and just above we have test cases and one representative here. So, it will be $4 * 2$ parameters + 1 which is equal to 9. There are 2 parameters here and each one, we need 4 and 1 is the common and therefore, we need 9.

(Refer Slide Time: 13:28)

Weak Testing: Single Fault Assumption

- **Premise:** “Failures rarely occur as the result of the simultaneous occurrence of two (or more) faults”
- Under this:
 - Hold the values of all but one variable at their nominal values, and let that one variable assume its extreme values.

NPTEL ONLINE
CERTIFICATION COURSES 120

All the weak testing that we have been discussing with respect to equivalence class, multiple parameter equivalence class and boundary value, the weak testing actually implicitly assumes a single fault that is only one of the parameter can have fault. If that is assumption, then weak testing is alright because for every input, every representative of one parameter, we combine with the representative for the other parameter.

Therefore, if we have two parameters for weak testing if you remember, if there are let's say 3 equivalence classes for the first parameter and 3 equivalence classes for the second parameter, then we may select this and this (respective cell in the above figure). So, we have all the parameters of all the representatives for the parameter1 and parameter2 represented in our test cases. And therefore, if there is a fault in any parameter representative parameter that will be detected, but what if specific combinations of parameters have fault?

Let's say when parameter1 has representative from this and the corresponding one is here, if these two are chosen then there is a fault. Of course, weak testing will not be able to detect that. It will detect only when one of the parameter, any one of the parameter if we select a value there, then it is faulty. In those cases, weak testing is satisfactory; otherwise we need to go for strong testing.

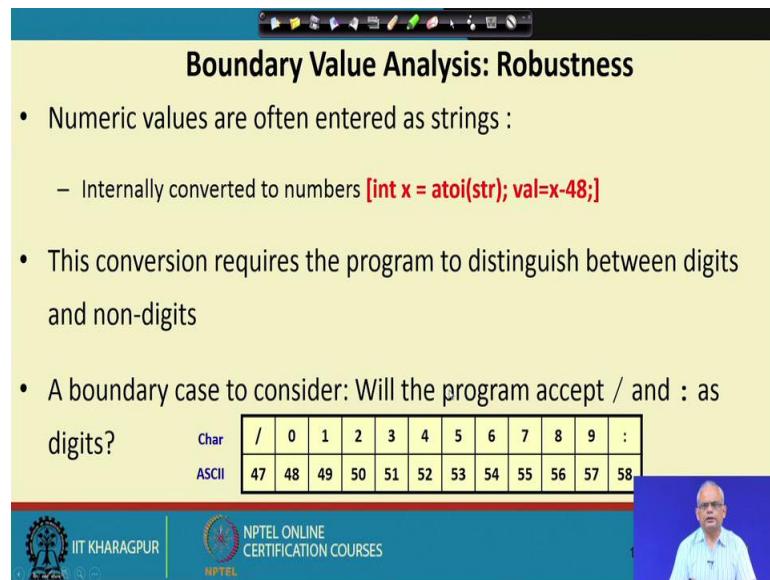
(Refer Slide Time: 15:48)

Boundary Value Analysis: Robustness

- Numeric values are often entered as strings :
 - Internally converted to numbers [int x = atoi(str); val=x-48;]
- This conversion requires the program to distinguish between digits and non-digits
- A boundary case to consider: Will the program accept / and : as digits?

Char	/	0	1	2	3	4	5	6	7	8	9	:
ASCII	47	48	49	50	51	52	53	54	55	56	57	58

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

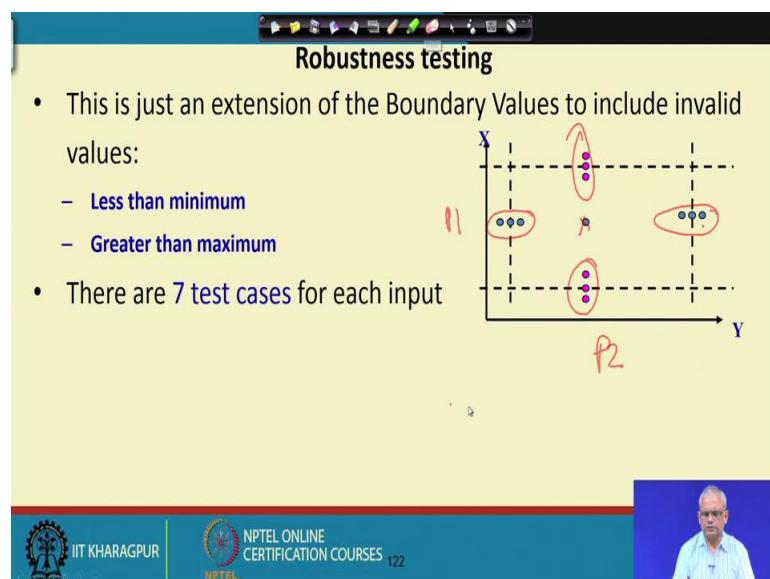


Now, let's look at the robust testing. In robust testing, we need to include a value which is invalid. Let us say, we have a function called as ASCII to integer and it takes a string let's say character it takes and it gives us an integer value for that. If we look at the ASCII table, that 0 to 9 are the characters that are valid which are the valid characters for integers and they have values 48 to 57. But then, if we give an invalid value such as slash or colon, then we can check that it gives us a wrong value. Therefore, we must include the one which is invalid integer representation that we call as the robustness testing.

(Refer Slide Time: 17:03)

Robustness testing

- This is just an extension of the Boundary Values to include invalid values:
 - Less than minimum
 - Greater than maximum
- There are 7 test cases for each input



For robustness testing, we consider the invalid values as well. Therefore, the number of elements at the boundary are three on each of the boundary. For the parameter1, we have 2 boundaries and for both boundaries we need to select three elements each. Similarly, for parameter2, we need to select three elements each, for parameter1 we also select 3 elements each and then this is the general element with which we combine and therefore, we need 7 test cases for each input.

(Refer Slide Time: 18:19)

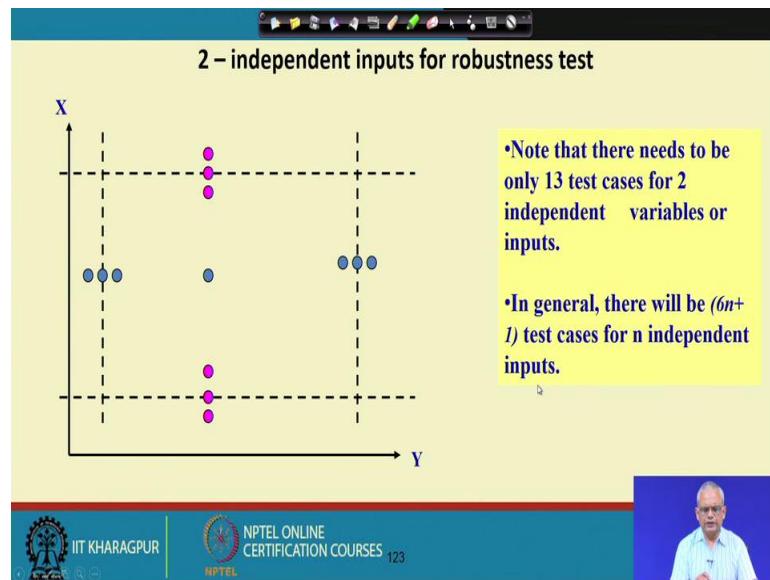
Robustness testing

- This is just an extension of the Boundary Values to include invalid values:
 - Less than minimum
 - Greater than maximum
- There are **7 test cases** for each input
- The testing of robustness is really a test of “error” handling.
 1. Did we anticipate the error situations?
 2. Do we issue informative error messages?
 3. Do we support some kind of recovery from the error?

NPTEL ONLINE CERTIFICATION COURSES 122

if we have n parameters, then we can easily compute how many test cases we need. The main idea behind robustness testing is that do we anticipate the error situations, do we display informative error messages when the value is invalid, is there some kind of recovery from the error? So, those are the main objectives of the robustness testing to check whether the programmer has handled invalid value satisfactorily.

(Refer Slide Time: 18:50)



By considering invalid values as well, we see that for two input, two parameters we need 13 test cases i.e., $6 + 6 + 1$, 13 test cases. So, in general, if we have n parameters, we need $6n + 1$ test cases.

(Refer Slide Time: 19:24)

Some Limitations of Boundary Value Testing

- How to handle a set of values?
- How about set of Boolean variables?
 - True
 - False
- May be radio buttons
- What about a non-numerical variable whose values are text?

Select a size for pizza

Small
Medium
Large

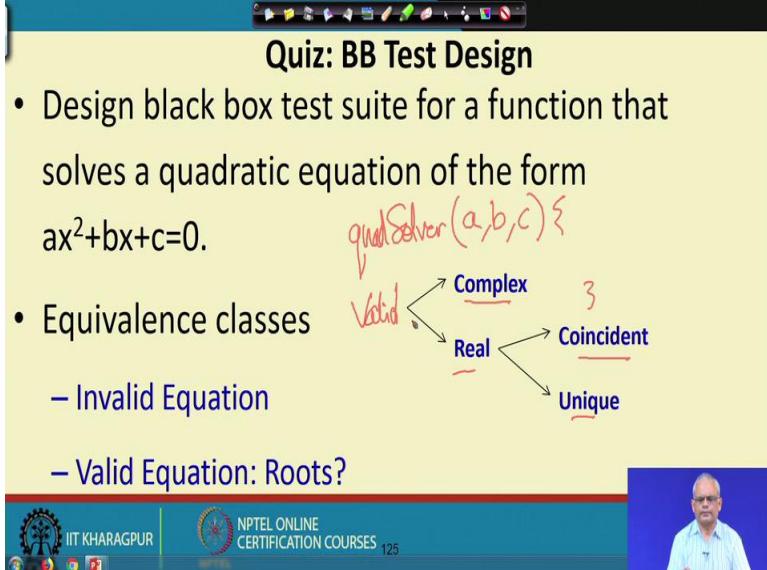
CANCEL OK

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 124

But the main problem with boundary value testing is that if we have a set of values, we do not have boundary there. What about Boolean values, there also we do not have boundaries; what about radio buttons like this, these can be considered like Boolean variables either they are on or off. Here also, we do not have boundary for this cannot be

defined. What about a non-numerical parameter, let's say character string? Again, it's hard to define boundaries for that.

(Refer Slide Time: 20:13)



The slide is titled "Quiz: BB Test Design". It contains two bullet points:

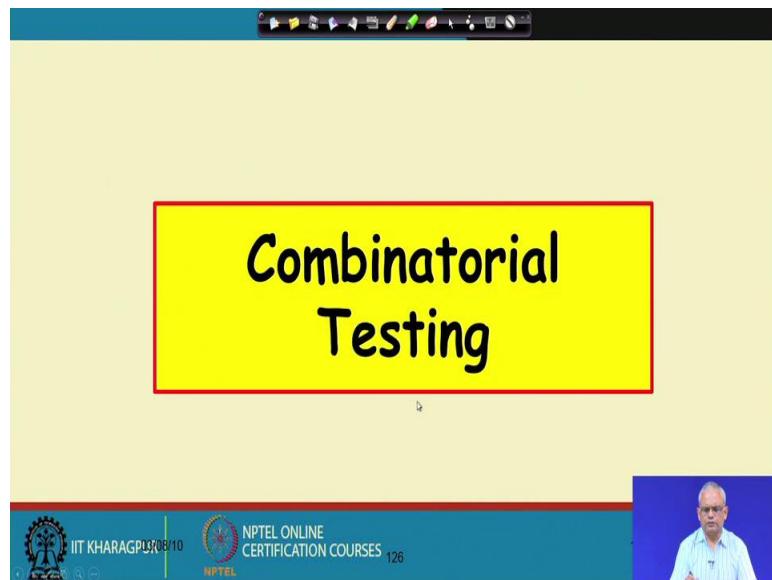
- Design black box test suite for a function that solves a quadratic equation of the form $ax^2+bx+c=0$.
- Equivalence classes
 - Invalid Equation
 - Valid Equation: Roots?

Handwritten notes on the slide:
quadSolver(a,b,c) {
Valid → Complex
Valid → Real → 3
Real → Coincident
Real → Unique

At the bottom left, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. At the bottom right, there is a video player showing a man speaking.

Now, let's conclude our discussion on the equivalence class and boundary value testing, but before that let's do a small quiz. Let's say we have a function which solves the quadratic equation of the form $a x^2 + bx + c$. The name of the function let us say, quadSolver and it takes 3 parameters a , b , c and it returns the roots. What will be the equivalence classes for this? If we examine the scenarios here, then one is that the root may be complex, the root may be real. If it is real, it may be coincident or unique, the valid set of values can be classified, the equivalence classes will be for complex root, real root and the real root can be coincident and unique and we can also have the invalid set of values and so on.

(Refer Slide Time: 21:55)



Now, let's look at Combinatorial Testing. We will see that if the number of parameters to a function is large and it uses many global variables and so on, the number of test cases required for equivalence class testing and boundary values testing can be large. And we will see how to handle that problem and let's look at combinatorial testing.

(Refer Slide Time: 22:28)

A screenshot of a presentation slide titled 'Combinatorial Testing: Motivation'. The slide contains a bulleted list of points explaining why combinatorial testing is necessary due to the complexity of program behavior influenced by multiple factors like input parameters, environment configurations, and state variables. The footer bar includes the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 127', and a small search icon.

The behavior of a program is affected by many factors; how many input parameters are there, how many global variables are there, state variables and so on. And for equivalence partitioning, we identify for every parameter what are the equivalence

classes. And when we combine it, we saw that there is a weak testing, strong testing and robust testing. If we do a robust testing then it may become impractical, if the number of input parameters are many.

(Refer Slide Time: 23:16)

Combinatorial Testing: Motivation

- Many times, the specific action to be performed depends on the value of a set of Boolean variable:
 - Controller applications
 - User interfaces

Font

Latin text font: Body

Font style: Regular

Size: 10

Font color: black

Underline style: (none)

Effects:

- Strikethrough
- Double Strikethrough
- Superscript
- Subscript
- Offset: 0%
- Equalize Character Height

OK Cancel

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 128

But then, in many situations we find that the number of input parameters are actually too many specially in Controller applications and User interfaces. Let's look at this specific user interface just see how many parameters are there, input parameters they are just too many and if we combine the equivalence classes here, then we will get too many test cases.

(Refer Slide Time: 23:52)

Combinatorial Testing

- Several combinatorial testing strategies exist:
 - Decision table-based testing
 - Cause-effect graphing
 - Pair-wise testing

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 129

For combinatorial testing, we will see the Decision table-based testing, Cause-effect graphing and Pair-wise testing which will help us to handle a small number of test cases when we have many parameters or many different types of inputs.

(Refer Slide Time: 24:19)

• Applicable to requirements involving conditional actions.
• This is represented as a decision table:

- Conditions = inputs
- Actions = outputs
- Rules = test cases

• Assume independence of inputs
• Example

- If c1 AND c2 OR c3 then A1

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No		

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 130

First let's look at the Decision table based testing. The decision table based testing is applicable when there are many conditional actions. Here we develop a table and in the table we represent the conditions. The conditions represent inputs and then the actions. The actions represent the output and each of these column under table is called as a rule

and the rules are the test cases. Let's see how to generate the decision table and then once we generate the decision table, we can easily form the test cases. Each of the column will become a test case.

(Refer Slide Time: 25:15)

Decision table-based Testing (DTT)

- Applicable to requirements involving conditional actions.
- This is represented as a decision table:
 - Conditions = inputs
 - Actions = outputs
 - Rules = test cases
- Assume independence of inputs
- Example
 - If $c_1 \text{ AND } c_2 \text{ OR } c_3 \text{ then } A_1$

Handwritten note: c_1, T, F, F, F
 c_2, T, T, F, F
 c_3, T, T, T, F
 A_1, A, A, A, T

	c_1	c_2	c_3	Action
1	T	T	T	A
2	F	T	T	A
3	F	F	T	A
4	F	F	F	T

IT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

Let's take the example of this condition here.

c_1 and c_2 or c_3 , then A_1 we can observe here that c_1, c_2, c_3 are the input and A_1 is the output. We write here the input c_1, c_2, c_3 these we call as the condition and then we have the action. And we combine all possible values of the condition with action and let's say the condition is for c_1 , it is true, c_2 is true and c_3 is true; c_1, c_2, c_3 are all true and then the action is A_1 . We can look at the expression here. Now what if c_1 is false, c_2 is true and c_3 is true. In this case, it is also A_1 now what if c_1 is false c_2 is false and c_3 is true again here this is A_1 , action is A_1 , but what if all are false then there is no action and so on. We can complete this table and this is the decision table where the top part of the table contains the condition outcomes, the lower part this displays the corresponding actions. And each of the row is a rule and they form the test cases. So, as many rows will be there on our table, those many test cases we will need. So, this is a test case, this is a test case c_1 is false, c_2 is true and c_3 is true and so on. We are almost at the end at the end of this lecture. We will stop here and continue our discussion from this point.

Thank you.

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

Lecture – 51
Combinatorial Testing

Welcome to this lecture! In the last lecture, we were discussing the Combinatorial Testing. As the first testing technique the combinatorial testing, we are discussing about the decision table based testing. We had said that if there is a function whose output depends on different types of conditions, then we can represent the values of the conditional inputs and the corresponding actions, in the form of a decision table and every column in the decision table become a test case that we call as a rule.

(Refer Slide Time: 01:04)

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No	No	Yes

131

So, to recall what we were discussing last time. We can represent the outcome of different conditions or the values of different conditions. The corresponding actions let's say 3 actions and sometime depending on the values of the condition, the action may take place or does not take place. This forms our decision table and each of the rule will become a test case. For some of the values of the conditions, we have written don't care. So, in that case irrespective of the condition 2 value the rule 2 will result in this action.

(Refer Slide Time: 02:05)

The slide includes the following text:

- A decision table consists of a number of columns (rules) that comprise all test situations
- Example: the triangle problem
 - C1: a, b, c form a triangle
 - C2: a=b
 - C3: a=c
 - C4: b=c
 - A1: Not a triangle
 - A2: scalene
 - A3: Isosceles
 - A4: equilateral
 - A5: Right angled

determineTriangle(a,b,c)

Sample Decision table

	r1	r2	...		rn
C1	0	1			0
c2	-	1			0
C3	-	1			1
C4	-	1			0
a1	1	0			0
a2	0	0			1
a3	0	0			0
a4	0	1			0
a5	0	0			

NPTEL ONLINE CERTIFICATION COURSES 132

Now, let's do some example problems. First we start with something very simple. Again, we take the problem of a function, which takes 3 parameters a, b, c representing the sides of a triangle. Determine triangle type and we have 3 parameters side1, side2, and side3, these are the 3 parameters for the determine triangle function and then depending on the specific values of the 3 sides it can display not a triangle, scalene, isosceles, equilateral, right angled etc.

Now, the conditions here are that a, b, c form a triangle or C2 is a = b, C3 is a = c, C4 is b = c. The parameters represented as a, b, c that maybe easier to understand in this context. So, the parameters are a, b, c. These are the 3 sides, a = b, a = c, b = c; these are the 3 conditions in the parameters. Now we represent it in a decision table form. If C1 is 0 i.e., it does not form a triangle then we have a1.

Now, if it forms a triangle and then C2, C3, C1 are 1 then it is equilateral i.e., all sides are equal and similarly we just keep on filling this as in the above diagram and for each column, we make a test case. So, that is the essential idea behind the decision table testing, whenever we see that some relation between the input parameter help define the output parameter. We write that in the form of conditions and those conditions, we represent here all possible values of conditions and then we note the corresponding actions and then these form our test cases.

(Refer Slide Time: 05:04)

Test Case ID	a	b	c	Expected output
TC1	4	1	2	Not a Triangle
TC2	2888	2888	2888	Equilateral
TC3	?)	Impossible
TC4				
...				
TC11				


IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES
133

NPTEL



We can write the specific values for the parameter a, b, c for which the columns that define the test case. For example, a \neq b, b \neq c and c \neq a, which is not ok, not a triangle we can take 4, 1, 2 as a specific values here and so on.

(Refer Slide Time: 05:44)

More Complete Decision Table for the Triangle Problem		Conditions									
		C1: a < b+c?									
		F T T T T T T T T T									
		C2: b < a+c?									
		- F T T T T T T T T									
		C3: c < a+b?									
		- - F T T T T T T T									
		C4: a=b?									
		- - - T T T T F F F									
		C5: a=c?									
		- - - T T F F T T F									
		C6: b=c?									
		- - - T F T F T F T									
		Actions									
		A1: Not a Triangle									
		X X X									
		A2: Scalene									
		X									
		A3: Isosceles									
		X									
		A4: Equilateral									
		X X X									
		A5: Impossible									


IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES
134

NPTEL

But, this is a more complete decision table, because earlier we had one of the condition is not a triangle, but what are the conditions, which define not a triangle? The conditions are if none of these are true, then this becomes not a triangle. If either of these becomes true that $a > b + c$ or $b > a + c$ and so on, then, it is not a triangle and we represent this in

C1, C2, C3, which help us to form a more concrete decision table and this will become easier to translate into test cases. If any of this is false C1, C2, C3 the corresponding, which as straight not a triangle so, compared to our previous decision table this is a more complete decision table, because it helps us to easily define the test cases.

(Refer Slide Time: 07:05)

Test Cases for the Triangle Problem	Case ID	a	b	c	Expected Output	
	DT1	4	1	2	Not a Triangle	
	DT2	1	4	2	Not a Triangle	
	DT3	1	2	4	Not a Triangle	
	DT4	5	5	5	Equilateral	
	DT5	?	?	?	Impossible	
	DT6	?	?	?	Impossible	
	DT7	2	2	3	Isosceles	
	DT8	?	?	?	Impossible	
	DT9	2	3	2	Isosceles	
	DT10	3	2	2	Isosceles	
	DT11	3	4	5	Scalene	

Based on the decision table, we can form the test cases, we can give the values, what is the expected output. Each of these is test case and we get 11 test cases.

(Refer Slide Time: 07:22)

Decision Table – Example 2		Printer Troubleshooting								
Conditions	Printer does not print		Y	Y	Y	Y	N	N	N	N
	A red light is flashing		Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized		Y	N	Y	N	Y	N	Y	N
	Check the power cable				X					
	Check the printer-computer cable		X		X					
Actions	Ensure printer software is installed		X	X		X		X		
	Check/replace ink		X	X			X	X		
	Check for paper jam			X		X				

Now, let's take another decision table example. Let's say the function is printer troubleshooting. We give some input to the function; those are the conditions for example, if the printer does not print, red light is flashing and printer unrecognized, based on the logical combinations of these input the output is defined.

For example, if the printer does not print' red light is flashing and printer is unrecognized, then the output will be check the printer computer cable and see your printer software is installed and check or replace the ink, but if the printer does not print and the red light is flashing then, we will display that check and replace ink or check for the paper jam and so on.

For every combination of the input parameter, we write the corresponding actions. This forms the decision table for the printer troubleshooting. Once we develop this decision table, each of the column in the decision table forms a test case.

(Refer Slide Time: 09:00)

Quiz: Develop BB Test Cases

- Policy for charging customers for certain in-flight services:

If the flight is more than half-full and ticket cost is more than Rs. 3000, free meals are served unless it is a domestic flight. The meals are charged on all domestic flights.

more than half full?
Domestic?
Ticket cost > 3000?
Free Meal

IIT KHARAGPUR | **NPTEL ONLINE CERTIFICATION COURSES** 137

A video player window shows a person speaking.

Now, based on our understanding of the decision table, let's try to solve a very simple problem. Let's say certain airline provides in-flight services that is food, free meals are served if the fight is more than half-full and the ticket cost is more than 3000.

If the flight is more than half full and ticket cost is more than 3000, free meals are served unless it is a domestic flight. If it is a domestic flight, then meals will not be served. And the meals are charged on all domestic flight. So, how do we develop the decision table

for this? If we look at the conditions, the conditions are whether more than half full, it is a domestic or international flight and ticket cost greater than 3000; these are the 3 conditions and the action is free meal served or not.

Now, we can form all possible combinations of this; more than half full - yes, domestic - yes, ticket cost 3000 - yes, free meal no. More than half full - yes, domestic - yes, ticket cost > 3000 - no and in that case free meal is no. More than half full - no, domestic - no, ticket > than 3000 - no, and then no free meal is served, but what about more than half filled is yes, domestic is no, ticket cost is greater than 3000, then it becomes yes and so on. So, we can develop the set of test cases for this by looking at the rules.

But, how do we optimize the number of test cases? Let's look at that.

(Refer Slide Time: 11:53)

		POSSIBLE COMBINATIONS							
CONDITIONS	<i>more than half-full</i>	N	N	N	N	Y	Y	Y	Y
	<i>more than Rs.3000 per seat</i>	N	N	Y	Y	N	N	Y	Y
	<i>domestic flight</i>	N	Y	N	Y	N	Y	N	Y
ACTIONS									

Fill all combinations in the table.

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES 138



So, we can identify the 3 conditions more than half full, more than 3000 per seat and the domestic flight and then the action is free meals are charged. First is we form all possible combinations each one has 2 values yes and no. So, 2^3 , we will need 8 of them here.

(Refer Slide Time: 12:31)

		POSSIBLE COMBINATIONS								
CONDITIONS	more than half-full	N	N	N	N	Y	Y	Y	Y	
	more than Rs. 3000 per seat	N	N	Y	Y	N	N	Y	Y	
	domestic flight	N	Y	N	Y	N	Y	N	Y	
	serve meals					Y	Y	Y	Y	
ACTIONS	free							Y		



So, we can represent now the corresponding actions. As long as it is a more than half full, meals are served and if it is an international flight, it is not a domestic flight, then it is a free meal.

(Refer Slide Time: 13:09)

		POSSIBLE COMBINATIONS								
CONDITIONS	more than half-full	N	N	N	N	Y	Y	Y	Y	
	more than Rs. 3000 per seat	N	N	Y	Y	N	N	Y	Y	
	domestic flight	N	Y	N	Y	N	Y	N	Y	
	serve meals					X	X	X	X	
	free							X		
ACTIONS										



But, one thing we can observe here is that if we represent it in the form of a decision table, we will look at the actions, for both these rules the action is same and the only difference between the conditions here is that this is No and this is Yes. So, we can combine these 2, we can combine these 2 and we can write don't care for this as shown

in the figure above. Similarly, for these 2, there the rules are similar, I mean the conditions are similar accepting that these 2 are different rest are similar and also the action is the same. Therefore, we can combine these 2 and write don't care for this. Similarly, you can combine these 2 because they differ only with respect to more than 3000 per seat, but the action is the same they serve meals. Now based on this idea we can combine this and form a decision table.

(Refer Slide Time: 14:32)

		Combinations			
		N	Y	Y	Y
CONDITIONS	<i>more than half-full</i>	N	Y	Y	Y
	<i>more than 3000 per seat</i>	-	N	Y	Y
ACTIONS	<i>domestic flight</i>	-	-	N	Y
	<i>serve meals</i>		X	X	X
				X	

 IIT KHARAGPUR
  NPTEL ONLINE
 CERTIFICATION COURSES 141

Which is more compact, number of rules are less, we have use the don't care here. Therefore, the number of test cases is reduced to 4.

(Refer Slide Time: 14:54)

The slide has a yellow header area containing the title 'Assumptions regarding rules'. The main content area is light yellow with black text. A red bullet point is present. A video player interface is visible at the bottom right.

–Rules need to be complete:

- That is, every combination of decision table values including default combinations are present.

–Rules need to be consistent:

- That is, there is no two different actions for the same combinations of conditions

NPTEL ONLINE CERTIFICATION COURSES 142

IIT KHARAGPUR

NPTEL

NPTEL ONLINE CERTIFICATION COURSES 142

IIT KHARAGPUR

NPTEL

Now, let's see some assumptions made regarding the rules; one is that when you form the decision table all possible combinations of the conditions are represented, that is how we must develop the decision table so that we do not miss out on specific combinations. One thing we must guard against is that for the same combination of conditions, same values of the conditions, we cannot have 2 different columns and have 2 different actions, taking that will be a contradictory thing. So, we should guard against this while developing a decision table.

(Refer Slide Time: 15:53)

The slide has a yellow header area containing the title 'Guidelines and Observations'. The main content area is light yellow with black text. Red bullet points are used for certain items. A video player interface is visible at the bottom right.

Guidelines and Observations

- Decision Table testing is most appropriate for programs for which:
 - There is a lot of decision making
 - Output is a logical relationships among input variables
 - There are calculations involving subsets of input variables
 - There are cause and effect relationships between input and output
 - There is complex computation logic
- Decision tables do not scale up very well

NPTEL ONLINE CERTIFICATION COURSES 143

IIT KHARAGPUR

NPTEL

The problems for which the decision table based testing is applicable is that from the function it appears that there is lot of decision making, if then else kind of thing happens. The output is a logical relation among inputs or there is a calculation involving subset of input variables or there is a cause and effect relation between input and output or the computation logic is complex.

In all these cases we need to develop the decision table, but one thing we must understand that, developing the decision table when the number of conditions is large becomes very cumbersome or can make mistakes. So, for small problems involving few conditions decision table is a very helpful to design the test cases.

(Refer Slide Time: 17:01)

Quiz: Design test Cases

- Customers on a e-commerce site get following discount:
 - A member gets 10% discount for purchases lower than Rs. 2000, else 15% discount
 - Purchase using SBI card fetches 5% discount
 - If the purchase amount after all discounts exceeds Rs. 2000/- then shipping is free.

Member?
Purchase > 2000?
SBI Card?

Now, let's do a small quiz to design the decision table based testing. Let's say the customer on an e-commerce site gets the following discount. A member of the site gets 10 percent discount for purchases lower than 2000, if he is not a member he gets and the purchase is more than 2000 gets 15 percent discount, purchase using the SBI card fetches 5 percent discount. If the purchase amount after all discount exceeds 2000, then shipping is free. How do we design the decision table for this?

The first step is to identify the conditions. The first condition is that, is the customer a member? The second condition is that the purchase amount > 2000 or not? The third condition is that whether SBI card is used or not? the purchase amount after discount? So, these are the 4 conditions based on which we can develop the decision table here.

Member is no and let's say purchase > 2000 is yes, SBI card is yes and the total amount after discount > 2000 is yes, then the total discount will be 5 percent for SBI card and then shipping is free and so on. We can design the decision table and that will help us to generate the test cases.

(Refer Slide Time: 19:55)

The slide has a yellow background with a black header bar at the top containing various icons. The title 'Cause-effect Graphs' is centered in bold black font. Below the title is a bulleted list:

- Overview:
 - Explores combinations of possible inputs
 - Specific combinations of inputs (causes) results in outputs (effects)
 - Represented as nodes of a cause effect graph
 - The graph also includes constraints and a number of intermediate nodes linking causes and effects

At the bottom of the slide, there is a footer bar with three sections: 'IIT KHARAGPUR' with its logo, 'NPTEL ONLINE CERTIFICATION COURSES 145' with its logo, and a video player showing a man speaking.

The cause effect graph is a testing technique, which just systematizes the decision table development. If we understand the problem well, we can easily generate the decision table. Sometimes the cause effect graphs can help us generate the decision table easily. Here we have specific symbols for input and then we will represent that, the symbols and form of output and then we develop the cause effect graph. By looking at the cause effect graph, it becomes easy to generate the decision table.

Let's look at one example and based on that, we can see that cause effect graph is just a technique, which helps us to develop the decision table. Finally, the test cases are generated based on the decision table, but if we are able to generate the decision table without using the cause effect graph then well and good, but if we cannot really design the decision table, then we can use the cause effect graph represent the causes and effects in the form of a graph. From that graph it becomes very easy to generate the decision table.

(Refer Slide Time: 21:32)

• If depositing less than Rs. 1 Lakh, rate of interest:

- 6% for deposit upto 1 year
- 7% for deposit over 1 year but less than 3 yrs
- 8% for deposit 3 years and above

• If depositing more than Rs. 1 Lakh, rate of interest:

- 7% for deposit upto 1 year
- 8% for deposit over 1 year but less than 3 yrs
- 9% for deposit 3 years and above

**Cause-Effect
Graph
Example**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 146

NPTEL

Let's explain this technique using a simple example. Again we will take the case of depositing less than 1 lakh, rate of interest is 6 percent for 1 year, 7 percent for deposit 1 year to 3 year and 8 percent for 3 year and above. If deposit is more than 1 lakh, then 7 percent for up to 1 year, 8 percent for deposit over 1 year, but less than 3 year and 9 percent for deposits above 3 years.

(Refer Slide Time: 22:11)

Cause-Effect Graph Example

Causes	Effects
C1: Deposit<1yr	e1: Rate 6%
C2: 1yr<Deposit<3yrs	e2: Rate 7%
C3: Deposit>3yrs	e3: Rate 8%
C4: Deposit <1 Lakh	e4: Rate 9%
C5: Deposit >=1Lakh	

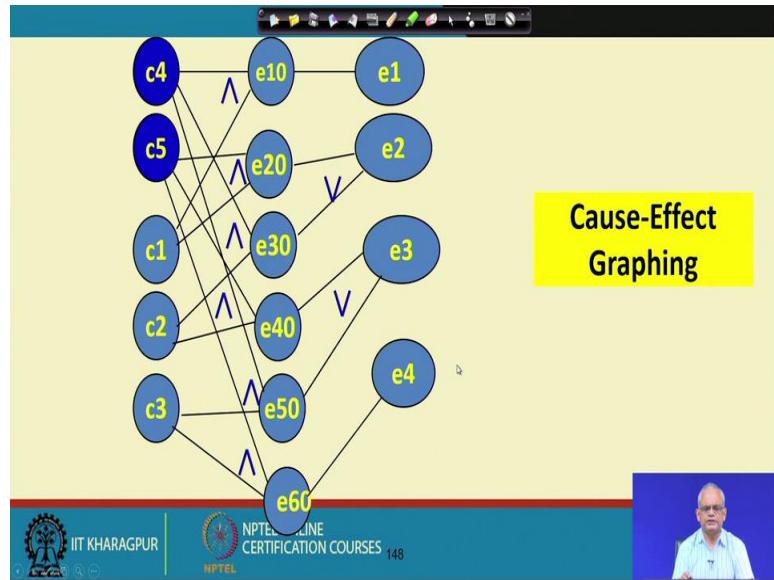
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 147

NPTEL

Now, we can write the causes which are the conditions basically and the corresponding effects. So, deposit is less than 1 year, deposit is between 1 to 3 year, deposit is greater

than 3 years, deposit is less than 1 lakh and deposit is greater than 1 lakh. So, these are the different conditions or causes. Then we will have the corresponding effect whether rate is 6 percent, 7 percent, 8 percent or 9 percent. Now, we present in the form of a Graph.

(Refer Slide Time: 22:48)



If you look here that C1, C2, C3 are the duration of the deposit C4, C5 are the amount of deposit, whether it is > 1 lakh or < 1 lakh, c4, c5 this is whether the deposit > 1 lakh or < 1 lakh and c1, c2, c3 are the duration. Now, if the deposit < 1 year and the amount deposited < 1 lakh, then the rate of interest is 6 percent. If, the deposit is for > 1 year and the amount deposited < 1 lakh then it is 7 percent and so on.

So, this is the AND condition I have represented here. To indicate the fact that if both of these hold, then this action takes place. Here for this we have a OR condition represented here in the figure above, because if any of this holds for any of this action takes place. If we have a AND here then both of these takes place then this will hold, but we have used a OR here.

So, cause effect graph is a very simple technique, which kind of helps in developing a decision table. Once we have get this, then developing the decision table become straight forward, we form these conditions and these are the actions and then we look at this and form the different condition values of the conditions.

(Refer Slide Time: 25:09)

C1	C2	C3	C4	C5	e1	e2	e3	e4
1	0	0	1	0	1	0	0	0
1	0	0	0	1	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	0	0	1	1	0	1	0

- Convert each row to a test case

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 149

So, this is the representation of decision table. These are the actions, these are the conditions and different combinations of the conditions are easily identified from the cause effect graph.

(Refer Slide Time: 25:24)

Pair-wise Testing

IIT KHARAGPUR 150 | NPTEL ONLINE CERTIFICATION COURSES 150

In the combinatorial testing so far we have discussed about the decision table based testing. We also saw that the cause effect graphing is a systematic technique, which develop a graph from the input problem description. The graph then can be easily

translated into a decision table that helps us to generate the test cases, because every column will become a test case.

In our subsequent discussion, we look at the pair wise testing. Sometimes the number of conditions and the actions are too many. We were just discussing about the case of an equivalence partitioning and the boundary value testing. The numbers of parameters are too many, which normally occurs in the case of user interface and controller type of programs, then the number of test cases can become too many. In that case is there any way we can reduce the number of test cases.

But, still achieve as much effective testing as either a robust testing or let's say decision based testing, we will discuss the pair wise testing, which will help us to reduce the number of test cases substantially and still achieve good testing. So, that topic we will discuss in the next lecture, which is pair wise testing.

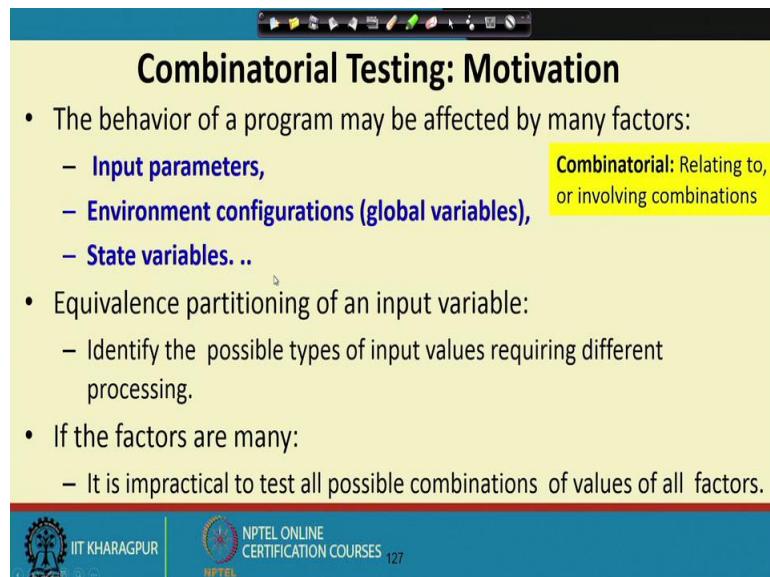
Thank you.

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

Lecture – 52
Decision Table Testing

Welcome to this lecture! In the last lecture, we had started to discuss about combinatorial testing. We will discuss these testing techniques today, in further details. We will first revisit what we discussed in the last lecture and then we will discuss further.

(Refer Slide Time: 00:40)



Combinatorial Testing: Motivation

- The behavior of a program may be affected by many factors:
 - **Input parameters,**
 - **Environment configurations (global variables),**
 - **State variables...**
- Equivalence partitioning of an input variable:
 - Identify the possible types of input values requiring different processing.
- If the factors are many:
 - It is impractical to test all possible combinations of values of all factors.

Combinatorial: Relating to, or involving combinations

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 127

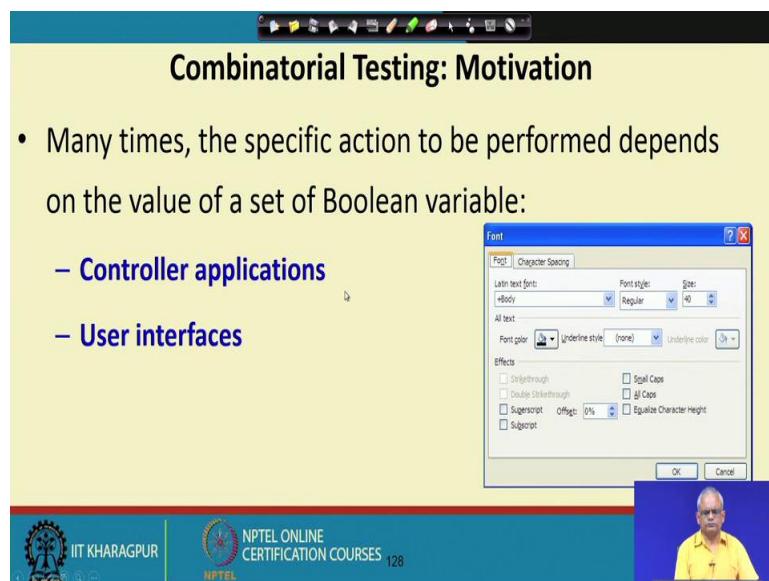
The word combinatorial, if you look in the dictionary, it means relating to or involving combinations. The idea here is that, in testing the software, there are multiple inputs and conditions. In the previous lectures, we had seen how to design black box test cases based on the input values, you looked at equivalence class boundary values and so on. But, then when there are multiple inputs, we need to design test cases based on various possible combinations among these input. And there can also be global variables and so on. Therefore, the combinations must be considered and of course, the combinations can become too many and then it becomes difficult to test. We will today see how to reduce the number of combinatorial test cases.

First, we will discuss the combinatorial test case design and later we will see how to reduce the number of test cases that are designed. The behavior of a program is affected

by many factors. For example, the specific parameters we give as input and there can be data stored as global variables for example arrays in a library example may be books, members, these are large arrays and their state changes depending on whether a book is issued, returned and so on. There can be state variables, which define the state of the software.

We had seen in the previous lecture that for every input variable, using the black box testing technique of equivalence class, boundary values and so on, we can design test cases. Then we have to consider various possible ways of combining this inputs, but the problem is that often the input maybe too many; there may be many parameters, environmental variable, state variable and so on. The number of test cases required for all possible combination may become trillions and the test suite become extremely large. We will see how to address this problem and reduce the number of test cases. But, first let's see the basic idea behind combinatorial generation of test cases and then we will see how to reduce the number of tests.

(Refer Slide Time: 04:07)



Let's look at one specific test situation; this is a user interface for a word processing software as shown in the above figure. You might have used similar software, where we can define the font characteristics. Here we have many things to choose. For example, whether the entered font is a superscript, subscript, small cap, all cap, then the font color, then the specific font type and whether it is a regular italic and so on and the size of the

font and so on. This kind of interface is very common in user interfaces, but then if we want to test based on all possible combinations of the input, then it may become too many. For example, we would like to check, whether it works well in superscript and when text font is something and text color is something. Whether it works as subscript when for the same font, font color size and so on. This kind of situation is also there in controller applications, where the controller behavior is defined by many factors. For example, temperature, pressure, user input and other factors.

(Refer Slide Time: 05:51)

Combinatorial Testing

- Several combinatorial testing strategies exist:
 - **Decision table-based testing**
 - **Cause-effect graphing**
 - **Pair-wise testing (reduced number of test cases)**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 129

With this motivation, we will discuss 3 main types of combinatorial testing. One is called as a decision table based testing, cause effect graphing and pair wise testing. The pair wise testing actually reduces the number of test cases, which may be obtained through a decision table or cause effect graphing.

(Refer Slide Time: 06:21)

Decision table-based Testing (DTT)

- Applicable to requirements involving conditional actions.
- This is represented as a decision table:
 - Conditions = inputs
 - Actions = outputs
 - Rules = test cases
- Assume independence of inputs
- Example
 - If c1 AND c2 OR c3 then A1

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No	No	Yes

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 130

Let's revisit the decision table based testing. In decision table, the upper rows of the table are the conditions, which are basically the inputs and the lower rows are the actions and these are the output. This table allows us to consider various possible combinations of the conditions.

There are 4 conditions here and all conditions are Boolean. If we consider all possible combinations, we will get 16 rows here, but the conditions may not be Boolean, it can be let say an integer taking value 1 to 10 for a choice and so on. So, you can know once we list down the conditions and actions, we know that what are the possible values of the input here and we can define the rules based on the specific values of the inputs distinct values of the input. Then identify which of the actions take place for that specific combination of conditions.

Each of the column here becomes a test case and this is called as a rule, because this is a specific combination of conditions and each of these become a test case, just to get a feel of it let's try to do the decision table for this expression, that is given here. Here, let's assume that c1, c2, c3 are Boolean expressions and if this is true then A1 is the action. But, if the outcome of this specific combination is false, then there is no action, but this is the simplest as I was saying, we might have much more complex where we might have several actions possible depending on specific outcome of the expression.

Just to explain the concept, how to draw the decision table for an expression like this, we need to write the conditions here C1, C2 and C3. These are the upper rows of the decision table and the lower row are the actions and we will write conditions or inputs and will write here the action as shown in the figure.

(Refer Slide Time: 09:11)

Decision table-based Testing (DTT)

- Applicable to requirements involving conditional actions.
- This is represented as a decision table:
 - Conditions = inputs
 - Actions = outputs
 - Rules = test cases
- Assume independence of inputs
- Example
 - If c1 AND c2 OR c3 then A1

Action	C1	C2	C3	A1
Y	T	T	T	Y
N	F	T	F	N
-	-	-	-	-

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 130

We will consider various combinations of conditions of the truth value of these conditions. Each of these is called as a rule or a test case let say C1 is true, C2 is true, C3 is true, therefore, the action A1 is yes; it will be taken. If C1 is true, C2 is true and C3 is false then still it is yes, but if C1 is true C2 is false and C3 is false, then it will be no and so on. So, this is the simple way to construct a decision table given a simple expression, we should be able to write the decision table for that and these become the test cases in addition table based testing.

(Refer Slide Time: 10:57)

	Rule1	Rule2	Rule3	Rule4
Condition1	Yes	Yes	No	No
Condition2	Yes	X	No	X
Condition3	No	Yes	No	X
Condition4	No	Yes	No	Yes
Action1	Yes	Yes	No	No
Action2	No	No	Yes	No
Action3	No	No	No	Yes

So, the idea here is that we write the conditions in the top rows, actions in the bottom rows and consider various combinations of conditions and each of these become a test case and we call them as a rules.

(Refer Slide Time: 11:20)

• A decision table consists of a number of columns (rules) that comprise all test situations
 • Example: the triangle problem
 – C1: a, b, c form a triangle
 – C2: a=b
 – C3: a=c
 – C4: b=c
 – A1: Not a triangle
 – A2: scalene
 – A3: isosceles
 – A4: equilateral
 – A5: Right angled

Sample Decision table

	r1	r2	...			m
C1	0	1				0
c2	-	1				0
C3	-	1				1
C4	-	1				0
a1	1	0				0
a2	0	0				1
a3	0	0				0
a4	0	1				0
a5	0	0				

Let's take a very simple example. Let's assume that we are trying to design the decision table based testing for a function, which is checkTriangle. The checkTriangle the name of the function and takes 3 integers a, b, c as it is input. The function will display that the specific triangle is a scalene, isosceles, equilateral, not a triangle and so on.

Now, here the conditions based on which the action had taken, can write here on the top rows, C1 is that it forms a triangle, C2 $a = b$, C3 $a = c$, C4 $b = c$. So, it is a triangle if it is $a = b$, $b = c$ and $c = a$ then, its A4 is an equilateral triangle. If it is not a triangle, then it will display not a triangle that is A1 and so on. We can easily develop this table, but the problem is that here we have written the condition 1 is a, b, c form a triangle, but that is not a very practical condition to check, we will see that we can refine the table.

(Refer Slide Time: 13:13)

Test cases from Decision Tables				
Test Case ID	a	b	c	Expected output
TC1	4	1	2	Not a Triangle
TC2	2888	2888	2888	Equilateral
TC3	?)	Impossible
TC4				
...				
TC11				

C1: a, b, c form a triangle
 C2: $a=b$
 C3: $a=c$
 C4: $b=c$

But, once we have drawn the decision table, we can design the test cases which are specific combinations of the conditions and the corresponding actions are the output to be checked. If C1 is that a, b, c is not a triangle is true then the display will be not a triangle irrespective of the condition C2, C3, C4.

We can do that, write the specific value which will satisfy condition 1, which is 4, 1, 2, then we say the expected output is not a triangle. Similarly, if it is a triangle and also $a = b$, $b = c$, $c = a$, then we write some specific values which satisfies that condition and the expected output is equilateral. Each of these is a test case, we give the test case number here, test case id and that is how we design the equivalent decision table based test cases.

(Refer Slide Time: 14:32)

Conditions											
C1: $a < b+c?$	F	T	T	T	T	T	T	T	T	T	T
C2: $b < a+c?$	-	F	T	T	T	T	T	T	T	T	T
C3: $c < a+b?$	-	-	F	T	T	T	T	T	T	T	T
C4: $a=b?$	-	-	-	T	T	T	T	F	F	F	F
C5: $a=c?$	-	-	-	T	T	F	F	T	T	F	F
C6: $b=c?$	-	-	-	T	F	T	F	T	F	T	F
Actions											
A1: Not a Triangle	X	X	X								
A2: Scalene											X
A3: Isosceles								X	X	X	
A4: Equilateral				X							
A5: Impossible				X	X			X			



This is a more complete decision table, because in the top row we have replaced, whether it forms a triangle or not with specific conditions, which are easier to check $a < b + c$, $b < a + c$, $c < a + b$. If all of this is true, then it forms a triangle. As long as it is false, the output is not a triangle, but if all the 3 conditions where C1, C2, C3 is true, then it forms a triangle. Therefore, we check the other conditions C4, C5, C6 and based on that we write whether it is a scalene, isosceles, equilateral or it is impossible, impossible is that it is not integer values and so on.

(Refer Slide Time: 15:58)

Test Cases for the Triangle Problem	Case ID	a	b	c	Expected Output	
	DT1	4	1	2	Not a Triangle	
	DT2	1	4	2	Not a Triangle	
	DT3	1	2	4	Not a Triangle	
	DT4	5	5	5	Equilateral	
	DT5	?	?	?	Impossible	
	DT6	?	?	?	Impossible	
	DT7	2	2	3	Isosceles	
	DT8	?	?	?	Impossible	
	DT9	2	3	2	Isosceles	
	DT10	3	2	2	Isosceles	
	DT11	3	4	5	Scalene	



Once we have developed the table, we can write specific values for the input and we write the expected output and the all of these form a test case.

(Refer Slide Time: 16:16)

Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			X					
	Check the printer-computer cable	X		X					
	Ensure printer software is installed	X		X		X		X	
	Check/replace ink	X	X			X	X		
	Check for paper jam		X		X				

NPTEL ONLINE
CERTIFICATION COURSES

Let's take one more example, which is the printer troubleshooting example. Here depending on some specific condition the program recommend some action to take. For example, if the printer does not print, red light is flashing and printer is unrecognized. It recommends to check printer cable, ensure printer software is installed and check replace check or replace ink. If printer is not printing and red light is flashing only, there is no printer unrecognized message and then need to check and replace ink and check for paper jam.

Similarly, if the printer does not print and printer is unrecognized, then we need to check the power cable for the printer and we need to check the printer cable and ensure that the printer software is installed and so on, we develop the table and each of the row columns here become a test case.

(Refer Slide Time: 17:33)

Quiz: Develop BB Test Cases

- Policy for charging customers for certain in-flight services:

If the flight is more than half-full and ticket cost is more than Rs. 3000, free meals are served unless it is a domestic flight. Otherwise, no meals are served. Meals are charged on all domestic flights.

	C1	C2					
C3	Y	Y	-	-	-	-	-
A1	Y	Y	-	-	-	-	-
A2	N	Y	-	-	-	-	-
Meals Served Free							

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 137

A video player window shows a man speaking.

Let's develop the decision table test case for a specific situation. If a flight is more than half-full and ticket cost is more than 3000, free meals are served, unless it is a domestic flight. In domestic flight meals are served, but these are not free, but if the flight is not half full, is less than half-full and ticket cost is less than 3000 or ticket cost is less than 3000, then no meals are served.

To develop the decision table for this, we need to identify the conditions. The specific condition here is that whether the flight is more than half-full, we will write that as condition 1, ticket cost is more than 3000 we will write that as condition 2, the actions is whether free meal is served and whether it is a domestic flight. So, this is the third condition here, the flight is more than half-full, ticket cost is more than 3000 and it is a domestic flight we can identify that these are the conditions and the actions are that whether meal is served and whether it is free.

So, we can write here C1, C2, C3 and action1 action2. Action1 is whether meal is served and Action2 is whether it is free. We will check whether C1 is that flight is more than half-full, ticket cost is more than 3000 and it is a domestic flight, then meal is served A1 and A2 is that it is free meal. So, will write here, yes and A2 is no. Similarly, if it is more than half-full and ticket cost is more than 3000, but it is not a domestic flight, then meal is served and also it is free and so on. Then find all possible combinations of the conditions and each of the columns here will become a test case.

(Refer Slide Time: 20:58)

		POSSIBLE COMBINATIONS							
CONDITIONS		N	N	N	N	Y	Y	Y	Y
	more than half-full	N	N	N	N	Y	Y	Y	Y
	more than Rs.3000 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	serve meals					Y	Y	Y	Y
	free								Y



NPTEL ONLINE
CERTIFICATION COURSES 138

So, I just drawn here for your reference that these are the 3 conditions and then the specific actions; write all possible combinations of the conditions. If there are 3 here and each is a binary then we will have 2^3 , 8 that is the number of columns here, it will give all possible combinations of the conditions of these 3 input Boolean variables.

(Refer Slide Time: 21:28)

		POSSIBLE COMBINATIONS							
CONDITIONS		N	N	N	N	Y	Y	Y	Y
	more than half-full	N	N	N	N	Y	Y	Y	Y
	more than Rs. 3000 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	serve meals					Y	Y	Y	Y
	free								Y

NPTEL ONLINE
CERTIFICATION COURSES 139

In this we have filled up the action part and these each of these become a test case and then the corresponding actions are checked when the test case is executed.

(Refer Slide Time: 21:46)

		POSSIBLE COMBINATIONS							
CONDITIONS	more than half-full	N	N	N	N	Y	Y	Y	Y
		N	N	Y	Y	N	N	Y	Y
ACTIONS	domestic flight	N	Y	N	Y	N	Y	N	Y
	serve meals	N Y		N Y		X	X	X	X
free		N Y		N Y		X			

IIT Kharagpur
NPTEL ONLINE
CERTIFICATION COURSES 140

But, even using a decision table, we can optimize the number of test cases, we can reduce little bit. For example, this is the decision table that we developed, but we can see here that the action part is the same here. There is no action for these 2 test cases or a rules, but the specific input values are same for both of these conditions and this is just no and yes. Therefore, we can combine this into a single rule and we will write don't care here. It is possible to combine these 2 because the action part is the same and irrespective of the domestic flight, nothing is done if it is less than 3000 and less than half-full.

Similarly, we can combine first two actions as drawn in the above diagram. The action part is the same and then they differ with respect to domestic flight. So, this becomes a don't care, but what about first two actions, we cannot combine with any other, because we do not have action. We have the same action here in column 5 and 6, but we can combine either column 5 with column 8 that these 2 are having same action or we can combine column 6 with column 8. So, they have the same action part. Then they differ with respect to only more than 3000. So, you can combine this into single rule and then we cannot combine more, because your already don't care here. So, I have drawn that reduced the number of test cases and I have written here don't care.

(Refer Slide Time: 24:12)

Final solution		Combinations			
CONDITIONS	<i>more than half-full</i>	N	Y	Y	Y
	<i>more than 3000 per seat</i>	-	N	Y	Y
	<i>domestic flight</i>	-	-	N	Y
ACTIONS	<i>serve meals</i>		X	X	X
	<i>free</i>			X	

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 141



So, the number of test cases required here is 4.

(Refer Slide Time: 24:36)

—Rules need to be complete:

Assumptions regarding rules

- That is, every combination of decision table values including default combinations are present.

—Rules need to be consistent:

- That is, there is no two different actions for the same combinations of conditions

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES 142



Now, in the decision table based testing, we must ensure that all possible combinations of the decisions of the conditions are taken care. Also we must ensure that the rules are consistent i.e., it should not be the case that for the same combination of condition, we are writing 2 different actions i.e., inconsistency. Basically if we write let say 3 conditions yes, yes and yes and we write that action A1 is yes, and A2 is no and for another rule we write same combination of condition and the actions are different, it

cannot happen because this is inconsistency. We must take care that the table is consistent and all possible combinations of conditions are present and that will give us our decision based test cases.

(Refer Slide Time: 25:53)

The slide has a yellow header and a red footer. The header contains the title 'Guidelines and Observations'. The main content is a bulleted list under the heading 'Decision table testing is appropriate for programs:' followed by a single bullet point in red.

- Decision table testing is appropriate for programs:
 - There is a lot of decision making
 - Output is a logical relationship among input variables
 - Results depend on calculations involving subsets of inputs
 - There are cause and effect relationships between input and output
- **Decision tables do not scale up very well**

The footer contains the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the number '143'.

The decision table based test cases are relevant for many problems. If we look at the code and see that there is lot of decision not code. If we look at the description of the soft program and see that the specification involves lot of decision making. The output is a logical relation among input variables.

The results depend on calculation involving subsets of input or there are cause and effect relationship between input and output, these are the cases where we have to design the decision table based test cases. But, the only problem with the decision table based test cases is that, as the number of input conditions become large, the table size grows exponentially, if 3 Boolean conditions we need 8 test cases.

If we have 4, we need 16, for 5, we need 32 and so on. If we have 10 then 1000 that becomes difficult to develop the table and the number of test cases grows very rapidly, it becomes difficult to optimize, manually reduce the number of test cases and so on. Also, the decision table based test cases even though it is very simple if we understand the problem very well. But if we do not understand the problem very well; it becomes difficult to develop the decision table.

We will look at another technique which is cause effect graphing, which gives us a graphical way to represent the input conditions, output conditions and from there we can develop the decision table. We are already at the end of this lecture, we will stop here continue in the next lecture.

Thank you.

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

Lecture - 53
Cause effect graphing

Welcome to this lecture. In the last lecture, we discussed the decision table based testing, where we considered various combinations of conditions. We will proceed from there. We will look at the Cause Effect Graphing. Then we look at the pair wise testing, P way testing and pair wise testing, but before that let us do a small quiz based on the decision table based testing.

(Refer Slide Time: 00:52)

Quiz: Design test Cases

- Customers on a e-commerce site get following discount:
 - A member gets 10% discount for purchases lower than Rs. 2000, else 15% discount
 - Purchase using SBI card fetches 5% discount
 - If the purchase amount after all discounts exceeds Rs. 2000/- then shipping is free.

So, the problem here is that we have an e-commerce site. Here a member gets 10 percent discount on purchases lower than 2000 rupees and if it is more than 2000 get 15 percent discount. Purchase using some specific card like SBI card fetches additional 5 percent discount. If the purchase amount after all discounts exceeds 2000 then shipping is free, otherwise the shipping is charged. Now to develop the decision table based testing, we need to identify the conditions and the actions. The next step would be to consider all possible combinations of conditions. Then identify the actions corresponding to those and we represent that on the decision table. Each column in the decision table becomes a test case.

Now, let's first identify the conditions. If we read here member gets 10 percent discount for purchases lower than 2000. So, a condition here is that if the purchase is lower than 2000, then we will get 10 percent discount if it is yes. If it is no then it is 15 percent discount, whether payment is made using SBI card this is another condition.

So, let me also write down the condition C1 and C2 and SBI card payment fetches an additional 5 percent discount. The third condition is that the purchase amount after all discount exceeds 2000, this is the third condition. And the action part is the decision is 10 percent, 15 percent, additional 5 percent and then the total amount exceeds 2000 and shipping is free.

We can write C1, C2, C3, C4 in the top rows of the decision table and A1, A2, A3, A4 are the actions. And then if C1 is true i.e., purchase is lower than 2000 C1 is yes, C2 is it is not purchased using SBI card, C3 is the purchase amount exceeds after all discount 2000 rupees. We will say no and then we will see that the discount is 10 percent yes and the rest are no and so on.

We need to design this table, but just one word of caution is that if C1 is yes, purchase amount is less than 2000, 10 percent then automatically the purchase amount after discount will be less than 2000. So, we cannot have 10% discount and this is also yes. If A1 is yes, A2 has to be no. So, there is a dependency between these 2 and that we have to take care while developing the decision table.

(Refer Slide Time: 05:03)

The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the title 'Cause-effect Graphs' is centered. To the right of the title, there is a small portrait of a man in a yellow shirt. The main content of the slide is a bulleted list under the heading '• Overview:'. The list includes the following points:

- Explores combinations of possible inputs
- Specific combination of inputs (causes) results in outputs (effects)
- Represented as nodes of a cause effect graph
- The graph also includes constraints and a number of intermediate nodes linking causes and effects

At the bottom of the slide, there is a footer bar. On the left side of the footer, there is the logo of IIT Kharagpur and the text 'IIT KHARAGPUR'. In the center, there is the text 'NPTEL ONLINE CERTIFICATION COURSES 145' next to the NPTEL logo. On the right side of the footer, there is a small video player window showing a video of the same man in the yellow shirt.

As we are mentioning earlier the cause effect graph provide a systematic technique and graphical symbol based on which we can explore the combinations of different inputs, represent them as nodes, edges in a graph. Once the graph is developed, we can easily translate the graph into a decision table. In the graph we can have constraints like and, or etc. and also we can have intermediate nodes between the input and output, which will help us to simplify the graph.

(Refer Slide Time: 05:53)

Cause-Effect Graph Example

- If depositing less than Rs. 1 Lakh, rate of interest:
 - 6% for deposit upto 1 year
 - 7% for deposit over 1 year but less than 3 yrs
 - 8% for deposit 3 years and above
- If depositing more than Rs. 1 Lakh, rate of interest:
 - 7% for deposit upto 1 year
 - 8% for deposit over 1 year but less than 3 yrs
 - 9% for deposit 3 years and above

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 146

To explain this cause effect graphing using a simple example, that we were discussing earlier that if the deposit amount is less than 100000, then the rate of interest is 6 percent for deposit up to 1 year, over 1 year and less than 3 year it is 7 percent, 8 percent deposit is 3 years and above. But, if the amount is more than 1 lakh, then the rate of interest is higher i.e., 7 percent, 8 percent and 9 percent.

Now, we want to develop the cause effect graph for this and once we develop the cause effect graph, we will translate that into a decision table. Then the decision table will give us the test cases.

First let's do the cause effect graph, the cause is the conditions on the input and the action is the output. So, here the conditions and the input, if it is the amount of deposit is less than 1 year, 1 to 3 year, 3 year and also another condition is that, whether it is less than 1 lakh or more than 1 lakh.

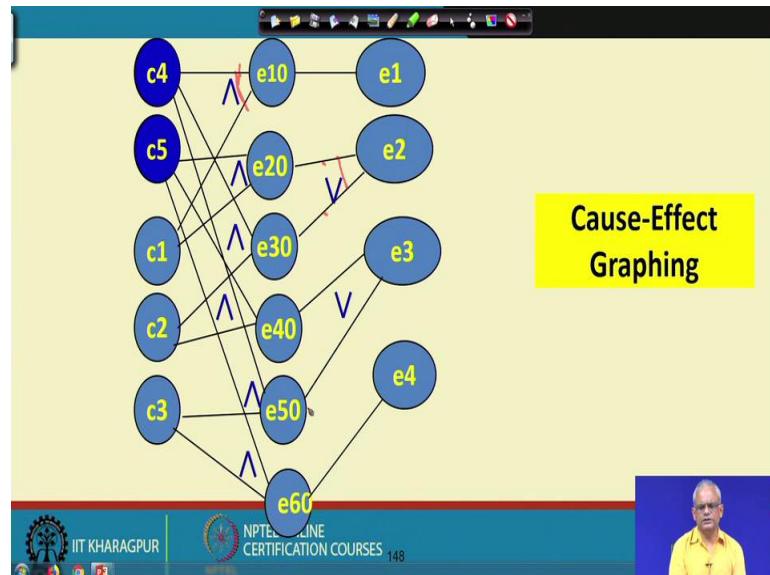
(Refer Slide Time: 07:16)

Cause-Effect Graph Example	
Causes	Effects
C1: Deposit<1yr	e1: Rate 6%
C2: 1yr<Deposit<3yrs	e2: Rate 7%
C3: Deposit>3yrs	e3: Rate 8%
C4: Deposit <1 Lakh	e4: Rate 9%
C5: Deposit >=1Lakh	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 147

So, we represent that here the causes, deposit < 1 year, 1 to 3 year, > 3 year and also, whether the deposit amount < 1 lakh or \geq 1 lakh. We also note down the different effects or the actions, which are whether the rate applicable is 6 percent, 7 percent, 8 percent and 9 percent. Now, each of these we represent in from form of a graph.

(Refer Slide Time: 07:51)



We write the input conditions and the actions and then specific combinations of the input conditions; they give rise to specific actions.

For example, if it is less than 1 lakh and it is less than 1 year we write a AND symbol here with C1 we write a AND symbol here with C4. So, that this is the constraint that both these conditions must hold. And then the corresponding effect or the action is e1. We have also a constraint in the form of or any of this if it holds then the action is e2 and so on. Can easily translate the input into this graph form cause effect graph and once we have done the cause effect graph.

(Refer Slide Time: 08:58)

Develop a Decision Table

C1	C2	C3	C4	C5	e1	e2	e3	e4
1	0	0	1	0	1	0	0	0
1	0	0	0	1	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	0	0	1	1	0	1	0

- Convert each row to a test case

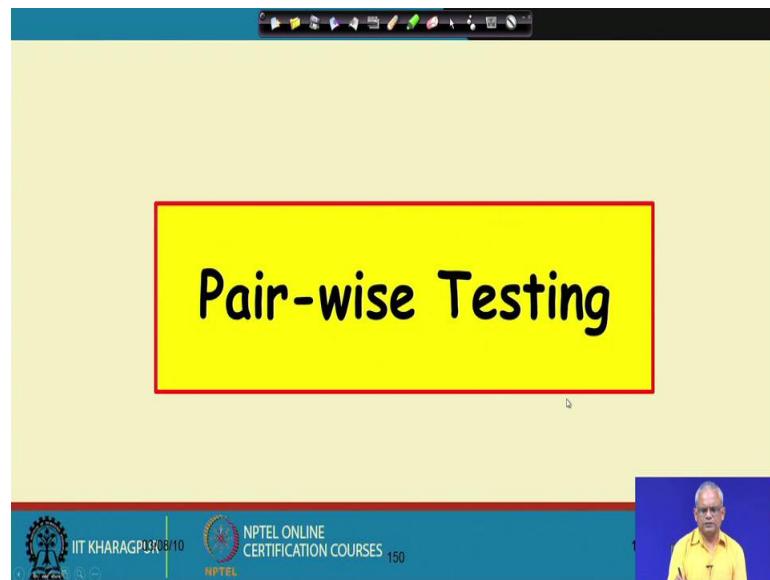

IIT KHARAGPUR

NPTEL ONLINE
CERTIFICATION COURSES
149



This can easily be translated into a decision table; we just identify from the graph that if certain causes are both 1, then the effect is 1 and so on. Once the decision table is stable out from cause effect graph, we get the test cases.

(Refer Slide Time: 09:20)



Now, let's discuss how to reduce the number of test cases, because in some situations where the input data are too many. Input conditions are too many, the number of test cases increases exponentially. And that can become a very large number, billions or trillions of test cases and it becomes impossible to test the software. Even though we are able to design the test cases using a decision table based testing. The pair wise testing helps us to reduce the number of a test cases.

(Refer Slide Time: 10:08)

A screenshot of a presentation slide titled 'Combinatorial Testing of User Interface'. On the left, there is a 10x10 matrix of binary values (0s and 1s) representing combinations of two input variables. A legend below the matrix states '0 = effect off' and '1 = effect on'. To the right of the matrix, a 'Font' dialog box is open, showing various font options like Times, Regular, Italic, Bold, and Bold Italic. A red oval highlights the 'Effects' section of the dialog box, which includes options like Strikethrough, Double strikethrough, Superscript, Subscript, Shadow, Outline, Emboss, and Hidden. Below the matrix, a note states ' $2^{10} = 1,024$ tests for all combinations'. At the bottom, a calculation ' $* 10^3 = 1024 * 1000 \dots$ Just too many to tests' is shown. The slide also features the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 151', and the NPTEL logo at the bottom.

Let's see, how do we go about developing the pair wise test cases. In the motivation for pair wise test cases, we just take the same example here that we are discussing earlier. That we consider the font in a word processing software and the user has to choose the specific font to be used, the style to be used and the size of the font. And also can use the checkboxes to indicate whether it should be a strikethrough, double strikethrough, superscript, subscript and so on.

Now, based on this how many test cases can be designed on a decision table based testing. We count here to be 10 checkboxes. Each of these is either checked or unchecked. If we consider all possible combinations of this, then it becomes 2^{10} . So, only based on the checkboxes it becomes 1024, but then what about the other options? That also we must consider.

Let's assume that there are only 10 here it may be more, but for our simplicity. We just consider that there are only 10 font types to be selected and 10 styles to be selected and 10 font size to be selected and then, how many test cases? We need to multiply 2^{10} with 10^3 that becomes 1024000. Too many test cases for anybody to test the software. It will take years or 10s of years, but how do we go about testing this kind of software, because, these are very common and I was saying that even controller software also have similar characteristic that many input variable combinations arise and different actions corresponding to those.

Let see at how to reduce the number of test cases such that it becomes possible to test this easily and also at the same time we do not want to lose the thoroughness of testing. We need a technique, which will reduce this 1024000 into just 8 or 10 test cases and also the thoroughness of testing should not suffer too much of course, it cannot be exactly as thorough as testing with all possible combination of conditions, but it should be 99 percent. Also, the thoroughness must be achieved and that is achieved by the pair wise testing, let's see the ideas behind this.

(Refer Slide Time: 13:44)

The diagram illustrates a combinatorial testing problem. At the top, five input variables are labeled x_1 , x_2 , x_3 , \dots , and x_n . Arrows point from each of these variables down to a central box labeled "System S". The background of the slide is light yellow.

Combinatorial Testing Problem

$x_1 \downarrow \quad x_2 \downarrow \quad x_3 \downarrow \quad \dots \quad x_n \downarrow$

System S

*Combinatorial testing problems generally follow a simple input-process-output model;
*The “state” of the system is not the focus of combinatorial testing.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 152

A video player interface is visible at the bottom right, showing a man in a yellow shirt speaking.

So, here the problem that we are addressing is that the system has only 1 state and there are many inputs and we need to consider the combinations on these input, but you may say that what if the system is state based; and there are multiple states of the system and transitions among state. Then, the problem is more complicated we need to design the combinatorial test for every state of the system. And also every transition and therefore, the problem becomes more complex, but we are now restricting to a simpler problem, where the system has only 1 state and x_1, x_2, \dots, x_n are the inputs. And we need to test using various possible combinations of the inputs.

(Refer Slide Time: 14:40)

The diagram discusses t-way testing. It lists several points:

- Instead of testing all possible combinations:
 - A subset of combinations is generated.
- Key observation:
 - It is often the case that a fault is caused by interactions among a few factors.
- t-way testing can dramatically reduce the number of test cases:
 - but remains effective in terms of fault detection.

t-way Testing

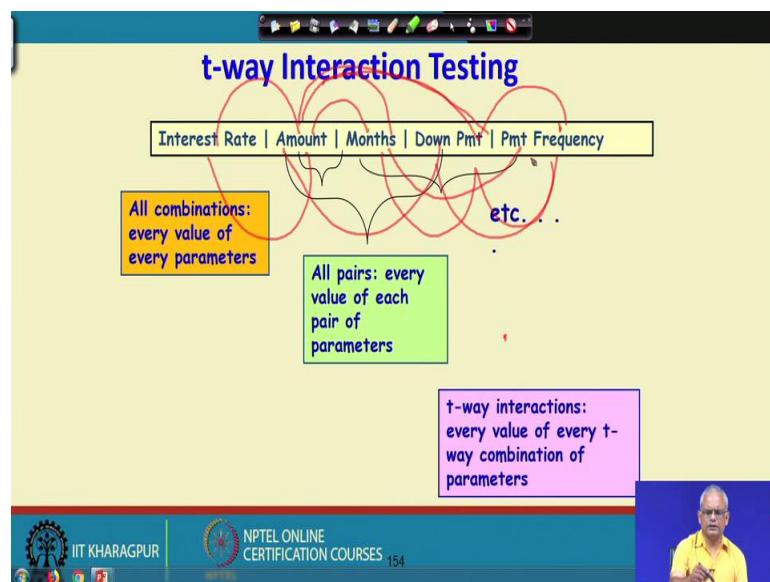
IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES 153

A video player interface is visible at the bottom right, showing a man in a yellow shirt speaking.

The key observation here is that often the bugs are caused by interaction among few factors. Even though we have 100s of input, but some specific 2 combination among this might cause the bug or may be 3 combinations or in general t combinations may cause the bug.

If we consider all possible t combinations among the input, then we can dramatically reduce the number of test cases. And still we will have an effective bug detection, even very reduced number of test cases. Let's discuss this idea further.

(Refer Slide Time: 15:57)



Let's assume that a program or a function takes these as the input. Let's say it takes input interest rate, the amount borrowed, the number of months for which the amount is borrowed and the down payment and then the payment frequency. Based on that, it will give us how much to be paid every month. That is the function.

If we look at the parameters here, there are 5 parameters. There are many possible equivalence classes boundary values and so on for each of this. If we consider all possible combinations of conditions we represent this in a decision table and test, then it will become too many, but we can consider pair wise testing in which we have all possible combinations among 2 of the input present. So, we have all possible combinations between any 2 conditions. Any 2 if they are present, then it is called as a 2 way testing.

We similarly can have 3 way testing. If we have all possible combinations of any 3. All possible combinations among the different parameters, then we have a 3 way testing, similarly we can have t way testing.

(Refer Slide Time: 18:19)

Pairwise Testing					
	Pressure	Temperature	Velocity	Acceleration	Air Density
A	T1	1	10	1.1	
B	T2	2	20	2.1	
	T3	3	30	3.1	
Pressure	Temperature				
A	T1	4	0		
A	T2	5			
A	T3	6			
B	T1				
B	T2				
B	T3				

Now, let's understand the pair wise testing further. Let's for simplicity assume that there are 2 possible classes of input for pressure A and B, 3 possible input equivalence class input for temperature T1, T2, T3. For velocity there are 6 velocity classes acceleration are 10 20 30 etc., air density is some 1.1, 2.1, 3.1.

Now, let us say we want to consider pair wise testing and we need to have test case, which will have any combination between 2 of these input present. Let us consider these 2 i.e., pressure and temperature. So, a test case must have pressure A temperature T1, pressure A temperature T2, A T3 and B T1, B T2, A T3 and B T3. So, that is all possible combinations of pressure and temperature and there are 6 entries here.

Similarly, we might have temperature and velocity. So, we will have 18 entries here. It is not necessary that, they may be there on a different test case, but it may be that they are on the same test case, some of these are present let's say T1 when this is T1 this is let us say velocity is 1 velocity, when temperature is T2 velocity is 2 and so on.

So, given the set of test cases, we should be able to identify given any 2 possible input values any 2 parameters, we should be able to find all possible combinations of

conditions. Any 2 may temperature, air density, velocity acceleration, or velocity air density.

In our setup test cases we should be able to identify all possible combinations of conditions. This is different from all possible combinations, where we have 0 0 0 0 0 0 0 1 etc., all possible combinations of conditions and that is too many. Here, it is possible to drastically reduce the number of test cases.

(Refer Slide Time: 21:15)

Number of inputs	Number of selected test data values	Number of combinations	Size of pairwise test set
7	2	128 $(2^7)^2$	8
13	3 3^{13}	1.6×10^6 $3^{13} - 1$	15 3×2
40	3	1.2×10^{19}	21

Now, let's see what is the reduction that is possible? If the number of input is 7 and each is a Boolean, then if we do a decision table based testing. Then the number of rules or combinations of conditions is 128.

But, if we develop the pair wise tests, then we can do with 8. If the number of input is 13 and each can take 3 possible values, then the number of possible combinations or the number of the columns on the decision table becomes 3^{13} . It's 1.6 million test cases, but then if you do pair wise testing it becomes just 15. Significant reduction can be achieved million test cases it is very difficult to test, it will take years to run a million test case, but just see here we can test with almost as much effectiveness using only 15 test cases.

Similarly, if the number of inputs is forty each 1 tests 3 only, then the number of test cases is 1.2×10^{19} . If there is a team testing these test cases in their lifetime they cannot complete testing, but if you consider pair wise testing, then we can do with only 21 test

cases which is a manageable number. So, the point that trying to convey through this table is that the reduction in the number of test cases is significant, is dramatic in pairwise testing over combinatorial test case design like decision table.

(Refer Slide Time: 24:34)

- A t-way interaction fault:
- In practice, a majority of software faults consist of simple and pairwise faults.

– Triggered by a certain combination of t input values.
– A simple fault is a 1-way fault \100 50
– Pairwise fault is a t-way fault where t = 2.

Now, to gain further understanding into the t way testing, 2 way or pair wise testing etc. Let's look at some more concepts, a simple fault is called as a 1-way fault. So, as long as a parameter has certain value this fault occurs, but if certain combination of 2 inputs. Let's say when the temperature becomes 100, only when the temperature is 100 and also at the same time the pressure is 50 only in this situation the error occurs. If the pressure is 100 and temperature 60 and so on the error may not occur. So, this is the 2 way testing will detect this.

A 1- way testing of course, cannot detect this because, we will have hundred as long as it reaches 100 in some other test case it has reached 50, we would have a 1- way testing. It is an important observation that in a majority of software it is an experimental, empirical observation, that in majority of software the fault consists of simple and pair wise faults. Several researches have conducted experiments and they found that vast majority of the faults are either simple faults, where as long as that value is taken that input value is taken then it fails. Whereas a pair wise fault, when 2 specific combination of input is taken then the software fails.

(Refer Slide Time: 26:57)

The slide has a yellow background with a black header bar containing icons. The title 'Single-mode Bugs' is centered at the top. Below the title is a bulleted list:

- The simplest bugs are single-mode faults:
 - Occur when one option causes a problem regardless of the other settings
 - Example: A printout is always gets smeared when you choose the duplex option in the print dialog box
 - Regardless of the printer or the other selected options

Let's see some example of a single mode bug. These kinds of bugs are very common. Here as long as the condition is satisfied, then the software fails one specific condition is satisfied the software fails. For example, as long as the duplex mode is chosen in a printer irrespective of all other possible combinations, as long as it is a duplex option, then the printout gets smeared irrespective of all other options for example printer type, color and so on. So, to test this we just need to check that every option is taken by the test cases, the test cases give every option for all of the input.

(Refer Slide Time: 28:17)

The slide has a yellow background with a black header bar containing icons. The title 'Double-mode Faults' is centered at the top. Below the title is a bulleted list:

- Double-mode faults
 - Occurs when two options are combined
 - Example: The printout is smeared only when duplex is selected and the printer selected is model 394

A small video window in the bottom right corner shows a man speaking.

A double mode fault here only if 2 specific combination of conditions are present then the problem occurs. Just to give an example the printout is smeared when the duplex is selected and the printer selected is 394. So, if you just select duplex and with some other model number problem does not occur, few use simplex with 394, problem does not occur, but only for this specific combination these occurs. So, this can be detected by pair wise testing and in general.

(Refer Slide Time: 29:04)

The slide has a dark blue header bar with various icons. The main content area has a light yellow background. On the left, there is a list of pseudocode steps. In the center, the title 'Example of Pairwise Fault' is displayed in bold black font. On the right, there is a video player showing a man in a yellow shirt speaking. The footer is dark blue and contains the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 161', and the NPTEL logo.

```
• begin
  - int x, y, z;
  - input (x, y, z);
  - if (x == x1 and y == y2)
    • output (f(x, y, z));
  - else if (x == x2 and y == y1)
    • output (g(x, y));
  - Else      // Missing (x == x2 and y == y1) f(x, y, z) - g(x, y);
    • output (f(x, y, z) + g(x, y))
  • end
```

We can have a n-way testing. We will just look at that in the next lecture and we will look at an example of pair wise fault. We are at the end of this lecture and we will stop here and continue in the next lecture.

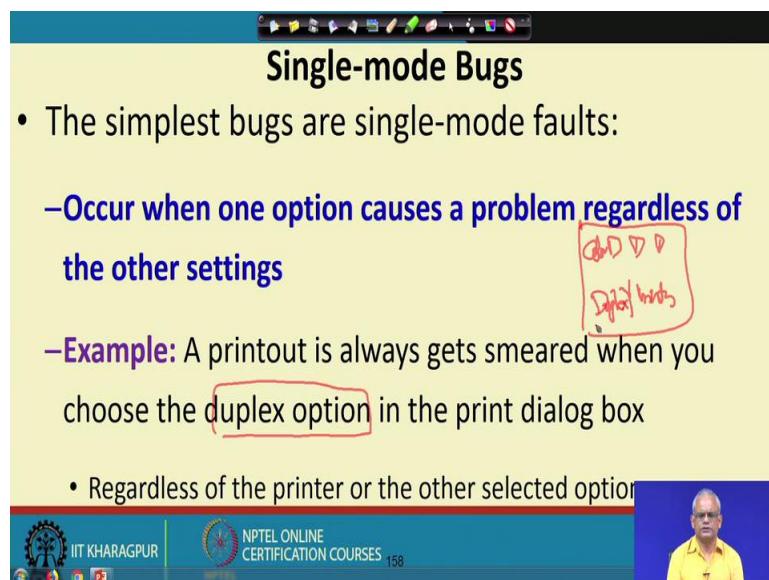
Thank you.

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

Lecture – 54
Pairwise Testing

Welcome to this lecture. In the last lecture we discussing Pairwise Testing and we had mentioned that the combinatorial testing techniques like a decision table based testing have the tendency to generate a large number of test cases. When the number of input conditions are large, actually the number of test cases are exponential in the number of input conditions.

(Refer Slide Time: 00:47)



Single-mode Bugs

- The simplest bugs are single-mode faults:

–Occur when one option causes a problem regardless of the other settings

–Example: A printout is always gets smeared when you choose the **Duplex** option in the print dialog box

- Regardless of the printer or the other selected options

And we were discussing about reducing the number of test cases and pairwise testing is a very promising way of reducing the number of test cases. And we are trying to get some insight into how does the pairwise testing produce such a thorough testing with a very small number of test cases; a drastic reduction from a million test cases to about 7 or 8 test cases and so on. For that we were trying to discuss the characteristics of the bugs, we were saying that the bugs are such that as long as input condition has some value, if the bug is expressed as a failure then we call it as a single mode bug. So, a single mode bug, when there is certain input value is given, then irrespective of all other input the failure

occurs. Just to give an example: as long as the duplex option is chosen on the printer dialogue box. There may be other options, this duplex is the printer type color or black and white and so on. There are many options here. Irrespective of all other options as long as you duplex is selected: yes, then the print gets smeared.

So, if we think of the code, if this becomes yes, then in the printer code there is some problem as long as this becomes yes. So, this is called as a Single-mode bug. The bug expresses itself when as long as some input condition is given value, certain value.

(Refer Slide Time: 03:16)

The slide has a title 'Double-mode Faults'. It contains a bulleted list:

- Double-mode faults

Handwritten notes above the list show a grid with 'Text' at the top left and boxes containing '5' and '8'. Below the list is a note: '-Occurs when two options are combined' followed by a handwritten '7'. Further down, there is a note: '-Example: The printout is smeared only when duplex is selected and the printer selected is model 394'. Handwritten notes next to this note show 'Duplex' with a checked box and 'Printer' with a box containing '394'. The bottom of the slide features the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES 159', and a video frame showing a speaker.

A double-mode fault occurs when 2 options have a specific combination. Let's say, we have many inputs parameters. And as long as this is given a value of 5 and this is given 7 as shown in the figure above, then the problem occurs. But if we have given them separately a value of 5 and at that time it is 8 and this is given 6 and this is 7. So, even though the 5, 7 it is tested, but these are test cases, which have values for all the parameters defined.

As long as they are getting the value 5 and 7 for these 2 parameter there is a failure, but for all the other combinations it works. An example is that the printout smeared only when the duplex is selected and the printer model is 394. So, a duplex option is checked yes and also the printer model. There are many printer, the printer model is selected to be

394, but if same 394 works when it is not duplex and with duplex any other printer will work.

But, then to get a further insight into how does this. How does this happen? That a specific combination of condition causes the problem, some input variable a specific combination of the values of some 2, 3 or so input values. They express themselves as a failure whereas, individually they may be taking the value, but that does not cause the problem.

(Refer Slide Time: 05:32)

Multi-mode Faults

- **Multi-mode faults**

- Occur when three or more settings produce the bug
- This is the type of problems that make complete coverage necessary

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 160

We will see a code example and understand why that situation occurs. In general you can have multi-mode faults. As long as all the input parameters that are given, they have some specific combination of all the input parameter causes the problem. It can be 3 way faults when 3 specific inputs have some value combination; it can be 4 way fault and so on.

(Refer Slide Time: 06:10)

The slide has a yellow background and a title 'Example of Pairwise Fault' in bold black font at the top right. On the left, there is a list of code snippets in a bullet-pointed format. The code is as follows:

```
• begin
  - int x, y, z;
  - input (x, y, z);
  - if (x == x1 and y == y2)
    • output (f(x, y, z));
  - else if (x == x2 and y == y1)
    • output (g(x, y));
  - Else      // Missing (x == x2 and y == y1) f(x, y, z) - g(x, y);
    • output (f(x, y, z) + g(x, y))
• end
• Expected: x = x1 and y = y1 => f(x, y, z) - g(x, y);
  x = x2, y = y2 => f(x, y, z) + g(x, y)
```

At the bottom left, there are logos for IIT Kharagpur and NPTEL. The NPTEL logo includes the text 'NPTEL ONLINE CERTIFICATION COURSES 161'. On the right side, there is a small video window showing a man in a yellow shirt speaking.

Now, let's try to gain the insight into why this occurs. That we have pair wise i.e., single mode and double mode faults are the majority of the faults. Let's look at this code here a simple code here and we have some 3 inputs x, y, z and then we have caused we have conditions defined on this input.

If $x = x_1$ so, we are reading 3 values x, y, z these are our input and if x is x_1 and y is y_2 then output is some $f(x, y, z)$. If x is x_2 and y is y_1 then output is $g(x, y)$, but then the programmer has missed one of the condition here. If x is x_2 and y is y_1 then the output should be $f(x, y, z) - g(x, y)$, but then he has written here else output is $f(x, y, z), g(x, y)$. So, this will work fine as long as we give $x = x_1, y = y_2$ etc., all possible combinations it will work, but it will fail only when x is x_2 and y is y_1 . So, a specific combination between x and y, it will fail and that is here.

Instead of displaying $f(x, y, z) - g(x, y)$ it will come here under the else and it will display $f(x, y, z), g(x, y)$. And this is a typical code that appears in many programs. And therefore, the expressions are formed using pair or 3 combinations of conditions. And if there is a problem under that then if specific values are given such that this becomes true, then only the condition in the failure will appear.

So, in this case when x is x_2 , when x is x_2 and y is y_1 , $f(x, y, z) - g(x, y)$ should have been updated, but it actually displays $f(x, y, z) + g(x, y)$, that is the problem and unless this x is given x_2 and y is given y_1 the problem will not be detected.

(Refer Slide Time: 09:30)

HARDKEYBOARDHIDDEN_NO	ORIENTATION_LANDSCAPE
HARDKEYBOARDHIDDEN_UNDEFINED	ORIENTATION_PORTRAIT
HARDKEYBOARDHIDDEN_YES	ORIENTATION_SQUARE
	ORIENTATION_UNDEFINED
KEYBOARDHIDDEN_NO	SCREENLAYOUT_LONG_MASK
KEYBOARDHIDDEN_UNDEFINED	SCREENLAYOUT_LONG_NO
KEYBOARDHIDDEN_YES	SCREENLAYOUT_LONG_UNDEFINED
KEYBOARD_12KEY	SCREENLAYOUT_LONG_YES
KEYBOARD_NOKEYS	SCREENLAYOUT_SIZE_LARGE
KEYBOARD_QWERTY	SCREENLAYOUT_SIZE_MASK
KEYBOARD_UNDEFINED	SCREENLAYOUT_SIZE_NORMAL
NAVIGATIONHIDDEN_NO	SCREENLAYOUT_SIZE_SMALL
NAVIGATIONHIDDEN_UNDEFINED	SCREENLAYOUT_SIZE_UNDEFINED
NAVIGATIONHIDDEN_YES	TOUCHSCREEN_FINGER
NAVIGATION_DPAD	TOUCHSCREEN_NOTOUCH
NAVIGATION_NONAV	TOUCHSCREEN_STYLUS
NAVIGATION_TRACKBALL	TOUCHSCREEN_UNDEFINED
NAVIGATION_UNDEFINED	
NAVIGATION_WHEEL	

Example: Android smart phone testing

- Apps should work on all combinations of platform options, but there are $3 \times 3 \times 4 \times 3 \times 5 \times 4 \times 4 \times 5 \times 4 = 172,800$ configurations

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES 162 | NPTEL

This occurs in many examples; let's consider android based smart phone testing.

The android has many environmental variables, global variables. The global variables are whether the hard keyboard, hidden no, undefined yes keyboard, keyboard is 12 key, no key, qwert, navigation, navigation touch screen, whether it is a finger, no touch screen it is a stylus or undefined screen layout small, large etc. So, just see here that the number of options required to do an exhaustive testing is if we see that this is 3 option, 3 option, and so on.

To do an exhaustive testing we need 172800 test cases, which is a large number, but if we consider pairwise or 3 way testing or 4 way testing, then the number can be drastically reduced to a manageable number. And we would have also detected almost every bug and that is why the t-way testing is a very attractive way, it makes combinatorial testing techniques practical.

(Refer Slide Time: 10:58)

The image shows a presentation slide with a yellow header containing the title 'White-Box Testing' in black font. Below the title is a small cursor icon. At the bottom of the slide, there is a red footer bar. On the left side of the footer, there is a logo for IIT Kharagpur and text 'IIT KHARAGPUR/08/10'. In the center of the footer, there is a logo for NPTEL and text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a man in a yellow shirt speaking.

Now, let's discuss about white box testing. So, far we have been discussing black box testing and we looked at several test strategies of black box testing. Now, let's look at white-box testing.

(Refer Slide Time: 11:21)

The image shows a presentation slide with a yellow header containing the title 'What is White-box Testing?' in black font. Below the title is a red footer bar. On the left side of the footer, there is a logo for IIT Kharagpur and text 'IIT KHARAGPUR'. In the center of the footer, there is a logo for NPTEL and text 'NPTEL ONLINE CERTIFICATION COURSES'. The main content area of the slide contains a bulleted list: '• White-box test cases designed based on:' followed by two points: '-Code structure of program.' and '-White-box testing is also called structural testing.'

White-box testing as we mentioned earlier, here the test cases are designed based on the code structure of the program. We must have the code available to us, we examine the code and based on the code we designed this test cases, the white box test cases. And

therefore, it is also called as the structural testing, because the test cases are designed based on the code structure.

(Refer Slide Time: 11:52)

The slide has a yellow background. At the top, the title 'White-Box Testing Strategies' is centered. Below the title, there are two main bullet points: 'Coverage-based:' and 'Fault-based:'. Under 'Coverage-based:', there is one bullet point: '– Design test cases to cover certain program elements.' Under 'Fault-based:', there is one bullet point: '– Design test cases to expose some category of faults'. At the bottom of the slide, there is a footer bar with three sections: IIT Kharagpur logo, NPTEL Online Certification Courses logo, and a video player showing a man in a yellow shirt.

Again, we have 2 major categories of white box testing; one is called as coverage-based testing, the other is fault-based. In coverage-based testing, we see the program elements, we observe the program elements and based on that, we design test cases such that the program elements are covered. The program element can be a statement, it can be some decisions that occur in the software or in the program, it can be some paths control flow in the program and so on. So, the program element is a generic term we have used and based on what we consider as program element, we will have different strategies of coverage based testing.

We can also have fault based testing here, we design test cases to expose some category of faults. We observe the program and if we find that there are arithmetic operators let's say + and so on. By using fault based testing, we can check whether the there are any problems with respect to the + operator becoming some other operator are those getting tested. So, in coverage based testing we will try to design test cases to cover certain program element and program element can be a statement, it can be a decision, can be a control flow and so on.

Whereas, in fault-based testing we observe the code and identify what are the ways in which faults can occur? The faults may get introduced, because an arithmetic operation was not done correctly, it may be because a variable type was not correct, maybe that some statement was missed; so the faults here at the typical faults that a programmer makes writing the program. And here in fault based testing we check whether the test cases are able to expose these specific types of faults that the programmer might commit.

(Refer Slide Time: 14:46)

The slide has a yellow background. At the top right, there is a yellow box containing the text "White-Box Testing". To the left of the box, there is a bulleted list of testing strategies. A red vertical line with a small red arrow points from the bottom of the list towards the word "Mutation". Another red vertical line with a small red arrow points from the bottom of the list towards the word "Data".

- Several white-box testing strategies have become very popular :
 - Statement coverage
 - Branch coverage
 - Path coverage
 - Condition coverage
 - MC/DC coverage
 - Mutation testing
 - Data flow-based testing

At the bottom of the slide, there is a blue footer bar. On the left side of the bar, there are logos for IIT Kharagpur and NPTEL Online Certification Courses. On the right side of the bar, there is a small video frame showing a man in a yellow shirt.

Now, first let's look at the coverage based testing. We will discuss statement coverage, branch coverage, path coverage, condition coverage, MC/DC coverage and also data flow coverage. So, these are the different coverage based testing techniques that we will discuss. We will also discuss a fault based testing which is called as the mutation test.

(Refer Slide Time: 15:35)

Why Both BB and WB Testing?	
Black-box	White-box
<ul style="list-style-type: none">• Impossible to write a test case for every possible set of inputs and outputs• Some code parts may not be reachable• Does not tell if extra functionality has been implemented.	<ul style="list-style-type: none">• Does not address the question of whether a program matches the specification• Does not tell if all functionalities have been implemented• Does not uncover any missing program logic
 IIT KHARAGPUR	 NPTEL ONLINE CERTIFICATION COURSES NPTEL
	

But, before we discuss about the white box testing, the coverage based testing techniques, a fault base testing and so on. Let's be clear about one issue, that if we are doing black box testing, do we need to do white box testing or if we are doing white box testing is it necessary to do black box testing.

If we want to answer that, we need both this testing. Then we should be able to give an example, that what cannot be detected by black box testing will be detected by white box testing. And also an example of what cannot be detected by white box testing, but will be detected by black box testing. That will justify our claim that we need to do both the testing; if, we cannot even give one example of a bug, which can only be detected by white box testing not by black box testing.

And similarly some category of bugs, which can only be detected by the black box testing and not the white box testing, then we would have somehow justified that we need both. But, if we can give no example that, which can be detected by only black box or only white box then we will not be justifying why we need both testing? We might as well do just one case testing.

So, based on that motivation, let us discuss the difference between black box and white box and why we need both? One of the main shortcoming of the black box is that that we cannot do exhaustive testing, black box testing. And also even if we do a thorough

testing based on the techniques that we discussed, we cannot guarantee that all code parts have been executed. It may be possible that the programmer who has written the code, he has written some extra code which is a Trojan.

Only the programmer knows, we tested according to the test cases designed based on the input output behavior that is specified, but the Trojan will not get exposed. Because the programmer has written a secret key, only when that key is pressed the Trojan gets activated. It is very difficult to discover Trojans by only black box testing, we need to examine the code and see whether all parts of the code are thoroughly being tested.

So, this is the justification, why we need to do white box testing? Because, if the programmer has written some extra functionality which is not there in the specification, we cannot detect that using black box testing, because it is not listed in the specification and we cannot design test cases for that. But, let's see why we need to do black box testing. One problem with white box testing is that we have been given the code and we design test cases exactly based on the code, but then just by examining the code we design test cases, but we can find whether all the code is executed and so on. But, we cannot tell if all functionality has been implemented, this is there in the requirement specification.

If the programmer has forgotten to write the code for some program logic, we cannot detect that using only white box testing, because the code itself is absent, unless we check the black box specification and see design test cases for every function, we cannot check whether some functionality has been omitted by the programmer.

So, that gives us the justification, why we need both black box and white box testing? Using just black box testing, we cannot know whether Trojans have been implemented or the programmer has added some extra functionality. Whereas, using white box testing we cannot know whether there is a missing functionality, the programmer has forgotten to write code for some specific functionality.

(Refer Slide Time: 21:14)

Coverage-Based Testing Versus Fault-Based Testing

- Idea behind coverage-based testing:
 - Design test cases so that certain program elements are executed (or covered).
 - Example: statement coverage, path coverage, etc.
- Idea behind fault-based testing:
 - Design test cases that focus on discovering certain types of faults.
 - Example: Mutation testing.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we said that the white box testing mainly the coverage based testing techniques and the fault based testing techniques. In the coverage based testing techniques, we said that when we check whether the designed test cases, execute some program elements. For example, whether all the statements are getting executed, we call it as statement coverage. All the paths in the program are executed that, we call as path coverage. So, here we cover or execute all the program elements that we consider.

In fault based testing, we focus on discovering certain types of faults. That is whether, if the programmer has done a mistake, in writing the logical expressions, are the test cases thorough enough to design all errors in logical expressions and so on. One prominent example of fault based testing is the mutation testing.

(Refer Slide Time: 22:26)

- **Statement:** each statement executed at least once
- **Branch:** each branch traversed (and every entry point taken) at least once
- **Condition:** each condition True at least once and False at least once
- **Multiple Condition:** All combination of Condition covered
- **Path:**
- **Dependency:**

Types of program element Coverage

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

And we said that in the coverage based testing the type of coverage is defined based on the type of the program element. And we will consider statement based coverage, branch coverage, condition coverage, multiple condition coverage, path coverage, and dependency coverage.

Thank you.

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

Lecture – 55
White box Testing

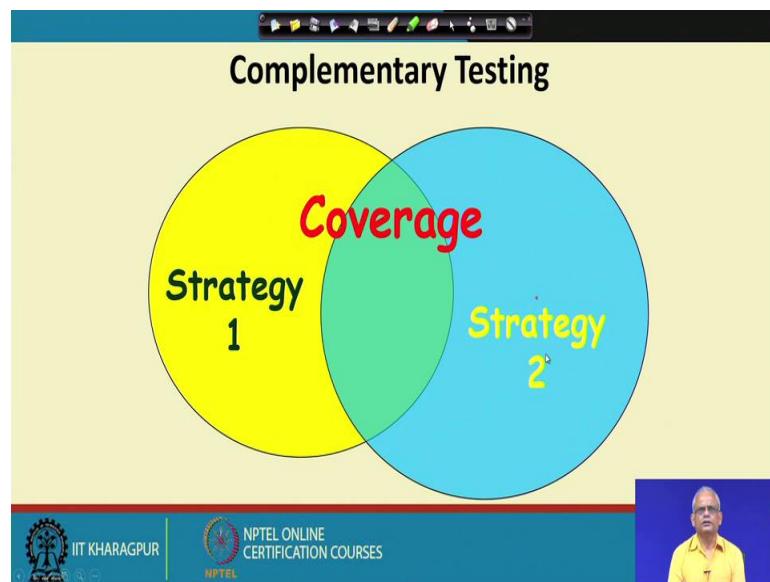
We had discussed that there are many white box testing techniques, but we need to discuss one important concept, that whether we need to test all of these different testing techniques. Whether, we need to do testing using all this half a dozen or a dozen testing techniques. For that we must understand the concept of a stronger testing and a weaker testing.

(Refer Slide Time: 00:46)



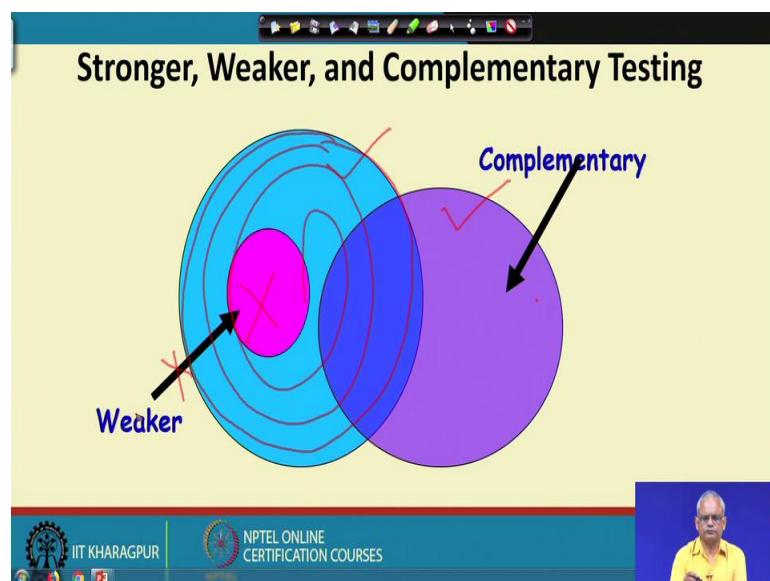
The idea here is that different testing techniques cover certain program elements. A stronger testing covers all the elements that have been covered by a weaker testing. That means, if you are doing a stronger testing, all the program elements that have been executed by a stronger testing includes the weaker testing. In other words, if you are doing a stronger testing, weaker testing is not necessary. We need not again design test cases for weaker testing. As long as we do a stronger testing, weaker testing need not be done as it is ensured automatically by a stronger testing.

(Refer Slide Time: 01:52)



It may be possible that two white box test strategies are complementary i.e., they execute some program elements which are common. Test strategy 1 executes elements some of which are overlapping with strategy 2. If 2 test strategies are complementary, then we need to conduct testing using both strategies.

(Refer Slide Time: 02:30)



If we represent the stronger, weaker, and complementary, we can see that the test strategy, which is covered by the green all the elements here by the green strategy. They cover the weaker strategy which is given in pink. Therefore, as long as we are doing the

testing for the blue coverage we need not do the weaker testing. But see here this is a complementary test strategy, which execute some elements here for the other strategy, but it also executes additional elements. Therefore, we need to do testing with both these blue and green strategies; we only can eliminate the weaker strategy.

(Refer Slide Time: 03:34)

Statement Coverage

- Statement coverage strategy:
 - Design test cases so that every statement in the program is executed at least once.

Now, based on the basic concepts that we discussed, now let's look at the different white box testing strategies. The simplest strategy is statement coverage. The idea here is that we need to execute every statement in the program at least once. We just look at the source code and design test cases such that every statement is executed at least once.

(Refer Slide Time: 04:12)

Statement Coverage

- The principal idea:
 - Unless a statement is executed,
 - We have no way of knowing if an error exists in that statement.

The principal idea behind this technique is that, if some statements are not executed we never know if there is a bug in that. That is a reason why we want every statement to be executed by the test cases. Such that, if there is a bug in the test case, it will probably get expressed when we execute that test cases.

(Refer Slide Time: 04:43)

Statement Coverage Criterion

- However, observing that a statement behaves properly for one input value:
–No guarantee that it will behave correctly for all input values!

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

A small video window in the bottom right corner shows a man in a yellow shirt speaking.

But, we must also understand that the statement coverage criterion has deficiency, because just by executing a test case once is no guarantee that all the bugs have been expressed, but then it is better than not executing all statements. So, we must ensure that all statements are executed at least once, but what we are trying to say here that just statement coverage may not be enough, we might achieve statement coverage, but it may not guarantee that good number of bugs, majority of the bugs have been exposed. Because just by executing the statement once may not be exposing the bugs present in that statement.

(Refer Slide Time: 05:43)

Statement Coverage

- Coverage measurement:

executed statements
statements

$$\frac{100}{500}$$

20%

Rationale: a fault in a statement can only be revealed by executing the faulty statement

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's see how the statement coverage is computed. Here, we count the total number of statements in the program and we track how many statements are executed by the test cases. So, the percentage of executed statements by the total number of statements, that gives us what is the coverage achieved. If we have 500 is the total number of statements and we have executed 100 statements by the test cases, we have achieved only 20 percent statement coverage. It is a very simple strategy, but it is one of the basic strategies and other strategies that we will see, they will achieve a more stronger testing than statement coverage. Let's discuss those techniques.

(Refer Slide Time: 06:55)

Example

- int f1(int x, int y){
- 1 while (x != y){
- 2 if (x>y) then
- 3 x=x-y; **Euclid's GCD Algorithm**
- 4 else y=y-x;
- 5 }
- 6 return x; }

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let's take an example; we have this simple code as shown in the above picture, while $x \neq y$, if $x > y$ then $x = x - y$ else $y = y - x$. If you remember, this is the Euler's GCD computation code. We want to see, what are the input values for which statement coverage will be achieved. There is a decision here, only if $x > y$, then x this statement will get executed. If $x \neq y$, statement 2 in the code will get executed. If we test it with $x = y$, statement 2 through statement 6 will not be executed. So, first need to check whether $x \neq y$ and then we must have $x > y$ and $x < y$.

So, this is the Euclid's GCD Algorithm and we know the specific values for which statement coverage will be achieved, but then given a large enough program, it is very hard to identify what are the specific values for which the statement coverage is achieved. Fortunately for white box testing, we do not really design test cases by looking at the specific values and see what will execute which one and so on. We have software to measure statement coverage. We just keep on giving values to this test cases to the software. Then the coverage tool will tell how much cover is achieved. We keep on testing until we achieve sufficient coverage.

(Refer Slide Time: 09:41)

The screenshot shows a presentation slide with a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the title 'Euclid's GCD Algorithm' is centered in a bold black font. The main content of the slide is a bulleted list:

- By choosing the test set $\{(x=3,y=3), (x=4,y=3), (x=3,y=4)\}$
 - All statements are executed at least once.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and the NPTEL logo. To the right of the footer, there is a small video window showing a man in a yellow shirt speaking.

Now, for this specific example, the Euclid's GCD program, if we choose these are the different values as given in the figure above, then statement coverage achieved and as I was saying that in reality for large programs, we do not have to identify the values for which statement coverage is achieved, but to understand the concept of the statement

coverage maybe for a small program. We might have to identify value such that statement coverage achieved or given some values or test cases, we should be able to tell that whether statement coverage will be achieved.

(Refer Slide Time: 10:22)

Branch Coverage

- Also called decision coverage.
- Test cases are designed such that:
 - Each branch condition
 - Assumes true as well as false value.

Handwritten notes on the right side of the slide:

- 'for(D)'
- 'if(c)' with an arrow pointing to it
- 'else' with an arrow pointing to it
- 'while(s)' with an arrow pointing to it

IT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

The next white box strategy that we will discuss is the branch coverage. Another name for branch coverage is decision coverage. The main idea here is that, in very program there are many decision statements, the decision statements can be of the form, if some condition is satisfied, execute some statement else, execute some other statement or it may be while some condition is satisfied, keep doing something or it may be for and so on. These are examples of decision statements. Here the idea in branch or decision coverage is that every decision statement must be taking true and false values. So, it should at least the test cases ensure that it goes inside the loop and also it exits the loop.

Similarly, if both this take place i.e., the condition is true and false. And similarly for the for loop, it should enter into the loop and also for some test input it should not enter into the loop. So, every condition in a conditional statement, every branch condition must take true and false values that is our test cases should ensure that. Now, if we look at the same function here, the Euclid's GCD algorithm, we find that the decision statements are at statement 2 and statement 4. There are 2 decision statements and our test cases must make statement 2 once true, make statement 4 once false. And similarly this should be true and false and for very small programs we can design the branch coverage test cases.

But, in normal practical situation we do not have to design the branch coverage test cases, we just keep on giving input and coverage tool will tell us, what is the extent of branch coverage that is achieved. And we keep on giving data until you get sufficient branch coverage.

(Refer Slide Time: 13:18)

The slide has a yellow background. At the top, the word 'Example' is centered in a bold black font. Below it, there is a bulleted list of test cases:

- Test cases for branch coverage can be:
- $\{(x=3,y=3), (x=3,y=2), (x=4,y=3), (x=3,y=4)\}$

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a person speaking.

So, for that simple program Euclid's GCD, we can design some test cases which will achieve the branch coverage.

(Refer Slide Time: 13:33)

The slide has a yellow background. At the top, the title 'Branch Testing' is centered in a bold black font. Below it, there is a bulleted list:

- **Adequacy criterion:** Each branch (edge in the CFG) must be executed at least once
- Coverage:
$$\frac{\text{# executed branches}}{\text{# branches}}$$

Handwritten notes are present on the right side of the slide:

- n decision points
- 2^n
- Write (with a circled 5)
- 3

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a person speaking.

But, how do we measure coverage? If we are executing the program using test cases, how does the coverage tool that will report the extent of coverage achieved?

(Refer Slide Time: 13:52)

Quiz 1: Branch and Statement Coverage:
Which is Stronger?

- Branch testing guarantees statement coverage:

—A stronger testing compared to the statement coverage-based testing.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

A small video player window shows a man in a yellow shirt speaking.

It will find out all the possible branches that are there on the code, number of branches and how many branches have been executed. If it is a while condition then there are 2 branches; one branch is that it is true, it enters into the loop and for false, it exits. Similarly, every conditional statement must achieve true and false values, during the execution of the test cases.

So, each of this will be computed twice here, if we have n decision statements, then the number of branches will be 2^n . Then the coverage tool can find out how many of these branches, true and false are taken and then it will report the extent of branch coverage achieved.

But, then one thing we must be clear is that, which is a stronger test, branch coverage or statement coverage? Because, if we can say that which is stronger testing, we need not do the other testing, but then if we say that something is stronger, we must be able to show that it is stronger testing. Now, here between branch coverage and statement coverage, we can say that branch coverage is stronger than statement coverage, because every statement must be there on some branch. So, that is our argument here that if there is a statement in a program it must be there on some branch. And therefore, if we are

covering the branches, then all statements must have been covered. So, branch coverage ensures statement coverage.

But, then the question comes that is it possible that branch coverage ensures statement coverage, but is it possible that branch coverage guarantees statement coverage that we could show now, but we have to also show that there are some branches which are not ensured by statement coverage. Otherwise they will become identical, to show that it is stronger, we have to show not only that branch coverage ensures statement coverage, but we have to also show that statement coverage does not ensure branch coverage. Or in other words, there are some branches, which may not get executed even though we achieve statement coverage, how do we show that?

One way we could show that, branch coverage guarantees statement coverage because every statement must lie on some branch.

(Refer Slide Time: 17:47)

Stronger Testing

- Stronger testing:
 - Superset of weaker testing
 - A stronger testing covers all the elements covered by a weaker testing.
 - Covers some additional elements not covered by weaker testing

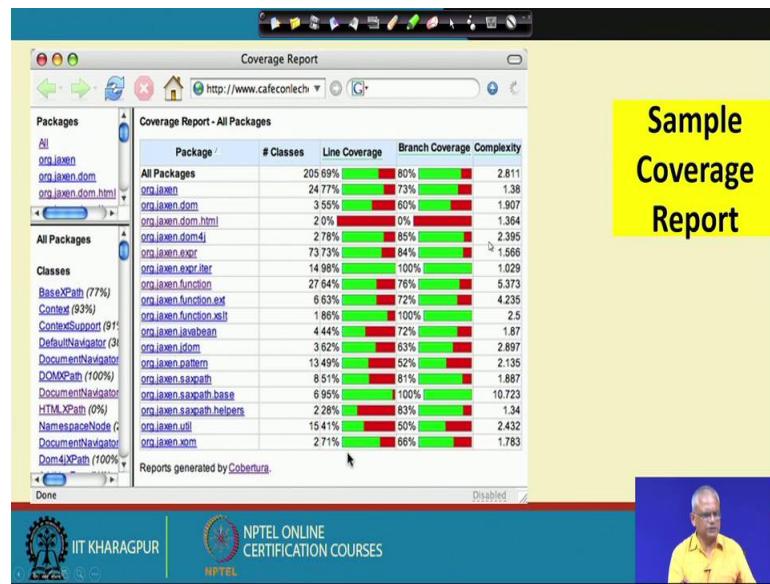
if(c) a=b

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

A small video window in the bottom right corner shows a professor speaking.

So, a stronger testing as we said is a superset of weaker testing, if we are doing stronger testing, we need not do the weaker testing. If we are doing the branch testing, we need not do the statement coverage testing, but then so far we have showed that state branch coverage ensures statement coverage, but we have to also show that statement coverage does not achieve branch coverage.

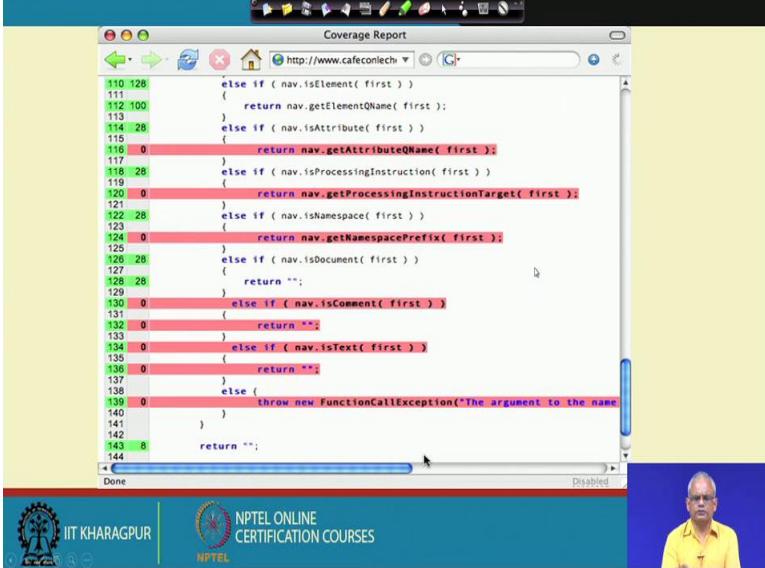
(Refer Slide Time: 18:21)



So let me just give one example here that will ensure that statement coverage does not achieve branch coverage. Let say we have a statement if c_1 , then $a = b$. Now, let us say that c_1 is true, then we achieve the statement coverage $c_1 = \text{true}$ achieves statement coverage, but it does not achieve branch coverage because c_1 is false has not been ensured. So, just achieving statement coverage and this code is not ensuring branch coverage and the code. So, this a simple example, which says that branch coverage does not ensure statement coverage.

There are many coverage tools and coverage tool as the test case are executed, it displays coverage report. In the coverage report it displays that, what are the statement coverage, branch coverage, etc. achieved for different functions.

(Refer Slide Time: 20:14)

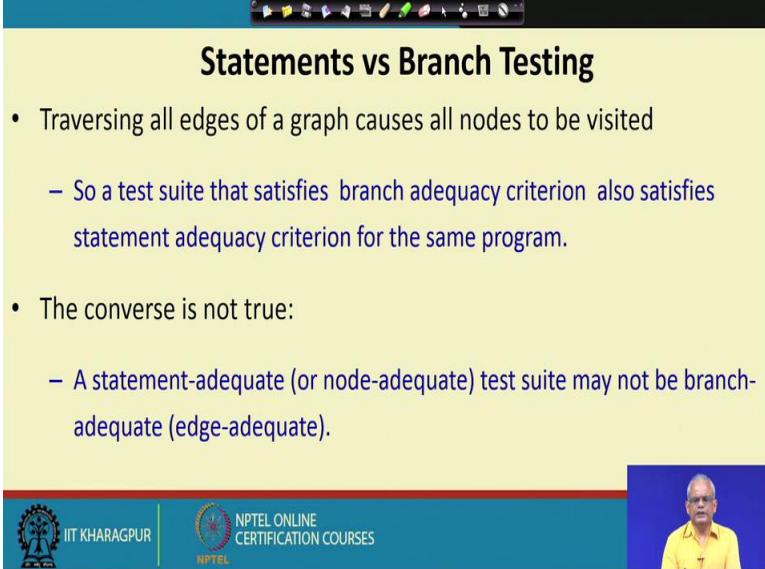


The screenshot shows a 'Coverage Report' window with a Java code editor. The code is a switch statement with many branches. Lines 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, and 140 are highlighted in red, indicating they have not been executed. Lines 110, 112, 114, 116, 118, 120, 122, 124, 126, 128, 130, 132, 134, 136, 138, 140, 142, and 144 are highlighted in green, indicating they have been executed. The code is as follows:

```
110 128     else if ( nav.isElement( first ) )
111         {
112             100             return nav.getElementQName( first );
113         }
114 28     else if ( nav.isAttribute( first ) )
115         {
116             0                 return nav.getAttributeQName( first );
117         }
118 28     else if ( nav.isProcessingInstruction( first ) )
119         {
120             0                 return nav.getProcessingInstructionTarget( first );
121         }
122 28     else if ( nav.isNamespace( first ) )
123         {
124             0                 return nav.getNamespacePrefix( first );
125         }
126 28     else if ( nav.isDocument( first ) )
127         {
128             0                 return "";
129         }
130 0     else if ( nav.isComment( first ) )
131         {
132             0                 return "";
133         }
134 0     else if ( nav.isText( first ) )
135         {
136             0                 return "";
137         }
138 0     else
139         {
140             throw new FunctionCallException("The argument to the name
141         }
142         }
143 8     return "";
144 }
```

And also, many tools that display what are the statements that are still not got executed. So, these are shown in the red and we know that statement coverage has not been achieved and these are the statements that have not been executed so far.

(Refer Slide Time: 20:38)



Statements vs Branch Testing

- Traversing all edges of a graph causes all nodes to be visited
 - So a test suite that satisfies branch adequacy criterion also satisfies statement adequacy criterion for the same program.
- The converse is not true:
 - A statement-adequate (or node-adequate) test suite may not be branch-adequate (edge-adequate).

So, we have seen that branch coverage is stronger testing, a branch coverage achieves statement coverage, but the converse is not true. The statement coverage, if we achieve statement coverage that does not mean that branch coverage has been achieved.

(Refer Slide Time: 21:01)

All Branches can still miss testing specific conditions

- Sample fault: missing operator (negation)
- Branch adequacy criterion can be satisfied by varying only digit_low
 - The faulty sub-expression might not be tested!
 - Even though we test both outcomes of the branch

Now, so far we have looked at the statement coverage and branch coverage, but is a branch coverage is a good enough testing, can it miss some bugs i.e., we achieve branch coverage, but then some types of bugs are not exposed. And this is what we are going to discuss now. That we have a certain branch condition, if digit is high, if digit high is 1 or digit low is -1 then do some action A.

Now, the problem here is that we achieve. If digit high = 1 or digit low = -1, then the action is A else action is A1 and so on. Now, let's say we achieve branch coverage here, that we have first branch condition is true. So, A is executed. If branch condition is false, then A1 is executed, but then just observe here that to get this true, we might have digit high = 1, that will make it true irrespective of digit low = -1. And let's say we have digit high = 0 and digit low != -1 and then this becomes false. So, we have achieved both true and false for this specific branch.

But, then let's say that we have some action here, which will encounter a failure only when digit low = -1. In that case we will not be able to discover that bug by testing. And therefore, not only we must ensure that the decision is both true and false, we must ensure that all component conditions here are clauses are true and false. So, branch coverage even though it is a strong testing, but then it may not achieve, it may not expose many types of bugs and we might have to do a stronger testing and a stronger testing is component clause and the decision statement must achieve both true and false values.

(Refer Slide Time: 24:24)

The slide has a yellow background. At the top center, it says 'Basic Condition Coverage'. Below that is a bulleted list:

- Also called **condition coverage**.
- Test case design:
- Each component of a composite conditional expression
- Made to assumes both true and false values.

Handwritten notes in red ink are present: 'if(a||b)' with 'T' under 'a' and 'F' under 'b'; and 'T' under 'a' and 'F' under 'b' again.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

And that brings us to the basic condition coverage; in the basic condition coverage each component condition must achieve true and false values. So, if we have a or b, then a must be true and false, the test cases must ensure that a test true and false values be also true and false values.

In the basic condition coverage, we must ensure that the component clauses here on the decision statement take true and false values. So, each component of a composite conditional expression must take true and false values.

(Refer Slide Time: 25:35)

The slide has a yellow background. At the top center, it says 'Basic Condition Testing'. Below that is a bulleted list:

- **Simple or (basic) Condition Testing:**
 - Test cases make each atomic condition to have both T and F values
 - Example: **if (a>10 && b<50)**
 T
 F
 F
 - The following test inputs would achieve basic condition coverage
 - a=15, b=30
 - a=5, b=60
- Does basic condition coverage subsume decision coverage?

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

Let's consider the example; if $a > 10$ and $b < 50$. Now, if $a = 15$ and $b = 30$. So, $a > 10$ becomes true and $b < 50$ becomes true. Now, for the other test case $a = 5$, $b = 60$, $a > 10$ is false, and $b < 50$ is false.

So, these two test cases ensure that both the component clauses, ensure achieve the true and false values. If there is any expression including dozens of clauses, it is possible that only 2 test cases may be able to achieve basic condition coverage. Because one test case may give all true and another test case may assign false to every condition.

Therefore, it is possible that in some large decision statement, consisting a many clauses just 2 test cases may be able to achieve basic condition testing that gives us a hint that this may not be a very strong coverage criterion, but then does it ensure if we achieve basic condition testing, does it ensure branch coverage? We need to answer that question is the basic condition testing, a stronger testing than the decision coverage or branch coverage, or is it a weaker testing or is it a complementary testing. Now, we are almost at the end of the lecture, we will discuss this point and other white box testing strategies in the next lecture.

Thank you.

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

Lecture – 56
Condition Testing

Welcome to this lecture. In the last couple of lectures, we have been discussing about the white box testing. We had said that white box testing has to be carried out on a unit, even though we are also performing black box testing. There are several types of white box tests strategies. We had discussed about statement coverage based testing and then the branch coverage based testing.

If you remember, we said that branch coverage is a stronger form of testing, if we are doing branch coverage testing, then we need not do the statement coverage testing. But is branch coverage testing good enough or in other words are there stronger testing strategies, which we need to carry out and if we carry out those branch or decision testing becomes not necessary. Let's see the other types of testing.

(Refer Slide Time: 01:34)

The slide is titled "White-box Testing". It contains a bulleted list of coverage types:

- Statement coverage
- Branch coverage (aka decision coverage)
- Basic condition coverage
- Condition/Decision coverage
- Multiple condition coverage
- MC/DC coverage
- Path coverage
- Data flow-based testing
- Mutation testing

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and a video player showing a person speaking.

In the last few lectures discussed about statement coverage based testing and then we had discussed about branch coverage testing, which is also popularly known as the decision coverage testing. Now, we will look at the basic condition coverage testing. We will later after basic coverage condition testing, look at the condition decision coverage based

testing. Then we will look at the multiple condition coverage testing, MC/DC coverage testing, path coverage testing, and then the data flow coverage testing, and finally, the mutation testing.

(Refer Slide Time: 02:27)

All Branches can still miss testing specific conditions

- Assume failure occurs when $c==\text{DIGIT}$
- Branch adequacy criterion can be satisfied by $c==\text{alphabet}$ and $c==\text{splchar}$
 - The faulty sub-expression might not be tested!
 - Even though we test both outcomes of the branch

First, let's address the question that is the branch coverage testing good enough or to tell that in other words if all branches have been tested can we miss some specific test conditions or can bugs remain even after we have completed branch coverage testing or decision coverage testing?

Let's take a simple example; let's assume that we have a conditional statement if c is equal to alphabet or c is equal to digit, where c is a character variable and we have read c from the keyboard. And then we are checking here, if c is either an alphabet or c is a digit. We were doing some special processing for that. Maybe we are recovering the integer value of c . If it is a digit, then $c = c-a$. Let's say, which will retrieve the integer value.

But, then if we are doing the branch testing we need to have the branch condition assume true and false values, to make it true if we have c is an alphabet i.e., A to Z and a to z. Then, if either becomes true this condition does not matter because this is a OR condition anyone of that becoming true will make the branch condition true. Now, let's say that the second test case is a special character like a control character or something.

So, now the expression when we evaluate $c \neq$ alphabet and $c \neq$ digit, because it is a special character. Therefore, it becomes false, but the fault is when $c =$ digit, we are not doing it properly, let's say, we have made it $c = c - a$. So, we made a small mistake here, to get the integer value we made a small mistake, but then our 2 test cases, that is c is a alphabet and c is a special character achieves decision coverage. And the faulty sub expression, which is $c =$ digit that is not tested. And the bug remains even if we are achieving decision coverage testing both the outcomes of the branch condition, but still we are not able to detect the bug and that the reason for that, we are not considering this sub expression becoming true any time.

If we had a test case which not only turns the first sub expression true and false, we should also have test cases making, the second sub expression true and false and in that case we could have detected the bug. So, that gives us a hint that we should look at the sub expressions. We should ensure that the test case makes it sub expression assumed true and false values. And exactly based on that idea the basic condition coverage testing is designed. Let's look at the basic condition coverage testing.

(Refer Slide Time: 07:04)

Basic Condition Coverage

- Also called **condition coverage or simple condition coverage**.
- Test case design: $((c == \text{ALPHABET}) || (c == \text{DIGIT}))$

—Each component of a composite conditional expression

- Made to assume both true and false values.

Page 2/2

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In the basic condition coverage testing for a branch condition having multiple clauses, which we will call as a complex condition; each of the sub expressions or the clauses, we will require them to achieve true and false values. You can see that if we make that, then the bug which was earlier not getting caught will get caught, but then is it a good testing

First of all, is it stronger than the statement coverage, is this kind of testing the basic condition coverage testing, is it stronger than simple decision coverage testing? Let's try to answer those questions and is it a good testing strategy.

(Refer Slide Time: 08:11)

Basic Condition Testing

- Simple or (basic) Condition Testing:
 - Test cases make each atomic condition assume T and F values
 - Example: if (a>10 && b<50)
- Following test inputs would achieve basic condition coverage
 - a=15, b=30 ✓ ||
 - a=5, b=60 ✓ ||
- Does basic condition coverage subsume decision coverage?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

First let's see, how do we design the basic condition testing strategies. Let's take an example of conditional expression, if $a = 10$ and $b < 50$. And our objective is that each of the sub expressions will achieve true and false values. Now to make $a > 10$ to make it true, we will make a is 15. And to make $b < 50$, we will assign some value $b = 30$ or something like that. And therefore, our first test case is a is 15, b is 30. So, this is the test case1, which will make the 2 sub expressions true. Now, we need another test case which will make both the sub expressions false. So, let's choose $a = 5$, $5 < 10$. And therefore, this is false and let's say b is 60, $60 > 50$. And therefore, the second sub expression is also false. So, these two test cases will achieve the simple condition testing or the basic condition testing, where each sub expression is made into true and false values.

But, can we always perform basic condition testing? Yes, we can perform basic condition testing and if the 2 sub expressions are independent. If there is a dependency between these, we may or may not achieve basic condition testing. Let's say $a > 10$ and $a < 5$, that we cannot for that condition there is a dependency between two sub expressions and it becomes difficult to design the test case as it cannot be at the same time greater than 10 and a is less than 5.

But, if we have independent clauses, we should be able to design basic condition testing and how many test cases would be needed? In one test case, we can assign true to all and another test case, we can assign false to all. So, 2 test cases would be good enough to achieve basic condition testing, but how does this testing. Compare with the branch coverage and the statement coverage testing; let's try to examine that.

(Refer Slide Time: 11:49)

Example: BCC

- Consider the conditional expression
 $\neg((c1.\text{and}.c2).\text{or}.c3):$
- Each of $c1$, $c2$, and $c3$ is exercised with all possible values,
 - That is, given true and false values.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL

There is another example; before we check how effective it is, there is one more example let's say we have $c1$ and $c2$ or $c3$. So, there are 3 clauses here and 3 sub expressions or clauses. And we will make each one of this true and false. Even though there maybe 3, 4 5 whatever, if they are independent then we can design test cases such that all are true and all are false. So, each sub expression gets true and false values. So, basic condition coverage is satisfied, but we just use 2 test cases.

(Refer Slide Time: 12:48)

Basic condition testing

- Adequacy criterion: each basic condition must be executed at least once
- Coverage:

$$\frac{\text{# truth values taken by all basic conditions}}{2 * \text{# basic conditions}}$$

$\xrightarrow{T} ((a=10 || b>5) \& c < 100) \xrightarrow{T}$
 $\xrightarrow{T} T \quad F$
 $= \frac{4}{6 \cdot 3} = \frac{2}{3}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, if we use some test cases to test a branch condition, what is the percentage of coverage achieved? Let's define a metric, which we call as the BCC metric, Basic Condition Coverage metric. Here we have this simple expression, the number of truth values that is true and false taken by all basic conditions over $2 * \text{number of basic conditions}$.

So, if we have an expression like $a = 10$ or $b > 5$ and $c < 100$, we can see here that there are 3 atomic conditions or the basic conditions and then possible outcomes of each of these is 2^3 so that is 6. Now, let's say we designed a test case which made this true, false, false, the second test case made true, true, false. So, this is the first test case outcome this is the second test case, and using two test cases we achieved this coverage of the basic conditions.

And then what is the percentage coverage achieved. So, we achieved true, false, false and also true value for the second condition. So, we have achieved 4 of the decisions out of the 6. So, the coverage will be $2/3$.

(Refer Slide Time: 15:12)

Is BCC Stronger than Decision Coverage?

- Consider the conditional statement:
-If((a>5).and.(b<3)).or.(c==0)) a=10;
- Two test cases can achieve basic condition coverage: (a=10, b=2, c=2) and (a=1, b=10, c=0)
- BCC does not imply Decision coverage and vice versa

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, is it a strong enough testing, how does it compare with the decision coverage and statement coverage? Let's take an example, if $a > 5$ and $b < 3$ or $c = 0$ and then we have some action here, $a = 10$. And we can use 2 test cases to achieve basic condition coverage, let's say $a = 10$ if this is true, $b = 2$ is true and $c = 2$ is false.

So, the first test case achieves true, true and false. The second test case $a = 1$ which is false, $b = 10$ which is false, and $c = 0$ which is true. Now, we have achieved basic condition coverage using these 2 test cases. So, each of the sub expressions achieves both true and false values, but then let's see if decision coverage is obtained true and true. So, the whole clause will become true and irrespective of the first one, even though it is false. So, the branch outcome will be true. Now, let's see the second one false and false. So, the clause here will become false.

But, then the second sub expression becomes true. So, the outcome is again true. So, even though the 2 test cases are able to achieve basic condition coverage, but they are not able to achieve the decision coverage. From this we can conclude that even though basic condition coverage may be satisfied, but still decision coverage may not be obtained. Or in other words we cannot say that basic condition coverage is a stronger form of testing than decision testing.

But is decision coverage testing stronger than the basic condition coverage ? No, that is also not true, because you had already seen that, even though the decision coverage was

achieved for $c = \text{alphabet}$ or $c = \text{digit}$. We could achieve the decision coverage, but that did not achieve the basic condition coverage. We can therefore, say that the basic condition coverage and decision coverage are complimentary, they are not really comparable testings, one cannot really be stronger than the other testing.

(Refer Slide Time: 18:35)

Condition/Decision Coverage Testing

- Condition/decision coverage: Sc BCC BC
– Each atomic condition made to assume both T and F values
– Decisions are also made to get T and F values
- **Multiple condition coverage (MCC):** $\begin{cases} & \text{if } (a>5) \\ T & \parallel \\ F & \end{cases} \cdot \begin{cases} & \text{if } (b>20) \\ T & \parallel \\ F & \end{cases}$
– Atomic conditions made to assume all possible combinations of truth values

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's see the condition decision coverage testing. Let me just pose another question to you that the basic condition coverage testing and the branch coverage testing these 2 are not really comparable. We cannot say that any of that any of these 2 is stronger than the other, but what about the statement coverage testing? We know that branch coverage testing is stronger than statement coverage testing, but what about the basic condition coverage and the statement testing. Can we say that basic condition coverage is stronger than statement coverage testing or is it vice versa that statement coverage testing is stronger than basic coverage testing?

Of course, from common sense we can say that statement coverage testing cannot be stronger than basic condition coverage testing, because one which is stronger than the statement coverage that also is not stronger than basic condition testing.

But, what about this basic condition coverage, does it achieve statement coverage? Please try to reason it out, give examples or prove, what is the relation between basic condition coverage and statement coverage testing? But, our idea while discussing about the branch coverage testing was to come up with a test strategy, which is stronger than

branch coverage testing, but at the same time achieves condition coverage. And basic condition coverage could not do that, can we impose the condition that we should achieve the basic condition coverage and also the decision coverage.

So, that is the strategy that we will discuss now, it is called as condition decision coverage. The condition decision coverage strategy here, we achieve basic condition coverage and also the decision coverage. And here, each of the atomic condition is made to assume true and false values, that is the basic condition coverage achieved and also the decision as a whole is also made to achieve true and false values. So, we need test cases for those. Therefore, this is a stronger testing than the basic condition testing or the branch coverage testing because it achieves both of this.

But is it the strongest testing? We can see that this is stronger than both branch testing and the basic condition coverage, but is it the strongest? No the strongest is multiple condition coverage. Here all possible atomic conditions, the atomic conditions are made to assume all possible combinations of truth values. If we have, if $a > 5$ or $b < 20$, in that case our test cases should make all the possible combinations of these two conditions.

In general, if we have n sub expression or clauses in a complex decision statement, then we will need 2^n test cases to achieve the multiple condition coverage. So, the multiple condition coverage testing should ensure that all the sub expressions, they achieve all possible combinations of truth values.

(Refer Slide Time: 23:33)

MCC

- Test cases make Conditions to assume all possible combinations of truth values.
- Consider: **if (a || b && c) then ...**

Test	a	b	c
(1)	T	T	T
(2)	T	T	F
(3)	T	F	T
(4)	T	F	F
(5)	F	T	T
(6)	T	T	F
(7)	F	F	T
(8)	F	F	F

20 Test Cases

3 = 8

Exponential in the number of basic conditions

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

A video thumbnail of a professor speaking is visible in the bottom right corner.

Now, let's look at an example, if we have this example $a \parallel b \& c$. we have 3 sub expressions, 3 Boolean values and if we need to achieve the multiple condition coverage how many test cases do we need? We need 2^3 test cases, which is 8. Let's see what will be the test cases; a b c these are the 3 sub expressions or Boolean variables and then all possible combinations the truth outcomes of these 3 sub expressions. We can say that to achieve the multiple condition coverage the number of test cases is required is exponential in the number of the sub expressions or the basic atomic conditions.

But, then this can be huge because in many applications. Especially in the controller embedded control applications, in user interfaces etc. It is quite common to have expressions involving 10 or 20 sub expressions. If we have 20 sub expressions in an expression in a decision statement, we will need 2^{20} test cases or which is a million test case. Just imagine that just to test one branch we need a million test cases to achieve multiple condition coverage, but if a unit has 10 of that the number of test case will be enormous, even million is a large number a tester would have to spend couple of years to run the million test cases.

We can therefore, say that the multiple condition coverage is a strong form of testing when there are branches, but then it is impractical, because it requires too many test cases when we have multiple clauses or multiple sub expressions 10 or more or even 8, 7 this also lead to large number of test cases.

So, can we have a compromise a test strategy which will achieve as thorough testing as the multiple condition coverage, but then the number of test cases should be few for expression it should be 3, 4, 5 something like that, should not be 1000s. Now, we are almost at the end of the lecture. Let's stop here and we will continue from here in the next class.

Thank you.

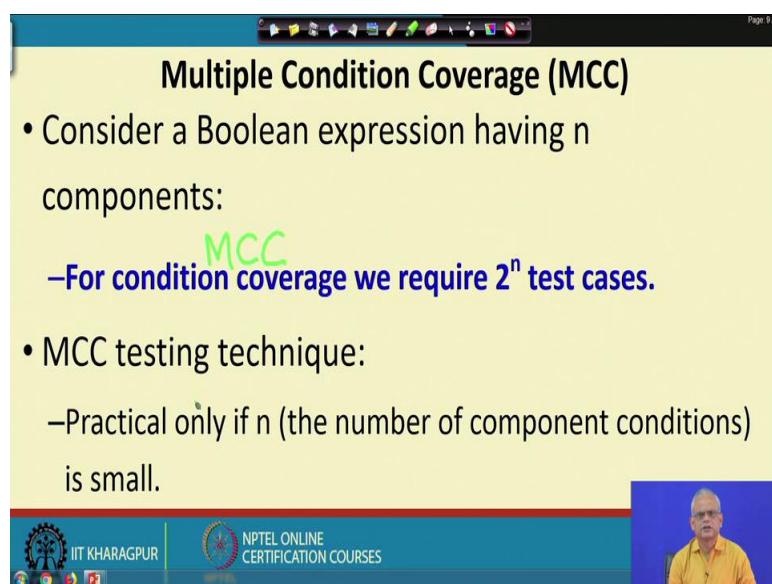
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 57
MC/DC Coverage

Welcome to this lecture. In the last lecture, we were discussing testing branches and we said that the simple branch decision coverage testing is not really a good enough testing. Bugs can remain, if the branch condition contains multiple sub expressions. Then we looked at the basic condition coverage testing and then we said that even though it overcome some short comings of the branch coverage testing, but it is not really a stronger testing than branch coverage. Therefore, we considered the condition decision coverage testing.

Where, the basic condition testing is achieved and the same time branch coverage is achieved but then we were trying to see, which is the strongest branch testing strategy and we said multiple condition coverage testing, where each of the sub expression is given all possible combinations of truth values, but then the number of test cases became exponential. Let's continue from that point.

(Refer Slide Time: 01:45)



Multiple Condition Coverage (MCC)

- Consider a Boolean expression having n components:

MCC

—For condition coverage we require 2^n test cases.

- MCC testing technique:
 - Practical only if n (the number of component conditions) is small.

Let's consider a Boolean expression having n sub expressions or components. The multiple condition coverage MCC, The MCC or the multiple condition coverage will require 2^n test cases. And that is the reason why MCC testing is not considered practical, if the numbers of sub expressions are more.

(Refer Slide Time: 02:37)

Test Case	a	b	c	d	e	$2^5=32$
(1)	T	-	T	-	T	
(2)	F	T	T	-	T	
(3)	T	-	F	T	T	
(4)	F	T	F	T	T	
(5)	F	F	-	T	T	
(6)	T	-	T	-	F	
(7)	F	T	T	-	F	
(8)	T	-	F	T	F	
(9)	F	T	F	T	F	
(10)	F	F	-	T	F	
(11)	T	-	F	F	-	
(12)	F	T	F	F	-	
(13)	F	F	-	F	-	

- Short-circuit evaluation often reduces number of test cases to a more manageable number, but not always...



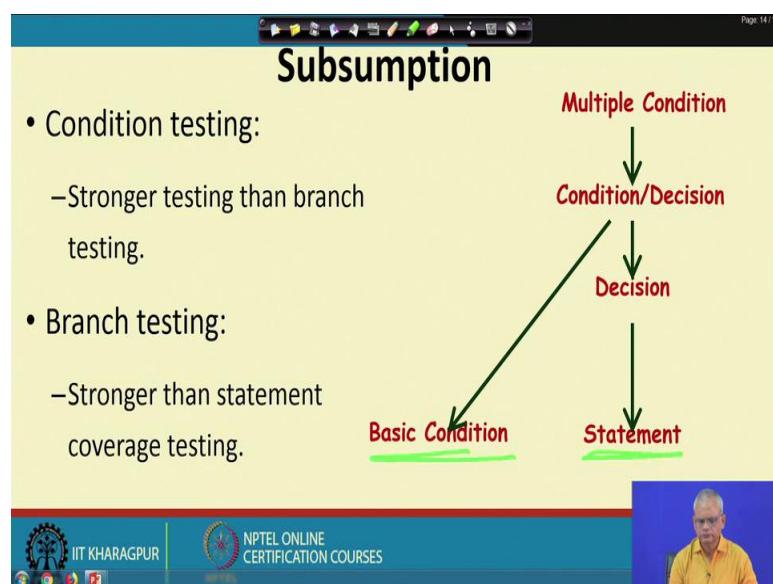

Even, if we have a 5 clauses here this example expression just 5 clauses. And to test for multiple condition coverage, we need 2^5 test cases and that is 32.

But, then we will see that short circuit evaluation reduces the number of test cases to a more manageable level. The short circuit evaluation means, that compiler while evaluating, it infers whether it needs to evaluate further. For example, let's say a is true and let's say c is true. Then the others the compiler does not really bother because they will have no effect if a is true then this expression is true. And if c is true then this is true and if e is true and then terms have to be true, d does not matter. So, b and c whether it is true and false it does not matter. So, you can just consider one of that. And therefore, the number of test cases reduces by 1, but what about if let say a is true and c is false. Then this becomes false irrespective of the value of the b, and let say d is false so that means, this is false. And therefore, e does not matter.

So, we can just consider one of the 2 possible test cases. And therefore, the number of test cases may not be 32 due to short circuit evaluation technique used by the compilers, because they want to execute fast and they see that if the truth value can be inferred as

they evaluate, then they do not evaluate the other ones. So, even though we are saying that 32 test cases are needed, but then depending on the compiler the short circuiting approach that, it takes we might need much less than 2^n to achieve the multiple conditions coverage testing.

(Refer Slide Time: 06:01)



So, based on our discussion so far we can say that the statement coverage is the weakest testing. And so is the basic condition coverage testing, but then they are not comparable because basic condition testing does not ensure statement coverage testing and vice versa, but then the branch coverage testing or the decision testing is stronger than the statement coverage testing.

And the condition decision coverage testing is stronger than both basic condition testing and the decision testing. And the multiple condition testing is the strongest of all this, but the only problem is that is not practical, when the number of sub expressions is large.

(Refer Slide Time: 07:04)

Shortcomings of Condition Testing

- **Redundancy of test cases:** Condition evaluation could be compiler-dependent:
 - Reason: Short circuit evaluation of conditions
- **Coverage may be Unachievable:** Possible dependencies among variables:
 - Example: $((\text{chr} == 'A') || (\text{chr} == 'E'))$ can not both be true at the same time

MCC

$\begin{array}{cc} T & T \\ F & F \end{array}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We had seen that the short circuit evaluation approach used by compilers often reduces the number of test cases to achieve multiple condition coverage. Also, we must remember that certain coverage may not be achievable. Let's say we have expression like character is A or character is E. Here, we cannot achieve the basic condition coverage also, because we cannot have, we can achieve basic condition coverage, because we can have A as true and E as false and E as true and, A as false as shown in the figure above.

We can achieve basic condition coverage testing, we can achieve decision coverage testing, but can we achieve condition decision coverage testing? That also you can achieve, but what about multiple condition coverage testing? Can we achieve both of these true at the same time? No, because the character cannot be same at the same time A and E. So, multiple condition coverage testing, we cannot achieve for this.

And, that we must keep in mind that we cannot achieve multiple condition coverage testing even though we deploy 1000s of test cases, we cannot achieve multiple condition coverage even for simple expressions.

(Refer Slide Time: 09:31)

Page 16 / 16

Short-circuit Evaluation

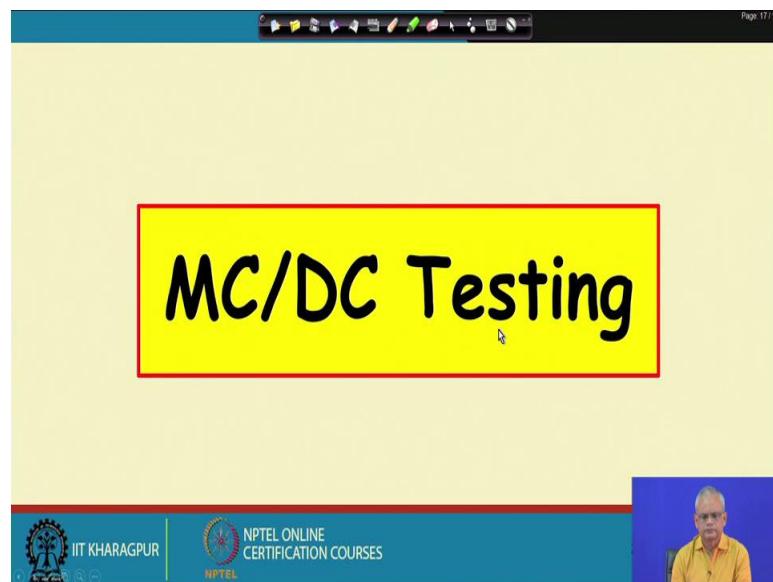
- $\text{if}(a>30 \&\& b<50)...$
F T
– If $a>30$ is FALSE compiler need not evaluate ($b<50$)
- Similarly, $\text{if}(a>30 \mid\mid b<50)...$ T
– If $a>30$ is TRUE compiler need not evaluate ($b<50$)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let's look at few more short circuit evaluations. In this expression, there is a `&&` here. And therefore, wherever there is a `&&`, if one of the sub expression evaluates to false, then the clause has to the decision has to evaluate to false irrespective of the other clauses.

The compilers take advantage of this and as soon as they find a false here in this kind of expression, they will not evaluate the others. They become don't cares and the outcome will be false. Whenever there is a `||` condition, whenever there is any sub expression evaluates to true, then the others become don't care and the decision evaluates to true. The decision will become true, even if one of that is true. And if the compiler deploys short circuit evaluation, it can save lot of time and skip evaluation of sub expressions.

(Refer Slide Time: 11:01)



Now, let's look at a test strategy, which is the strong testing for decisions. It overcomes the problem of the multiple condition coverage testing, which required large number of test cases. When we had a decision involving let say 10 sub expressions for multiple condition coverage might need 2^{10} test cases which is about a 1000 test case, but if we use the MC/DC testing that is multiple condition and decision coverage. Then we might need 7 or 8 test cases, but this 7 or 8 test cases, we can see experimentally that it is almost as good as the multiple condition coverage testing.

Undoubtedly the multiple condition decision coverage and MC/DC coverage testing is a very powerful strategy with very small number of test cases; it can achieve a thorough testing of the branches. And therefore, it is a very popular technique and incorporated into many testing standards now, let's look at the MC/DC testing.

(Refer Slide Time: 12:38)

Page: 17 / 17

Modified Condition/Decision Coverage (MC/DC)

- **Motivation:** Effectively test important combinations of conditions, without exponential blowup to test suite size:
 - “Important” combinations means: Each basic condition should independently affect the outcome of each decision

	T	
((c == CHAR) (c == DIGIT))		F
	F	

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Page 17 / 17

Here, we need to test all possible outcomes for each of the conditions. These are the basic conditions, but then we require that they independently affect the outcome of the entire decision. What it means is that, we not only require that each of the sub expression achieve true and false values, but also while the rest of the expression is held at some truth value. If you make this true to false then the decision should also change. The decision outcome will become false, if we make C == CHAR as the false and if we make c == CHAR as the true then the decision outcomes would become true. Our test cases would be able to achieve that and then we will say that this satisfies MC/DC coverage criteria.

(Refer Slide Time: 14:01)

Modified Condition/Decision Coverage (MC/DC)

- **Motivation:** Effectively test important combinations of conditions, without exponential blowup to test suite size:
 - “Important” combinations means: Each basic condition should independently affect the outcome of each decision
- **Requires:** $((c == \text{CHAR}) || (c == \text{DIGIT}))$
 - For each basic condition c , Compound condition as a whole evaluates to true or false as ac becomes T or F

Subsumption hierarchy

```
graph TD; MCC[MCC] --> MCDC[MC/DC]; MCDC --> CDD[Condition/Decision]; CDD --> Decision[Decision]; Decision --> Statement[Statement]; BCC[BCC] --> CDD
```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So, here the MC/DC coverage requires that for every basic condition in a complex conditional statement. The compound condition evaluates to true and false as the basic conditions are given true and false values. Let's explore further about this test technique.

(Refer Slide Time: 14:29)

Condition/Decision Coverage

- Condition: true, false.
- Decision: true, false.

Multiple Condition coverage (MCC)

- all possible combinations of condition outcomes in a decision
- for a decision with n conditions

2^n test cases are required

Modified Condition/Decision coverage (MC/DC)

- Bug-detection effectiveness almost similar to MCC
- Number of test cases linear in the number of basic conditions.

Subsumption hierarchy

```
graph TD; MCC[MCC] --> MCDC[MC/DC]; MCDC --> CDD[Condition/Decision]; CDD --> Decision[Decision]; Decision --> Statement[Statement]; BCC[BCC] --> CDD
```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

So far we looked at the condition decision coverage testing, it achieves both condition coverage and decision condition coverage. The multiple condition coverage testing which required exponential number of test cases, but it was a strong strategy. Now we are going to discuss about the MC/DC, where the bug detection effectiveness is very

similar to the MCC. But, the number of test cases is linear in the number of basic conditions. If we can draw the subsumption hierarchy, we can see here in the above figure that the MC/DC testing requires small number of test cases. It is stronger than condition decision coverage testing, but of course, it is weaker than the MCC, multiple condition coverage testing. But multiple condition coverage testing requires huge number of test cases, but MC/DC testing achieves testing, which is very close to the MCC testing. So, that is the reason why MC/DC testing is an important test strategy used extensively during testing.

(Refer Slide Time: 16:01)

Page 16 / 18

- MC/DC stands for **Modified Condition / Decision Coverage**
- It is a condition coverage technique
 - **Condition:** Atomic conditions in expression.
 - **Decision:** Controls the program flow.
- **Main idea:** Each condition must be shown to independently affect the outcome of a decision.
 - The outcome of a decision changes as a result of changing a single condition.

What is MC/DC?

Handwritten annotations: $(a > 10) | (c = 5) | (d > 50)$ with T over the first two and F over the third; F is written under the first two; T is written under the third; and F is written under the last two.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Now, let's try to understand, what is MC/DC and how to design the MC/DC test cases. As we have been already using the terms condition and decision, the condition is the atomic condition in expression or is a sub expression. The decision is the outcome of a decision statement and controls the program flow, already seen what are the basic conditions and what is the decision. The main idea in the MC/DC testing is that given an expression having multiple atomic expressions or conditions, each condition must be shown to independently affect the decision.

That is if we have an expression like $a > b$, $a > 10$ or $c = 5$ and $d > 50$. Let say this is our expression or the decision statement, now for achieving the MC/DC will hold it with some value let say $c = 5$ and $d = 60$. Then as we make $a > 10$ that is true, then the outcome should be true and if I make this is false the outcome will become false.

And similarly if you hold first and second to some truth outcomes let say first clause is true and second clause is true. And then if we make third clause as true and false, the decision output, the decision outcome should also become true and false. Similarly, for the second clause, if we hold these first and third in some true and false values as shown in the figure above let say second clause is true and this is true. And as we toggle, this second sub expression to true and false then the outcome will toggle to true and false. Let see does it really do? We have held this first sub expression to false. The last sub expression to true, now if we make the second sub expression to true then first and second evaluates to true and then third is also true. So, the outcome will be true, but what if we make first and second clause as false? So, first and second become false and the third becomes true. And therefore, the outcome will be false. So, we can say that if I hold the first sub expression to false, the last sub expression to true, then if we toggle the second sub expression to true and false the decision outcome also toggles to true and false. We need to do that for every sub expression, that it will independently affect the outcome of the decision. So, that is the main idea here in the MC/DC testing.

(Refer Slide Time: 20:24)

Three Requirements for MC/DC

Requirement 1:

- Every decision in a program must take T/F values.

Requirement 2:

- Every condition in each decision must take T/F values.

Requirement 3:

- Each condition in a decision should independently affect the decision's outcome.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

If we formally write down what we need during MC/DC testing, the first requirement is that the decision should have got true and false value by the test cases. Every condition or the sub expression must have got true and false values during the testing and also each sub expression should independently affect, determine the outcome of the decision. So, these are the 3 requirements.

(Refer Slide Time: 21:11)

Page 20 / 20

MC/DC Requirement 1

- The decision is made to take both T/F values.

If $((a > 10) \&\& ((b < 50) \&\& (c == 0)))$ then

true false

This is as in Branch coverage.

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

```
graph TD; If["If (( a > 10 ) && (( b < 50 ) && ( c == 0 ))) then"] -- true --> Cond1["(( a > 10 )"]; Cond1 -- true --> Cond2["( b < 50 )"]; Cond1 -- true --> Cond3["( c == 0 )"]; Cond2 -- true --> IfEnd["then..."]; Cond3 -- true --> IfEnd;
```

Let's try to understand with some examples. Let's look at this expression $a > 10$ and $b < 50$ and $c = 0$. And we want to test it such that the test cases will achieve MC/DC coverage. So, we need to assign the first sub expression true and false values such that the decision will become true and false. So, the first thing is that the decision as a whole should become true and false.

(Refer Slide Time: 22:01)

Page 20 / 20

MC/DC Requirement 2

- Test cases make every condition in the decision to evaluate to both T and F at least once.

If $(a > 10) \&\& ((b < 50) \&\& (c == 0))$ then...

true false

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

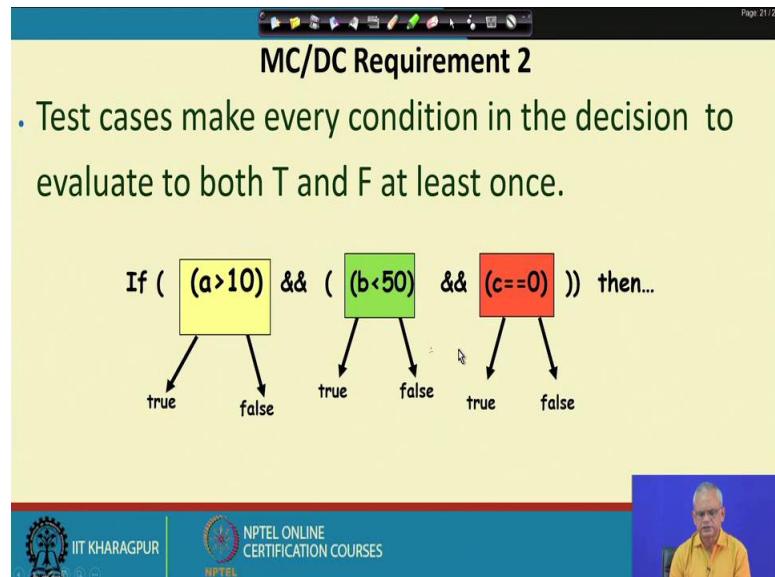
```
graph TD; If["If (( a > 10 ) && (( b < 50 ) && ( c == 0 ))) then..."] -- true --> Cond1["(( a > 10 )"]; Cond1 -- T --> Cond2["( b < 50 )"]; Cond1 -- F --> Cond3["( c == 0 )"]; Cond2 -- T --> IfEnd["..."]; Cond3 -- T --> IfEnd;
```

If we assign true and false to first clause i.e., a is 20, which is true and a is 5 false then the outcome should also become true and false. So, for what values of these 2, if you

hold them constant, will this toggle the output as it become true and false. So, if we have second clause is true and third clause is true i.e., b is 20 and c is 0, then this sub expression becomes true and see here there is a `&&` here. So, is this is true then the outcome the decision becomes true, if this is false then this is true and outcome becomes false.

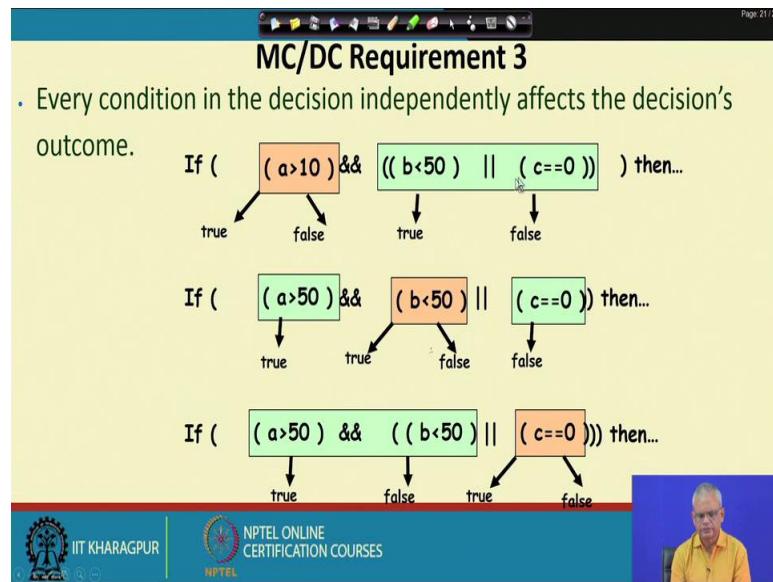
So, as the first expression becomes true and false, if we hold the second and third clause as true, the outcome the decision outcome toggles. Similarly, we need to do for the other two sub expressions.

(Refer Slide Time: 23:32)



So, for these as it becomes true and false, we need to hold these 2 such that the outcome becomes true and false and also for the last sub expression so, that is the requirement for MC/DC.

(Refer Slide Time: 23:52)



And, what are the values which will achieve MC/DC coverage for this; one is that we hold second to true and third to false, if this is \parallel and then as it toggles between true and false, the decision will toggle between true and false. And for second situation, for the second sub expression, if we hold second to true and third to false then the outcome decision outcome will toggle between true and false. Similarly, if you hold the first one to true and second one to false, then the third sub expression as it becomes true and false the decision will toggle between true and false, but then the question remains that given a complex expression, which is having a large number of sub expressions are conditions. How do we design the test cases such that MC/DC coverage will be achieved?

(Refer Slide Time: 25:03)

MC/DC: An Example

- N+1 test cases required for N basic conditions
- Example:

$$(((a>10 \mid\mid b<50) \&\& c==0) \mid\mid d<5) \&\& e==10$$

Test Case	a>10	b<50	c==0	d<5	e==10	outcome
(1)	<u>true</u>	false	<u>true</u>	false	<u>true</u>	<u>true</u>
(2)	false	<u>true</u>	true	false	true	<u>true</u>
(3)	true	false	false	<u>true</u>	true	<u>true</u>
(6)	true	false	true	false	<u>false</u>	<u>false</u>
(11)	true	false	<u>false</u>	<u>false</u>	true	<u>false</u>
(13)	<u>false</u>	<u>false</u>	true	false	true	<u>false</u>

- Underlined values independently affect the output of the decision



 IIT KHARAGPUR |
  NPTEL ONLINE CERTIFICATION COURSES

Let's first look at an example so, it is another example so that we can get the idea about how to achieve MC/DC coverage. We first develop the truth table and then we check that if the first one is true, then how can the output, this is true? And for second one the outcome is true. For third one, first sub expression is true and the outcome is true and scan here in the first column and see that for same values of the other sub expressions in the last row; here as we toggle between true and false the outcome becomes false.

So, these 2 test cases together will achieve the independent evaluation of the first condition. What about the second condition? The second condition, we scan through the truth table and find that when second cell is true and others are true, false, true etc., the outcome is true. And, when second cell at last row is false then the others are held true false, true etc., we get false. So, the second and last row together achieve the independent evaluation of the second clause, the third clause is when first cell is true and the others are the false, true, the outcome is true and when this is false in fifth cell of the third column and others are false, true, then the outcome toggles. So, we also need this and so on.

So, for every condition we look at the truth table and then we examine that when does the output toggle. The decision outcome toggle when the specific condition toggles. So, this is the main idea behind MC/DC. We are almost at the end of this lecture and will stop here and continue in the next lecture.

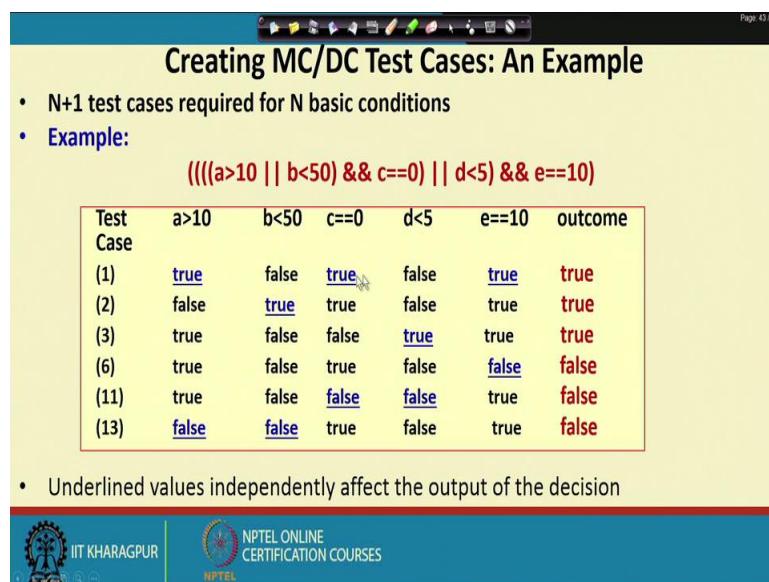
Thank you.

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

Lecture – 58
MC/DC Testing

Welcome to this lecture. In the last lecture, we were discussing about the MC/DC Testing.

(Refer Slide Time: 00:26)



Creating MC/DC Test Cases: An Example

- N+1 test cases required for N basic conditions
- Example:

$$(((a>10 \mid\mid b<50) \&\& c==0) \mid\mid d<5) \&\& e==10$$

Test Case	a>10	b<50	c==0	d<5	e==10	outcome
(1)	true	false	<u>true</u>	false	true	true
(2)	false	<u>true</u>	true	false	true	true
(3)	true	false	false	<u>true</u>	true	true
(6)	true	false	true	false	<u>false</u>	false
(11)	true	false	<u>false</u>	<u>false</u>	true	false
(13)	<u>false</u>	<u>false</u>	true	false	true	false

• Underlined values independently affect the output of the decision

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES
NPTEL

We said that it is a very important test criterion, it is particle and also it gives a thorough testing and therefore, very popular. Now, let's proceed with what we were discussing last time. We had seen the basic definitions of the MC/DC testing and we had seen few examples about how to develop MC/DC test cases given a conditional statement, but remember that there may be several conditional statements in a program and of course, we need to design for every conditional statement. Now, let's look at more systematic way to develop the MC/DC test cases.

One thing need to mention that, if a conditional expression has n sub expressions, here we have 5 sub expressions. Then for multiple condition coverage MCC, we need 2^5 which is 32, but for MC/DC test cases it will be just $6, 5 + 1 = 6$. And, if there are 10 basic conditions or sub expressions, then for multiple condition coverages we need 2^{10} which is a 1024 test cases, but for in MC/DC testing we need 11 test cases. We will not

give any proof for $n+1$ test cases required for n basic conditions we will just take this result.

Now, let's look at this example expression and see how we go about creating the MC/DC test cases. The first step of course, is to draw the truth table as the different sub expressions or the basic conditions change the truth values. All possible combinations of truth value we need to explore, but that will be 2^5 , which is 32 rows in the truth table. But as screen is small, I just drawn only the important ones in the above figure and I have not included all the possible combinations. I have omitted the ones which are not required for inclusion in the MC/DC test suite.

Once, we draw the complete truth table i.e., all possible combinations of the truth values of the basic conditions and the corresponding outcome. We need to check for every basic condition, which are the 2 rows, where the truth value of the basic expression is true and false. And the output also toggles, true and false and the remaining conditions basic conditions have the same value just see here in the first row false, true, false, true and for last row also false, true, false, true. So, the other conditions truth value of other conditions remaining constant, a change of the truth value of the condition under consideration as it toggles from true to false the outcome toggles from true to false.

Similarly, let's look at the second clause. We check all rows and see that when it is false in the first row, the outcome is true, but when it is true here in the second row, the outcome is still true. So, these two cannot be the ones, we just check all possible combinations here. And then we find that the second one along with the thirteenth-one, as the change from true to false is the condition $b < 50$ changes from true to false. And the other ones remaining constant true, false, true, false, true, false, true and this is false true, false true.

As $b < 50$ toggles from true to false the outcome toggles from true to false. So, we need row no1, 13 to so, the independent evaluation of the first condition basic condition. So, this shows the independent this first condition, independently toggles the outcome, when the others are remaining constant. Similarly, row no 2 and 13 toggles the outcome when the rest are held constant. Similarly, the third-one, as it toggles from true or false the outcome toggles from true to false, the other things remaining constant.

Similarly, the fourth-one as it toggles from true to false, the outcome toggles from true to false, false rest are held constant. And similarly the last one as it toggles from true to false. The outcome toggles from true to false. Now, we need all this out of the 32 rows, we need 6 test cases, for achieving MC/DC coverage and we have 5 basic conditions and we need 6 test cases.

(Refer Slide Time: 07:27)

Sytematically Creating MC/DC test cases

- Create truth table for conditions.
- Extend the truth table to represent test case pair that lead to show the independence influence of each condition.

Example : If (A and B) then ...

Test Case Number	A	B	Decision	Test case pair for A	Test case pair for B
1	T	T	T	3	2
2	T	F	F		1
3	F	T	F	1	
4	F	F	F		

- Show independence of A :
 - Take 1 + 3
- Show independence of B :
 - Take 1 + 2

$\{1,2,3\}$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, how we can systematically create the MC/DC test cases? Let's take an example; this is the example as shown in the above diagram. A and B are Boolean variables and we need to create the truth table of course, but then will create a extended truth table. In the extended truth table for each basic condition, we will keep a note of which 2 rows independently influence the output for a specific basic condition. Let's first see how to have the extended truth table? We have the truth table here a numbered the rows 1, 2, 3, 4, there are only 2 basic conditions. So, we have 4 rows in the truth table and this is the outcome.

And, then we just check here, that when A is true and B is true, the outcome is true. Now, if A is false and the B is remaining true, the outcome is false. So, 1 along with 3 will show the independent influence of this condition, first condition on the outcome. And as we go round as we go check all the possible rows of course, we will find 3 along with 1 also independently influences the outcome. And the other combinations are not able to independently influence the outcome, but for larger sub expressions and other

examples, we will see that there may be several pairs of rows which show the independent influence of the first condition on the outcome.

Similarly, we check the second condition. We see that when A is true and B is true the outcome is true and in the next one we say that A is held true, but if B becomes F the outcome toggles. And therefore, 1 along with 2 will show the independent influence of B on the outcome.

Similarly, 2 with 1 will also show the independent evaluation. So, we have to take 1, 3 tests 1, 3 for independent influence of A on the output. And, 1, 2 for the independent influence of B on the output, and our MC/DC test suite will be the first row, second row, and third row. So, we had so, this is the MC/DC test suite. So, we had 2 basic conditions and we need 3 test cases. The test suit is 1, 2 and 3.

(Refer Slide Time: 11:25)

Sytematically Creating MC/DC test cases

- Create truth table for conditions.
- Extend the truth table to represent test case pair that lead to show the independence influence of each condition.

Example : If (A and B) then ...

Test Case Number	A	B	Decision	Test case pair for A	Test case pair for B
1	T	T	T	3	2
2	T	F	F		1
3	F	T	F	1	
4	F	F	F		

- Show independence of A :
 - Take 1 + 3
- Show independence of B :
 - Take 1 + 2
- Resulting test cases are
 - 1 + 2 + 3



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES



Now, let's look at another example.

(Refer Slide Time: 11:34)

	A	B	C	Result	A	B	C	MC/DC
1	1	1	1	1			*	*
2	1	1	0	0			*	*
3	1	0	1	1	*			*
4	0	1	1	1		*		*
5	1	0	0	0				
6	0	1	0	0				
7	0	0	1	0	*	*		*
8	0	0	0	0				

Another Example



This is slightly larger example, 3 Boolean variables A, B and C. And therefore, we need 2^3 rows in the truth table. I drawn the truth table here, extended truth table where not only drawn the truth table, but also we are keeping note of which rows result in independent influence of A on the outcome, independent influence of B on the outcome, independent influence of C on the outcome. And then we will identify the MC/DC test suite. So, this is the extended table.

Now, let's take this the first Boolean variable A. Now, as we scan through here A is 1, when B and C are held at 1 result is 1. Now, let us see where B and C are held at 1 here, but then this does not help A has toggled, but the outcome has not toggled. So, that does not help us. Now, let's check the second one. When A is 1, B and C is 1 and 0 respectively, outcome is 0 now let's check where 1, 0. So, this also does not help us. Now, let's check the third-one. So, A is 1 B, C are held at 0, 1 respectively and the outcome is 1, now see let's see where it is 0, 1. So, A has toggled from 1 to 0 and B and C are remaining 0 and the outcome has toggled. And therefore, we select these 2 how the independent influence of A on the outcome.

Similarly, if we scan through for B we will find that 4 and 7, that show the independent influence of B on the outcome of the expression evaluation. Similarly, for C if we scan through the truth table, we will see that 1 and 2 will show independent influence of C on

the outcome just C here, C has toggled from 1 to 0 outcome has toggled from 1 to 0 the rest A, B are held at 1, 1. So, this two will do.

Now, let's create the MC/DC test suite, we need 1 that will show 1 and 2 that will show the independent influence of C we need 3, 4 and we need 7. So, that is 5 test cases. So, it is linear in the number of test suite even though it is not $n+1$, because here that does not hold, but it will be around that.

(Refer Slide Time: 15:19)

If (A and (B or C)) then...

Minimal Set Example

We want to determine the MINIMAL set of test cases

Here:

- {2,3,4,6}
- {2,3,4,7}

Non-minimal set is:

- {1,2,3,4,5}

Page 44 / 44

TC# ABC Result A B C

1	TTT	T	5		
2	TTF	T	6	4	
3	TFT	T	7		4
4	TFF	F		2	3
5	FTT	F	1		
6	FTF	F	2		
7	FFT	F	3		
8	FFF	F			

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

NPTEL

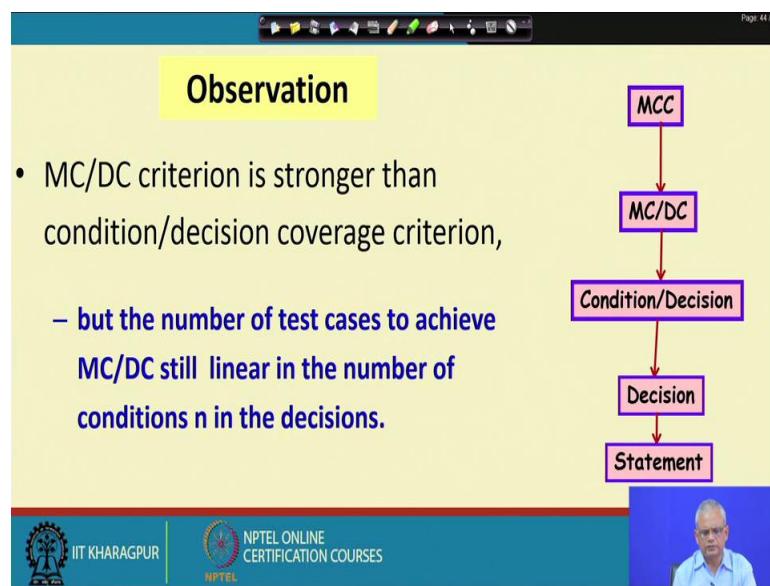
Now, let's see when there are multiple rows can show the independent influence of a variable, we will have alternate possibilities for the MC/DC test suite. And we need to select the minimal test suite in that case. Let us look at this expression. A B C are three Boolean variables, and there are $2^3 = 8$ rows in the truth table, and we have written the truth table here. We see here that 1 along with 5 shows the independent influence of A on the outcome. Similarly, 2 along with 6 also shows the independent influence of the basic condition A on the outcome. Similarly, 3 along with 7 also shows independent influence of A on the outcome. And naturally will have 5 and 1, 6 and 2 and 7 and 3. Thus the same thing here, because this is 1-5, 2-6 and 3-7 we have 5-1, 6-2, 7-3. So, this of course, can be easily find out based on this.

Now, let's see the independent influence B on the outcome, we have 2-4 and of course, 4-2. Similarly, for C it is 3-4, 4-3. Now when we choose the MC/DC test suite we can

find that 2, 3, 4 and 6. So, 2-6, 4-3 and 4-2 this has to be included anyway, 4-3 and 4-2 will be there and we can either choose 2 and 6 or 5 and 1 etc.

So, if we see select 2 and 6 for the A and then we have these 2 possibilities, 2-3, 4-6 and 2-3, 4-7 and also there is a non-minimal set where we include 2-4, 4-2 and 3-4, 4-2 and 3-4, that has to be there 4-2 and 3-4 and also we include 1-5. So, this is a non-minimal set requiring 5 test cases, but we can have the minimal set, which is 4 test cases.

(Refer Slide Time: 18:47)



Now, let's note down our observation, MC/DC criterion shows the independent influence of every basic condition and the outcome. The number of test cases is linear in the number of basic conditions. Typically for n basic condition we need $n + 1$ test cases. Of course, sometimes we have non minimal test cases where we need 1 more, but this is a very strong testing.

It subsumes the condition decision coverage, which in turn subsumes decision coverage and which subsumes statement coverage. Of course, MC/DC is subsumed by MCC, but MCC is impractical. So, MC/DC is a strong enough test criterion and also requires small number of test cases and therefore, it is very popular.

(Refer Slide Time: 20:07)

• MC/DC essentially is :

MC/DC: Summary

- basic condition coverage (C)
- branch coverage (DC)
- plus one additional condition (M):
every basic condition must independently affect the decision's output

- It is subsumed by MCC and subsumes all other criteria discussed so far
 - stronger than statement and branch coverage
- **A good balance of thoroughness and test size and therefore widely used...**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, MC/DC just to summarize, we require basic condition coverage, branch coverage, and also we show that every condition independently affects the decisions output. It is subsumed by MCC and subsumes all other criteria and is stronger than statement and branch coverage, it is a good balance of thoroughness and test sizes widely used.

(Refer Slide Time: 20:40)

Assignment

- Design MC/DC Test cases for the following branch condition:

((a>10)&&(b>=20)) || (c==5)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Here is just a small assignment for you, just note it down a simple branch condition having 3 basic conditions. And please develop the MC/DC test suite, you can of course,

frame further branch conditions on your own involving 3, 4, 5 basic conditions, develop the truth table and decide on the MC/DC test suite.

(Refer Slide Time: 21:24)

The slide has a yellow background and a blue header bar. The title 'Path Coverage' is centered at the top. Below it, there are two main bullet points:

- Design test cases such that:
 - All linearly independent paths in the program are executed at least once.
- Defined in terms of
 - Control flow graph (CFG) of a program.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player window showing a man speaking.

Now, let's look at path testing. Path testing is based on the control flow, the main idea here is that we need to design test cases such that, all linearly independent paths, all linearly independent paths are executed at least once. So, we need to define this term, what we mean linearly independent paths. And as mentioning the paths in a program are defined in terms of the control flow graph of a program.

So, we need to first discuss, how to draw the control flow graph of a program. Or the CFG of the program, and once we are able to draw the control flow graph of a program, we will see what are the linearly independent paths and, how to estimate the number of linearly independent paths in a program and that will tell us about the number of test cases required to achieve path coverage.

(Refer Slide Time: 22:50)

- To understand path coverage-based testing:
 - We need to learn how to draw control flow graph of a program.
- What is a control flow graph (CFG)?
 - **The sequence in which different instructions of a program get executed.**
 - The way control flows through the program.

The most basic question here is that what exactly is the control flow graph? What does it indicate? The control flow graph indicates how the control flows through the program, which in turn is the sequence in which the different instructions of the program get executed, when a test case is executed.

(Refer Slide Time: 23:23)

- Number all statements of a program.
- Numbered statements:
 - Represent nodes of control flow graph.
- Draw an edge from one node to another node:
 - If execution of the statement representing the first node can result in transfer of control to the other node.

int f1(int x,int y){
1 while (x != y){
2 if (x > y) then
3 x=x-y;
4 else y=y-x;
5 }
6 return z;
}

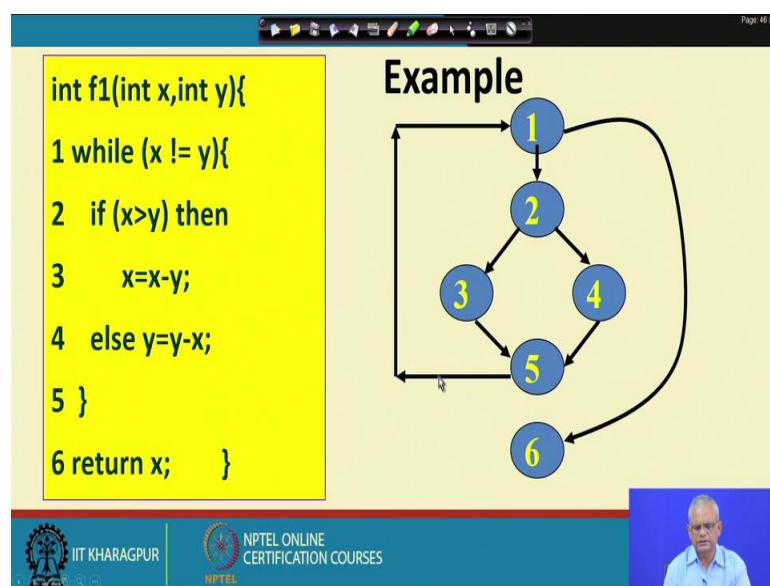
```
graph TD; 1((1)) --> 2((2)); 2 --> 3((3)); 3 --> 4((4)); 4 --> 5((5)); 5 --> 6((6)); 6 --> 1;
```

Now, the next question is how to draw the control flow graph? The first step, given a unit or a function, the first step is to number all the statements. We can develop the control flow graph either for a segment of a unit or the full unit; we have not numbered the first

statement here, because anyway that will get executed. And the rest once we have numbered here and these numbered statements become nodes in the flow graph. Now, then we draw the edges between the nodes. So, this we draw the nodes here 1 2 3 4 5 6 as shown in the figure and the next work is to draw the edges between these nodes, we draw an edge from 1 node to another, when we see that execution of the statement representing the node, when transfer control to the other node.

Now, let's look at this case, we have one, the while statement from here it can enter the loop the control can come to the next statement. So, 1 to 2 it can come and also if while becomes false the control might come to 6. So, I drawn here 1 to 6, 2 is a decision statement and from 2 either the control can come to 3 or it can come to 4. So, that is what I have represented. So, that is the basic idea here. That given a program segment or a unit, we first number the statements here and then these become the nodes and the control flow graph representing each statement. And then we draw an edge between the nodes, if in the program control can flow from one node to the other.

(Refer Slide Time: 26:16)



So, the same example here that, we have numbered the statements and then we have drawn the corresponding control flow graph, but let's look at a systematic way, how we can develop the control flow graph for any given program.

(Refer Slide Time: 26:50)

The slide contains the following text:

- Every program is composed of:
 - Sequence
 - Selection
 - Iteration
- If we know how to draw CFG corresponding these basic statements:
 - We can draw CFG for any program.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a small video thumbnail of a professor.

To draw the control flow graph for any given program, we need to understand that every structured program consists of 3 types of statements; sequence, selection, and iteration and if we can understand, how the control transfers for these 3 types of statements, then on the control flow diagram we can represent those edges. Whenever we find a sequence type of statement in a program and we know how to represent sequence, which is then we can easily draw that edge. For selection if we know how to draw the edges. We will do the same thing on the control flow graph and for the iteration. So, these are the basic 3 constructs.

Let's understand how to draw the edges and the CFG corresponding to these 3 types of statements. And then given any program we should be able to easily develop the control flow graph, because they will after all be consisting of these types of statements. We are almost at the end of this lecture. We will stop here and we will continue in the next lecture from this point.

Thank you.

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

Lecture - 59
Path Testing

Welcome to this lecture. In the last lecture, we were discussing about the Path Testing. We said that in path testing, we need all linearly independent paths to be exercised by the test suite. Then only we can say that the test suite achieves path coverage, but we said that to define linearly independent paths, we need to be able to draw the control flow graph of the program. We were discussing how to draw the control flow graph of the program. Then we said that number the statements and then each number becomes a node and we need to draw the edges among the nodes. We were discussing a systematic technique about how to draw the control flow graph. We said that there are 3 types of statements in any program the sequence, selection and iteration.

(Refer Slide Time: 01:24)

The slide has a yellow background. At the top, there is a toolbar with various icons. Below the toolbar, the title 'How to Draw Control flow Graph?' is enclosed in a yellow box. The main content consists of two bullet points:

- Every program is composed of:
 - Sequence
 - Selection
 - Iteration
- If we know how to draw CFG corresponding these basic statements:
 - We can draw CFG for any program.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'.

If we know, how to represent the edges for these three types of statements, then we should be able to draw the control flow graph for any program. Now, let's see how to draw the edges for these three types of statements.

(Refer Slide Time: 01:43)

How to Draw Control flow Graph?

- Sequence:

1 a=5;
2 b=a*b-1;

```
graph TD; 1((1)) --> 2((2))
```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

Page 46 / 46

First let's look at sequence type of statement; this is an example of sequence type of statement. Here the control flows just from one statement to the other in sequence. So, this is the easiest of all 3 types of statements and we have numbered this 1 and 2 and we just draw the edge showing the control flow occurs from statement 1 to statement 2.

(Refer Slide Time: 02:11)

How to Draw Control Flow Graph?

- Selection:

1 if($a > b$) then
2 $c = 3$;
3 else $c = 5$;
4 $c = c * c$;

```
graph TD; 1((1)) --> 2((2)); 1 --> 3((3)); 2 --> 4((4)); 3 --> 4
```

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

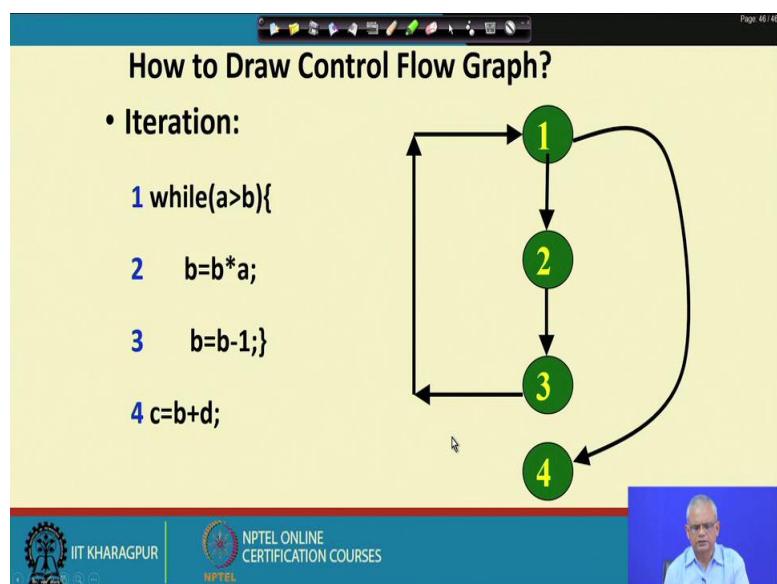
Page 46 / 46

Now, let's look at a selection type statement, if $a > b$ then $c = 3$ else $c = 5$, so this is the statement. Then the subsequent statement is $c = c * c$. So, this is the selection statement

that follows the sequence statement based on the outcome of the decision here the control transfers either to 2 or to 3.

But, after 2 or 3 are completed, the control goes to the next statement in sequence. So, drawn that 1 to either 2 or 3 and then from both 2 and 3 control comes to 4 so, this also easy to draw.

(Refer Slide Time: 03:16)



Now, let's look at iteration type of statement. So, this is a while loop and here if the while condition is true, then it enters the loop and statement 2 will be executed, but if the while condition is false it goes out of the loop and goes to the next statement in sequence, but the only thing to mark here is that each time it completes a loop the control comes back to check what the outcome of the decision is. Only when the decision becomes false, it comes to the next statement.

The decision never transfers from 3 to 4, but from 3, it always goes back to 1 where the decision is evaluated and only if it is false, it comes to 4. So, the same thing we have represented here, if the decision is true, enters the loop 1-2-3 and it compulsorily transfers to 1 to check what is the outcome of the decision. And if it is true, again it look continuous the loop and only when evaluates to false goes to the next statement.

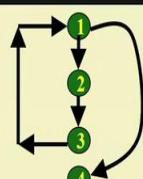
So, if you draw edge between 3 and 4, that will be erroneous many who start to draw control flow graphs make that mistake they draw an edge between 3 and 4, but see here

that control does not transfer from 3 to 4, but it goes back to 1 where the decision is checked and only when it is false the control transfers to 4.

(Refer Slide Time: 05:15)

Path

- A path through a program:
 - **A node and edge sequence from the starting node to a terminal node of the control flow graph.**
 - There may be several terminal nodes for program.



IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

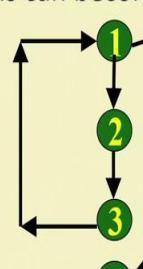


Now, once we draw the control flow graph, we can define a path. A path is any node and edge sequence from the start node to a terminal node of the control flow graph. For this example, from 1 we can have a path 1-2-3 and then 4 another may be 1-2-3,1-4. So, several paths are possible given a program. And also there can be several terminal nodes in a program, because there may be written statements and so on.

(Refer Slide Time: 06:12)

All Path Criterion

- In the presence of loops, the number paths can become extremely large:
 - This makes all path testing impractical



IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES



The all path criterion says that, in presence of loops the number of paths can be infinite or very large. The reason for that is that for given this kind of a loop, we can have 1 2 3 4, 1 2 3, 1 2 3 4 or 1 2 3, 1 2 3, 1 2 3, 3 times and 4 and so on. All are qualifiers paths. So, we can say that in presence of loops the number of paths becomes infinite.

And therefore, if you trying to achieve all path criterion, it will be very difficult, even if we know how many,, what is the maximum number of iterations that can occur here, but sometimes we cannot determine for example, it might be that while true repeat. So, all path criterion even though it is a strong criterion, but then it is impractical because we do not know how many paths are required and the paths can be extremely large.

(Refer Slide Time: 07:47)

Linearly Independent Path

- Any path through the program that:
 - Introduces at least one new edge:
 - Not included in any other independent paths.

From this consideration that all path criterions are not a practical criterion, the linearly independent path was defined, the linearly independent path is defined as a path belongs to the set of linearly independent paths.

If it is introduced, at least one new edge, which is not included in any other independent path so, 1 2 3 4 will be a linear linearly independent path, but 1 2 3, 1 2 3 4 will not, because it does not cover any new edge not covered by the previous test case that is 1 2 3 4. So, the same edges are repeating and get excluded by linearly independent path.

(Refer Slide Time: 08:47)

- It is straight forward:
 - To identify linearly independent paths of simple programs.
- For complicated programs:
 - It is not easy to determine the number of independent paths.

Page 46 / 46

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Given a control flow graph finding a linearly independent path for very simple programs, we can easily find it, but for complicated programs it becomes extremely difficult. For example, if I draw a graph like this in the above figure. So, just for this finding the number of independent path is nontrivial. Remember that the number of statements can be 100 and the number of edges can be let's say 150. Then even if you spend a month trying to find out what is the number of linearly independent paths you may still be struggling at the end of the month. So, just from visual inspection trying to compute, linearly independent path set is an extremely complicated task for nontrivial programs.

(Refer Slide Time: 10:22)

McCabe's Cyclomatic Metric

- An upper bound:
 - For the number of linearly independent paths of a program
- Provides a practical way of determining:
 - The maximum number of test cases required for basis path testing.

Page 47 / 47

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

McCabe based on graph theoretical results, could find the number of linearly independent paths in a straight forward manner. Rather than trying to count all the linearly independent paths, he used graph theoretic results. It is extremely simple to use his result to find the number of linearly independent paths in a program that provides a practical way of determining the maximum number of test cases required for the basis path testing. So, the set of linearly independent paths are also called as basis paths and that is the reason McCabe's Cyclomatic Metric is an important metric for any program.

(Refer Slide Time: 11:22)

McCabe's Cyclomatic Metric

- Given a control flow graph G , cyclomatic complexity $V(G)$:
- $- V(G) = E - N + 2$

• N is the number of nodes in G
 • E is the number of edges in G

$$4 - 4 + 2 = 2$$

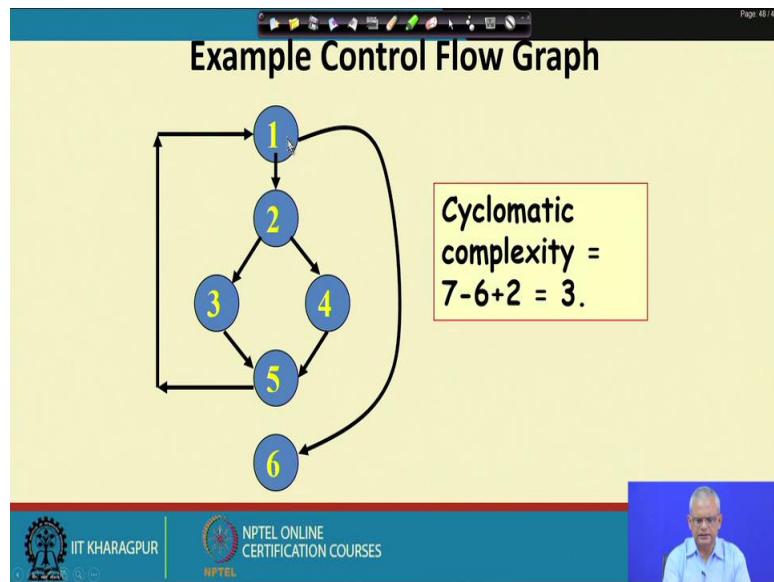
Hand-drawn diagram of a control flow graph G with 4 nodes (1, 2, 3, 4) and 4 edges. Node 1 has an incoming edge from the left and an outgoing edge to node 2. Node 2 has an outgoing edge to node 3. Node 3 has an outgoing edge to node 4 and an incoming edge from node 4. Node 4 has an outgoing edge back to node 1, forming a loop.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let's see how to compute the McCabe's metric. Given a control flow graph the McCabe's metric is defined as $V(G)$, G is the graph and $V(G)$ is the metric, which is the number of edges in the graph - the number of nodes + 2. Here the number of edge is 4 and there are 4 nodes for. So, $4 - 4 + 2 = 2$. So, the McCabe metric for a control flow graph, for a loop or iterative statement is 2. Given any complex control flow graph we can just compute the number of edges, number of nodes and then you can just use the formula $E - N + 2$ and get the cyclomatic complexity value.

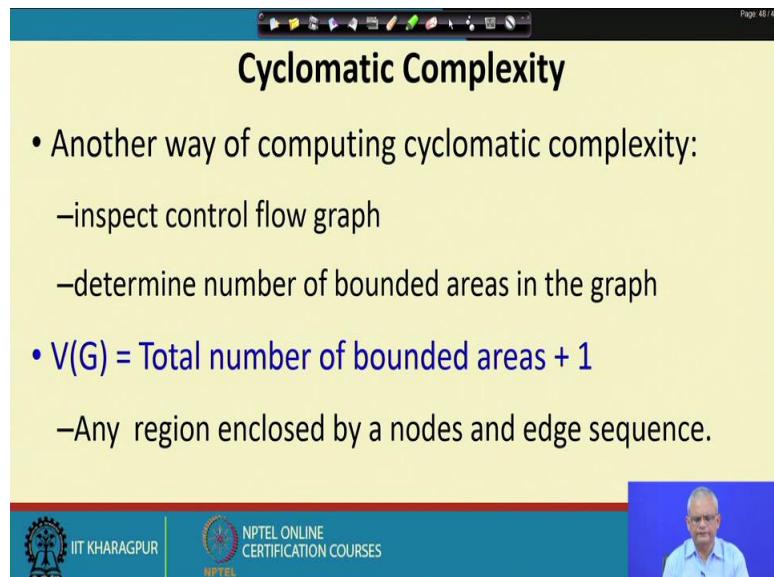
This is also easily automated because, given a graph, we can easily compute the number of nodes and the number of edges and then compute the McCabe's metric.

(Refer Slide Time: 12:59)



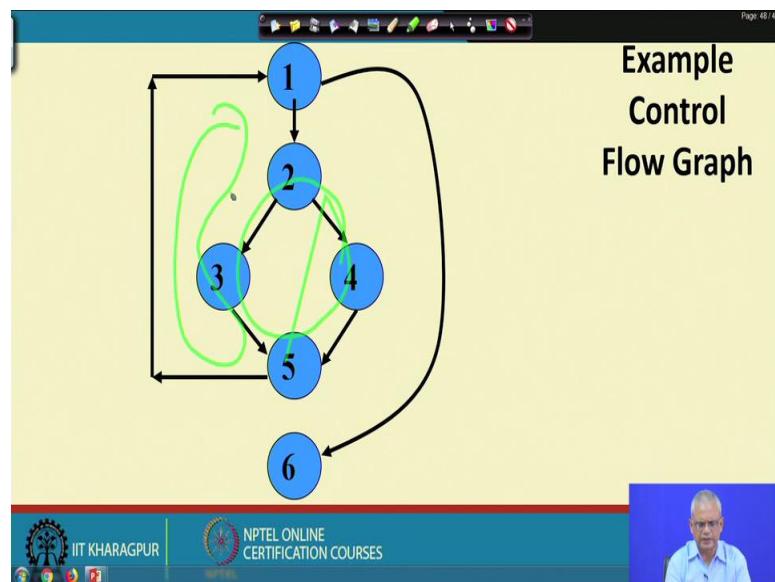
Let's take this example. So, for this program we find that the number of edges is 7, number of nodes is 6 you can count and then plus 2 which is 3.

(Refer Slide Time: 13:20)



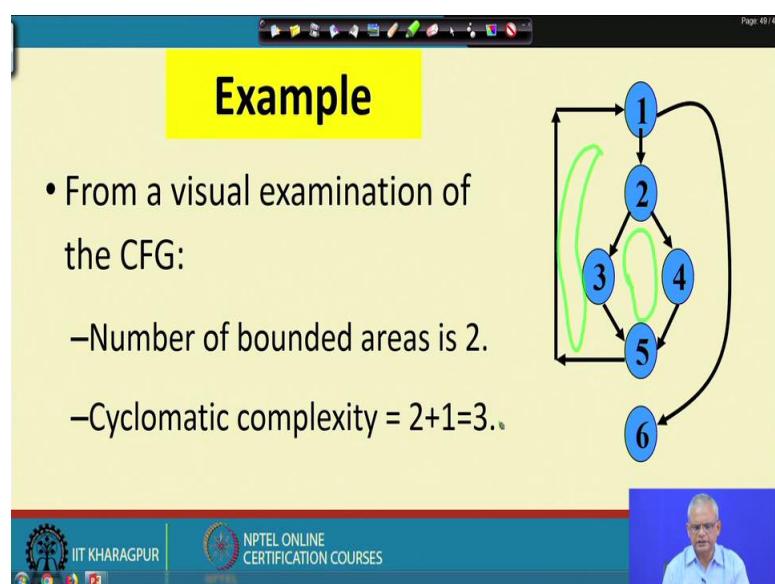
There are alternate definitions of the cyclomatic complexity, we can visually examine the number of bounded areas and add 1 and that also will give the cyclomatic complexity.

(Refer Slide Time: 13:40)



For example, for the same control flow graph. We can find the number of bounded areas. So, the number of bounded areas as represented in the above graph is 2. So, $2+1$, number of bounded areas plus 1. So, $2+1=3$ and this is easy to compute from a visual examination of the CFG. The other one, $E-N+2$. So, that is easy to compute using a computer program.

(Refer Slide Time: 14:23)



So, here the number of bounded areas are 2 and $2+1$ is 3 and that is the cyclomatic complexity we might also compute $E-N+2$, which will also give the same result.

(Refer Slide Time: 14:50)

The slide has a yellow background. At the top, the title 'Cyclomatic Complexity' is centered. Below the title is a bulleted list:

- McCabe's metric provides:
 - A quantitative measure of testing difficulty and the reliability
- Intuitively,
 - Number of bounded areas increases with the number of decision nodes and loops.

At the bottom of the slide, there is a footer bar with three sections: 'IIT KHARAGPUR' (with a logo), 'NPTEL ONLINE CERTIFICATION COURSES' (with a logo), and a video player showing a man speaking.

The McCabe's metric provides a quantitative measure of the testing difficulty, because if the McCabe's metric is 15; that means, there are 15 basis paths or linearly independent paths. We need 15 test cases. So, the McCabe's metric value provides the difficulty level of testing and also the reliability, because that many paths are there and we might miss bugs and that also corresponds to the reliability.

(Refer Slide Time: 15:38)

The slide has a yellow background. At the top, the title 'Cyclomatic Complexity' is centered. Below the title is a bulleted list:

- The first method of computing $V(G)$ is amenable to automation:
 - You can write a program which determines the number of nodes and edges of a graph
 - Then use the formula to find $V(G)$.

At the bottom of the slide, there is a footer bar with three sections: 'IIT KHARAGPUR' (with a logo), 'NPTEL ONLINE CERTIFICATION COURSES' (with a logo), and a video player showing a man speaking.

Computing the bounded areas is easy from a visual inspection and by the $E-N+2$ formula , it is easy to compute using a computer program.

(Refer Slide Time: 15:52)

The slide has a yellow background and a blue header bar at the top. The title 'Cyclomatic Complexity' is centered in bold black font. Below the title is a bulleted list:

- The cyclomatic complexity of a program provides:
 - A lower bound on the number of test cases to be designed
 - To guarantee coverage of all linearly independent paths.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man speaking.

But, one thing we must remember that it provides a bound on the number of test cases to be designed, we can test even more test cases.

(Refer Slide Time: 16:12)

The slide has a yellow background and a blue header bar at the top. The title 'Cyclomatic Complexity' is centered in bold black font. Below the title is a bulleted list:

- Knowing the number of test cases required:
 - Does not make it any easier to derive the test cases,
 - Only gives an indication of the minimum number of test cases required.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer bar, there is a small video window showing a man speaking.

But, the problem is that knowing the number of test cases required does not make it any easier to derive the test cases, because if we say that there are 15 linearly independent paths, but then how do you identify those paths? We know that there are 15, but then identifying those 15 is an extremely tough problem.

So, one way is that we keep on testing and see if a new edge is covered by a test case, which was not covered by earlier test cases then we say that a new linearly independent path is being executed. And we can count the number of linearly independent paths and we can find the percentages of linearly independent paths that have been executed.

(Refer Slide Time: 17:11)

The slide has a title 'Practical Path Testing' and two main bullet points:

- The tester proposes initial set of test data :
 - Using his experience and judgment.
- A dynamic program analyzer used:
 - Measures which parts of the program have been tested
 - Result used to determine when to stop testing.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the NPTEL logo, and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right side of the footer, there is a small video window showing a person speaking.

So, in that, the tester proposes an initial setup test data and then use a dynamic program analyzer, which counts the number of linearly independent paths being executed and then it displays the percentage of linearly independent paths that have been covered and that gives the path coverage metric.

(Refer Slide Time: 17:40)

Page 50 / 50

Derivation of Test Cases

- Draw control flow graph.
- Determine $V(G)$.
- Determine the set of linearly independent paths.
- Prepare test cases:
 - Force execution along each path.
 - Not practical for larger programs.

 IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL


But, can we derive the test cases for very trivial program? Yes, if there are 1 or 2 control statements and so on becomes easy, but if there are dozens of control statements becomes extremely difficult to derive the test cases.

(Refer Slide Time: 18:05)

Page 50 / 50

```

int f1(int x,int y){
1 while (x != y){
2   if (x>y) then
3     x=x-y;
4   else y=y-x;
5 }
6 return x; }
```

Example



 IIT KHARAGPUR
NPTEL ONLINE
CERTIFICATION COURSES
NPTEL

For this example above, we see that there are 3 linearly independent paths; one occurs when it enters in it executes the statement 3 and the other one when it executes the statement 4. And we can check that 2 test cases should be able to cover all the linearly independent paths here. Even though the McCabe's metric is 3, that is an upper bound and we might need 2 or 3 test cases, we will design test cases to achieve path coverage, but just need to mention 1 shortcut here to count the number of linearly independent

paths, other than drawing the CFG counting the number of edges, number of nodes and performing E-N+2.

Or computing the number of bounded areas + 1, the other method is look through the program statement; whenever you find a decision statement count it, number of decision statements + 1. So, that is also equal to the McCabe's metric. Let me repeat again to be able to easily compute the McCabe's metric look through the program code, whenever you encounter a decision statement like while, if etc., count it and then add 1 that will also be equal to E-N+2 or number of independent bounded areas + 1.

(Refer Slide Time: 20:08)

The slide has a yellow background. At the top, there is a navigation bar with icons. The title 'Derivation of Test Cases' is centered in bold black font. Below the title is a bulleted list:

- Number of independent paths: 3
 - 1,6 test case (x=1, y=1)
 - 1,2,3,5,1,6 test case(x=1, y=2)
 - 1,2,4,5,1,6 test case(x=2, y=1)

At the bottom, there is a blue footer bar. On the left, it features the IIT Kharagpur logo and the text 'IIT KHARAGPUR'. In the center, it features the NPTEL logo and the text 'NPTEL ONLINE CERTIFICATION COURSES'. On the right, there is a video player window showing a man in a light blue shirt speaking.

So, for the previous program we can we have number of independent paths here 3 and we can use 3 test cases for achieving the path coverage.

(Refer Slide Time: 20:29)

An Interesting Application of Cyclomatic Complexity

- Relationship exists between:
 - McCabe's metric
 - The number of errors existing in the code,
 - Time required to correct the errors.
 - Time required to understand the program

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The cyclomatic complexity metric other than giving an estimate of the number of linearly independent paths, it has several other applications. It correlates with the number of errors existing in the code after testing, it correlates with the time required to correct an error and it correlates with time required to understand the program. You might ask why there is a correlation between the McCabe's metric and these 3 issues.

It is not hard to make out why it is so? Remember, that McCabe's metric gives the number of paths in the program and to understand the program, you need to trace all the paths. So, if there are independent paths to be able to understand this behavior of the program, you need to check that from the output how do you trace to the input. So, all paths you have to check in the code and see how the control it gets the values and so on and that is the reason why the understanding difficulty correlates well with the McCabe's metric. And that is also the reason, because to be able to correct the error you need to understand the program. And therefore, it correlates with the cyclomatic metric and similar is the error existing in the code.

(Refer Slide Time: 22:30)

The slide has a yellow background. The title 'Cyclomatic Complexity' is at the top. Below it is a bulleted list:

- Cyclomatic complexity of a program:
 - Indicates the psychological complexity of a program.
 - Difficulty level of understanding the program.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player window showing a man in a light blue shirt.

The cyclomatic complexity corresponds to the difficulty of understanding a program. And therefore, we say that it indicates the psychological complexity of a program or the structural complexity of a program.

(Refer Slide Time: 22:51)

The slide has a yellow background. The title 'Cyclomatic Complexity' is at the top. Below it is a bulleted list:

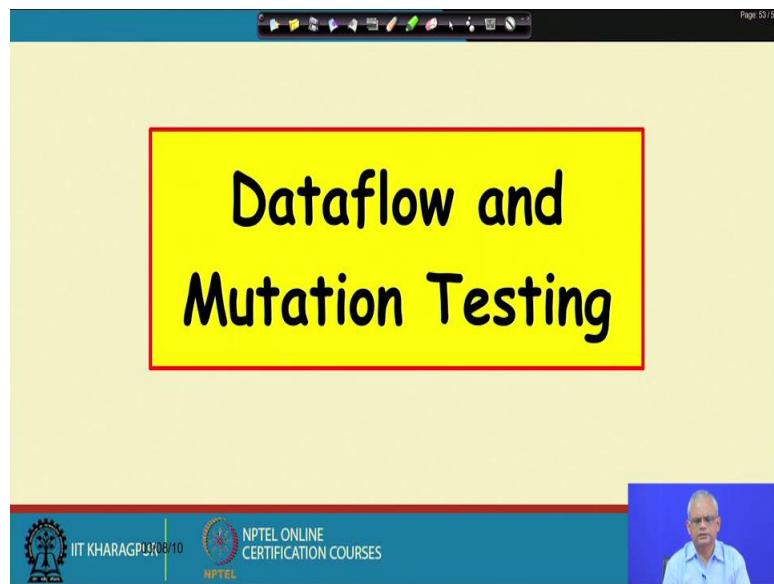
- From maintenance perspective,
 - Limit cyclomatic complexity of modules
 - To some reasonable value.
 - Good software development organizations:
 - Restrict cyclomatic complexity of functions to a maximum of ten or so.

At the bottom, there is a blue footer bar with the IIT Kharagpur logo, the text 'IIT KHARAGPUR', the NPTEL logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player window showing a man in a light blue shirt.

Normally, the companies require their programmers to keep the cyclomatic complexity to a reasonable value. And of course, if you think of it means that every function should have only a few loops and decision statements.

Many organizations require that programmer should not write functions having more than 10 cyclomatic complexity, because otherwise the complexity of understanding debugging becomes extremely large.

(Refer Slide Time: 23:40)



So, we saw that the cyclomatic complexity metric has many applications. We saw how to measure the cyclomatic metric for a program and the applications of the cyclomatic metric; it indicates the difficulty level of testing the program, the difficulty level of understanding the program, it correlates with the final reliability of the program and so on.

We saw that the cyclomatic metric is important, many organizations restrict it to 10 or so for a unit and it is called as a structural complexity metric or the psychological complexity metric. The structural metric is a more popular term that is used. Now, let's look at another test technique that is a white box test technique called as the data flow testing and then we look at the mutation testing.

(Refer Slide Time: 24:58)

White Box Testing: Quiz

1. What do you mean by coverage-based testing?
2. What are the different types of coverage based testing?
3. How is a specific coverage-based testing carried out?
4. What do you understand by fault-based testing?
5. Give an example of fault-based testing?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, before that a small quiz here what do you mean by coverage based testing? We had seen that coverage based testing implies that certain program elements should be executed by the test suite.

We measure the extent to which the program elements are executed by test case and that we give a percentage coverage achieved. The elements can be the statements, the control transfer, edges, paths and so on. Now, what are the different types of coverage testing that we discussed? We discussed statement coverage, we discussed decision coverage, the basic condition coverage, condition decision coverage, MC/DC coverage, multiple condition coverage, and then the path coverage.

How is a specific coverage-based testing carried out? We saw that designing test cases for coverage based testing is non-trivial except for extremely small programs. And therefore, the test cases are designed and as they are executed impossibly random test cases, random values. As they are executed, the dynamic analyzer tool keeps track of what is the path coverage achieved.

We keep on giving inputs until the path coverage becomes acceptably large, but one thing need to mention here is that 100 percent path coverage may not be achievable for non-trivial programs, because there can be infeasible paths. What do you understand by fault based testing? That is the next question. In a fault based testing we introduce faults into the program and run the test cases and see if those faults are getting detected.

The way to introduce fault is called as mutation. Each time introduce a fault we call it as a mutation of the program. And then for on the mutated program, we run the test cases and see if the test cases are able to detect the fault that we have introduced. Give an example of a fault based testing? So, mutation testing is an example of a fault based testing. We are at the end of this lecture, we will stop here and we will discuss about the data flow and mutation testing in the next lecture.

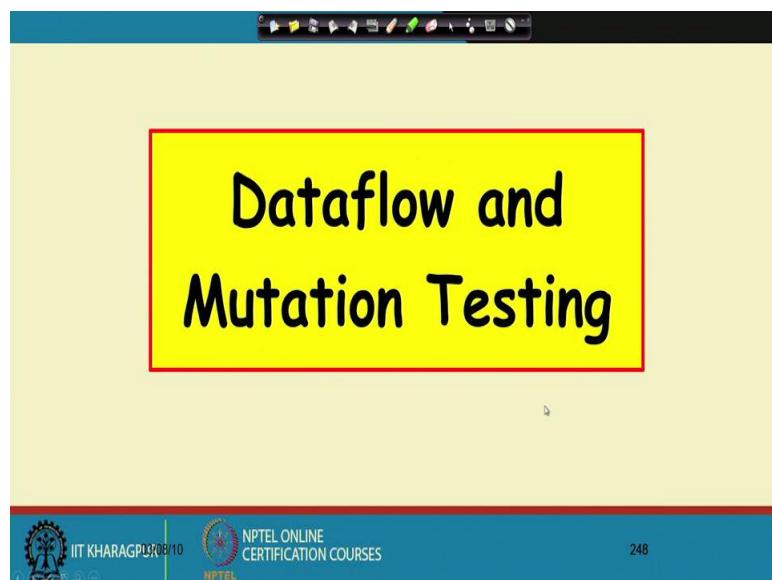
Thank you.

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

Lecture – 60
Dataflow and Mutation Testing

Welcome to this lecture. Over the last few lectures, we have looked at various white box testing techniques. Specifically, we have looked at the coverage based testing, various types of coverage, various elements. Today, we will look at one more coverage based testing which is Dataflow testing and then we will look at fault based testing which is Mutation Testing.

(Refer Slide Time: 00:49)



So, first we will look at Dataflow testing and then will look at Mutation Testing.

(Refer Slide Time: 00:53)

White Box Testing: Quiz

1. What do you mean by coverage-based testing?
2. What are the different types of coverage based testing?
3. How is a specific coverage-based testing carried out?
4. What do you understand by fault-based testing?
5. Give an example of fault-based testing?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

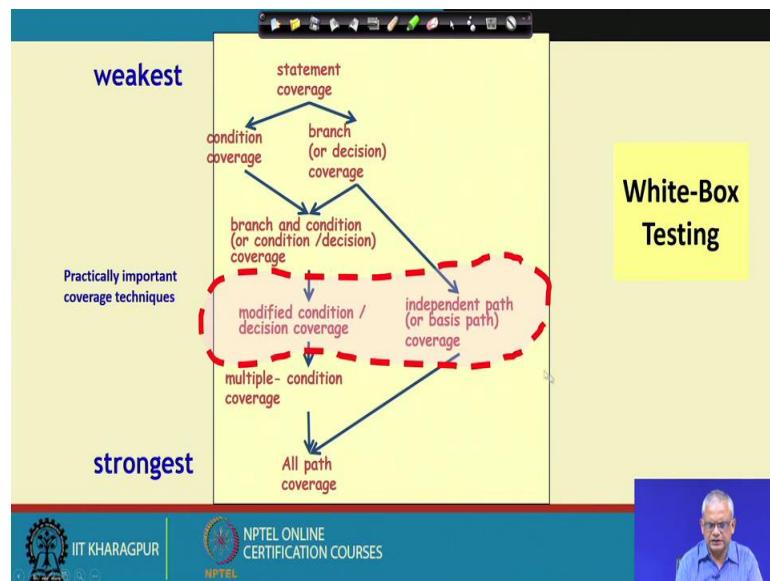
But before we get started, just a small quiz on White Box Testing. What do you mean by coverage-based testing? As you know by coverage-based testing, we mean that certain elements of a program need to be covered or executed by test case. The elements can be statements, control flows, decisions, conditions and so on. What are the different types of condition based coverage based testing? We had discussed many coverage based testing; starting with the statement coverage, then we had looked at the basic decision coverage, then multiple condition coverage, multiple condition decision coverage, MC/DC coverage and so on.

How is testing carried out in using a specific coverage-based testing? This we had said that you do not design test cases for a coverage based testing. But, you test the program using randomly generated test cases and then measure the coverage using a dynamic analysis tool and keep on giving test inputs, until you reach the desired coverage value. What do you understand by fault based testing?

Here, we had said that unlike the coverage based testing, here we check whether the test cases are able to detect certain types of faults and for this, we inject fault into the program which we call as a mutated program and then, test using all the test cases and see, if the fault is getting exposed; if the fault is getting exposed by the test cases, then we know that the test cases are ok. But if the test cases are not able to detect the fault, then we know that testing is inadequate and we need to design additional test cases to

detect the fault. Give one example of fault-based testing? And we had said that Mutation Testing is a prominent example of fault-based testing and today, we are going to discuss about Mutation Testing.

(Refer Slide Time: 03:50)



We had discussed various types of testing. Now, let's see what is the inclusion relation between various types of coverage? The strongest coverage is known by all path coverage and the weakest coverage is known as the statement coverage and then, intermediately, we have condition coverage and then the branch or decision coverage which are stronger than the statement coverage.

We have condition decision coverage which is stronger than both branch coverage and condition coverage and then we have MC/DC coverage which is stronger than the branch that is condition decision coverage. Then, we have path coverage, the basis path coverage or independent path coverage and that is stronger than the branch coverage and then we have the multiple condition coverage or the MCC which is stronger than the MC/DC coverage.

But, what about how do we compare MC/DC coverage with the path coverage? They are not strictly comparable; they are complimentary tests which means that for testing a program, we need to do both MC/DC test and the path test and this is popularly done for any non trivial software, we need to do both path testing and the MC/DC testing. Now, let's first look at the dataflow testing and then, we will look at the mutation testing.

(Refer Slide Time: 05:43)

Data Flow-Based Testing

- Selects test paths of a program:
 - According to the locations of
 - Definitions and uses of different variables in a program.

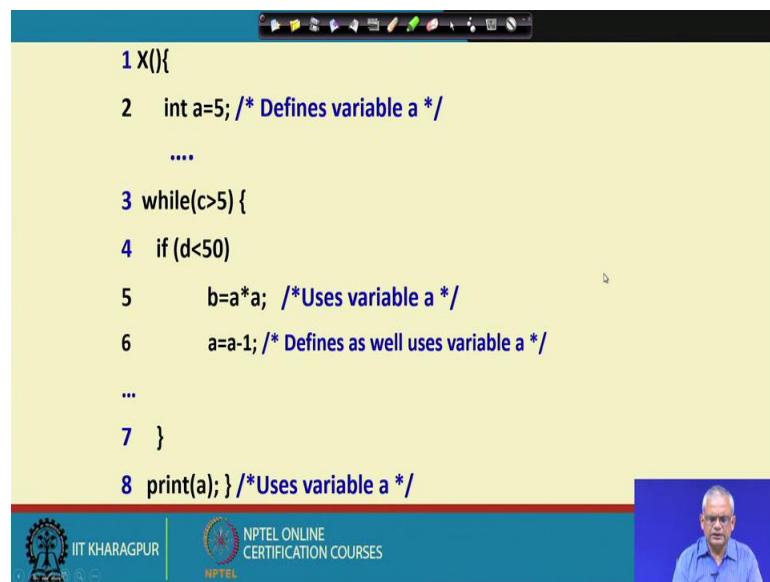
a=b;
b=c*d;
d=a;

IIT Kharagpur | NPTEL ONLINE CERTIFICATION COURSES

In data flow testing, we see that whether in the program, certain definitions and uses of variables exercised. So, here given a small program $a = b$, $b = c * d$, $d = a$; we see here that a is assigned the value here. We say that a is defined and the next statement b is defined, the third statement d is defined; but it uses a . So, a is defined in the first statement and is used in the last statement and this we call as a DU path.

So, the definition and use for various variables, we see that whether those are covered and that is the main idea between the dataflow testing. We can think of that the data is defined in the first statement and is used in the last statement here in the given program in the figure above or in other words, the data flows from the first statement to last statement and here, we check whether all types of data flows are getting tested or not.

(Refer Slide Time: 07:35)

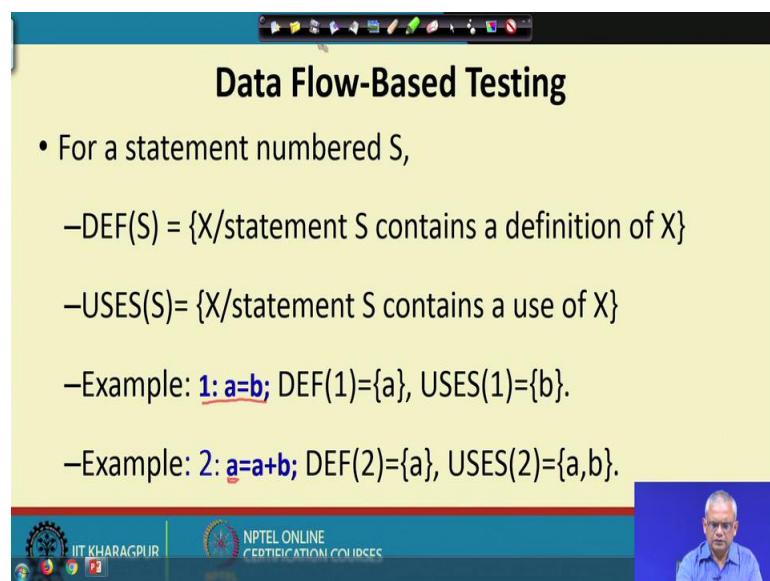


```
1 X(){  
2     int a=5; /* Defines variable a */  
    ...  
3     while(c>5){  
4         if (d<50)  
5             b=a*a; /*Uses variable a */  
6             a=a-1; /* Defines as well uses variable a */  
        ...  
7     }  
8     print(a); } /*Uses variable a */
```

The screenshot shows a computer desktop with a window displaying a C program. The program defines a function X() that initializes variable a to 5, contains a loop that increments a until it reaches 50, and prints the value of a. Below the code, there are logos for IIT Kharagpur and NPTEL, and a video player showing a man speaking.

Let's look at this program. This is the function and the first statement assigns a value to a we say that it defines the variable a. Now the fifth statement it uses the variable a and defines variable b; the sixth statement defines a and also uses a. Similarly, the third statement uses c; fourth statement uses d; the eighth statement uses a. So, given any statement in a program, we can find out what are the variables that the statement defines or uses.

(Refer Slide Time: 08:34)



Data Flow-Based Testing

- For a statement numbered S,
 - DEF(S) = {X/statement S contains a definition of X}
 - USES(S)= {X/statement S contains a use of X}
 - Example: 1: a=b; DEF(1)={a}, USES(1)={b}.
 - Example: 2: a=a+b; DEF(2)={a}, USES(2)={a,b}.

The screenshot shows a slide titled "Data Flow-Based Testing". It lists the definitions and uses of a variable for a statement S. It includes examples for statements 1 and 2, showing how DEF and USES sets are calculated. Below the slide, there is a video player showing a man speaking.

Here in the dataflow based testing, first we compute all the DEF and USES set. If S is a statement, a program statement, then the set $\text{DEF}(S)$ is the set of all variable X such that the statement defines the variable X. So, if $b=a$ is a statement, then the DEF set for the statement will be b; b is defined $b = a$ and USES will be the all the variables that are used in the statement.

Let's take some examples, if we look at a statement like $a = b$; then, $\text{DEF}(1)$ because the statement number is 1; $\text{DEF}(1)$ is a and $\text{USES}(1)$ is b. Similarly, if the statement number is 2 and it is $a = a + b$; then, $\text{DEF}(2)$ is a because a is defined and also if we look at the right hand side of this equality a and b are also used here and therefore, we write $\text{USES}(2)$ is a, b.

(Refer Slide Time: 10:27)

Data Flow-Based Testing

- A variable X is said to be **live** at statement S_1 , if
 - X is defined at a statement S:
 - There exists a path from S to S_1 not containing any definition of X.

IIT Kharagpur | NPTEL Online Certification Courses

Now, we define the term live at a statement S_1 . We say a variable to be live at a statement S_1 , if the variable is defined at some statement S and there is a path from S to S_1 not containing any definition of X. So, that means if there are statements in the program and we have 1 statement here which is S and $\text{DEF}(S)$ is X; that means, it must be having of the form $X = a + b$ or something.

There is another statement S_1 which has a use for X. So, $\text{USE}(S_1)$ is the set X. It must be something like $P = X^2$ or some such thing. So, that the value X, the value of the variable X is used here and then the statements in between they have no definition for X. So, if we define X to be 5, then the 5 value is available in S_1 and this is called as the

variable X is called as live at S1. It is defined at S and use at S1 and we say that X which is defined by at S is live at S1.

(Refer Slide Time: 12:56)

Definition-use chain (DU chain)

- $[X, S, S1]$,
- S and S1 are statement numbers,
- $X \in \text{DEF}(S)$
- $X \in \text{USES}(S1)$, and
- the definition of X in the statement S is live at statement S1.

Now, we can define a DU chain. It is the set of all variable which are live which is defined at S and used at S1. It is the set of all variables X which is defined at S and used at S1. So, in a program we might have different variables defined at different statements and which are live at other statements.

So, a variable might be defined at some statement and it is live at some other statements; the statement number maybe 1 and this maybe 5, 7, 20 etc. So, 1 to 5 is a DU chain; 1 to 20 is a DU chain; 1 to 7 is a DU chain. So, a program can contain large number of DU chains because there are many variables and each variable which is defined at a place maybe live at several statements.

(Refer Slide Time: 14:31)

```
1 X(){  
2     int a=5; /* Defines variable a */  
3     While(c>5){  
4         if (d<50)  
5             b=a*a; /*Uses variable a */  
6         a=a-1; /* Defines variable a */  
7     }  
8     print(a); } /*Uses variable a */
```

ADU Chain) consists of a definition, of a variable and all the uses, reachable from that definition without any other intervening definitions.

A DU chain consists of a definition of a variable and all uses that are reachable from the definition without any intervening definition. Just to give an example here, if this is a function; a is defined here in the second statement and a is used in the fifth statement. So, a, 2 and 5 is a DU chain. We can find other DU chains here; maybe b is defined here and used at some other place and so on. A program typically has a large number of DU chains.

(Refer Slide Time: 15:40)

Data Flow-Based Testing

- One simple data flow testing strategy:
 - Every DU chain in a program be covered at least once.
- Data flow testing strategies:
 - Useful for selecting test paths of a program containing nested if and loop statements.

The simplest dataflow testing strategy is to require that every DU chain in the program is executed at least once; every DU chain in the program is executed at least once. But what about the path testing; does it ensure that all DU chains are covered? No. But what about all DU chains covered, does it imply all paths covered? No.

(Refer Slide Time: 16:30)

A screenshot of a computer screen displaying a C program. The code is as follows:

```
• 1 X(){}
• 2 B1; /* Defines variable a */
• 3 While(C1) {
• 4   if(C2)
• 5     if(C4) B4; /*Uses variable a */
• 6   else B5;
• 7   else if(C3) B2;
• 8   else B3; }
• 9 B6 }
```

Red arrows highlight the flow of variable 'a' from its definition in B1 through various branches of the while loop. A yellow box in the bottom right corner contains the text "Data Flow-Based Testing".

Let's look at 1 example. So, this is the program, where we have several statements that we have written as a block. So, there are 5 blocks here B1, B2, B3, B4, B5 as represented in the above program and see here that once it enters a while loop, the different blocks can be reached based on the outcome of the conditions C2, C4, C3 and C1.

Now, if you want to compute the DU chains here, if a certain variable X is defined in all the blocks and used in all the blocks. Then, how many DU chains are there? So, we have B1 to if the variable is defined in B1 and used in all these B4, B5, B2, B3. So, it can reach B2; it can reach B5; it can reach B2; it can be live at B2 and it can be live at B5. Now, what about if it is defined that B4 and used in B5? Yes, in the loop it can be defined here and used here. Similarly, it can be defined in B4 and used in B2 and so on.

(Refer Slide Time: 18:08)

The slide has a yellow header bar with the title 'Data Flow-Based Testing'. The main content area contains the following text:

- [a,1,5]: a DU chain.
- Assume:
 - $\text{DEF}(X) = \{\text{B1, B2, B3, B4, B5}\}$
 - $\text{USES}(X) = \{\text{B2, B3, B4, B5, B6}\}$
 - There are 25 DU chains.
- However only 5 paths are needed to cover these chains.

At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

So, based on that computation, if B1, B2, B3, B4, B5 define the variable X and B2, B3, B4, B5, B6 use the variable X; then, we can see that from B1 it can come to B2, B3, B4, B5 and similarly, if it is defined in B2, it can also be used any of these. So, the total number of DU chains can become 25. But then, for the given program there are only 5 paths that are possible.

(Refer Slide Time: 18:43)

The slide has a yellow header bar with the title 'Mutation Testing'. The main content area is empty. At the bottom, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a video player showing a man speaking.

Because there are 4 conditionals 1, 2, 3, 4; 4 conditionals and that means, the cyclomatic complexity of this is 5 and 5 independent paths are possible and therefore, only 5 test

cases can cover all the 5 independent paths; but then, the DU chains can be many. Now, let's look at Mutation Testing.

(Refer Slide Time: 19:31)

Main Idea

- Insert faults into a program:
 - Check whether the test suite is able to detect these.
 - This either validates or invalidates the test suite.

(Refer Slide Time: 19:35)

Mutation Testing

- In this, software is first tested:
 - Using an initial test suite designed using white-box strategies we already discussed.
 - After the initial testing is complete,
 - Mutation testing is taken up.
 - The idea behind mutation testing:
 - Make a few arbitrary small changes to a program at a time.**

In mutation testing, an initial test suite is designed by using some coverage based testing and then, once coverage based testing is over; then mutation testing is taken up. The main idea is to inject faults i.e., small changes into the program at a time and run the set of test cases. See if the change that we made is being detected, if it is not being detected

by the test cases; then, we know that the test cases are inadequate. We need to introduce additional test cases until the change that was made is getting detected.

So, that is the main idea here. Given a program each time we inject faults by making small change in the program and then, check whether the test suite is able to detect the change and then based on that we either say that the test suite is ok and we inject one more fault or we just say that the test suite is ok and if it is not detected, we say that we need to add more test cases and we invalidate the test suite that is we say the test suite is not sufficient, we add more test cases to that.

(Refer Slide Time: 21:08)

The slide has a title 'Mutation Testing Terminology' and a bulleted list:

- Each time the program is changed:
 - It is called a **mutated program**
 - The change is called a **mutant**.

At the bottom, there are logos for IIT Kharagpur and NPTEL Online Certification Courses.

So, here each time we make one small change and the change is called as a mutant and once we have the mutant, the program is called as a mutated program.

(Refer Slide Time: 21:26)

The slide has a yellow header bar with the title 'Mutation Testing'. Below it is a white content area containing a bulleted list. At the bottom is a blue footer bar with logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

- A mutated program:
 - Tested against the full test suite of the program.
 - If there exists at least one test case in the test suite for which:
 - A mutant gives an incorrect result,
 - **Then the mutant is said to be dead.**

And the mutated program is tested using the full test suite and then if there is at least one test case which fails, then we say that the mutant is dead; but if the mutant is not dead, that is all test cases are exhausted and none of the test cases has failed. Then, we say that the mutant survives and we need to design additional test cases so that the mutant can get killed.

(Refer Slide Time: 22:05)

The slide has a yellow header bar with the title 'Mutation Testing'. Below it is a white content area containing a bulleted list. At the bottom is a blue footer bar with logos for IIT Kharagpur and NPTEL, and a video player showing a person speaking.

- If a mutant remains alive ---even after all test cases have been exhausted,
 - **The test suite is enhanced to kill the mutant.**
- The process of generation and killing of mutants:
 - **Can be automated by predefining a set of primitive changes that can be applied to the program.**

But here, for given program large number of mutants are possible because we can make many small changes to a program and therefore, when this is typically done by

automated tool. In the automated tool it makes small changes and it runs the test suite and then, if there is a test suite fails, then the mutant is discarded and the new mutant is created.

(Refer Slide Time: 22:39)

Mutation Testing

- Example primitive changes to a program:
 - Deleting a statement
 - Altering an arithmetic operator,
 - Changing the value of a constant,
 - Changing a data type, etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But then, what are the types of mutants that can be possible; one mutant is delete test statement in a program; another mutant may be alter an arithmetic operator that is substitute a + with - or a *, with a / and so on. The third type of mutant maybe changes the value of a constant. A fourth type of mutant maybe change the data type that is an int into float. A fifth type of mutant maybe to change the logical operator = with != or something.

So, given a program, we can have several primitives defined primitive types of changes and then we carry out throughout the program and this may lead to millions or billions of mutants getting created.

(Refer Slide Time: 23:51)

The slide has a yellow header bar with the title 'Traditional Mutation Operators'. The main content area contains a bulleted list of mutation operations:

- **Deletion of a statement**
- Boolean:
 - **Replacement of a statement with another**
eg. == and >=, < and <=
 - Replacement of **boolean expressions** with **true** or **false** eg. **a || b** with **true**
- **Replacement of arithmetic operator**
eg. * and +, / and -
- **Replacement of a variable** (ensuring same scope/type)

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video player window showing a person speaking.

Some of the traditional mutation operations are Delete a statement; Replace 1 statement with another. We can change some operators < with <=. We can replace the Boolean expression entire Boolean expression with either true or false. We can replace arithmetic operator * with + and so on. We can replace a variable a with b and ensuring that the scope and type of the variable are the same.

(Refer Slide Time: 24:42)

The slide has a yellow header bar with the title 'Underlying Hypotheses'. The main content area contains a bulleted list of hypotheses:

- Mutation testing is based on the following two hypotheses:
 - **The Competent Programmer Hypothesis**
 - **The Coupling Effect**

A red text at the bottom states: 'Both of these were proposed by DeMillo et al., 1978'

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a small video player window showing a person speaking.

Now, let's see the Underlying Hypotheses behind the mutation testing; there are 2 main hypotheses. The first one is called as Competent Programmer Hypothesis. So, this

hypothesis says that programmers write programs well, they are good programmers. But occasionally, they make small programming errors, maybe while writing, they may exchange a + with a - or they may exchange Boolean operator, forget to write a statement etc. And the other hypothesis is called as coupling effect which says that the more complex bugs in a program are caused by a combination of several simple bugs. The simple bugs are called as the programming bugs just a small programming error and a complex bug maybe algorithmic bug; the programmer did not understand the algorithm well and so on.

So, the complex bugs are hard to introduce, but the coupling effect says that if we introduce several small bugs, they may act like a complex bug and both these hypotheses were proposed by DeMillo in 1978. And this form the cornerstone of the mutation testing and this ensures that mutation testing is effective under these two assumptions the competent programmer hypothesis and the coupling effect.

(Refer Slide Time: 26:34)

The Competent Programmer Hypothesis

- Programmers create programs that are close to being correct:
 - Differ from the correct program by some simple errors.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The competent programmer hypothesis says that at anytime, the programmer may call, may introduce only very simple bugs.

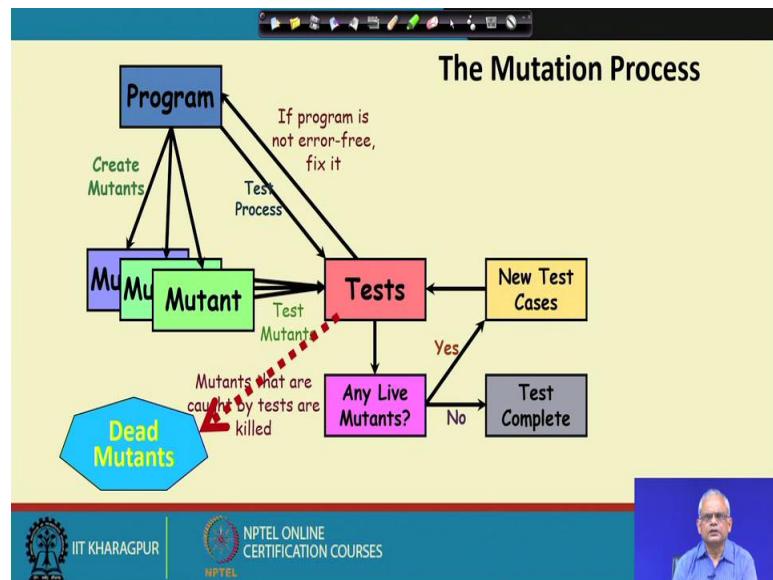
(Refer Slide Time: 26:46)

The Coupling Effect

- Complex errors are caused due to several simple errors.
- It therefore suffices to check for the presence of the simple errors

The Coupling effect, complex errors are caused due to several simple bugs and due to the coupling effect, it is enough if we check with simple mutants. We do not have to really simulate or create very complex mutants; just simple mutants several of them will suffice.

(Refer Slide Time: 27:16)



Now, we can pictorially depict the mutation testing process. We first use some initial test cases based on coverage based testing and so on, we test it and if there are problems we fix it and then, the mutation testing process starts. We create a large number of mutants

and for each mutant, we test using the test cases and if the test case fails; then, the mutant is dead.

But, if there are any mutant is live, then we need to design further test cases. We augment the test cases and then we again carryout testing with other mutants and finally, as all mutants are being killed, the test is complete. So, we discussed the dataflow testing which is a coverage based testing and we looked at the mutation testing process and with this, we will complete this lecture.

Thank you very much.

**THIS BOOK
IS NOT FOR
SALE
NOR COMMERCIAL USE**



(044) 2257 5905/08



nptel.ac.in



swayam.gov.in