OBJECT ORIENTED ANALYSIS AND DESIGN

Prof. Partha Pratim Das

IIT KHARAGPUR

INDEX

S.No.	Topic	Page No.
	WEEK- 1	
1	Challenges in Software Engineering	01
2	Complexity of Software	18
3	Complexity of Software (Contd.)	32
4	Structure and Attributes of a Complex System	37
5	Structure and Attributes of a Complex System (Contd.)	55
6	Object-Oriented Analysis and Design	63
7	Bringing Order to Chaos	78
8	Bringing Order to Chaos	93
	WEEK- 2	
9	Evolution of Object Models - Programming Languages and Paradigms	106
10	Foundations of the Object Model - OOA, OOD and OOP	126
11	Foundations of the Object Model – OOA, OOD and OOP	140
12	Elements of Object Model (Major): Abstraction and Encapsulation	152
13	Elements of Object Model (Major): Abstraction and Encapsulation (Contd.)	173
14	Elements of the Object Model (Major): Modularity and Hierarchy	179
15	Elements of the Object Model (Major): Modularity and Hierarchy (Contd.)	194
16	Elements of the Object Model (Minor): Typing, Concurrency and Persistence	206

17	Elements of the Object Model (Minor): Typing, Concurrency and Persistence (Contd.)	218

WEEK- 3

18	Nature of an object: State, Behavior and Identity	229
19	Nature of an object: State, Behavior and Identity (Contd.)	243
20	Relationships among objects	257
21	Relationships among objects (Contd.)	273
22	Nature of a class: Interface and Implementation	279
23	Nature of a Class: Interface and Implementation (Contd.)	294
24	Relationships among classes	301
25	Relationships among objects (Contd.)	314
26	How to Build Quality Classes and Objects	325

WEEK- 4

27	Tutorial I LMS	341
28	How to Identify Classes and Objects	349
29	Identification of Classes, Objects and Relationship in LMS	365
30	Identification of Classes, Objects and Relationship in LMS (Contd)	382
31	Identification of Classes, Objects and Relationship in LMS (Contd)	395
32	Identification of Classes, Objects and Relationship in LMS (Contd.)	406

WEEK- 5

33	Overview of UML	420
34	SDLC Phases and UML Diagrams	437
35	Use-Case Diagrams Part I	451
36	Use-Case Diagrams: Part II	463
37	Use-Case Diagrams: Part III	478

WEEK- 6

38	Class Diagrams: Part I (Class, Property and Operation)	496
39	Class Diagrams: Part 2 (Association, Weak and Strong Aggregation)	512
40	Class Diagrams – Part 3 (Generalisation, Dependency, Constraints)	532
41	Sequence Diagrams - Part 1	550
42	Sequence Diagrams - Part 2	569

WEEK- 7

43	Communication Diagram	586
44	Activity Diagrams Part I	603
45	Activity Diagrams Part II	619
46	Activity Diagrams Part III	633
47	Interaction Overview Diagram	648

WEEK- 8

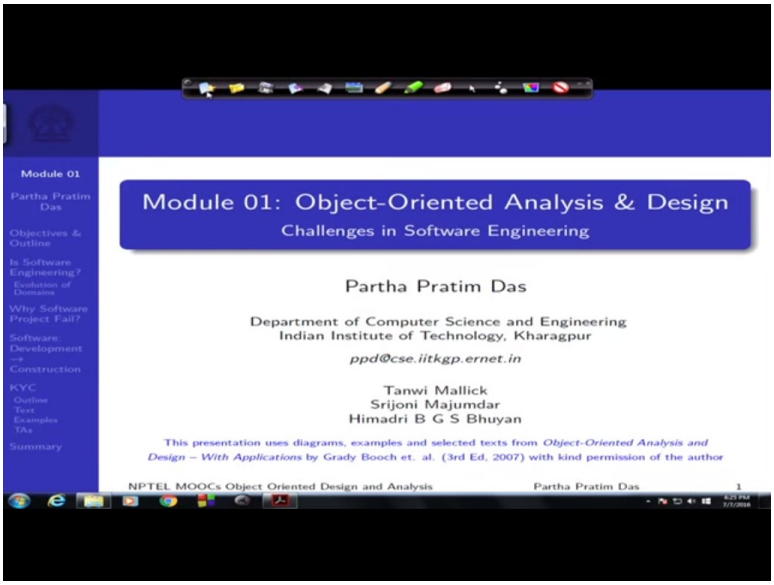
48	State Machine Diagrams: Part I	663
49	State Machine Diagrams: Part II	681
50	State Machine Diagrams: Part III	698
51	Various UML Diagrams	714
52	Closing Comments	731

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 01
Challenges in Software Engineering

Welcome to the moocs course on object-oriented analysis and design. This is module 1.

(Refer Slide Time: 0:46)



Here, we will start by setting the background and creating the motivation for, why is it required to study understand object-orientation? How does it plea with analysis of real life problems and how it can be effectively used in design of complex software. Certainly you all are familiar with object-oriented programming; at least I assume that you know glimpses of that. You aware that C++ or Java are some of the most popular object-oriented programming languages.

Now the fact of the matter is that while you design, implement and maintain a complex software, certainly using a object-oriented programming language turns out to be of great help but merely the use of a object-oriented programming language, that a programming language which supports definition, creation, management, destruction of objects, merely the use of that language as a programming vehicle does not solve all the issues of software system.

And in reality, much of the problems that software systems encounter or the developers and as well as a users of the software systems encounter are not just limited to the programming code. Several actually

significant part of the problems start at the specification of the problem itself, where the user talks to the developer and wants to express that this is what I want as a software. Now in some cases, users will directly do that if you are have an enterprise and you are trying to create,

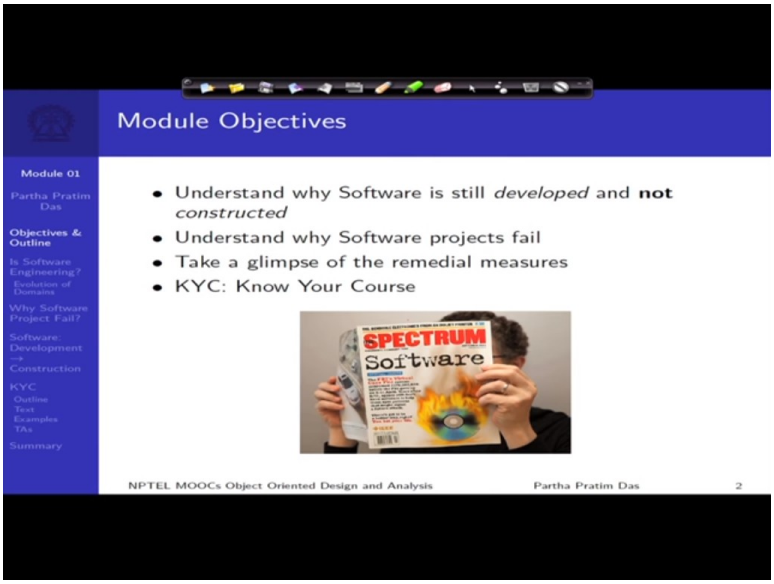
Get a system created by a software vendor, say to manage your employees, to manage the leaves that your employees take, you will directly talk to the vendor and get a system created for your organizational use or it could be that it is a generic software product where several independent unknown users have talked to several parts of the developers to express their requirements. And all those requirements had been canned into a complete set, a consistent set to develop the product.

An example could be Microsoft word, excel, this office suite and mail clients, browsers, they have not been developed for any specific needs of any specific organization or individual. They have been created for generic use by a large number of individuals, groups and organizations and therefore we call them the products, irrespective of which kind of software system. We talk about the specification, design, analysis and finally implementation, maintenance, preservation are all critical issues.

That needs to be effectively handled, particularly for a industrial strength large scale software. So you would be familiar that this whole exercise is a prescient in the name of software engineering like civil engineer's construct buildings, mechanical engineers construct automobiles, machines. Electrical engineers construct power plants, air conditioners, fans, software engineers are supposed to build software and that discipline is known as software engineering. So to understand the requirements.

Or rather more the motivation of why we need to engage in object-oriented analysis and design, we will start by taking a brief look into, what are the challenges that software engineering or software engineers face, that will be the main state of our current module.

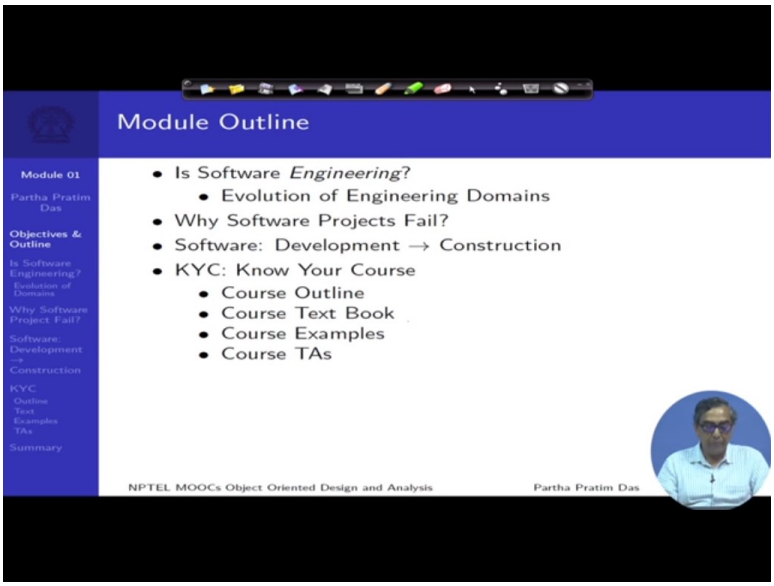
(Refer Slide Time: 05:09)



So to specify, the specific objective would be to understand why software is still developed and not constructed. Please note these two words development and construction. If you tell into any field of engineering other than software, then you will typically find that the word used at the, to define the activity to define the product is called constructions. Buildings are constructed, power plants are constructed but software is typically developed, not constructed.

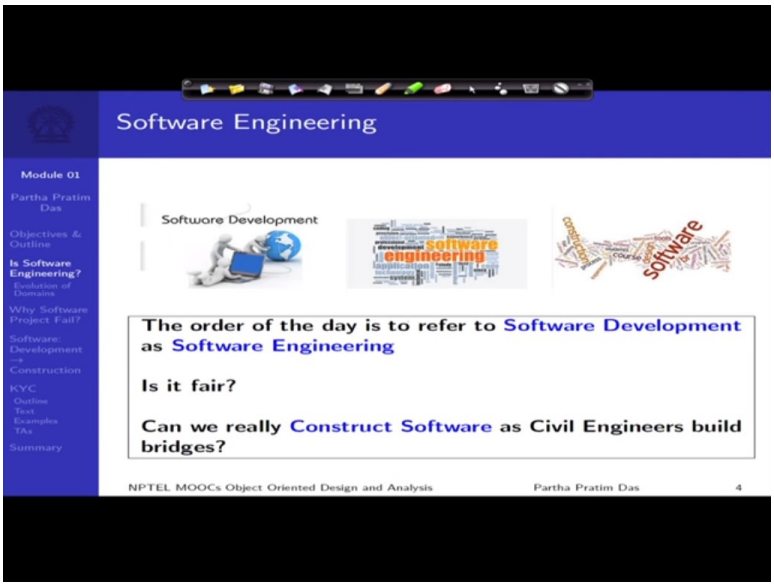
So we will take a little bit of round in terms of why this difference in semantics. We will try to understand why software projects fail and take a glimpse into some of the remedial measures and finally, we will end the module with the KYC, that is know your course. That is what we are going to cover in this whole.

(Refer Slide Time: 06:10)



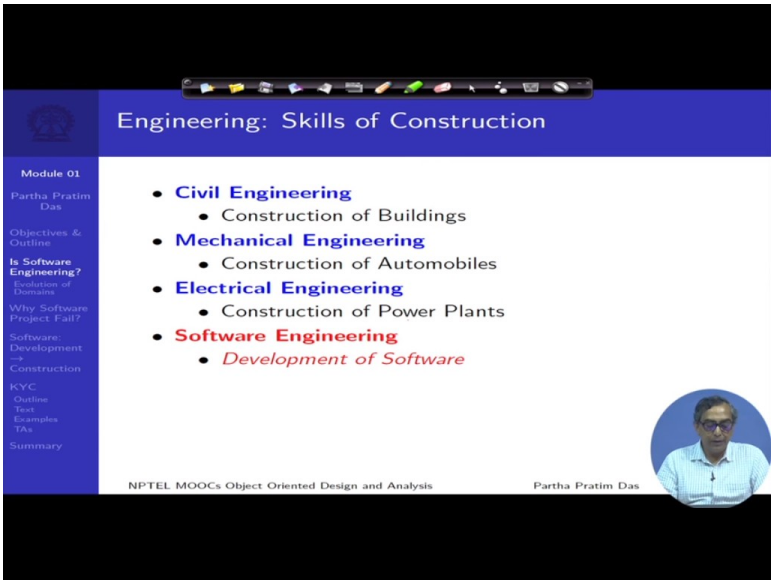
The outline would be is software. What is software engineering? Or rather is software really engineering and why projects fail and so on. And the convention that we will typically follow in all the modules is during the presentation, you will be able to see the outline of the module on the left hand side of every slide and the current subsection in the outline that is being discussed in a given slide will be highlighted on the left.

(Refer Slide Time: 06:43)



So this is the outline that you will have. So now let us look at software engineering, so the order of the day is to refer to the activity of the software development as software engineering. I start by raising a question that is it fair to really pronounce proclaim that we are doing software engineering? If (we)you are doing software engineering then we must be able to construct software as civil engineers build bridges.

(Refer Slide Time: 07:12)



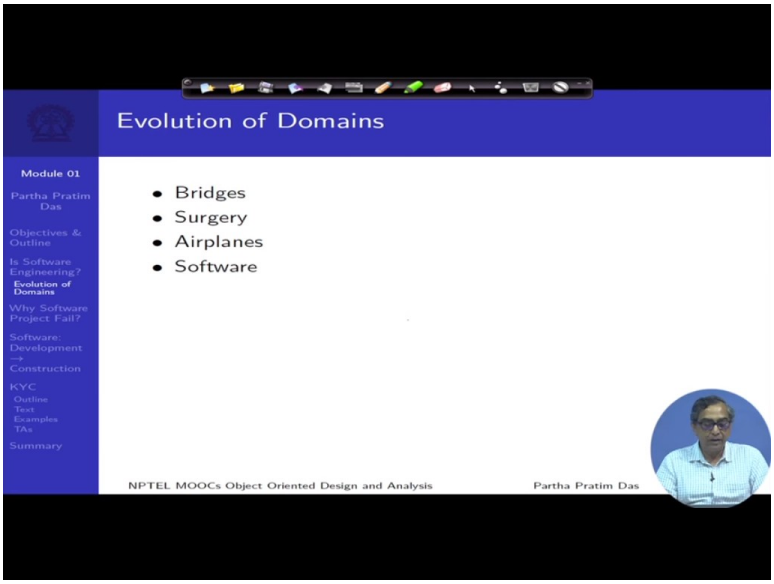
So what is engineering? Engineering in very simple terms can be look at from different angles but the core concept is, it is a scale of construction. Engineering also deals with the ability to manage complexity, If you ,what is the difference between science, theory and engineering? The science tells us the natural laws, principles, mathematics that govern, how systems will behave. We turn them into engineering when we add utility to what we do one or more people one or more organizations (nations groups 7.52) get benefited.

And the second very important factor is we create something which did not exist and in the whole process we really need to learn engineering because there is a complexity to handle if things are small, simple then you do not need to be an engineer. You can, If you just want to analyze as you throw a ball and want to hit a target, what is the force with which you will throw the ball and hit the target. You can do that from your study of physics by using Newton's law of motion.

But when you want to do that with by firing a missile which has to go over 5000 kilometers and still hit a building at the right point, you need to do engineering. Both are actually hitting a target over a distance but what is different in case of missile is, the complexity is very different. So when we talk about engineering we basically talk about how to manage complexity and that the core skill of an engineer is to learn how to what is the complexity and how to handle it.

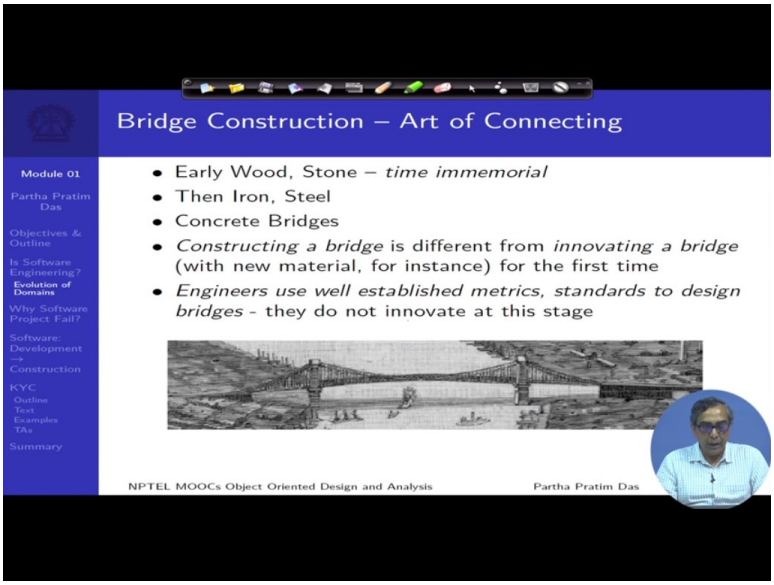
So just to take a glimpse we will take a quick tour this is more of a story telling a quick tour into some of the common fields of engineering and see how they have evolved.

(Refer Slide Time: 09:22)



So we will start with 4 different domains the bridges the surgery aeroplanes and certainly software.

(Refer Slide Time: 09:31)



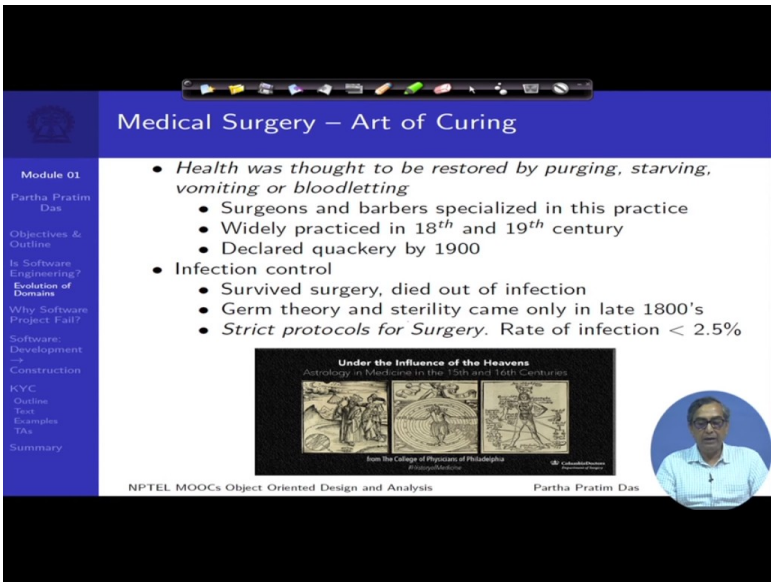
So if you look into bridges, this is one of the earliest constructions that the human civilization had to do from time immemorial we have been constructing bridges, first with wood, stone, then iron, steel, glass, concrete bridges and so on and by now if we are given to construct a bridge then we do not accept this outcome that the bridge may fall that the construction may not succeed. If I construct a flyover I would not expect the flyover to fall over. Let us not talk about incidents in Delhi and Kolkatta that we have seen in last couple of hours.

That's faults in engineering but the civil engineering knowledge guarantees that the bridges will not fall. So constructing a bridge is different from innovating a bridge. We will say we are innovating on a

bridge if we are experimenting with a new material, we have seen bridges with iron, steel, concrete but then if you look at the Google for Grand Canyon, you will find bridges which are completely in glass. So when civil engineers sit down and find out how a bridge can be constructed completely with glass.

And experimenting or the innovating a bridge but when they are actual engineers and constructing a bridge then they are working with established metrics standards proof and processes so that is there is no innovation particularly and the process must succeed.

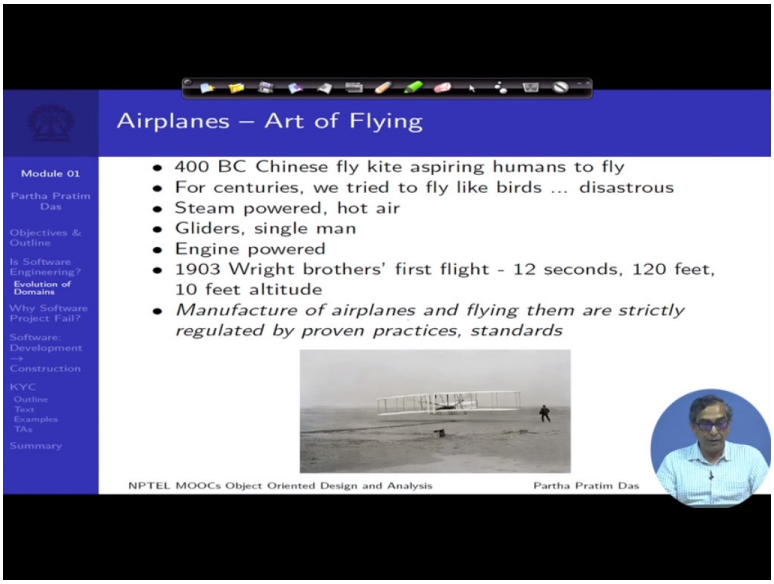
(Refer Slide Time: 11:17)



The same story will reveal itself in other fields, just consider medical surgery. Health naturally has been a issue of concern again from time immemorial and as is documented that surgery and the act of barbers were considered equivalent. Prior to 18th century you had to cut the human body and do something it was declared quackery by 1900 but people slowly, the doctors or surgeons started perfecting on their surgery but most patients did not survive because of the injury infection.

So we learnt how to manage infection in the late 1800s, now if you look into the way the surgery is done every surgery follows a strict protocol. The protocol is nationally and internationally be established proven and the doctor and the team has to specifically follow that protocols to ensure that the surgeries succeeds, the infection is not propagated to the patient and so on and that is the reason that surgery today has become so successful. So that is also a kind of engineering that we see.

(Refer Slide Time: 12:41)

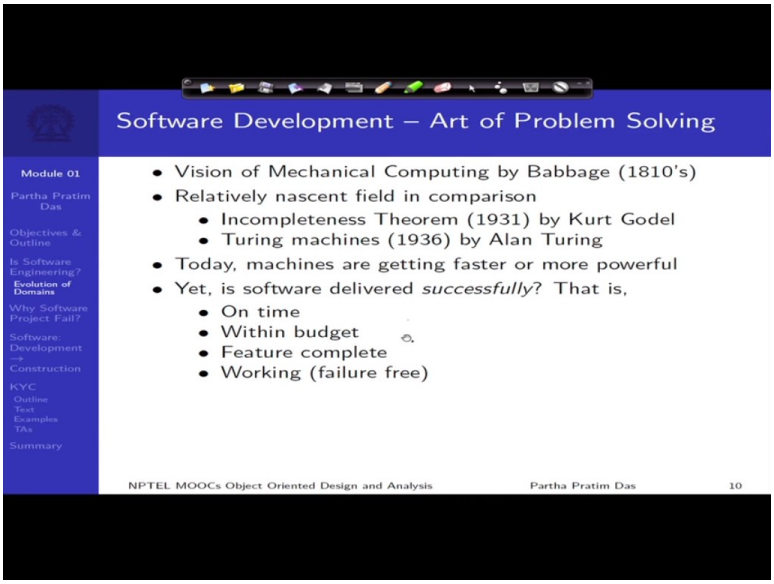


And you can see that the success rate is quite high and without that of course we would be very scared to go to the surgeons. A third domain is aeroplanes many of us certainly loves flying and again the attempt started at around 400 BC with Chinese trying to flying kite and aspiring that humans could also fly. For centuries people has tried to fly as birds with disastrous effects many people have lost their lives.

There have been different means of flying, steam powered, hot air gliding for single man, engine powered and so on. Till about little over a 100 years back the wright brothers finally made the today's prototype of aeroplanes and they could fly again if you look at how the aeroplanes are manufactured how they are flown is strictly regulated governed by proven practices, standards and if those standards are not adhered to then the aero plane will not or the aircraft will not be allowed to fly the pilot will not be able to take the seat and so on.

And that gives us the comfort the guarantee that if I fly in an aircraft then I will reach the destination safely. So in all of these fields the underlined factor is the the confidence or the surety or the guarantee of the success is very very very high and often when things go wrong when things fail they are they fail either because some human being has made a mistake or many several human beings have made some mistake or there is some intentional wrong doing.

(Refer Slide Time: 14:27)

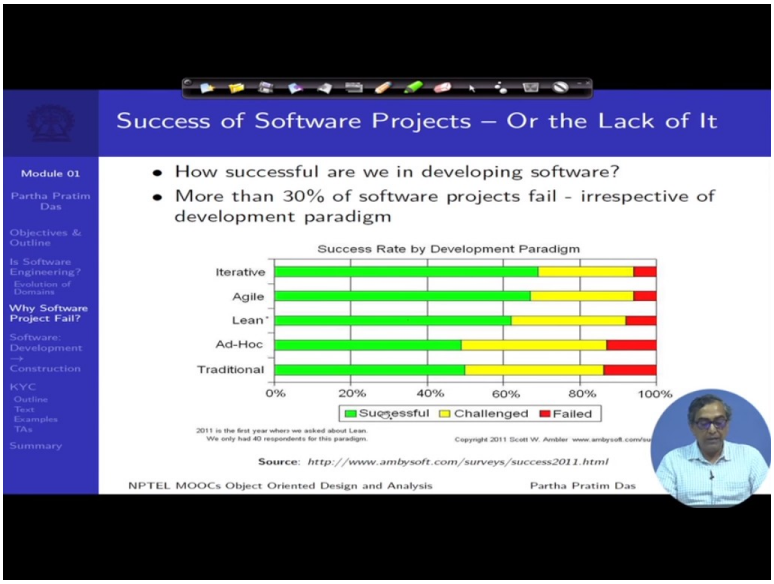


In contrast let us look at software development I love to call it the art of problem solving the vision for computing mechanical computing, automated computing started in early 1800 with Charles Babbage and Adalh and so on but incidentally there was not much development for over a hundred years afterwards. The first significant step towards software algorithms and software systems happened in the 1930s with the 2 seminal works, 1 by Kurt Godel's incompleteness theorem and alan turing , turing machines many of you have been heard of this.

So if you look at then software is not even 100 years old so its its compared to other fields of engineering its very very nascent, very very new and therefore it has not got the time to really prove its processes, setup, its standards and make sure that the success will always be at the end of the route, of course we see a lot of developments happening very fast developments machines are expanding, they are getting faster more powerful, lot of machines yet is the software delivered successfully we all understand that it is not.

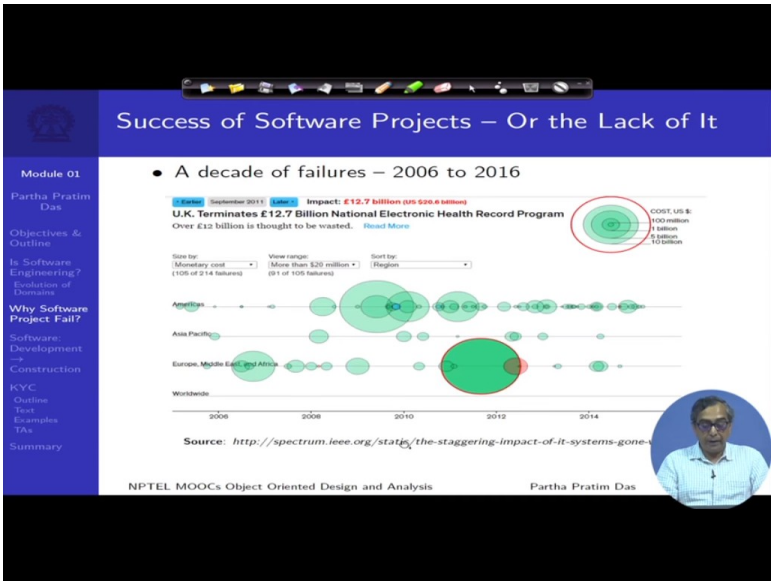
And let me for the sake of being formal in this course define what I mean by a successful delivery of a software it means primarily 4 factors that the project is completed on time. As it was planned there is no extension of the project, it is completed within budget that is there is overshoot of funds, all features that we have promised are completed. It is not that I will do it on time within budget and implement only 50% of the functionality that I wanted and finally it must a be working software that is it should be failure free fault free.

(Refer Slide Time: 16:24)



Now in this background if you look at some of the recent studies these I have picked up from the sources governance, these are 5 years to absolutely recent studies which show that more than 30% of the software projects fail and these this statistics also taken for really large projects if you take into medium size projects, the success rate would be even lower so that is quite alarming.

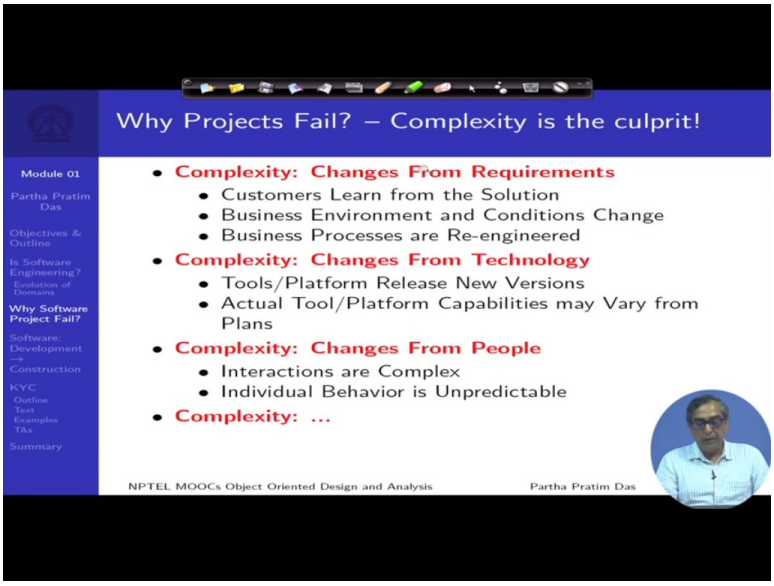
(Refer Slide Time: 16:51)



And this is another data which is very very recent, couple of days back which show that over the last decade 2006 to 2016 , what are the different large number of large projects that have failed. So here this bubbles this circles show the kind of budget that that particular software had. So the selected one that you can see here this one is actually 12.7 billion pound project which had been trminated by the UK government little around the middle of 2011.

And this show that there is continuous irrespective of which region of the world you are talking of there are series of large to very large to huge software projects, which are failing causing several damages to property loss of money and effort and resources so that that is the basic alarm point.

(Refer Slide Time: 17:58)



So we just tried to we will talk about this lot more but here in this introductory part I just quickly try to highlight why software projects fail that can be one line answer. The complexity is the culprit this complexity of several dimensions in software development. So some of them you can you can very quickly understand is the change of requirement when you start developing a software you are told something.

And then as you go through the development the customer would repeatedly come back to you and say that I want something different. Now is it that the customer is not a smart person is it that the customer is not being truthful to you? No please do not misunderstand the customer. The customer is changing the requirement because the customer originally did not know what she wanted she had only had a vague idea. And as you start building the software you show the prototype you show the design you show the first alpha version.

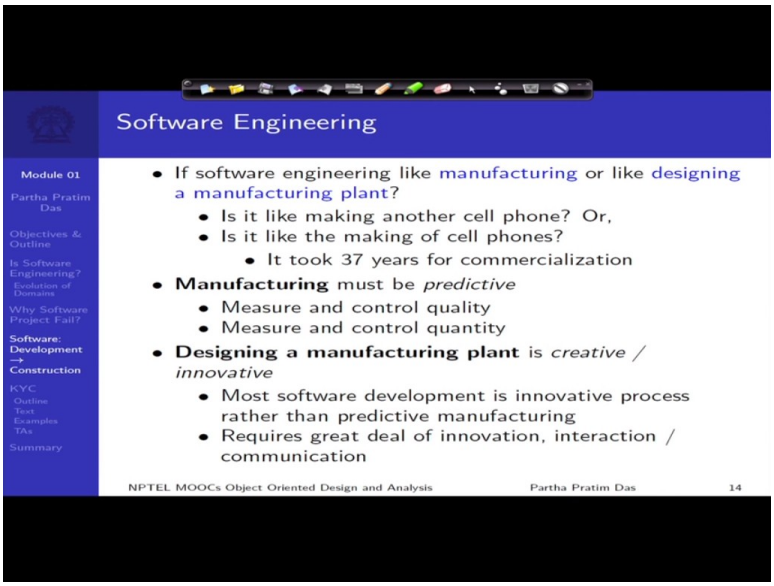
The customer gets more and more ideas, the business environment change, the condition change, processes are really changing. So one complexity big part of complexity come from the change of requirements, the different complexity comes from change from technology, yours software works on system, works on hardware, works on network components, works on different operating systems, third party protocols and so on.

Over the period of your development if it is 1 year, 2 years the target systems that started working with that system itself evolves and the new system is around this. When you started it was windows 8.1, by the time you are in the middle of the project its windows 10, things have changed. The things may change in terms of hardware so being able to adapt to those adds a new complexity to the whole process and there is a severe complexity from people.

Unlike most other fields of engineering where the process of construction itself can is significantly automated. Here in software the process of construction has automation aid but its finally people who develop software, who write the code, so people interaction of people as you will all understand are complex. 2 people have certain kind of interaction, when you put 4 people you get 4, say 2, 4 choose 2 kind of interactions and you have more and more people more and more groups more and more organizations more and more distributed development teams across geographies and so on interaction gets really complex.

And people are some 1 factor in the whole software industry which have an unpredictable behavior, you may everybody have certain uncertainty in terms of how you will behave today and how you will behave tomorrow. We say am in a not in a right mood, I may not be in the 100% fitness and so and so. Complexities in software development come from all of this different dimensions this is not an exhaustive list. We will talk little bit more about of the software complexity in the next module but this is just to give you glimpse in terms of why software projects fail and we will have to live with that.

(Refer Slide Time: 21:21)

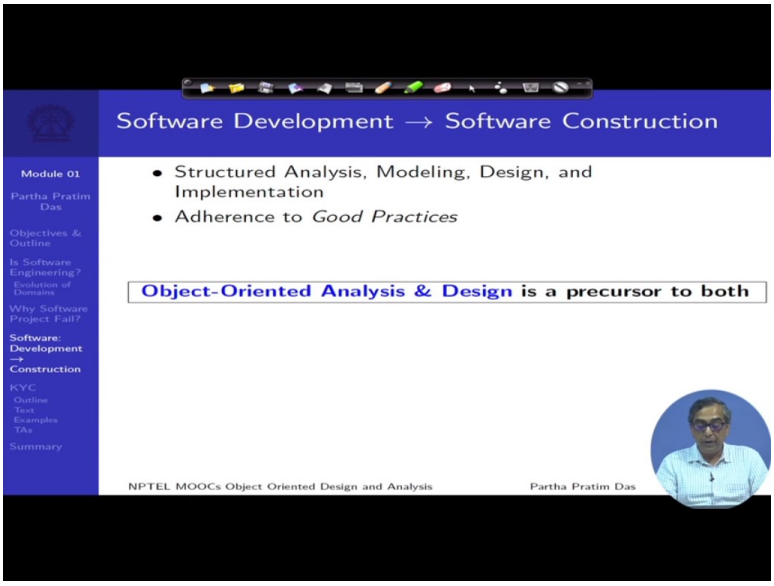


So the question is the more question is, is software engineering is like manufacturing where your have proven process and you just make a cell phone or it is like designing a manufacturing plant or like making a cell phone. So these are 2 different aspects 1 is when somebody started deciding that or dreaming that there should be cell phones, cellular phones which should be usable anywhere. To the point when today you start making Samsung galaxy phone, iPhone and so on.

The scenarios these 2 scenarios are different and for your statistics it took over 37 years to make commercial cell phones, so now the manufacturing process is predictive. It must be predictive, we must be able to measure and control quality as well as quantity but designing the manufacturing plant is a creative innovative process, interestingly most software development is continuous to be innovative process rather than predictive manufacturing. So that is what makes it difficult to view software building software development as a construction process because it does not have the reliability productivity of the manufacturing of construction.

It requires great deal of innovation interaction communication and all those factors impact the success of the projects. So we have a long way to go from the status of software development to the status of software construction.

(Refer Slide Time: 22:52)



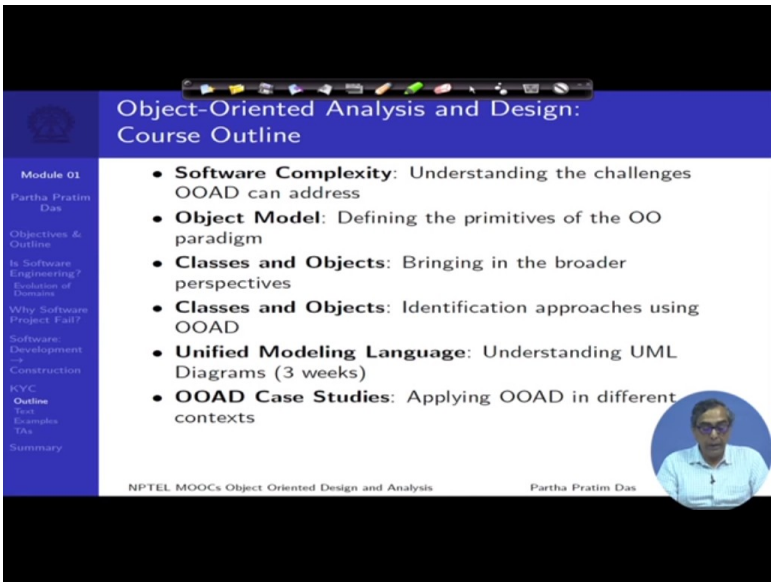
We look forward to a day when like civil engineers, electrical engineers will be able to say that we have constructed a software and if you have constructed a software, it will work the process will predictable quality, will be predictable quantum, will be predictable cost, will be predictable and we will adhered to all of that. The present state of the art says that this transition needs structured analysis

modeling design and implementation.

We will slowly understand through this course what these terms mean in depth and it also needs that we are we can adhere to good practices, if you look into any of the streams of engineering there is a large volume of handbooks and standard data and practices which the engineers must follow that. That is the code of conduct I mean, u just cannot evolve or just deviate from that software engineers are still not up to it. So we are still struggling even to follow the coding guidelines in programs.

So the fact of the matter is that object-oriented analysis and design is a precursor to achieving both these major tasks which will take us forward somewhat closer to software construction than the state of development that we are in and that is the main motivation of attending discussing this course.

(Refer Slide Time: 24:25)



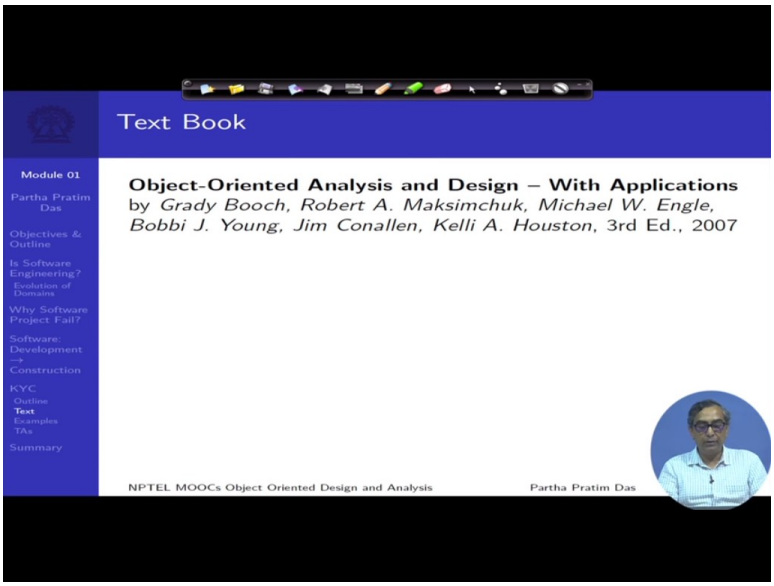
So before I close this module let us quickly take a look into knowing your course the course outline. We will talk about software complexity, understanding the challenges that OOAD can address, will discuss about object models how to define the primitives of the object-oriented paradigm, will talk about classes and objects in terms of bringing in the broader perspective of modeling and we will talk about how with all these definitions and background of theory

We will actually do hands on exercise and give you concrete principles in terms of how you can really identify what are the approaches to identify objects and how to analyze them how to design based on them. And in this whole process, we will start using it language a graphical language which is kind of a de facto standard to express object-oriented systems today known as a unified modeling language and

for 3 weeks we will learn this language we will take several examples to model in that language.

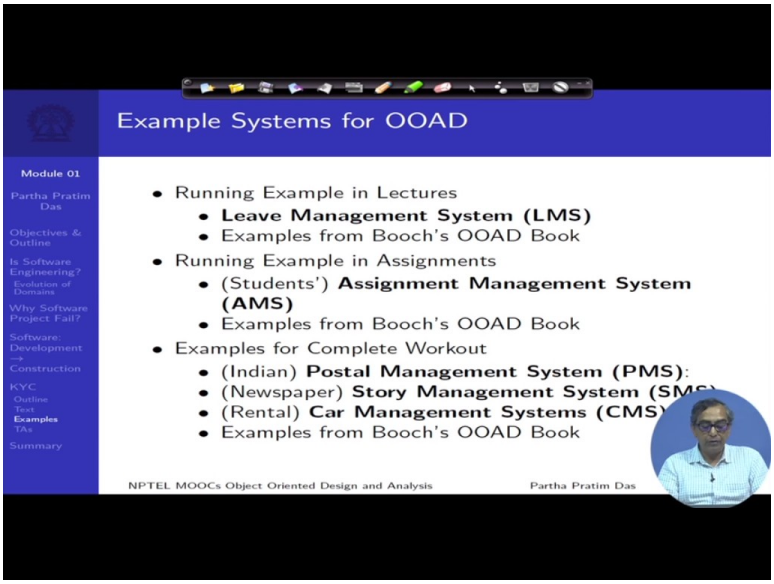
And will show you exactly how to use this, how to talk in the lingua franca of object-oriented community, finally before we end we try to discuss and make you work through some of the case studies where OOAD will be applied to different contexts.

(Refer Slide Time: 25:58)



This is the text books just to there are several books in object-oriented analysis design programming uml and so on just to keep you focused and certainly I would like to recommend you to follow this book as a text book. This is certainly in whatever I have seen this is the best book on this topic written by several stalwarts sphere added by Grady Booch during a verdict and he is with IBM now and please follow this book. This has most of the materials that we will discuss is will be borrowed from this book and also the whole discussion of UML has a good summary in this book and if you follow this am sure you will find this course quite easy to go.

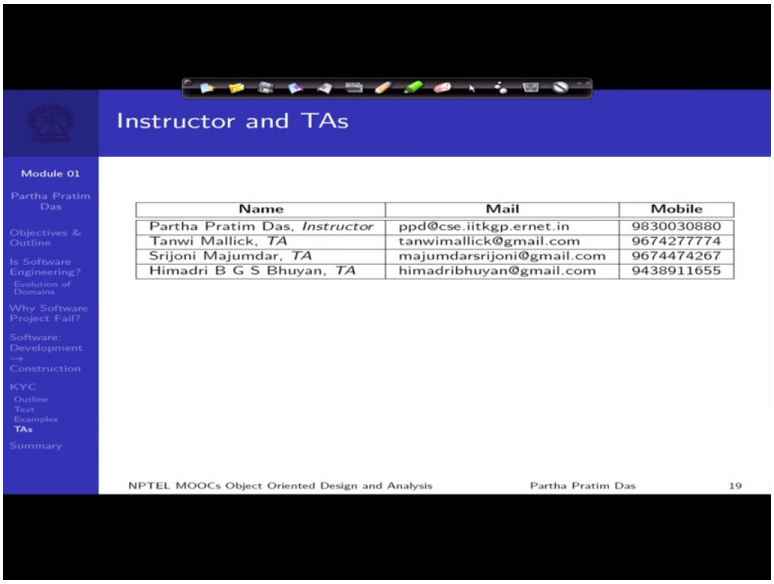
(Refer Slide Time: 26:52)



These are some of the examples that we will try to follow all through this course are the major part of the course, we will use a running example. So while we discuss about different points, we will take glimpses from one running example about a leave management system of an organization. So very soon we will introduce you to the general notion of the system, what does an organization think about leave management, what is required and then we will repeatedly use that side to side, we will have examples from Booch's book.

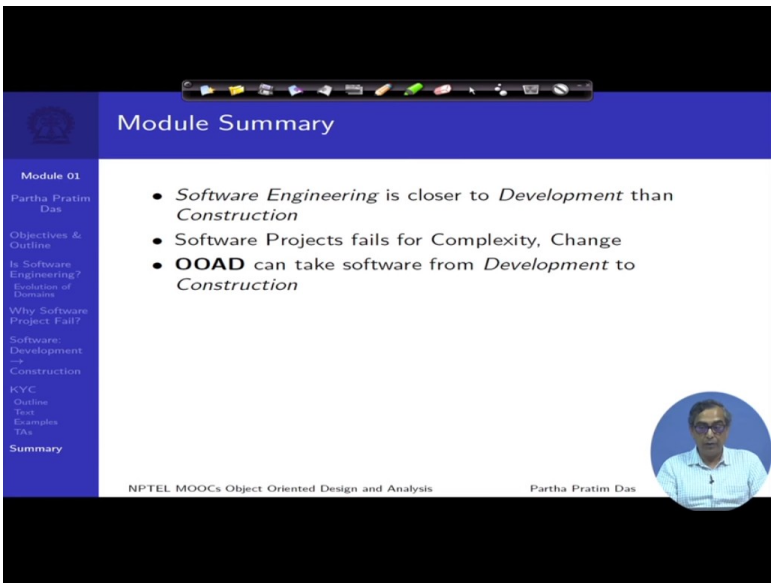
Will have a separate running examples in the assignments that you will do that is an assignment management system where the assignments that students do are managed through that system beside we have Booch's examples from Booch's book and for complete workout we will have couple of different example systems or postal management story management in a newspaper house car management for a rental car company and some of the Booch's OOAD examples.

(Refer Slide Time: 27:57)



Here am your instructor as you know so my contact is given here and am assisted by 3, Tanwi Mallick, Srijoni Majumdar and Himadri Bhuyan. So they will be accessible on these emails and phone numbers in case you need.

(Refer Slide Time: 28:18)



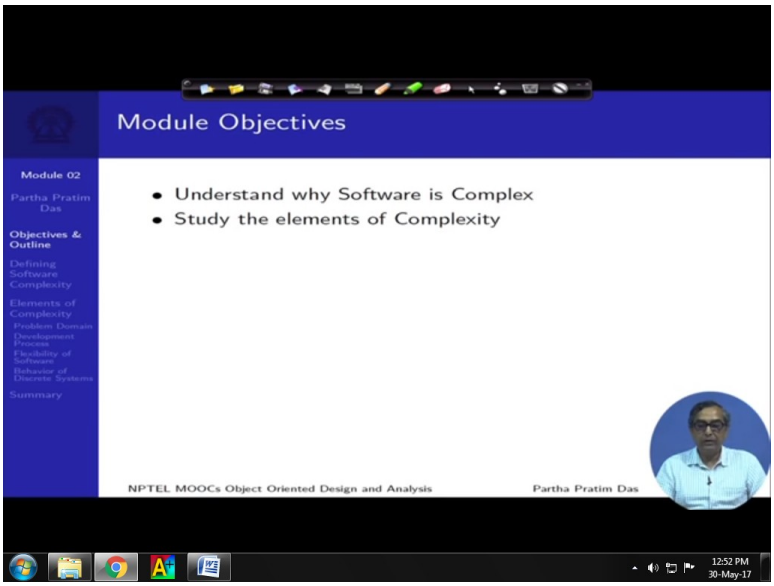
So we will conclude, summarize that software engineering is closer to development, that is than to construction. This is what we have seen through the simple discussion. We have also observed that software projects fails due to complexity due to regular need of change and we have noted and this is what we will demonstrate through the course that OOAD will take software from development to construction over a period of time.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 02
Complexity of Software

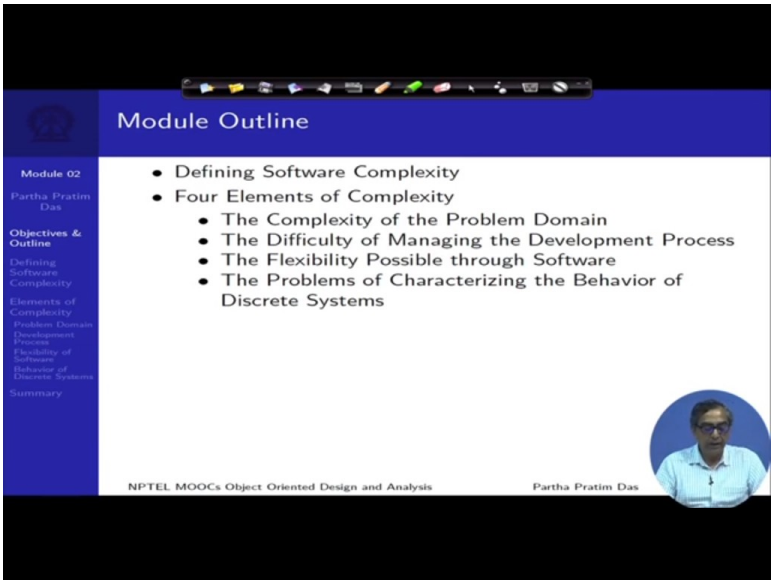
Welcome to module 2 of object-oriented analysis and design course. In module 1 we have set the stage for the requirements of OOAD. We have seen that software development is still at a nascent stage and has a long way to go for confirm success of projects and to achieve a status of a proper field of engineering which can guarantee correct construction, timely completion of projects and within budget delivery and so on. In this module, we look at what is the complexity of software.

(Refer Slide Time: 01:04)



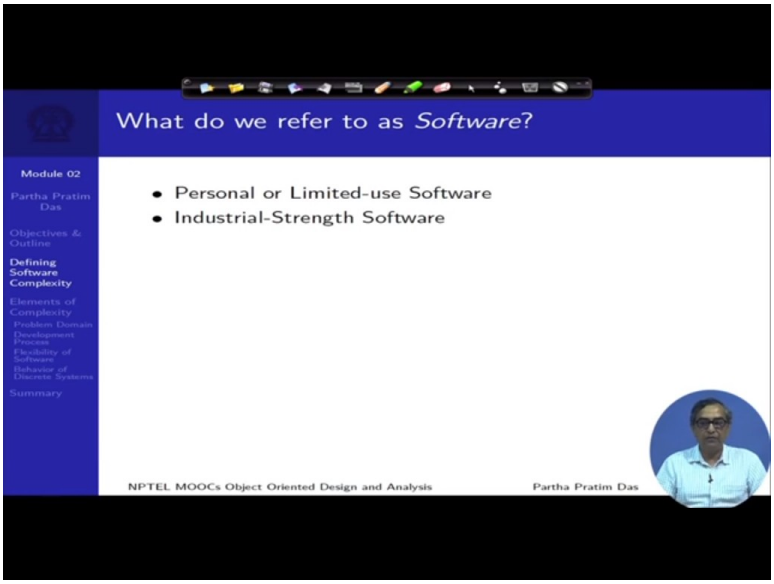
That is we take a look at why software is complex and to analyze little bit more, we try to understand the elements of the complexity of a software.

(Refer Slide Time: 01:19)



This is the outline and as I have already meant mentioned earlier, you will be able to see this outline on the left bar throughout. OK

(Refer Slide Time: 01:34)

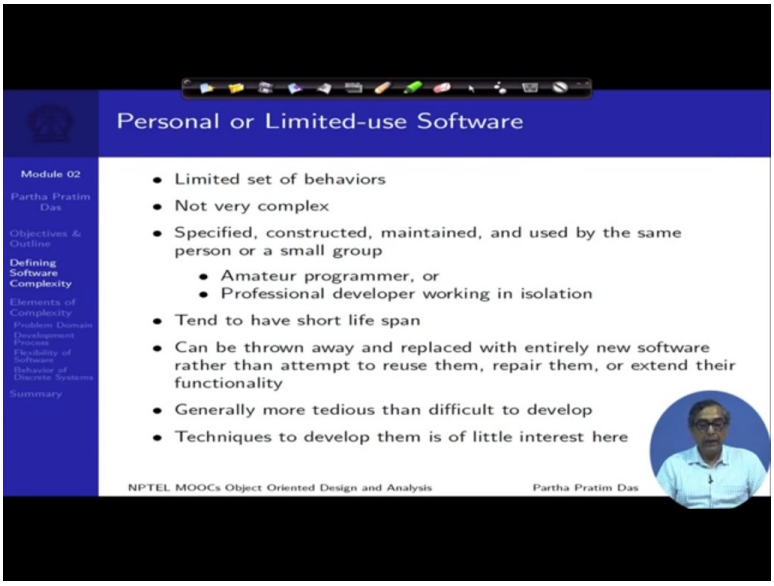


So before we start to take a look into the complexity of software, let us first try to understand what do you refer to a software. There has to be a, I mean this is just clarifying the jargon clarifying the understanding. So software as we know is a, you can always simplify and say that anything you say as a software is finally a program which takes inputs and generates output which is which is a quite a simplest and straightforward view of software.

But we all know that software gets lot more complex than that. So here for the purpose of this course we look at, we divide the software of the world in 2 buckets. One we say are personal or limited-use

software the other we say is industrial-strength software. So let me just quickly explain what these mean.

(Refer Slide Time: 02:35)

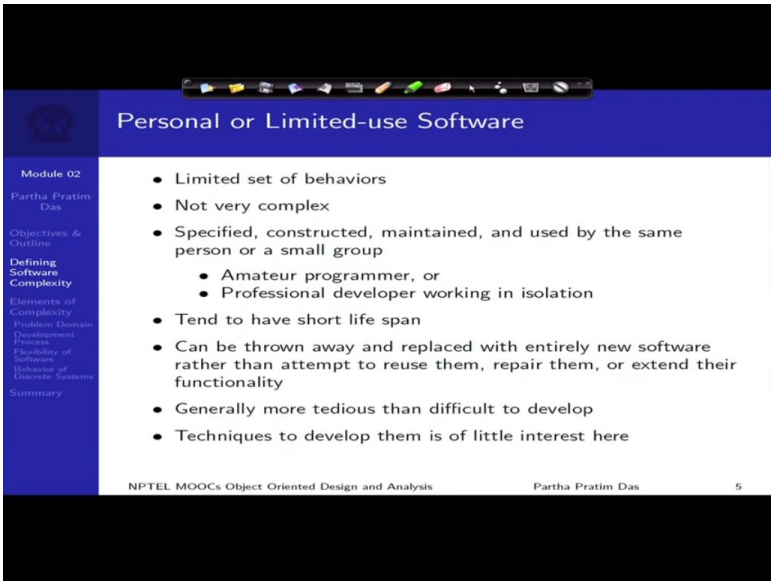


The personal or limited-use software has a limited set of behavior, it is not very complex and most importantly it is specified, constructed, maintained and used possibly by the same person or at most by a very small group. For example, I recall about 10, 15 years back or while working on a project 3 of us faced with the problem that at those days, there used to be floppy drives, they used to be very very small. So when we wanted to copy one big file from one system to the other.

It did not fit into one floppy. So what we needed is to divide that binary file into 4,5,10 different floppies then copy each one into a single floppy, take that to the other system, copy back and then merge that, 10 different parts together into 1. Now this is something which was required which is a engineering requirement of the setup that we had. So we struggled in doing those was, that was not that day when you could just go to Google and type and get a free software which has which somebody has written for you.

So we sat down and wrote a program in c I remembered and which will take a binary file ,divide into as many as parts as you want, then copy them one by one in different floppies and then you take those floppies, copy the individual parts and again another part of that program will combine them together. So this also so this is the software, this had functionality, this has served us for about 3, 4 years quite effectively and all, all of us some 4, 5 of us in the group used to regularly use that. So this is a kind of software which we will say is personal or limited software.

(Refer Slide Time: 04:40)



Because as I had described the behavior is quite simple, it is not complex at all. It is done by a small group ourselves, kind of this group is amateur programmer, we were not exactly armature programmer but we are professional developers working in isolation not as a part of a big team and such software tend to have a short life span and of course once or twice we needed to make changes, we did that ourselves and such software since they are not very complex has not been done with a lot of effort.

So it is possible that when they get dated is possible to scrap them and write new software all over again or make major changes, you do not have to worry about a lot of investment already existing in this software so that somehow even though the functionality is not matching exactly of what you need today, you still need to continue to use that and bend backwards to write some more wrappers, some more plug-in to make that old software, the decaying legacy software to work under the present conditions.

So the such software's may be often may be more tedious but they are not very difficult to build and the reason I characterize the software is we will not be concerned with, we will not talk about the development of such software in this course. This is not of much interest.

(Refer Slide Time: 06:07)

What we are interested in is, what we call is industrial strength software. Industrial strength software does not mean software that needs to be used in an industry. It is showing that the kind of strength that it shows is what industry you typically use. The sum of the characteristics would be it the software exhibits a very rich set of behaviors for example; it could be reactive systems that drive or are driven by events in the physical world.

For example you may be aware that a lot number of cars today when you break, press the break actually the break happens through software, there is no direct shaft which connects the pedal that you are pushing at the break and the shoes that press against the tyres, the rim of the tyre. So these are, these are different reactive systems that are driven by the events in the physical world. Second typical characteristic would be this software work with scarce resources.

By scarce resource it could mean several things, certainly typically it means time and space that is it must work in in a limited time, many of these are real time in nature, they have small memory to use and off let one of the most scarce resources stand out to be power. You are all using mobile phones, handsets, tablets and so on and am sure one common complaint, most of us have for most of the gadgets that we use, I need my battery runs out very fast.

Irrespective of what system we use irrespective of what use pattern you have, we are always very constraint on power so such software you will have to be aware of scarcity. The third is such software has to maintain integrity of millions of records I mean if you have a software which manages 100 students record or 200 bookings and so on, then it is not an industrial set software but it has to be 100s

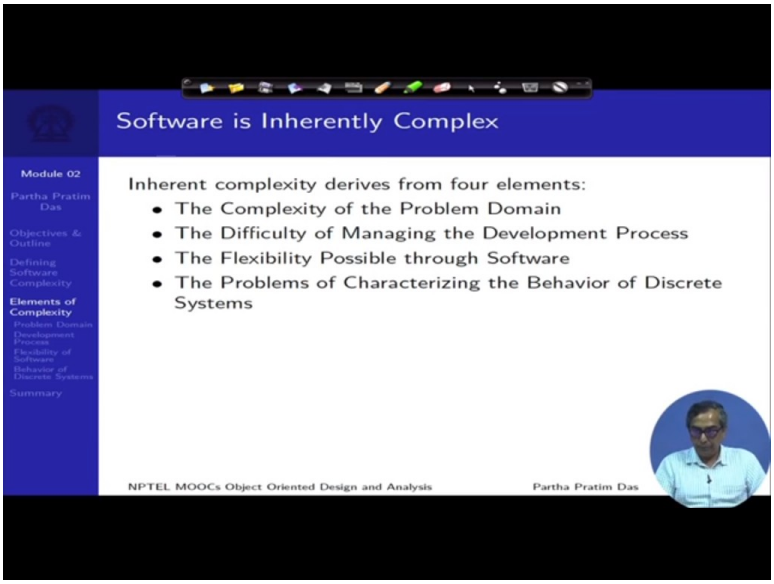
of 1000s, millions like air line bookings railway reservations and so and so.

It, the software could command and control real world entities but in like routing of airways, railway traffic and so on. Usually they should have long time span this should depend, many users should be depending on them, it will be critical for many users that the systems function properly and more often these are based on frameworks that simplify the creation of a domain-specific application. This last point would become, if you are not aware of frameworks, then this will become more clear as we go through the course.

But these factors, some of these or all of these factors decide that the software is industrial strength. So there could be simple examples like the Indian railway reservation that am sure all of us have used is will be a candidate here. AutoCAD will be a candidate here word will be a candidate here and if you think you will see that they all match the requirements or characteristics that I am putting down here but my split and match program for transferring big binary files from one pc to other will not be a candidate.

So the complexity of industrial strength software systems are important to study. Primarily because if you once you put all these conditions then there complexity exceeds the human intellectual capability. It is impossible for any single individual or even for a small group of individuals to comprehend understand the system in totality. So that is a very interesting scenario, we are talking about building a system, we are talking about building a software which nobody understands in full.

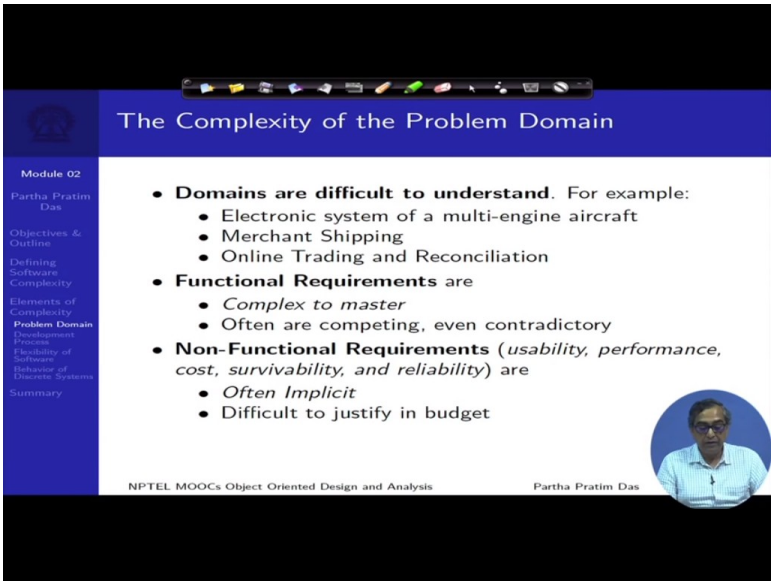
But the software together works. So that is where the challenge lies and it is those kind of software that we will be interested in and the development analysis, design, implementation techniques for those software can be facilitated by object-oriented analysis and design and with those aspects we will study the course. So when we will talk about a software henceforth, it will always mean this kind of software.
(Refer Slide Time: 10:45)



So the basic point is the software is inherently complex but we can look at this complexity in 4 different dimensions which are commonly known as elements of software complexity. The complexity of problem domain, difficulty of so if you just highlight the complexity arising from problem domain, the difficulty of managing development processes, the flexibility that software offers, the fact that software is soft is a strength of software and that itself is a basic complexity.

And kind of works often as the weakness of the software and finally the behavior of the discrete systems or the discrete event systems. So what we will do is we will take each of these elements and try to see what does a complexity mean in regard to this system, these elements.

(Refer Slide Time: 11:50)



So, first the complexity of problem domain, so basically if you look at the scenario then it is always a

scenario of 2 parties, on one side is the user of the software or that we in the industry, we will typically call them as customers or clients and on the other side is the developer or the vendor who will develop the system. Certainly the developer would have skills, domain knowledge in software but the user would come from her own domain of expertise requirements.

And to be able to develop the software, it is the requirement of the developer to understand the domain of the user. We will just take some random examples an electronic system of a multi engineer, now you do not expect the software engineer to understand this electronic system. You do not even expect the software engineer to understand the engine of an aircraft or the kind of control that the engine requires or the issues that arise when there are multiple engines on the aircraft.

And the engines need to switch from on to off and speed controls and all that. So it is the big challenge for the software developer to understand this domain a different very different thing, merchant shipping, as you will be aware that 1000s and 1000s tons of freight is regularly transported across seas all around the world and big companies, corporations manage that. Big ships manage that, and that is the activity of merchant shipping.

If you are asked to develop a software to manage merchant shipping or certain aspect of merchant shipping, am not even talking about everything in merchant shipping. Certainly you will find it very very difficult; because you do not understand that domain. Online trading and reconciliation, we use net banking, many of us may be using the online trading to trade on stock markets but if you want to really develop the software,

You will need understand all the nuance of what happens when you place a transaction, when you place a buy or sell order and how does it go through, what are the kind of synchronization that is require, what is said to be the match of a order between buy and sell orders, how these are reconciliated, how is money really is taken out from the bank account or credited to the bank account, how are demats operated and so on so forth.

So am just trying to randomly talk on different domains to make you understand that it is a really difficult to understand the domain. A second issue with the complexity that comes from the problem domain, is functional requirements are complex to master. In the background of the fact that you have not understood the domain, so all that you have done is possibly skim through some pages that the

customer has given you.

And then have done some blind search from google to quickly educate yourself or familiarize yourself with some jargons of that domain but when it comes to functional requirements so when you said that a dead fred will have to be reconciliated within the EOD, you will not find it easy to really understand what this whole functionality means. So they are complex to master and often the requirements are competing and contradictory.

To take it as well be contradictory just couple of days back, I have been sitting with a team to develop who are developing a digital library. This digital library should be available at a national level, several, 100s and 1000s of people will use that. So on one side, the the requirements of the security team is that each and every individual must be known to the system, each and every individual must have been authenticated through biometrics and so on

To make sure that nobody misuses the library. The other side of the requirement, the different page of the requirement says that this is the national resource, a national asset so this digital library should be assessable to one and all in the country. So you can easily make out that both of these are real requirements of the user or the client. The client is being honest but these requirements are contradictory they are competing requirements and this is just a simple example.

When you go to more and more complex domains, you have more and more complex ah situations to deal with. The third, we have a lot of non-functional requirements in a software. By non-functional, what is meant is which is not evident, that is, is the software user friendly, it could be functionally good but is it user friendly. Does it have a good performance that is does it perform the tasks within the expected time, what is the cost of the software.

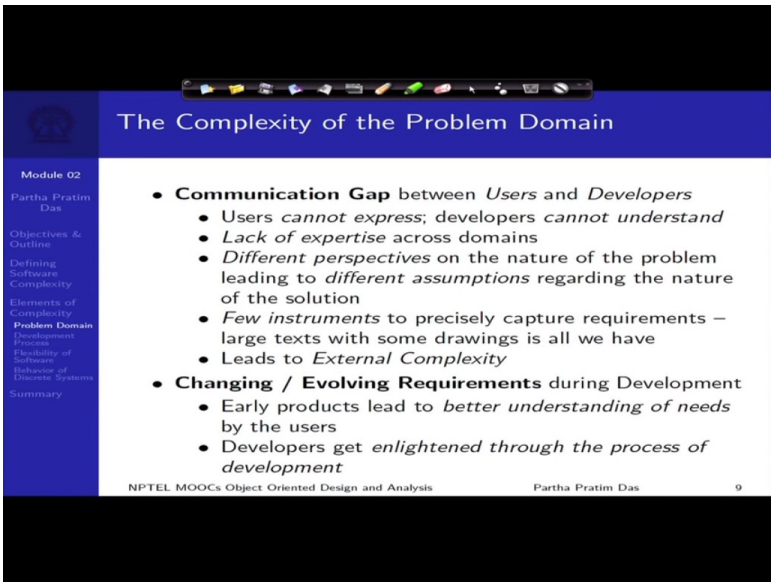
How long with the software survive, is it reliable, how often does it crash, so and so forth. Non functional requirements are very difficult to deal with because users do not understand non functional requirements so when users will tell us, the developers that, this is what I need. The user at best will give a vague idea of a functional requirement in incorrect terms but non functional requirements, most of the times the user will be silent about or will tell you something which is not at all very meaningful.

So a non functional requirements are often implicit and since they are implicit, when you will consider

them in your design and say well this is the budget that you will need to put up, the customer will come back strongly saying that well this is, this is not justifiable, am not getting any functionality in this. The customer will never understand why you want the software to survive 10 years and why there is a separate course to make the software survive 10 years.

The customer says yes I have get the software, I will use it for 10 years. So these are some of the core difficulties that happen in the, due to the complexity of the problem domain.

(Refer Slide Time: 18:34)



Now let us continue further. There is a there is a core notion and many people refer to it as external complexity which is basically the starting point of any software development project. So just think of the scenario, there is on one side is the user, the customer, who has some needs so would has called the vendor, the developer has gone there.

So the first thing what has to happen, the customer has to explain the requirements, the needs to the developer. Now in this, so this is a purely whatever way you do it this is a human-human communication process. And we know that any place you make a communication, the first thing that happens is there is a gap in communication. And the severe gap of communication between the users and the developers is one of the greatest bottlenecks of smooth development of software.

The first thing is users cannot express the other side is developers cannot understand. Users cannot express because users talk in their language, the language of their domain. Developers cannot understand because developers are software developers, they do not understand the language of the

domain. They talk in the language of their software. So when a broker talks to a company that I need a back office management system for demats, the broker very fluently says that well

These are the different kinds of demats it will have and there should be a possibility you should support a placing of demats and so and so forth. to the software poor software developer that is all geek the software does the developer does not understand what placing a demat mean. On the other hand when the software developer says that look you this has to be real time and so on and I will need this kind of a server, there will have to be such network router boxes and so on

Certainly the customer is totally at a loss, is to what is this going on. Where is my shares where are my demats, where is my back office. So the lack of expertise across domains is a major issue of communication gap and that gives rise to different perspective on the nature of the problem. So the perspectives of the developer and that of the customer that of the user are very different in terms of what the problem is.

The user sees one kind of a problem; the developer sees a different kind of a problem. The developer provides a solution to the problem that the developer sees and that is not the solution to the problem that the user sees. So it is a complete I mean it is a perfect recipe for disaster. Now in this process to minimize the communication gap unfortunately we have very few instruments and most projects till date works with large volume of text written in natural language with some drawing at the best is all that we have.

And certainly we know that even a simple sentence like I will go to your home tomorrow can be understood in several ways. I will go to, your home tomorrow. I will go to your, home tomorrow. I will go to your home, tomorrow. Same sentence but you all understand it means different things so when a complex requirement is written down in terms of national language, certainly it is open to several interpretations and developers more often make that interpretation which the user never meant.

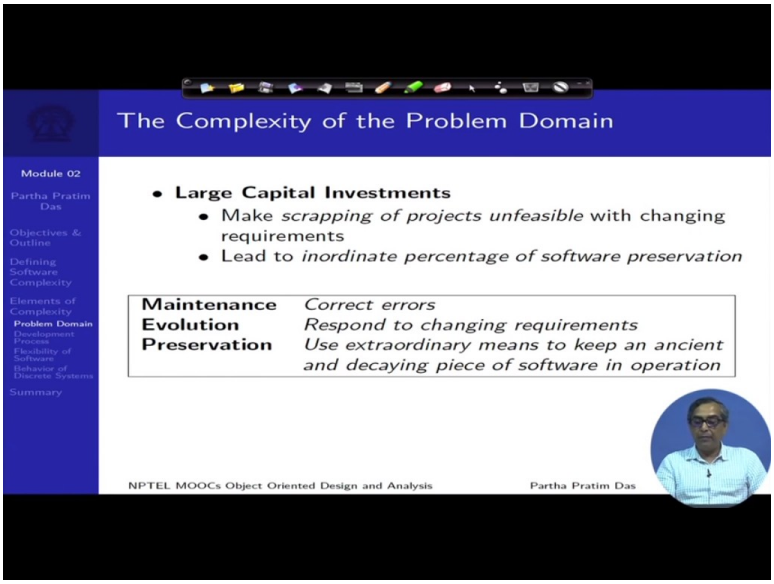
And this is one specific aspect which we will expand on significantly in this course and this will be the basis for, the lack of instrument is the basis for designing using and promoting the unified modeling language which is far away from any natural language, very close to diagrams. Diagrams, icons or what we understand universally irrespective of culture, age, education and our mother tounge. So but this remains to be a big complexity of the problem domain and all these lead to the external complexity as I

started saying.

The next issue is what I touched upon also little bit in the in module 1 is changing and evolving requirements is a reality of a software development and requirements change for 2 reason for the, on the user sides, the requirements keep on changing because more and more the software I see developed. I understand actually what I needed; I understand that what I see is not what I needed, so I change the requirement. Even on the other side, on the developer side, more and more the developer develops,

She starts becoming a better expert of the domain in which the software is being developed. So after 5 months or 6 months of development, then the developer realizes, understands very clearly that there is only one kind of demat, everything else is a specialization hierarchy where such and such and such can be done. Did not see that when this was introduced to her at the time of giving the requirement. So changing or evolving requirements is a reality which adds to a lot of complexity to the problem domain.

(Refer Slide Time: 24:12)



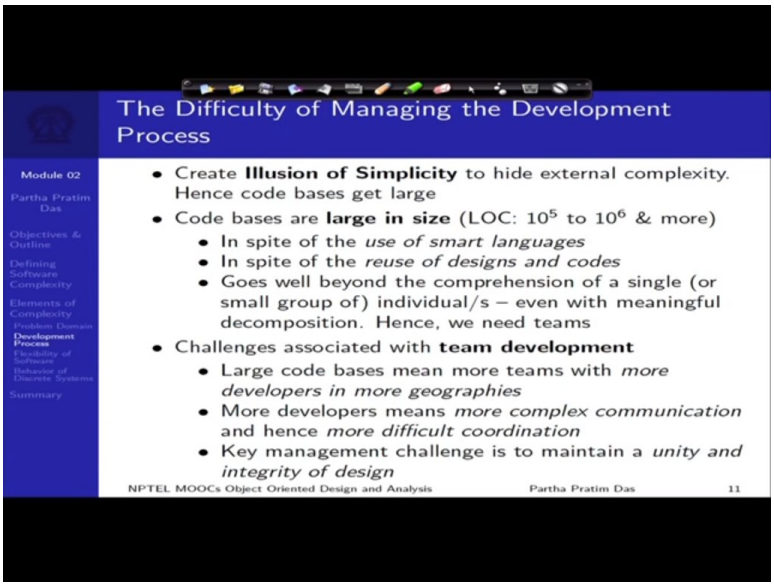
Last but not the least is software projects, large software projects have huge capital investment so when the requirements really change, when the scenarios really change because the business situations may have changed, the business process may have changed, the technology may have changed, it immediately does not remain practical to scrap the project it gets unfeasible with this changing requirements. So what we have to do?

We will have to somehow tweak around and make the old software work in the new way. More often

these changes are referred to as maintenance and I agree with Booch the author of the book who says that no, this is not maintenance. True, this is not. Maintenance is when you have errors in the software you correct that. That is maintenance but when genuinely you respond to changing requirements, that is evolution and finally when you bend you back, bend backwards to make the legacy software,

Decaying software, ancient software to fit the requirements of today somehow you are basically doing software preservation. Naturally the software preservation requires a inordinate percentage of effort by the developers and we need to also build techniques, we need to also make our processes such that you can not only analyze and design and implement but we should be able to maintain, evolve and preserve the software using our techniques. So when you focus on these, all these will have to be equally emphasized.

(Refer Slide Time: 26:11)



Let me now move on to the next element of software complexity that is difficulty of managing the development process. The whole, why do we make software? To solve problems. Why do we make software? So that complex problems can be solved in simple terms. So I need to do a, do an analysis of a truss for building a bridge. I want to do a finite element analysis which is a very complex process but I should be able to do that with the press of a button.

So the reality of what we do and the illusion of that the software has to produce are very different. Software has to be of illusion that is very easy. Matlab on instruction and you get the Fourier transform done but that Fourier transform is a very complex mathematics. So the software has to hide the external complexity the gap between the domain and the instrument of delivery in the software which results in

the software is getting large in size, huge code basis.

Complex software typically have the loc, loc stands for lines of code ranging from 100s or thousands to a million or more and this is given that we are not today not coding in assembly language or a low level language. We are coding in smart high level languages in spite of the fact that we will take all efforts to reuse designs, reuse codes but the fact remains that code base finally turns out to be finally huge that it goes beyond the comprehension of a single individual or groups.

Even when we have done a very meaningful decomposition so what is the consequence? Unit teams. Unit teams, software cannot be developed by individuals or very small group of people. Teams preferably should be created small but unit teams and the moment you needs teams, now you have a new dimension to software development which is team development. If the team does not work, the software does not work. So large code bases would mean more teams, more developers.

You will not get more developers at one place, economics will come in. You will have more developers in more geography, some in India, some in Brazil, some in Poland, some in US. More developers will mean more complex interaction, more misunderstanding, more difficult coordination and the key management challenge will remain to be how do you maintain a unity, how do you maintain a integrity of the design of the software in spite of all these challenges.

So huge amount of complexity is not from the technology or from the problem domain it is from the process of development because it is a human intensive, because it is a process where the intellectual capability has to go in very strongly. I will stop here and take the remaining points separately.

Object-Oriented Analysis and Design

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

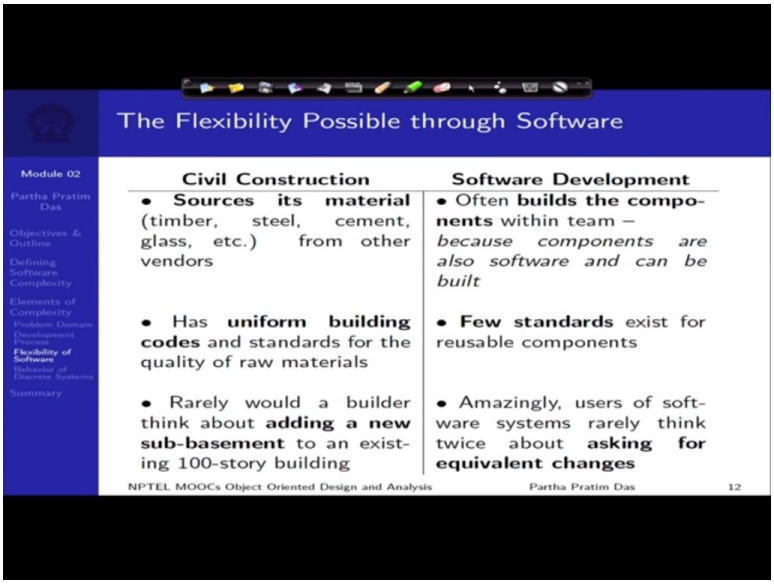
Lecture – 03
Complexity of Software (Contd.)

Welcome back to module 2 of object-oriented analysis and design. In the earlier part of this module, we have talked about industrial strength software. The software which is complex, large, resource constraint, needs a lot of people to develop and so on and we are focusing on the requirements of development, design, analysis of such software and we have noted that the software is a very complex phenomenon and the complexity comes from 4 different elements of which 2 elements we have discussed.

The complexity that arise from the problem domain including the difficulty of understanding the domain. The difficulty of functional requirements, non functional requirements as well as communication gap between the developer and the user and the external complexity and the other element that we have looked into is the difficulty of managing development teams and we have seen that complex softwares in spite of the use of the all different technology today.

The very high level language reusability, modularity, decomposition. In spite of all of that even a small component of software goes well beyond the comprehension of individuals and therefore it becomes. Software Development has become a major activity in team development and team management several teams in several corners of the world often and having different issues in interaction between themselves make it a real management challenge to keep unity and uniformity of the design process.

(Refer Slide Time: 02:17)



Now let us look into the third element of software complexity that is flexibility through possible through software. Software is soft it is flexible now the first there primarily 3 points which you should be sensitive about. The third one you are already aware am sure but the first point is if you just compare software development with civil construction if a building has to be constructed with this components it needs raw materials.

Now no civil construction vendor would start by planting trees so that timber can be used or by setting of a steel roll mill so that the steel can be used in casting the concrete and so on so forth. What you will do you will source your raw materials from several vendors and then just will apply your expertise to construct it. In software this is a big problem, the problem is that the components of a software or the software themselves the components of the component are in turn again software and so on so forth.

At every point you are faced with a challenge of whether to build or to buy, whether to build or to borrow, whether to build or to steal. So often it turns out that a team might end up building not only the software that the team needs to build but it also builds a whole lot of components ,second point which you should understand is any field of engineering like civil construction, when the source are raw material there are several uniform building codes and standards which control the quality of the law material.

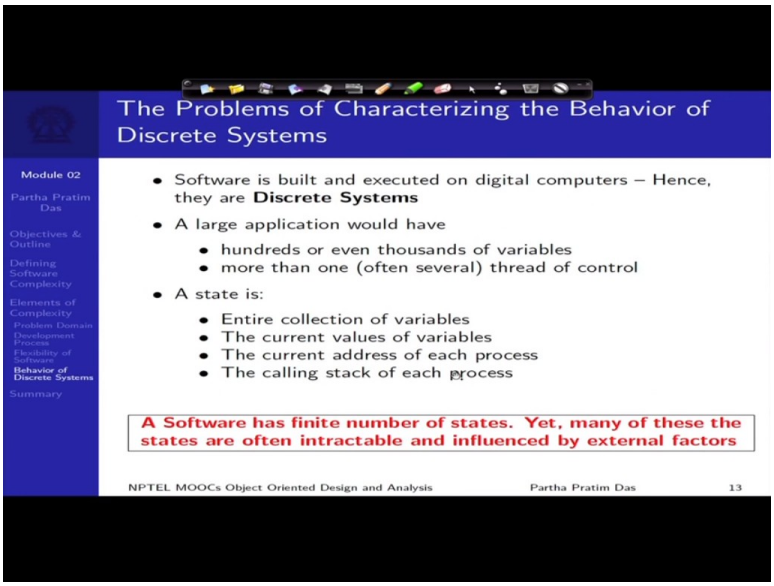
So if you have to cast a cast a concrete the kind of grain that the sand should have the kind of casting strength your cement should have are all documented and fixed unless you adhere to those you cannot construct and therefore when you take raw materials you will always check against those. In software

unfortunately there are very few standards really really hardly any standard to check if a component is really usable what is the reliability of a component that you are reusing and so on.

So the dimensions of flexibility at the major issue of component development becoming a part of the software development. The third point which all of us know is if you consider you have a 100 storied building, you would really even dream of going back to the builder and say that I need a new sub-basement that would be. There would be a question which most people will love it because of the cost because of risks because of the difficulty of doing that whole thing.

But in software similar requirements will very routinely ask for that so the fact that the software can be changed goes severely against the software that a product never can get remain stable and continue to deliver the goods that is required. So this flexibility adds a lot of complexity to the whole process of software development.

(Refer Slide Time: 06:03)



Finally is a problem of characterizing the behavior of discrete systems which adds a lot of complexity. Initial computing systems were mechanical which are very simple after electrical sciences came under our command and elementary electronics was possible, then we came up with lot of computing which is known as analog computing where basically you had analog systems doing the computation for you but really the evolution of computing happen when we move to digital computers that is we do not have continuous value.

So if we if I throw a ball in air and track the trajectory I would not expect the ball suddenly to start

going high at the top of the height or suddenly start going straight we will expect the ball to have a parabolic path but if you are simulating that same process in your digital computer, it is quite possible that you will see this kind of behavior. Why does it happen, why does it go that way because the software being build and executed by a digital system is modeled as a discrete event system.

I will not go into the depth of what discrete different systems are. They are more of study of system dynamics, what it simply means that if we talk in terms of automata theory, it has caught a finite set of states and when it works, it simply changes from one state to the other. Very nice so actually in that way the things should have been simple because an analog computing will have infinite number of states but there are two difficulties.

one analog computing has strict mathematical model if I try to compute the trajectory of the ball through analog means I will use simple equations derived from newtons laws and they will divide everything. But if I have to do it through digital means, I need a mathematics, I need tools which are very different from the some ordinary physical world laws and not all of those yet are very clearly understood. The second is of course discrete systems have predefined well-defined states.

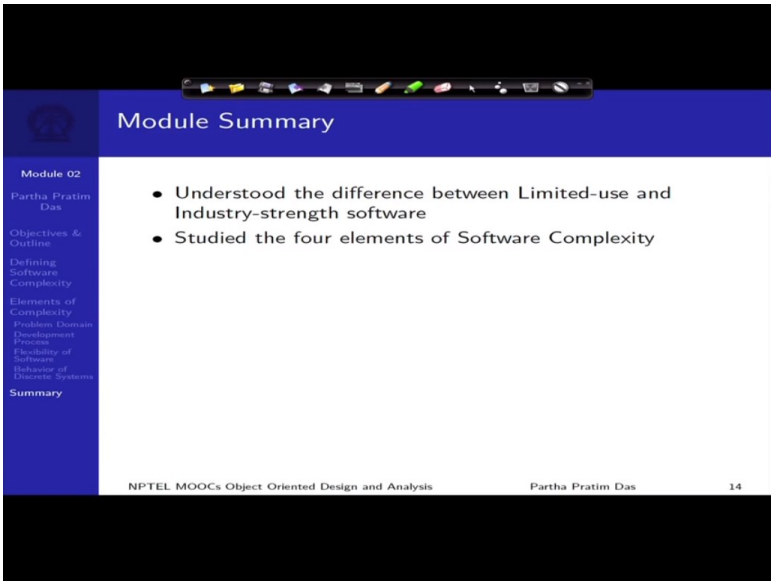
But if you consider a large application, then the application will have 100s of 1000s of variables and possibly in today's time multiple threads of control, multiple things can be happening at the same time given that the state of a system is the entire collection of these variables, the value that the variables has, the current address of each and every process, the current stack of each and every process and so on. So even though in theoretical terms, a software has infinite number of states.

Yet many of these states are intractable we just do not understand what these states are, how to enter into those states, what to do once your system gets into that state, how to get out of that. Even at times it is very difficult to even identify that I have got into a state which is not where I want to be in. So we take all those states together and say okay this is this is where the software is not working this is where the software is crashed, this is where we have an exception and so on so forth.

But the bottom line is there are several external factors which decide how your system run makes transition between these large number of states and the total behavior of the resulting discrete system that we make as a software is never understood. The tools today do not exist for doing that the skill does not exist with most of the developers to analyze them. So it adds a several dimension of

complexity to the software because of this behavior of discrete systems.

(Refer Slide Time: 10:41)



So these are the four different elements of software complexity. In this module we have started by trying to understand the difference between limited use personal kind of software and industrial strength software and then we have take a tour of the 4 fundamental elements which make software complex that the complexity that arise from the problem domain from the communication gap, the external complexity, the complexity that arise due to the difficulty of managing big teams which need to be managed.

They are the machines of software development, the flexibility that the software offers is the strength of the software but we have seen why that adds to the complexity of software development and finally the software being a discrete system, the behavior of the discrete system adds a new dimension to the complexity of software. So having understood this, we will stop now. In the next module, will come up and start understanding about what the structure of complex systems typically are and how to manage those complexities.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer science and Engineering
Indian Institute of Technology, Kharagpur

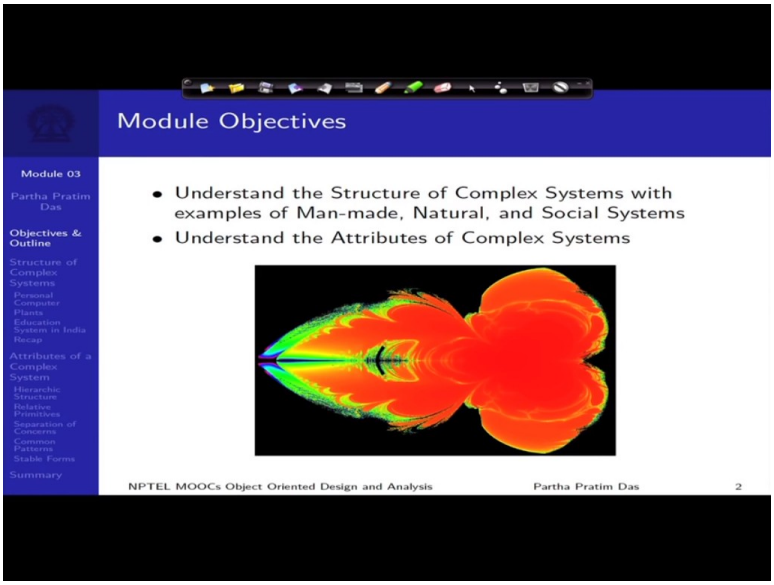
Lecture – 04
Structure and Attributes of a Complex System

Welcome to module 3 of object-oriented analysis and design. In the earlier modules we have taken a look at some of the big challenges that are faced by software engineering. Software projects having a high percentage of failure and based on that in the last module we tried to look at some of the typical complexities that face software development, the issues that arise from the problem domain, the external complexity, the difficulty of managing groups.

And the issues arising from discrete system behavior and flexibility and so on. So now we slowly start getting into how to handle how to manage this complexity. So in the module the current module and the subsequent ones we will slowly start characterizing how to handle this complexity. So first what we do in this module here is we take a look into a variety of complex systems and try to observe what is that inherent structure.

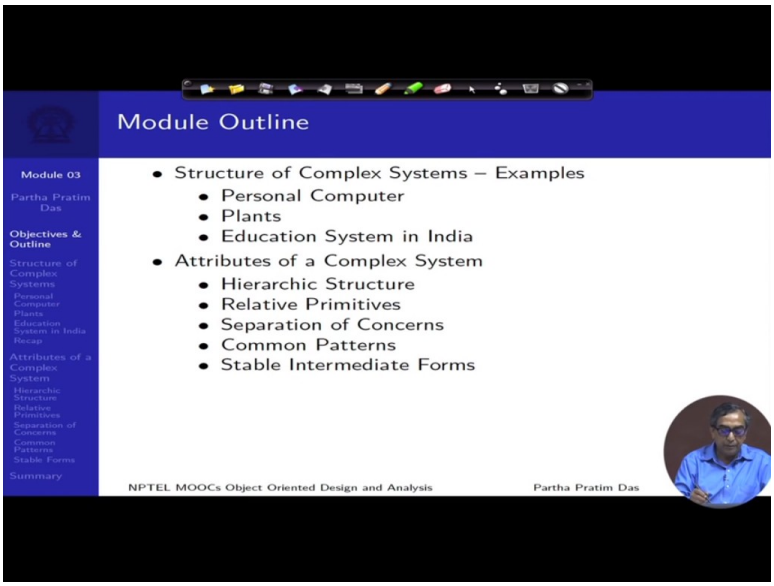
What is the inherent behavior and through that process, we will try to extract some commonality that we can canonicalize at a later point of time and then find out a strategy to handle the design, to handle the complexity.

(Refer Slide Time: 01:53)



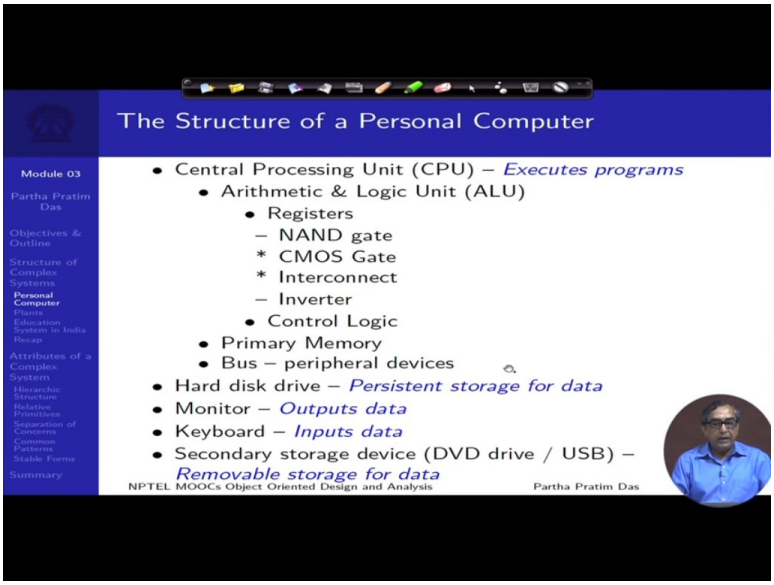
So this current module will specifically focus to understand the structure of complex systems and we take 3 examples, 1 manmade, 1 natural and 1 social system and from that we will try to understand attributes of a complex system.

(Refer Slide Time: 02:10)



The module outline as you already know is presented also on the left bar of every slide.

(Refer Slide Time: 02:18)



So the first system we take a look at is a man made system. We try to understand what is an outline of a personal computer. all of us know personal computer. So you will know that there are some a personal computer of whatever capacity you talk of, am not talking about any specific

personal computer, even this will include laptop and so on will comprise of a cpu which is a basic component which executes programs.

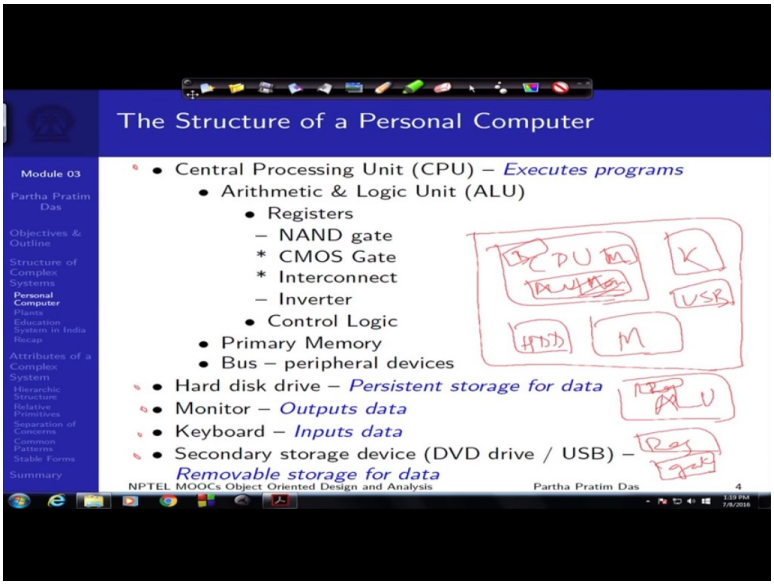
It must have a hard disk to store the programs, store the data on a persistent basis, it must have a monitor to display interact with us the output device, it must have a keyboard as an input device and typically it will have a removable storage, secondary storage like usb or DVD. At least at a, these are the minimum components which will define a personal computer. Now if I take each one of these components, say just as an example I take the CPU.

Then I can say that the CPU actually comprise a number of sub components or sub systems. It must have an arithmetic and logical unit which is a core unit which does the mathematical computation, the condition checking and so on. The CPU must have primary memory, the fast memory through which we operate and it has busses that will connect to the peripheral external devices of the CPU. Now in turn, again if we look at the ALU.

The arithmetic and logical unit, then we can decompose and find other sub systems that alus must have registers, it will have control logic, it will have other say floating point unit, integer unit, branch unit all these different components will be there. All are not listed here is not possible that will comprise and make the arithmetic and logic unit. Now again if we go deeper, look specifically into register.

We know that typically today the registers are built with digital logic of NAND gates inverters and other different kinds of gates. Now if you look into a particular NAND gate typically a NAND gate will be realized in terms of a cmos technology so it will have cmos gates and interconnects. So if we specifically look into so this was the the highest level of decomposition so a personal computer.

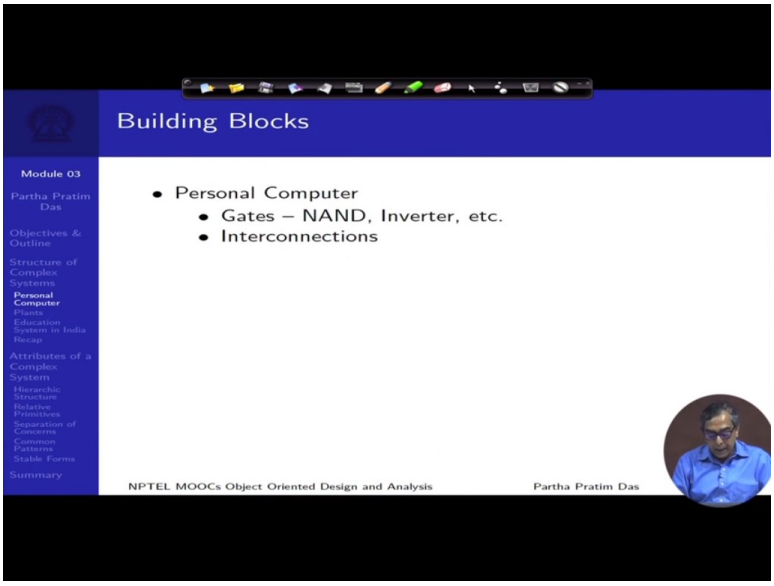
(Refer Slide Time: 04:53)



If I look at it this way, we have a cpu, I have hard disk, I have monitor, I have keyboard, I have usb then the cpu in turn has ALU, it has memory, another component as lets say it's a memory here, then it has bars all these different components. Now if I take ALU, then again I have registers in them. If I take registers, I have gates in them and so on. So if we if we just keep on going on this hierarchy.

You will find that the whole system can be described as a collection of hierarchical components, hierarchical subsystems where every system has certain subcomponents, subsystems in that and is a part of the higher level of systems.

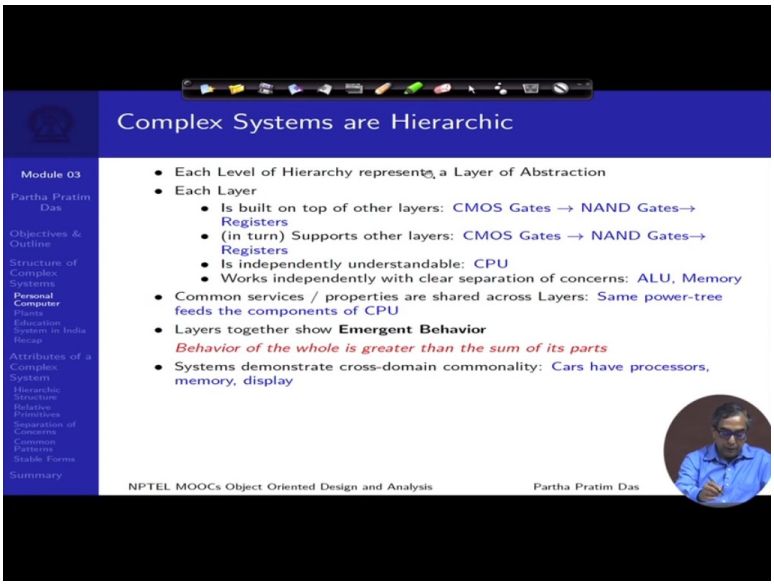
(Refer Slide Time: 06:02)



Now this is what we observe in case of a personal computer so but at a basic level we can see that the building block of a personal computer is something very generic. it is, these are basically gates, NAND gate, inverter so whether you talk about ALU or you talk about say the memory or you talk about hard disk (())(06:22) fundamentally these are collection I mean built with several primitive gates.

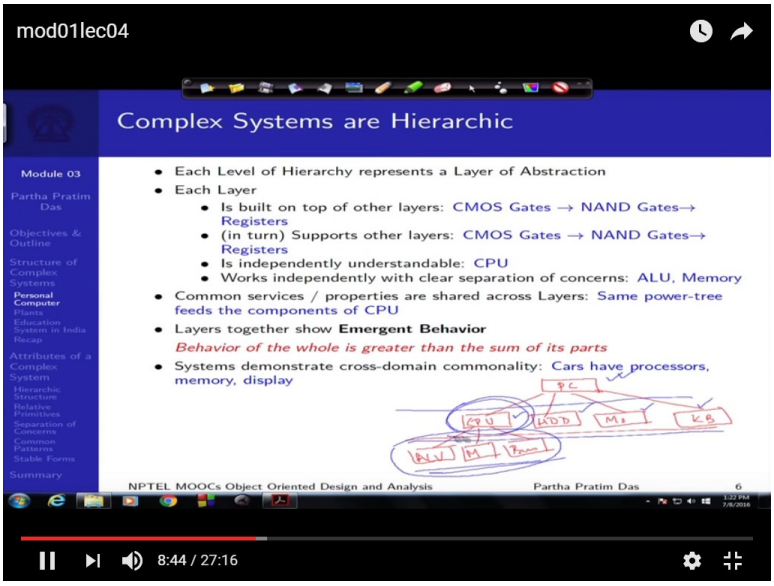
And on other side we have different interconnects to define the buses that exist on the CPU, the buses that connect from one CPU to the hard disk and so on. So while on one side, the personal computer has several other systems and subsystems of complexities on the other side, there are few primitive elements through which these computers are built up.

(Refer Slide Time: 06:54)



Now with this, with this observation about personal computer, we can try to make some generic properties and we will slowly build up on those properties. First thing we observe is each level of hierarchy represents a layer of abstraction which you mean is if I talk about a personal computer itself is an abstraction, is a concept and so I can say that the at this is at the top level of hierarchy, this is the personal computer.

(Refer Slide Time: 07:35)



Now I say that this is comprising a number of other levels hierarchically of HDD ,of monitor, of keyboard and so on. Now the CPU in turn has ALU, it has memory; it has bars and so on . So what am trying to point out is, if you look into every say if you look into this layer, i if I look into this layer, then these different components of this layer together buildup the higher layer in turn if you look into any specific component here, that again is built up by other subcomponents.

So each layer is built on top of other layer. This layer of CPU is built on top of this layer. In CPU, HDD, memory, monitor, keyboard all makes up the next layer. So these are just examples shown here of these multi layered structure like CMOS gates build NAND gates, NAND gates build registers and so on. But in spite of all of that, the major property is that every layer can be independently understood. I can study CPU independent of any other layer anything else.

I can they work independently with what you call is a clear separation of concerns that is when I look into the functioning of an ALU, I just assume that there is a memory through which I can send a data and keep it as long as I want. But I do not really need to know how the DRAMS in the memory, the flip-flops in the memory really work to make that happen and when I concentrate on organizing the memory, what are the different ports, how the address to be decoded and so on.

Am really not concerned with how numbers are added in ALU. All that I know is ALU might ask, the CPU might ask for some value to be stored or retrieve from the memory and at that point of time, we can just respond to that request. So this individuality, this individuality is as will see can be sort of is called the separation of concerns that is different systems within themselves can function independently.

And across systems there is an interconnection there is an interlinking which has to be clearly which is clearly specified. Further we can make some more observations that in addition to this hierarchy and this independence of components in subsystems, there are some common services and properties that are shared across layers. For examples many of you who have little background of electrical sciences would know that.

If I have a CPU, then the CPU needs power. all components of CPU, the memory needs power, BUS needs power, the ALU needs power, the registers needs power and each of the component do not separately get a power ,there is a power tree which kind of distribute the power to different components. This power tree is there for a common facility, common service that is shared by different components.

And in every many systems we will see that this particular concept will be applicable. Given all this, what we observe is we have several subsystems of every system and when we look at the system and the subsystem separately we find interesting observation that if we just take the subsystems as a collection and then we put them together, then the system that we get is actually more than just the collection of subsystem.

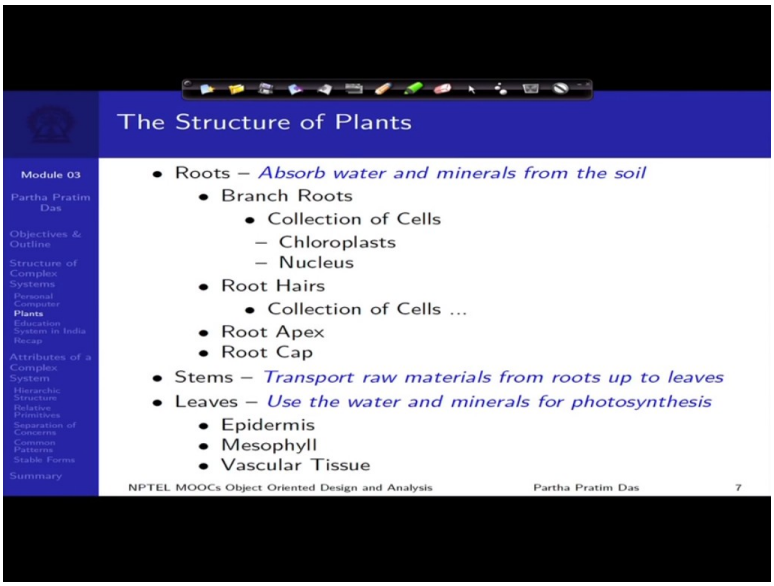
ALU by itself can only add 2 numbers in a register and so on. Primary memory by itself can store some values, retrieve those values, the bus can only take a value from one point to another. But when you put them together into a CPU, it can execute a program and this principle is known as the emergent behavior principle. The behavior when you say that the behavior of the whole is greater than the sum of its parts.

So behavior of the CPU is more than the behavior of the ALU plus the behavior of the memory plus the behavior of the bus and so on. Together when they are put or put together, properly interlinked then they build the whole system which has a high emergent behavior. Finally we will observe that the systems complex systems often demonstrate cross-domain commonality that is all this different hierarchy.

And commonality that am discussing in terms of personal computer if I just go to one of the modern day cars and try to look into different subsystems of steering control, break control, air condition control, the wind control and so on, we will find similar processors there having similar, very similar kind of functionalities so if I understand the functionality of a CPU kind of or a personal computer kind of complex system, and that will aid me a lot to understand different subsystems of car as well and that the same logic would apply.

For understanding mobile phones understanding air craft controls and so on. So there is a lot of cross-domain commonality that can be found so these are some of the generic observations that we are making from our example of personal computers.

(Refer Slide Time: 13:47)



Now let's move on to take up the next system which is the natural system and we look at the structure of plants. am not a botanist um exactly I don't expect a whole lot of knowledge of botany but all of us understand this much which we studied in school that a plant comprise of

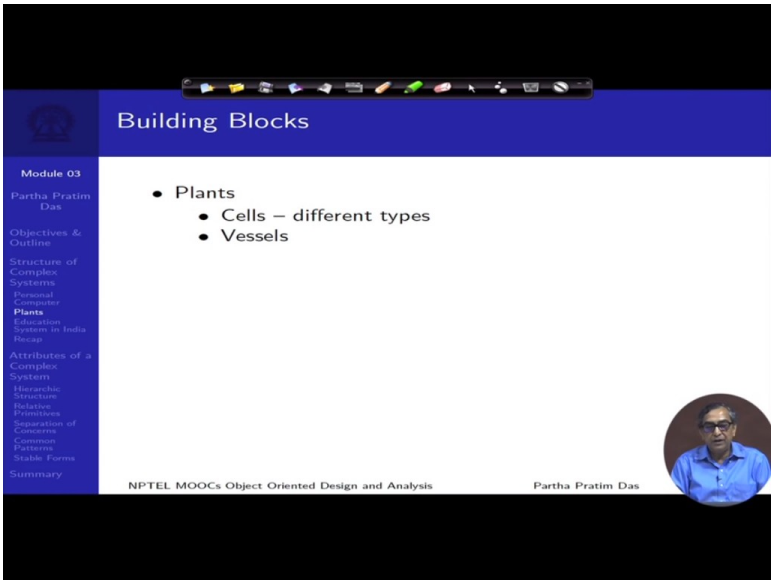
primarily three parts. There has to be roots that go into the ground usually there are stem and there are leaves. Roots observe the nutrition, the water, minerals from the soil.

Stem carry those from the root to the leaves and the leaves use water, minerals and the sun rays to do photosynthesis so that food can be prepared. These are the basics. So if I look at the plant as a system then these are the main 3 subsystems with their functionalities. Now again like in the case of personal computers we can start decomposing and look into subsystems. The roots have branch roots, they branch through the soil.

They are find at the end there are root hairs, at the end of the hair there is a root apex and a root cap. So any roots can be typically thought of to be comprising of four different subsystems as listed. In turn branch roots will be comprising a collection of cells. Every cell will comprise chloroplast & nucleus, and so on so forth. Am just expanded few few places only just to give a representative idea if we expand it all then it will really become an essay in botany.

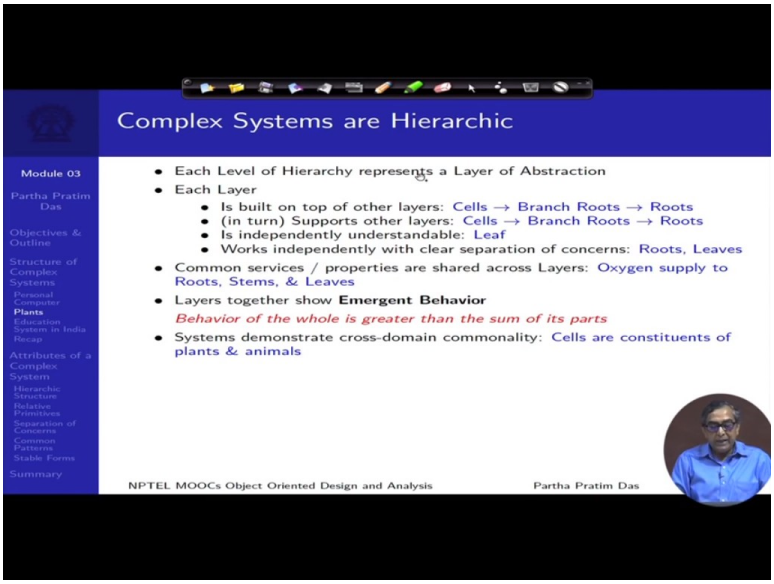
Similarly if we look at leaves they are formed of subsystems defining epidermis, mesophyll and the vascular tissues. Similar decomposition is possible for stems and multilayers. So this is what we find in a natural system and again if we look at how this is implemented we say that the building blocks are few.

(Refer Slide Time: 15:43)



The cells of different types and there are vessels to carry a fluid carry information from one part of the subsystem to another, something that we should note is is kind of very common with what we saw in personal computer that such a complex system that personal computer has very few building blocks like interconnects and gates and so on. Similarly such complex systems as plants has very few building blocks fundamentally in terms of cells & vessels and so on.

(Refer Slide Time: 16:19)



Now I take you back to the same characterization chart that I discussed in terms of a personal computer the same set of points but I just read these points now in the context of a plant system. Again we have seen that it is hierarchy it has different levels of abstraction the plant being in the highest level, then you have roots, stems and leaves roots have branch roots and so on. So each layer again is built on top of the other.

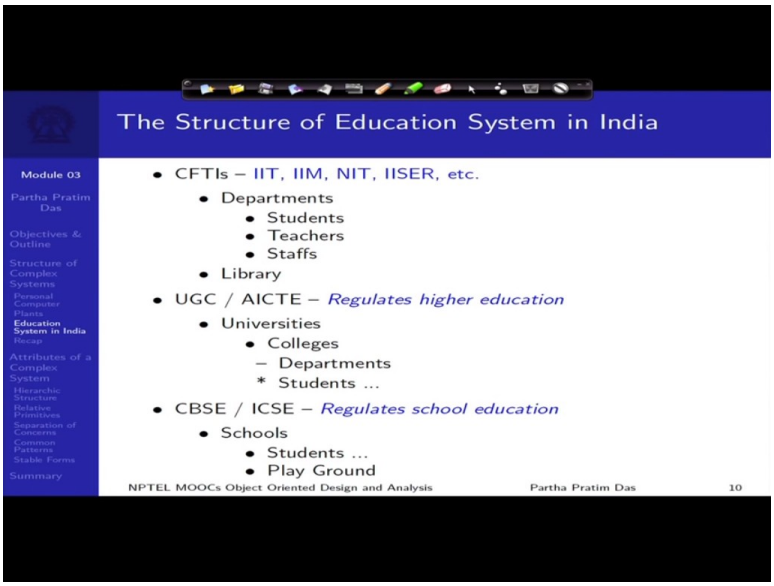
You can see that cells form branch roots, branch roots going to forming roots and so on. So if you look at branch roots, it has to use cells. It is build on top of that. If you look at branch roots, they help to build up roots and e a each one of them each one of these subsystems can be independently understood like a leaf and they work independently with a clear separation of concern. The way the roots function we does not need to know how photosynthesis happens in the leaf.

And when the le when we study the leaf, we do not really need to understand the osmosis that happens to observe water and minerals to the roots. Again together we can pick up a leaf we can pick up a stem, we can uproot a plant, but when you put them together in the system is properly interlinked, you find the emergent behavior of a plant which is certainly bigger than the independent behavior of these components.

And very interestingly the natural world show a significant amount of a cross-domain commonality. For example in parallel to plant systems if we just study animal systems, we find very similar hierarchy of different plot systems like digestive systems, like reproductive systems, like respiratory systems and so on. We can keep on decomposing that but when we go to the basic building blocks they are again cells. So the cells differ.

They are they are different kind of cells, animal cells have certain properties, plant cells have different properties, but fundamental part is they are undeniably these are cells. Those are the consequence of plants and animals both. So across domain, we find commonality here as again as we saw in case of the personal computer.

(Refer Slide Time: 18:37)



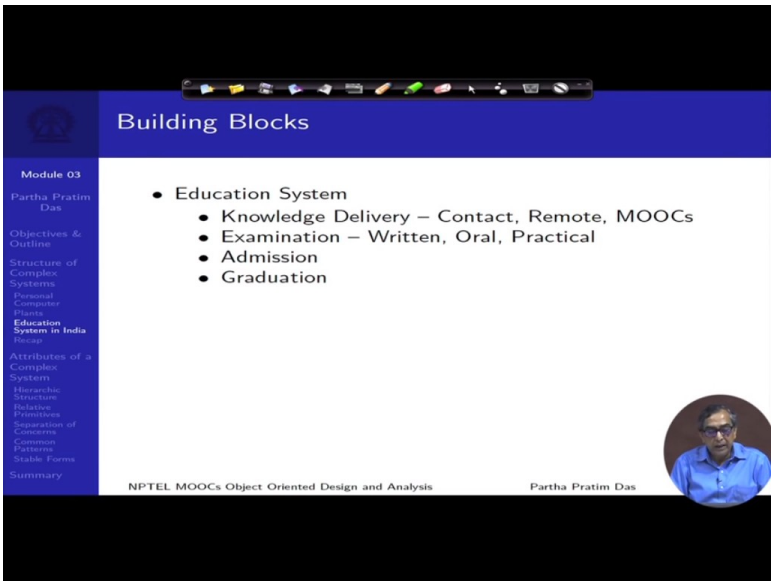
So let us move on and take a third example which is kind of a social administrative system and since we all are in in India and related to education and so I take an example of education system in India. So I we look at the education system in India if you take that as a whole system and try

to analyze you will find that they are significantly divided in terms of centrally funded technical institutes like CFTIs.

Then there are organizations like university grants commission or AICTE who regulates higher education and there are CBSE, ICSE and so on who regulates school education. Now if you look into any any one of this particularly CFTI you will find that CFTI, IIT is like a CFTI, is a CFTI. You will find different departments, libraries, stadium auditorium and so on. Every department will have students, teachers, staff.

Similarly if you go to UGC and start decomposing that you will find universities. Universities will have colleges, colleges have departments, departments have students and so on. CBSE has schools, schools has students, play grounds and so on. So again we see that in a system of social political administrative nature we again see on one side we do see it's a really complex system again but it again shows a same kind of hierarchy, same kind of system subsystem decomposition that exist.

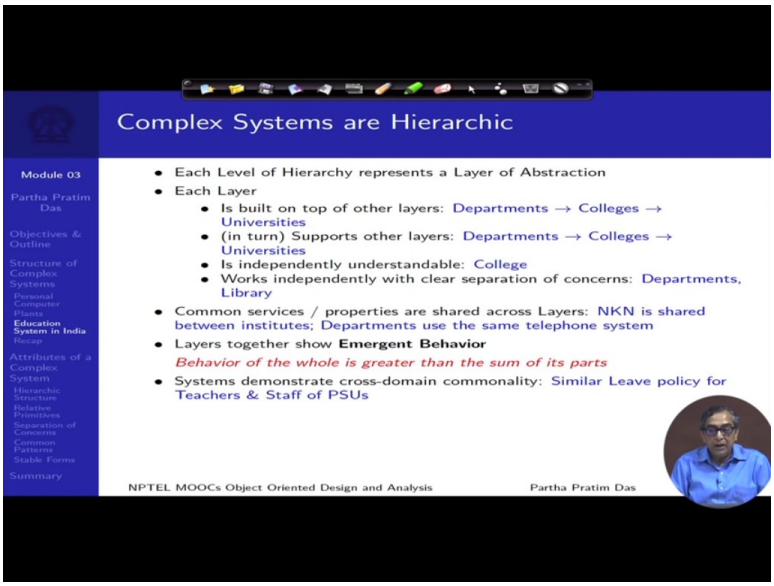
(Refer Slide Time: 20:07)



Building blocks there are very, in the education system there are few building blocks one is the knowledge delivery that is how people teach, one is examination how the knowledge acquired is checked and how people joined this institutes and how they graduate and go out of this institute. If you look at all of this institutes and educational system we will find that this is based on

defining our specializing this specific building blocks on which then you add different variabilitys, conditions in terms to buildup these systems.

(Refer Slide Time: 20:44)



So will be back to our characterization of complex systems the same slide we are interpreting this in terms of the education system. So again the highest level of abstraction is the education system in India, below that we have CFTI and UGC and all those and then the appropriate ones. For example if we look into this particular that departments build up colleges, colleges build up universities.

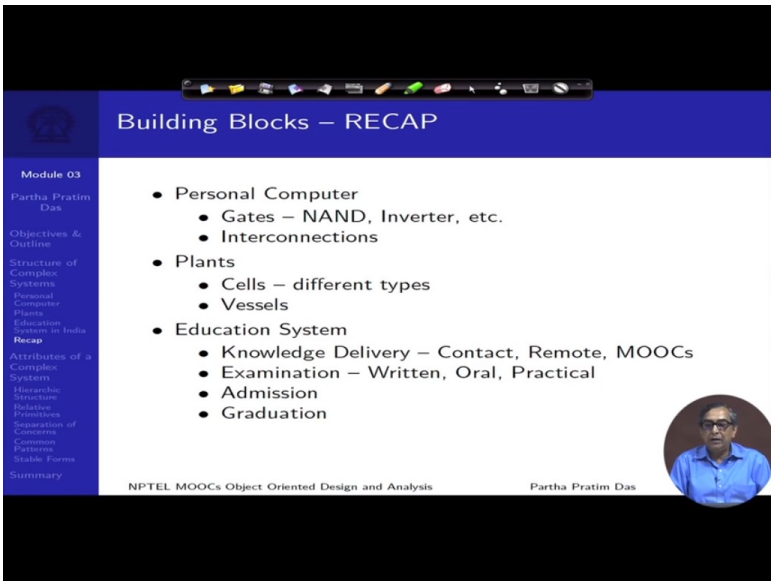
So if you look at colleges, then it is built in terms of the layer of departments and colleges go in terms of building the university layer. So the similar kind of the structure that we had seen for a personal computer and system and the plant system we do see here. We do see that this systems, many of this systems can be independently understood, there is separation of concern the department and library has to function together.

But how a department functions and it conducts its classes and examinations and all that is not required to be understood by a library system which has requirement of books, issue of books and so on and vice versa. So they enjoy a whole lot of common services like nkn is available between institutes ,many departments but they have the same, they use the same telephone system and finally they should demonstrate the emergent behavior.

When we just take the library separately, the department separately or the play ground separately, we have certain functionalities but when you put to them together then we have a college which has a bigger functionality. When you put 100 colleges together under a university we have a traditional behavior and certainly also in terms of cross-domain functionality, we find that different practices different properties that are held in the education system are also enjoyed by several other.

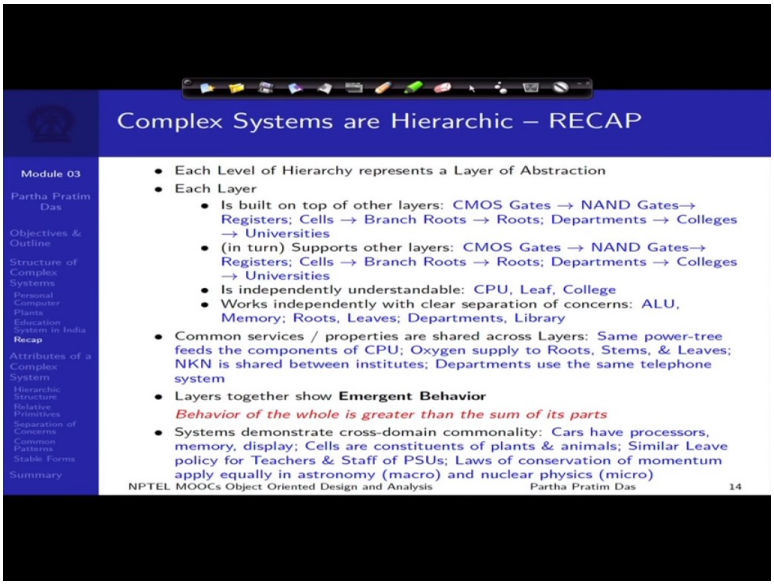
For example the leave policy of teachers and staff will be lot similar to the leave policy of teacher and staff in the public sector units psus.

(Refer Slide Time: 23:10)



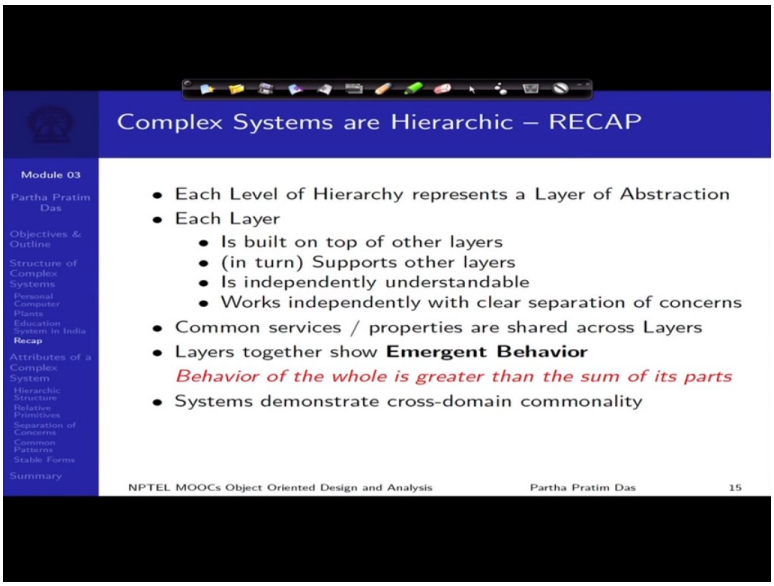
So in short what we want to conclude from these examples is that here is whole lot of similarities between all these different kind of system irrespective of whether they are manmade they are natural they are social or of other kinds.(23:13). So just recap that we saw that every system has a few primitive building blocks, personal computer, plant education system all of that.

(Refer Slide Time: 23:28)



And just putting everything together that complex systems are hierarchical and they demonstrate a whole lot of properties which are independent of whether it is a manmade system, it's a natural system or whatever kind of system, complex system typically show these properties. So these are collection of all the instantiation for your easy reference and I have some additional ones also included here. But I will just clean up and show you.

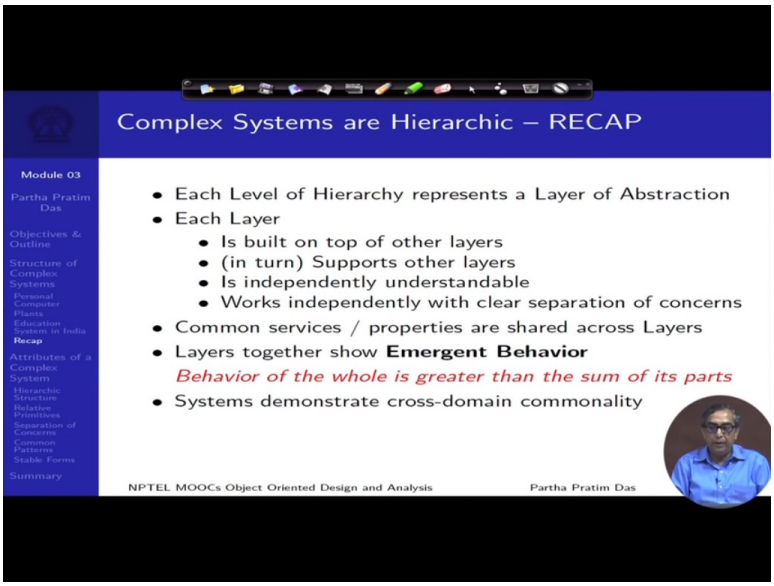
(Refer Slide Time: 23:57)



This is the take back we want to do from this short study of structure of different complex systems that to summarize complex systems are hierarchical. This is the major observation we are making and it has multiple layers of levels of abstraction and abs levels of abstractions are

interrelated. Every level supports the higher level and is supported by the lower level. Every level can be understood on its one.

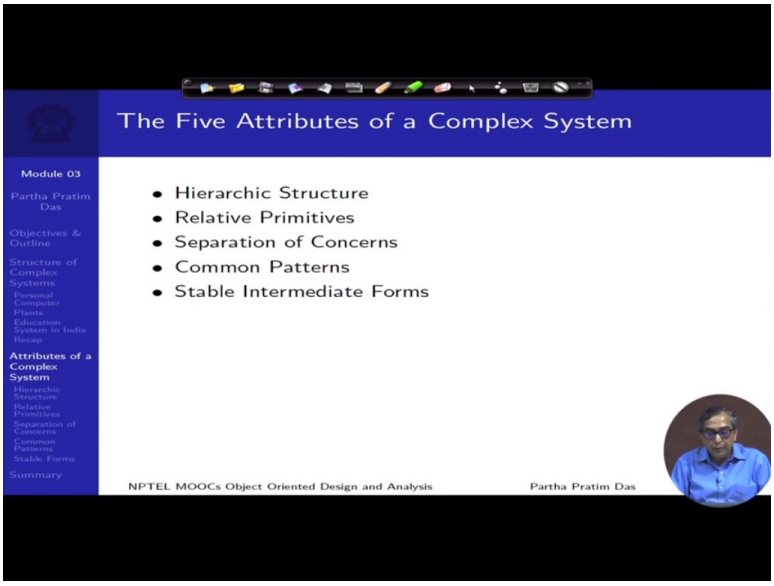
(Refer Slide Time: 24:24)



The different components in a level have separate concerns which can be independently studied put together every level build up a bigger functionality for the next level and there are lot of cross-domain commonality in terms of these systems. So we started with the issue of how we actually handle complexity and this was our first step to go forward analyzing the complex system and through some examples we analyze and come to the conclusion that while the complex systems are indeed complex but they do show a lot common properties.

So if we build up strategies, if we build up further analysis based on these properties, based on these characteristics, then we hope to be able to analyze a new complex system lot more effectively.

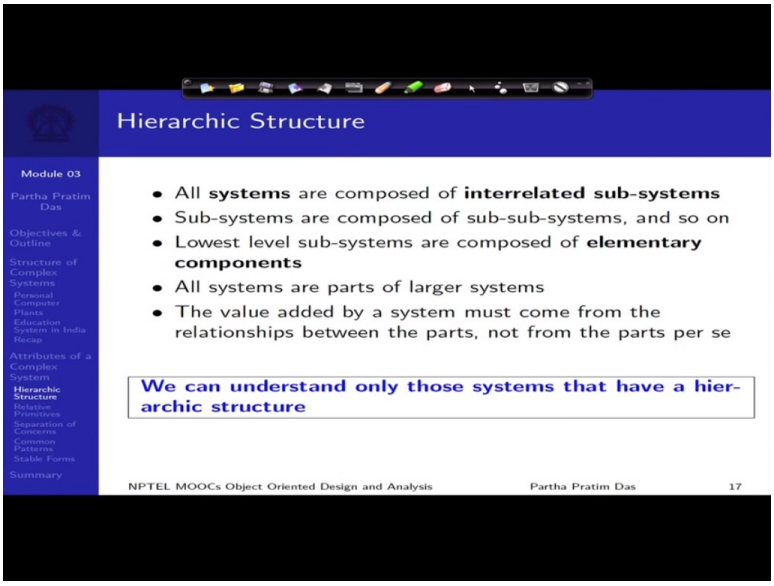
(Refer Slide Time: 25:17)



So with this intention we go forward to just put all these observations in a very crisp forms and this is typically called the 5 attributes of complex systems. That every complex system will have hierarchic structure and this hierarchy will terminate in a primitive which is directly defined. Now what is a primitive and what is not a primitive is kind of a subjective decision. There will be strong separation of concerns between these components.

Across domains there are similarities so there are common patterns that happen across domains and the whole development of such complex systems has to happen through a reted iterative refinement process. So at every stage we need to have certain intermediate forms.

(Refer Slide Time: 26:14)



So just to quickly illustrate a hierarchical system we have talked we have been talking from the beginning of this module, we have been talking about this systems. So they are composed of interrelated subsystems, sub-subsystems and so on and at the bottom they have the elementary components. And we can understand now you can question that it is it necessary that all complex systems be hierarchic in nature and show these behavior.

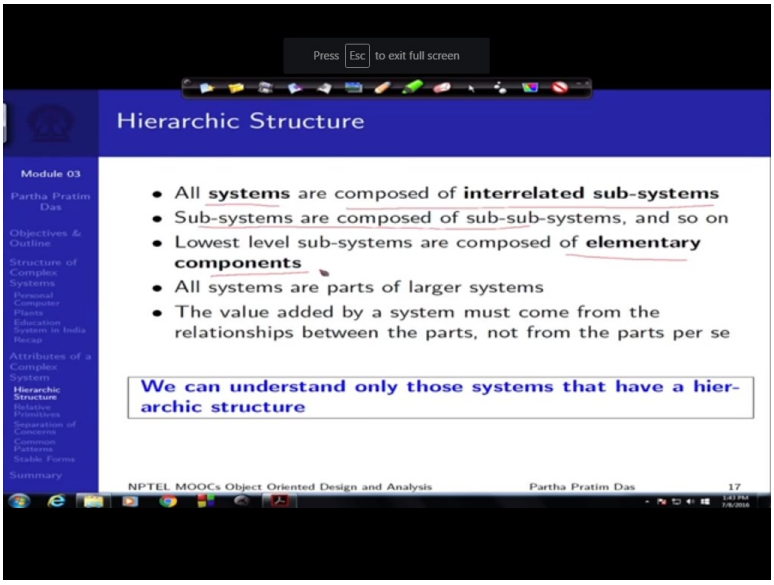
The reality is that the complex subsystems that we study do show this behavior or we can put this differently that only those complex systems we can study and you know try to think of putting up a software system on where this hierarchy exist. So it is kind of our basic assumption is hierarchy is the kind of base line, is kind of the foundational part of the complex system on which the remaining 4 attributes are built up.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture – 05
Structure and Attributes of a Complex System (Contd.)

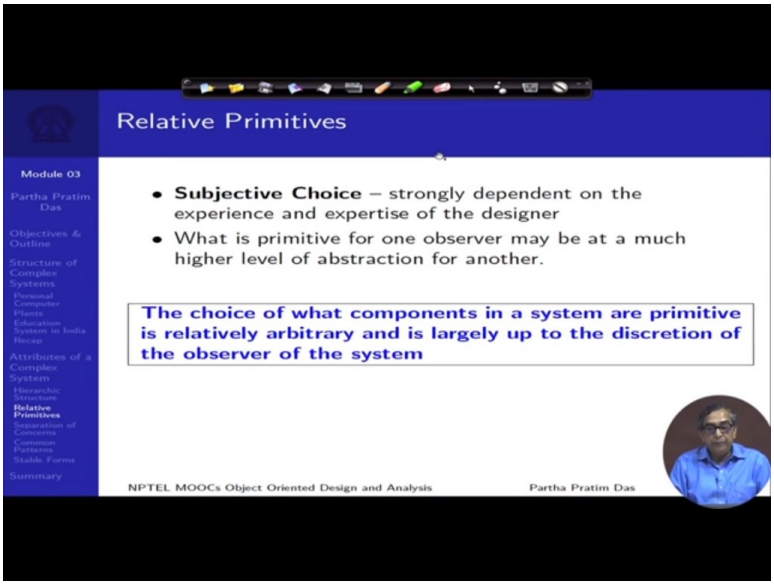
Welcome back to module 3 of object-oriented analysis and design in this module earlier we have looked at three examples of complex systems and tried to characterize their structure and we have come up with some common conclusion leading with the fact that every complex system has to be hierarchic in nature and with that we are now trying to conclude that there are 5 major attributes of complex systems. The hierarchic structure, the relative primitive, separation of concerned, common patterns and stable intermediate form.

(Refer Slide Time: 00:58)



And we have also taken a brief look into hierarchic structure that is we say that every system is composed of interrelated subsystems and sub-sub systems till you go to an elementary component level. This is a basic property that we say will exist in any complex system that we analyze.

(Refer Slide Time: 01:22)

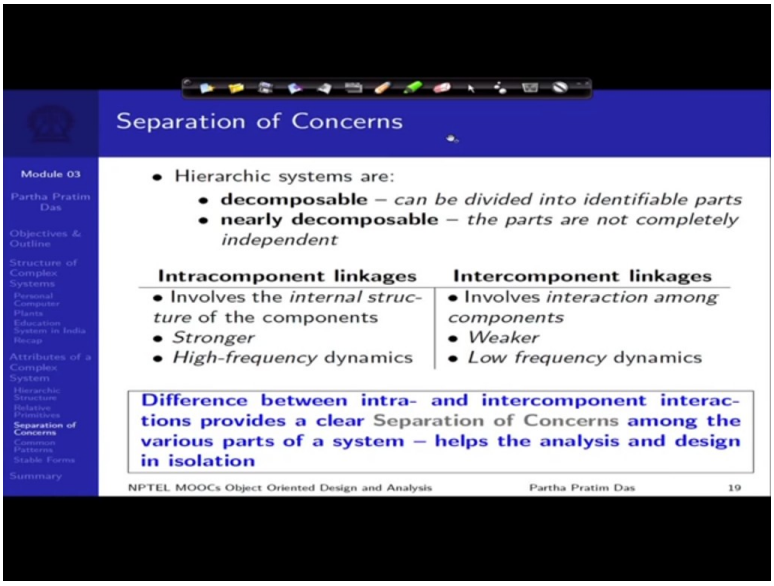


Now next is relative primitive that is if we keep on decomposing the system in the hierarchy manner as we have said and then there are some where you will have to stop and that stopping point is called the relative primitive or the primitive. It is relative because there is, there cannot be a unique definition of what is a primitive. It is a matter of perception of the observer of the designer as to deciding what is a primitive.

And what could be a primitive for me may not be a primitive for someone else and vice versa. So the choice of what component in a system is primitive is relatively arbitrary and is largely up to the discretion but you will have to for any any system if the description has to have some primitives in terms of which things are defined. For example in personal computer, electronic systems the primitive could be taken either as a digital logic gate or as a CMOS gate.

Or you can even go lower or you can become even higher and say that no it might primitive it is not up to the level of NAND gate but might primitive for certain electronic subsystems like maxers (02:38) and resistors and I am just doing a RTL kind of design so primitives are relatively arbitrary.

(Refer Slide Time: 02:46)



The next is, the next attribute of complex system is separation of concerns that is hierarchic systems are decomposable that is what we have seen then they can be divided into identifiable parts, personal computer decomposable in terms of CPU, hard disk, monitor, keyboard, and so on. At the same time, interestingly they are nearly decomposable in the sense that while we say that these are all decomposable but they are not completely independent.

If CPU is completely independent of the hard disk and completely independent of the monitor and so on then the CPU will not be able to access the file from the hard disk and use it. Monitor would not be able to display the values that the CPU wants to display. So on one side they are decomposable but other side they are nearly decomposable so this this kind of can be characterized by the amount of interconnection, inter component linkages that exists.

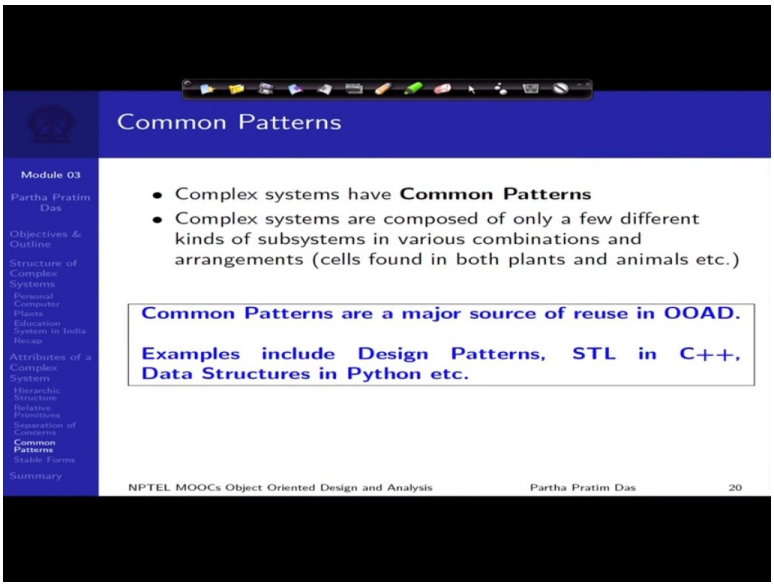
If you look into the CPU there are different components of ALU, primary memory and bus and all that naturally the interconnection that exists between them or inter component linkages that will have to exist between them is much higher compared to what will exist between the CPU and the hard disk. So if we look into this and try to characterize between this intra component linkages. So if you look at this, here we are talking about the intra component that is within component.

And this is inter component that is across component so this involves inter connect interactions between the internal structure and this involves interaction across components then we can always expect that this will be much stronger there will have to be lot more within the system than what happens across the system. This will have a much higher frequency; this will have a much lower frequency.

So the difference between the intra and inter component interaction will provide a clear separation of concern that is when I am looking into the interactions within the CPU that is highly dense and that defines what I understand to be the CPU and when I look at the interaction between the CPU and the memory, it is low frequency, it is weak and that defines that I am talking to a separate system and memory subsystem will have to take care of its own intra component interactions.

But the interaction between the CPU and the memory will be separate and can be separately handled. Now this separation of concern is very very important because that is what allows us to study the subsystems independently. If the separation of concern was not possible then we would not have been able to study, design and develop the subsystems independently. So this third attribute is very critical from our object-oriented analysis design context.

(Refer Slide Time: 05:51)



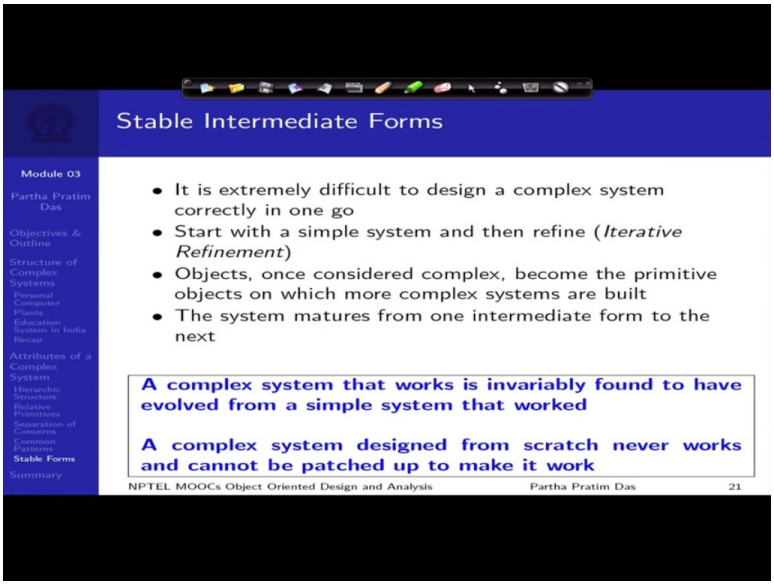
The third that we observe is common pattern complex systems have common patters we did mention about some of these in terms of saying that we have processors in personal computers we have processors in cards we have processors in mobile phones and so on so. If I understand processor, then I understand all of these. So finding out this commonality is one of the challenges or one of the major handles through which OOAD can simplify problem.

Because a lot of analysis experience of people show that at the end, there are not too many different kinds of subsystems amongst many of the common complex systems. So if we can understand those subsystems well over a period of time, then we can approach and analyze several varieties of complex systems quite easily. For example from the natural world the cells are found both in plants and animals, vessels are found both in plants and animals.

So if one understand cells then in plants then it is much easier for the person to go and understand cells in animals. So common patterns are major source of reuse, so hope you all would be aware with the term of reuse that something which is built can for one purpose can be engaged in another purpose. This reuse becomes easy this reuse becomes possible because there are common patterns.

And if I just talk about very specific examples you have different libraries in the c you have standard libraries in c plus plus you have template library, you have design patterns, you have 5 common data structures in python. These are all examples of common patterns that happen across domains across systems and allow us to make reusable solutions using the object object-oriented analysis and design. So this is fourth attribute which is critical for such systems.

(Refer Slide Time: 07:56)



The 5th and final attribute talks about stable intermediate form. It has to do more with how we should tackle the development of a software system, development of a system which is complex. It is extremely difficult to design a complex system correctly in on go I mean if we just think that we have understood, studied a lot and we can just go and design a complex; we will inevitably face failure.

So what we what succeeds is starting with a simple system and then refine, add some more complexity and then refine again. So this approach is typically called the iterative refinement so the when you start, you say certain objects are very very complex so you start you say okay I am going to make a MATLAB kind of system and when I start, I need to represent numbers. Now I need to represent number.

That representing a number itself is a very complex concept because you need to deal with bits. The bits are finite in size, they will overflow, underflow then it is high in all that so you are dealing with concepts objects which are just numbers and you find them to be really difficult to handle, they are complex in nature while your primitive remains to be the bits that you can manipulate. Once you have built up these objects.

Once you have built up these numbers of say integer, double, floating, floating point numbers, complex numbers, fractional numbers and so on. Then you can use them to build bigger

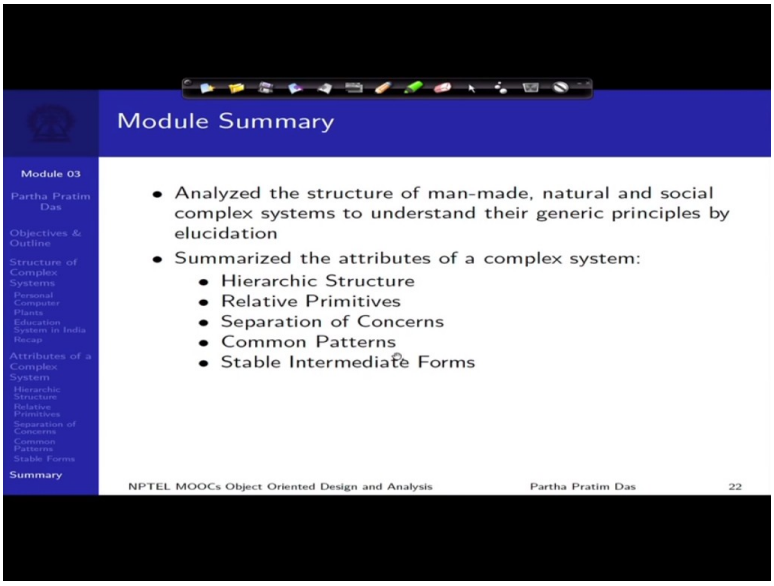
functionality now you are building up matrices, now you are building up vectors because you are doing a MATLAB. So now the objects like a floating point number which looked very complex and was not considered to be primitive, now in turn it will become primitive.

And now you are making matrix of double numbers, floating point numbers. So now double becomes a primitive and matrix is the next level of object, complex object that you are trying to build up. Once you have been able to build that up, then you say matrix becomes a primitive and I am making a system of linear equation solver where matrix is there, vector is there at every stage. So what is primitive and what is complex is a relative term.

And that is what we take into advantage when we want to, when we need to build up systems and we need to go from one state of the system, one stage of the system to the next one by refinement and this is what is called a intermediate form. So when I have started with representing number or by representing matrices or by representing systems of equations, I have not done my MATLAB but they are all intermediate forms but what we need is the intermediate form has to be stable by itself.

That is whatever I make of a system to represent and manipulate matrices must be stable to work when I build up more systems on that. So being able to handle stable intermediate forms or having stable intermediate forms is a necessary requirement of complex systems and any attempt to design, try to design complex systems from scratch will inevitably lead to disaster.

(Refer Slide Time: 11:20)



So these are the 5 attributes that is what we have studied so if we sum up, we have analyzed the structure of man-made ,natural and social kind of, administrative kind of complex system to try to understand the generic principle by elucidation. We are… There is no there is no proven theory that this will happen, things will happen the way we are trying to characterize but this is by illustration by example that we have tried to understand that there is a lot of common properties that exist.

And then we have scan through them, summarize them in terms of 5 basic attributes of a complex systems looking from the object-oriented analysis and design perspective, the hierarchic structure of the system, the relative primitives, separation of concerns, common patterns across different systems and across domain and the iterative refinement leading to stable intermediate forms at every stage of the system.

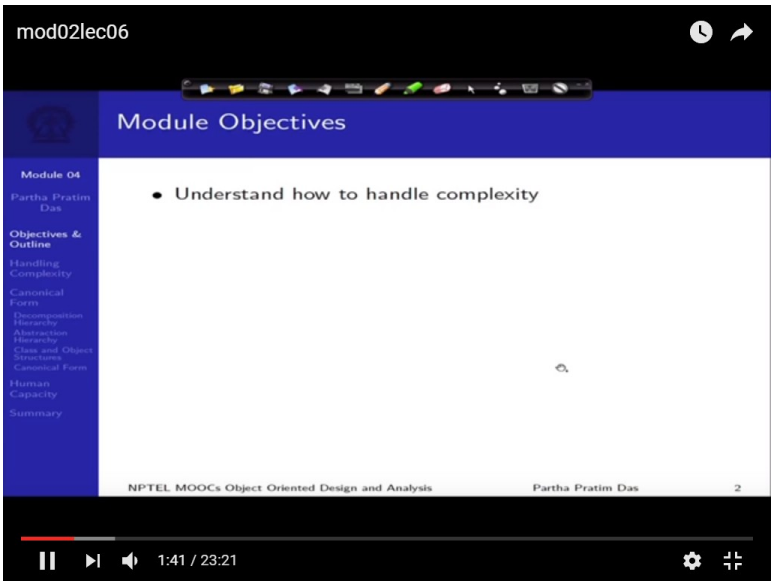
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 06
Object-Oriented Analysis and Design

Welcome to module 4 of object-oriented analysis and design. We have so far taken a look into the difficulties of software development and what are the, look at some of the complexities, sources of some of the complexities of software development and then in order to form adequate tools, instruments to be able to handle this complexity. In the last module, module 3, we have studied the structure of couple of sample systems and extracted from the study of those sample systems.

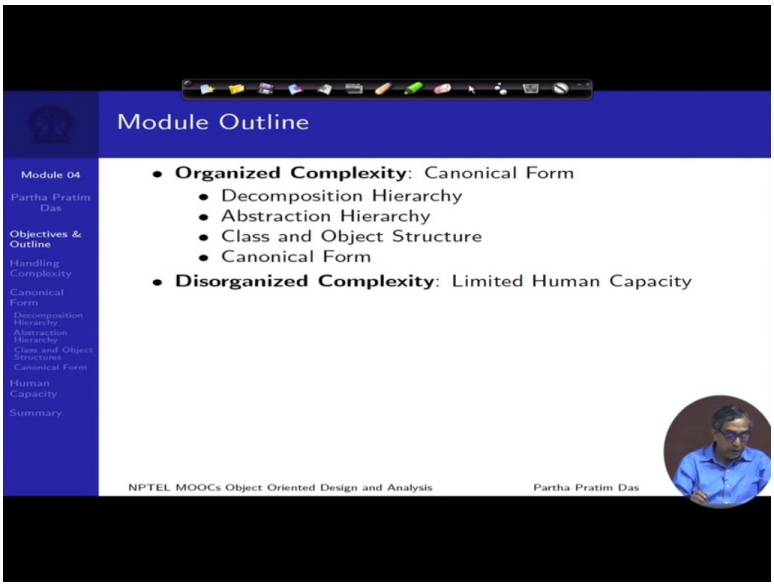
We have extracted 5 basic attributes of complex systems. We have understood that the complex systems necessarily has to be in hierarchical nature there has to be a definition of primitives, there will be separation of concern between its components, there will be certain common patterns which we will find across systems and we will need to define some stable intermediate forms for the development.

(Refer Slide Time: 01:42)



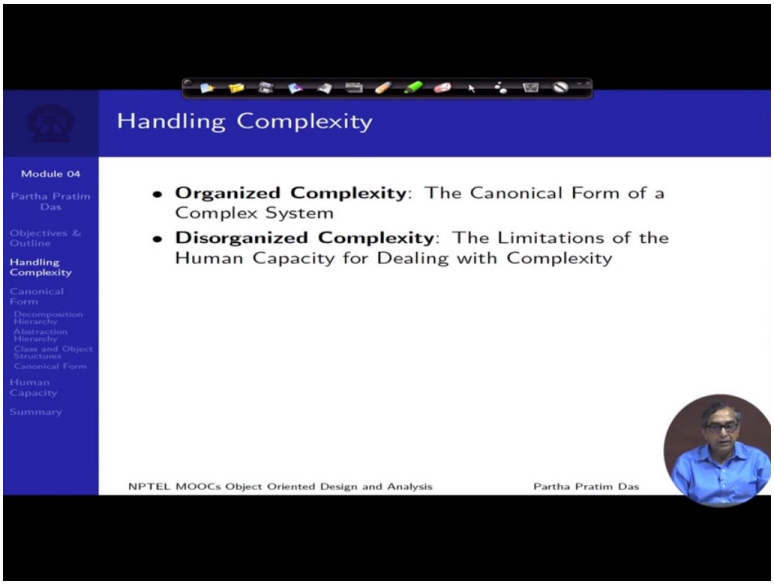
Equipped with that, now we try to get closer to the design problem, the analysis and design problem and look into how we handle the complexity better. So this module will primarily be concerned with the approaches to handle complexity

(Refer Slide Time: 01:44)



The module outline as it is in to the left of your slide.

(Refer Slide Time: 01:56)



Now in terms of handling complexity we just divide the approach in 2 parts or define rather we discover that there are 2 parts to handling complexity. 1 is we can say is organized complexity rather this is kind of complexity where we can we are able to find structure, we are able to find

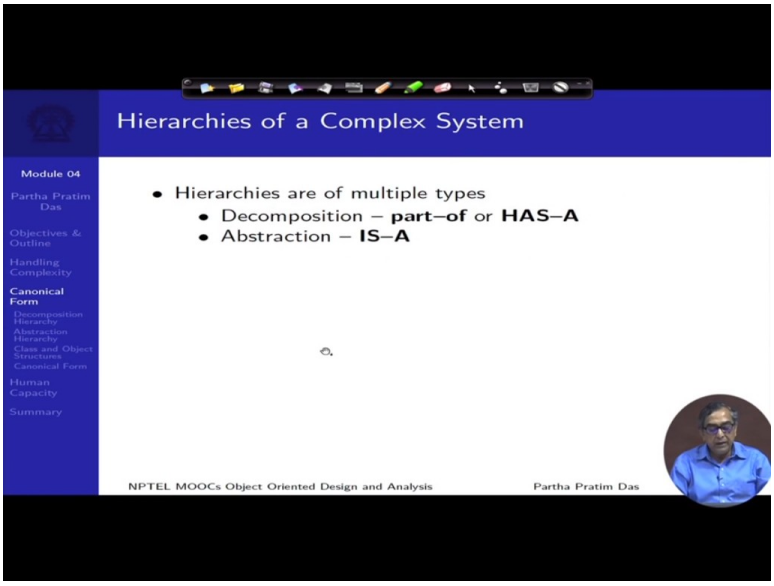
patterns, we are able to make device, tools and mechanisms by which we can approach the whole problem better.

So we will discuss those and leading to certain kind of a canonical form, standardized form for a complex system. So any complex system we want to design, we will try to express that in this canonical form and that is why it is called organized complexity. The other part of the complexity, unfortunately as it stands today is disorganized and that has to deal with us. A major or significant component or the most significant component in the whole process of analysis and design is a human game because it is a human process.

So unfortunately there is a severe limitation of the human capacity and that leads to a lot of complexity of the overall analysis and design problem. We cannot handle a lot of information together. So when we are thrown into a complex system for the first time, to put it in simple terms, we just get lost. No individual can master and capture that. So we kind of struggle like a like a child thrown into the water and trying to learn to swim.

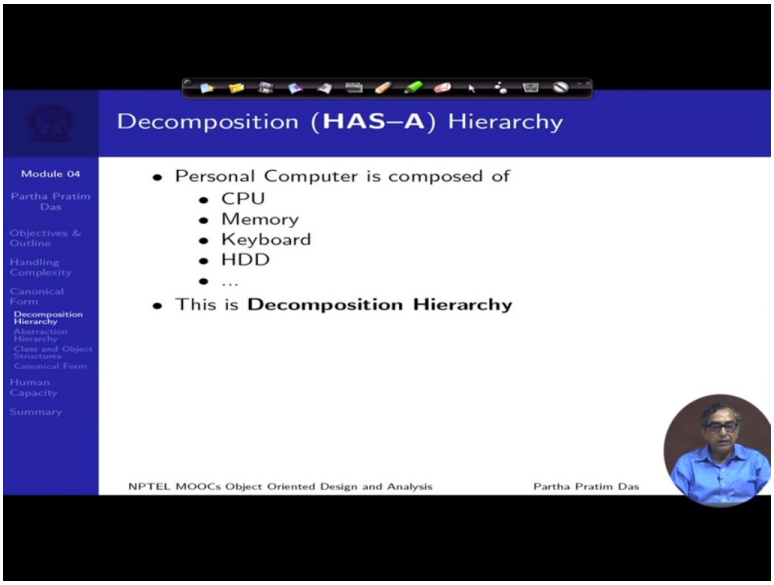
so this aspect of complexity is known as a or is characterized as a disorganized complexity because there is still a lot that needs to be developed in terms of techniques, tools and understanding before we can handle this more and this is one of the major source of issues that will continue to progress.

(Refer Slide Time: 03:50)



So the first let us take a look into the organized complexity we have seen that the complex systems are hierarchic now we go just one step further to say that hierarchy is not 1. there are multiple different hierarchies that are possible in a system and of this the 2 hierarchies which very significantly impact are analysis and design process are hierarchies of decomposition which is commonly called the part-of or has-a hierarchy and hierarchies of extraction.

(Refer Slide Time: 04:24)

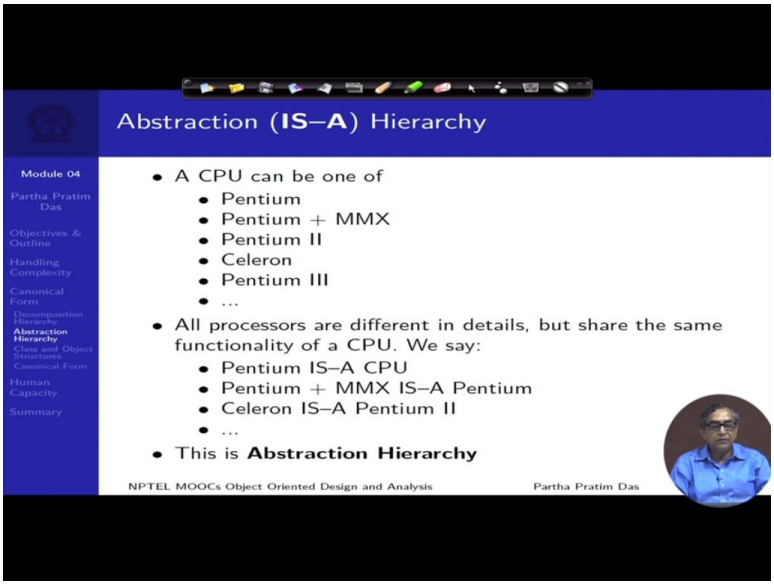


Let us take a look. So we did analyze the structure of a personal computer. The personal computer is composed of CPU, memory, keyboard, hard disk and naturally we identify that this is the hierarchy. so this hierarchy is kind of a compositional hierarchy so if you look at from the

top, we call this as a decomposition hierarchy that is if I break down CPU I will get this component, am, if I break down a personal computer, I will get these components.

If I break down a CPU, I will get ALU, I will get primary memory I will get parsers. If I break down an ALU, I will get resistors and so on. So such kind of hierarchy is called the decomposition or has-a hierarchy. has-a in the sense that we can just in terms of simple English phrase I can say that personal computer has a CPU, personal computer has a memory, CPU has a ALU so a short term form of decomposition hierarchy is has-a hierarchy.

(Refer Slide Time: 05:27)



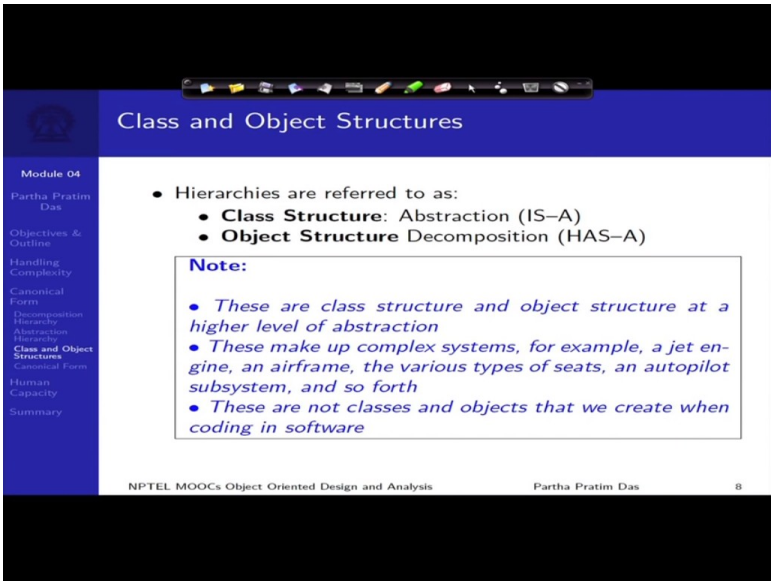
In contrast, let's say if I look at the CPU then I just say that CPU, I mean CPU has what is the functionality can execute programs. It will have some kind of an ALU, memory, bus and all that but we didn't say what CPU we have. We all know that there are different variations of CPU so it could be a Pentium, it could be a Pentium MMX Pentium II, Celeron, Pentium III so and so forth it could be. So this all these different kinds of alpha CPU and all of these different ones.

Now when we have talked about a CPU, we did not say which specific one is this. So all these processors are different in their details but they save the same functionality, the functionality of that of being able to execute a program, access the memory, work through the braces and so on so we can easily say that the Pentium is a CPU that it is an instance of, it is a kind of the CPU. But then we have Pentium MMX which is Pentium with multimedia functionality.

So we will say that Pentium MMX is a Pentium but then it has something more. Will say Celeron is a Pentium II because it whatever Celeron can do, whatever Pentium II can do Celeron can also do that. So we are saying here that comp notion of hierarchy is little bit different. We say this is kind of an easy hierarchy that is 1 concept say pent CPU has different kinds of variations all of which are actually CPU. One concept is Pentium has different variations but all of them share the same x86 architecture.

So this kind of a hierarchy of which we will talk lot more as we go along is known as an abstraction hierarchy or as you have seen that expressions use the simple English phrase of is a. so commonly it is called a is-a hierarchy.

(Refer Slide Time: 07:34)



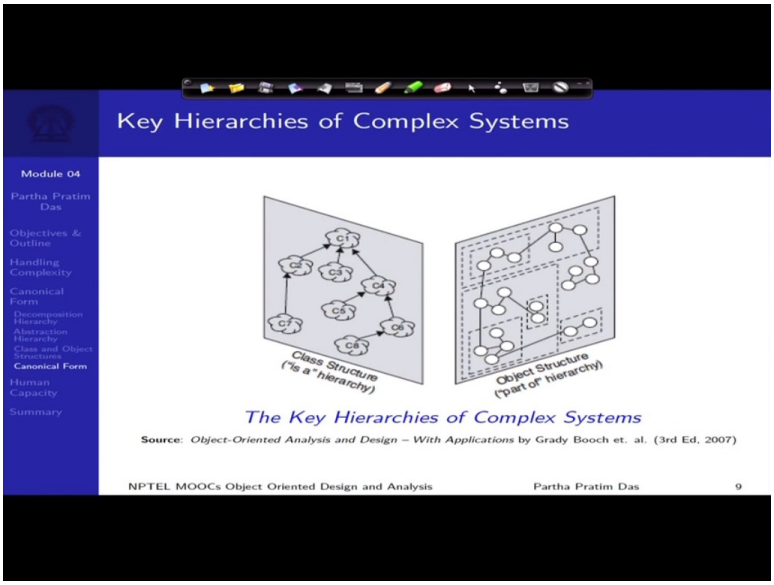
Now these 2 hierarchies are typically referred in terms of 2 structures, known as a class structure and object structure. The class structure gives us the abstraction hierarchy and the object structure gives us the decomposition hierarchy. We will show why it is this way so all that we are saying that kind of the complex system representation has kind of 2 orthogonal access 1 which talks of the abstraction hierarchy the other which talks about decomposition hierarchy.

The one which talks about the abstraction hierarchy we say is defines a class structure, one that talks about the decomposition hierarchy we say talks about the object structure. here please note

a couple of points these are if you are new to, completely new to object oriented analysis, design and programming particularly then you may ignore this note because you will not have this confusion but if you are if you have some background in op, oad

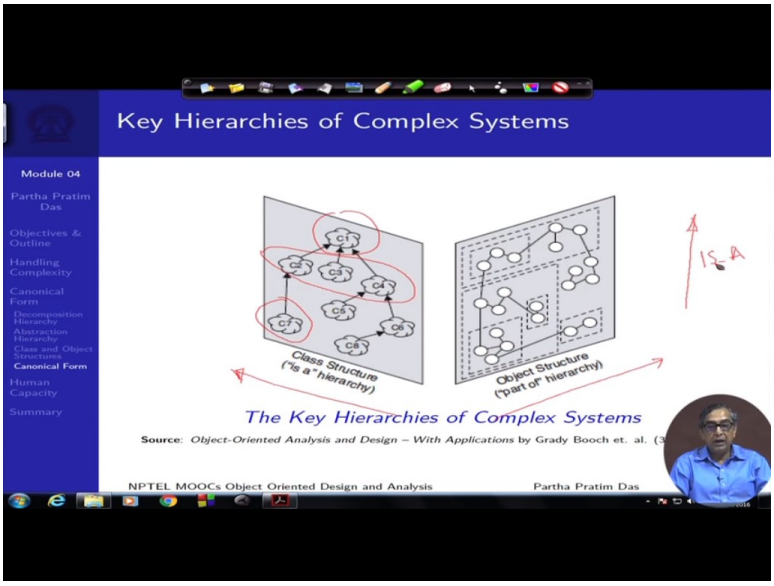
Then you will start confusing this use of the term class and object with what you know in say in programming language class in c++, java object doesn't stands and so on. But here we are not talking about those coding details. Here we are talking about high level concepts that are represented in terms of class and object structures.

(Refer Slide Time: 09:10)



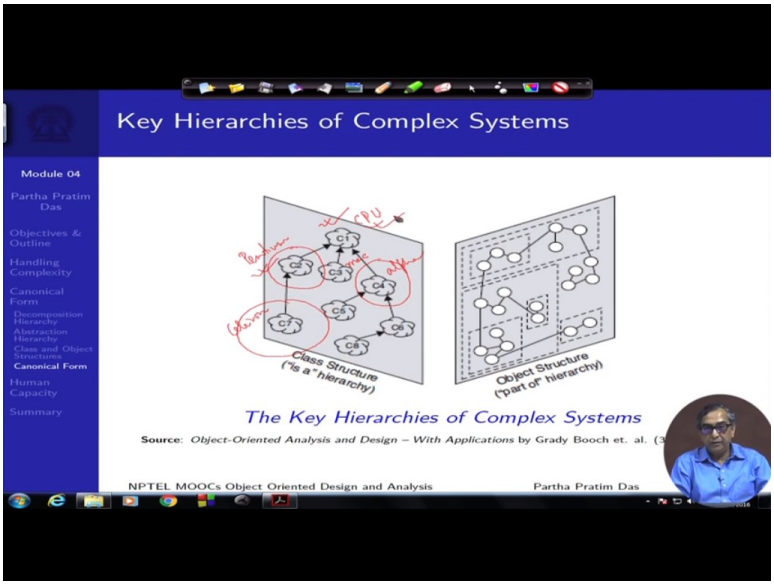
So this the very nice diagram from the text book that I have recommended to you from the booch's text book and you can see very clearly that here the 2 different axis are being drawn. This is the axis of abstraction and this is the axis of decomposition. So I can look at but we are looking at the some system if you look at the some system along this axis, then what you are saying is there is something like class c1. There is something concept which is the CPU.

(Refer Slide Time: 09:34)



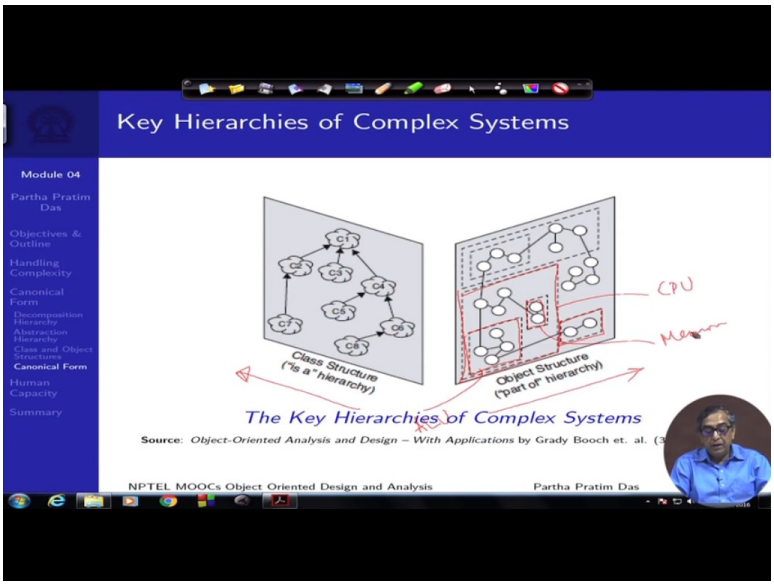
Then there are some classes concepts which have specializations of that that is classes c2, c3, c4 are like c1 but something more. So if c1 is CPU, c2 could be Pentium, c3 could be the mac processor, c4 could be the take alpha processor(10:15) and so on. Then we have c7 which is c2 these arrow basically its IS-A. So c7 is the c2, so the c7 is a Celeron is a Pentium.

(Refer Slide Time: 10:42)



So we have c so you have CPU here, we have Pentium ,we have mac, we have alpha, we have Celeron. So in terms of a personal computing system, say that these are different special kinds of CPU that are possible and that is what gives rise to the class structure which says that Celeron has all the properties that Pentium has which in turn has all the properties that the CPU need to have. Independently alpha has all the properties that CPU need to have.

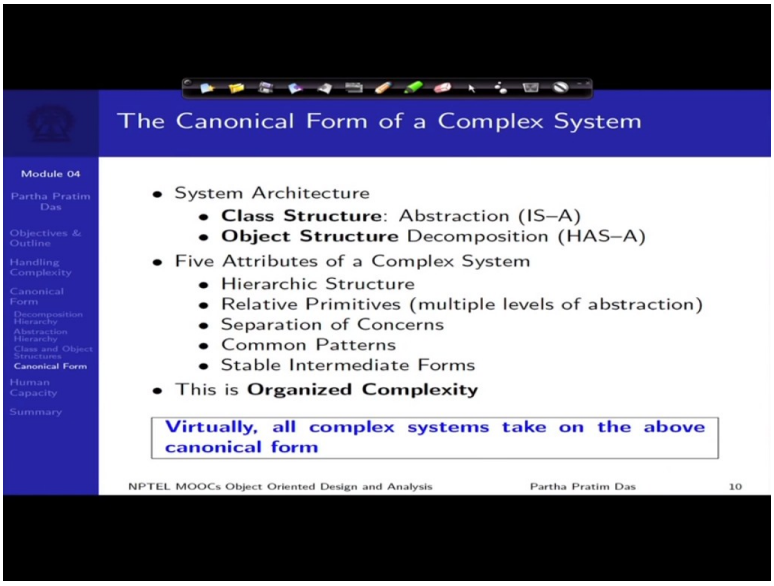
(Refer Slide Time: 12:25)



And when you look into the whole system in terms of this commonality of behavior, we say we are looking through the class structure that is the abstraction hierarchy. On the other if you look into this, you'll find that there are dotted boxes enclosing this it says basically this is this is one object; this is one entity we are dealing with. So this is another entity. So this if we look into this object or this entity then you have this, you have another.

So this is exactly a kind of decomposition that we are talking about so if this happens to be CPU, then this could be your ALU, this could be your memory and so on. So this is basically the decompositional hierarchy known as the part of hierarchy or the object structure so you can look at the whole system in terms of the different objects different component values it has and what is the sub component hierarchy it has. So these are the 2 key hierarchies through which complex systems of today will be characterized will be discussed.

(Refer Slide Time: 12:49)



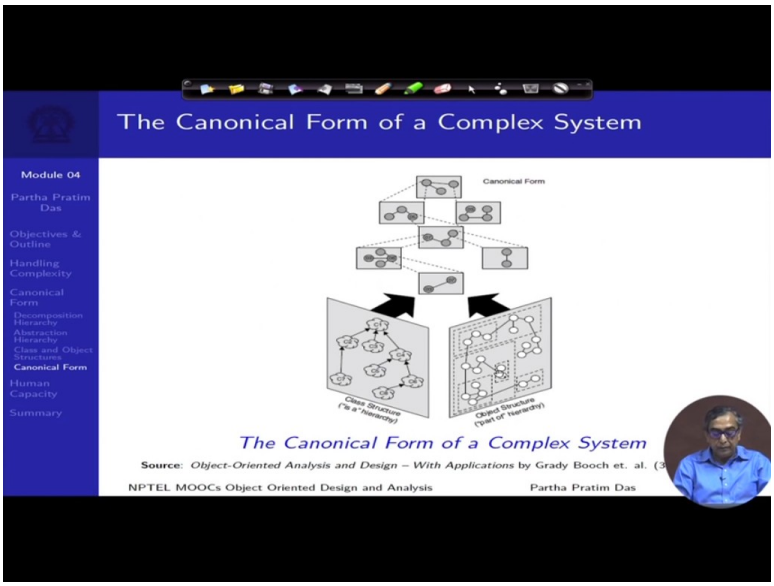
So with this, now we say the canonical form of a complex system, we can now describe that in terms of 2 major properties, 1 what we say is the system architecture that is the 2 orthogonal view of how does the class structure, how does the abstraction hierarchy look in that system and what is the object structure that is how does the decomposition hierarchies look in that system and these 2 together is commonly referred to as a system architecture for the complex system.

The other is what we have studied in the previous module, the 5 different attributes of the complex system and in terms of these attributes, we need to analyze the complex system and represent that so when we do all these together, the 5 attributes in the system architecture then we say that we have placed the complex system in a canonical form. So here the the main observation to be made the main advantage that we are trying to drive in is here we are not talking about any specific complex system.

No matter which complex system we talk about, we will be able to represent it in terms of these architecture and attributes and that is why this is called a canonical or standardized form of representation and once this representation can be made complete we will have much better understanding of the complex system. So in this process we have found that the overall complexity of a complex system in terms of understanding it, analyzing it dealing with it.

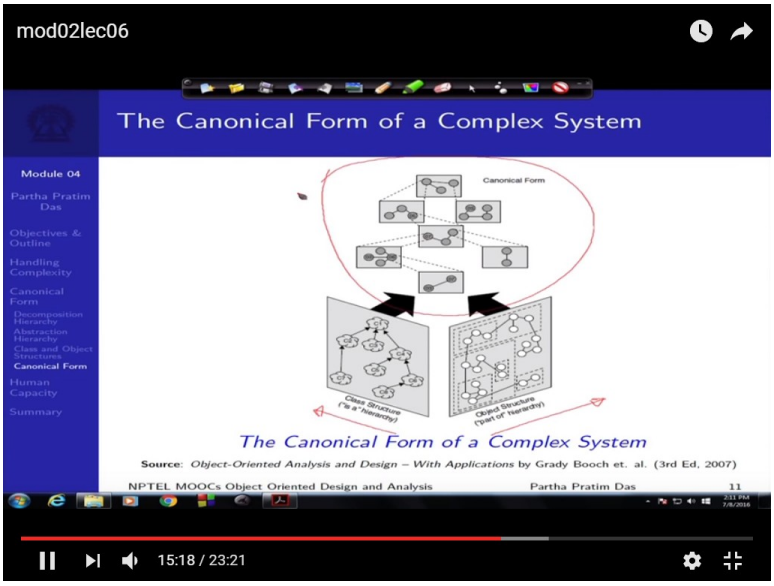
We have been able to somehow organize it under the canonical form and that is the reason we call this an organized complexity of the whole system.

(Refer Slide Time: 14:48)



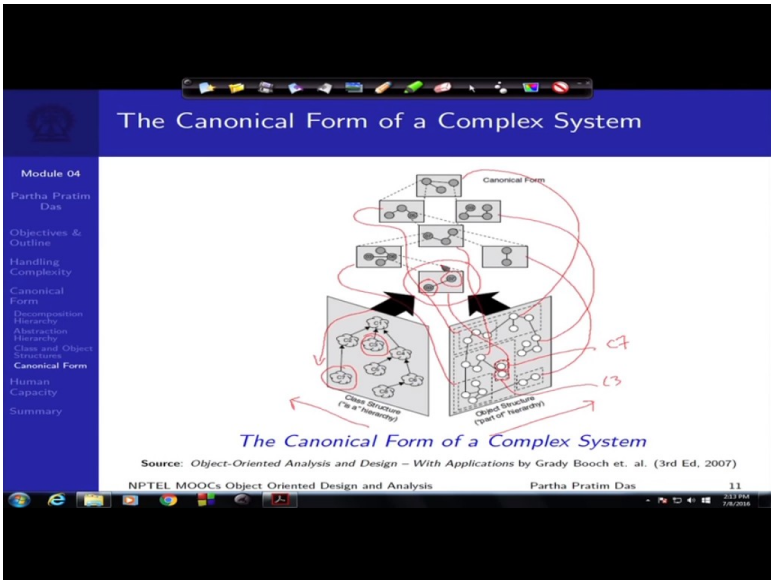
So this is an illustration of the canonical form with this is again from the same text book so if you if you look into this what it means, this is the class hierarchy, class structure that is the abstraction hierarchy, this is the decomposition hierarchy or the object structure and if you look at what these are these are actually the objects. So this this part which is which is being added new this part the canonical form part which is actually being added with those attribute is basically a cross product or combination of the these 2 axis.

(Refer Slide Time: 15:20)



So if you look at you have an object here and I think this is this object. The object here is this object; the object here is this object. You can just verify, the object here is this object, the object here is this object, the object here is this object. This means is basically the same the object here is this object. This is basically the same object but what you can read with a little bit of care with some am sorry it is not that clear in terms of the size.

(Refer Slide Time: 15:20)



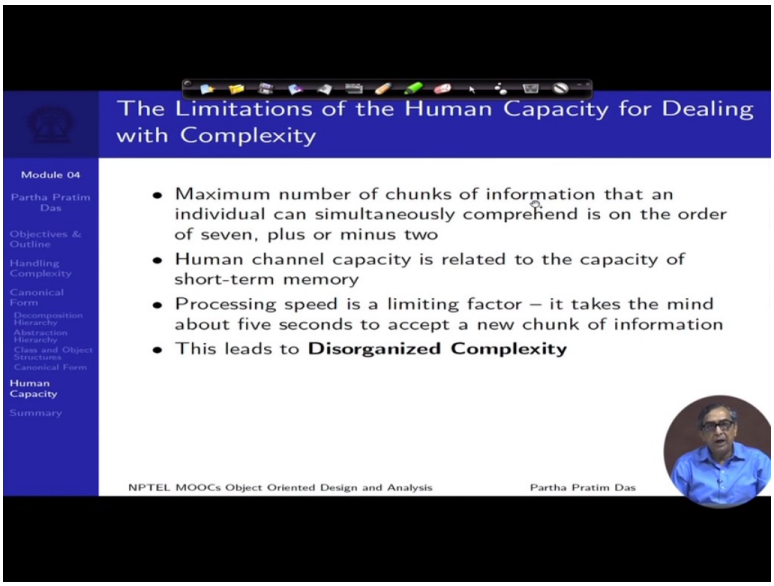
It is here it is written to be 3 which means that this is this object which in turn is this object is a c3 object from class state this is 7, so this is a class 7 object. So you can say that this is a so this is a c7 object. This is a c3 object. So this hierarchy tells you that how these objects are going to get their behavior from higher downwards and this hierarchy tells you which you represent here as well as here is how they are put together. How their component structure is?

And we will see when we go which of these approaches is into defining object models will see that these are what will start calling them as a basic abstraction or inheritance properties will start calling them as modularity, encapsulation and all those properties. But this is basic system structure that we need to keep in mind. So I would request that please carefully study this diagram. This is a very interesting illustration now.

How should a complex system canonically look like when analyzed from the 2 orthogonal axis of class structure or abstraction hierarchy which say what is the specialization amongst different

concepts in that system and the object structure or the decomposition hierarchy which says which is a component, sub component, sub-sub component relationship and as a intermix of these 2 canonical form emerge in the middle which is the system that we are actually trying to build up.

(Refer Slide Time: 18:00)



Now coming to the other side, so this was about the organized complexity. But finally the human beings will have to analyze and actually build up this canonical form from as we discuss mentioned earlier also from some vague descriptions from some vague interactions, half knowledge broken ideas and so on and when someone is thrown into a complex system, it is goes well beyond the comprehension of that person.

And here some of the observations that different psychologist and psychiatrist have tried to quantify over a period of time. What is commonly said is a maximum number of chunks of information that is an if you if you deal with the simple information that we can simultaneously comprehend is kind of a 7 for a human individual. Now for some it could be little less for some it could be little more that is around that number. Now this chunk of information could be anything.

It could be independent concepts, it could be number of variables in a in a function, it could be the different components or different important things that you see in a picture and so on. But at the same time, we cannot handle too many concepts together too many junks of information

together which makes it extremely difficult for human mind to be able to analyze or come to terms with the complex system.

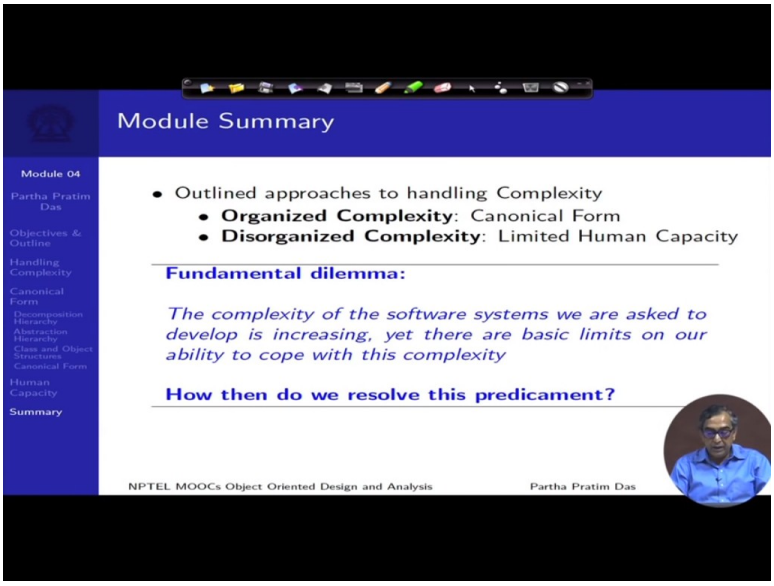
And it is believed that this you know limitation of the human capacity is has to do with limitation of short term memory which is which we quickly acquire and quickly forget and only small number of items can be kept in there. The other limitation which creates a significant bottle neck is the processing speed of the limiting is a limiting factor. The human brain is extremely slow so if you are given with a new chunk of information we take about 5 seconds to come to terms with what is it, what does it mean.

So only 7, 8 chunks of information each being processed with 5 seconds 10 seconds of time, the whole system has a disadvantage of time of not being able to handle the complexity that one need have and that is one reason that the whole process of the human engagement in dealing with complex system has to be done through different development of practices, processes, organized behavior, team building and so on.

And as I mentioned at the beginning of the module is these are still being formed and formulated. You must have heard about different paradigms of development. Some ad-hoc paradigm, traditional paradigm, waterfall model and then you have agile then you have iterative and all those. These are if you if you look into it these are actually attempts in trying to make better and better system.

So that from this kind of very disorganized behavior of the human mind, we can take it to a mode structure form where you can handle more information with the less amount of time. But this remains to be a concerned area and really we will not be able to add lot of value in terms of this particular aspect of the whole problem.

(Refer Slide Time: 21:52)



So to summarize we have outline here. Some approaches of handling complexity. Primarily we have talked about the canonical form comprising the system architecture and the 5 attributes and the canonical form is what you will have to very clearly understand and try to remember because in the next week when we start talking about object models and so on, we will frequently fall back on this canonical form for explanation and understanding.

And all these together is called the organized complexity and we have observed that due to the limitation of human capacity we have a big part of the complexity which is still quite disorganized and difficult to handle. So in this context in this background we are not posed with fundamental dilemma that the complexity of the software system that we want to develop is regularly increasing, constantly increasing.

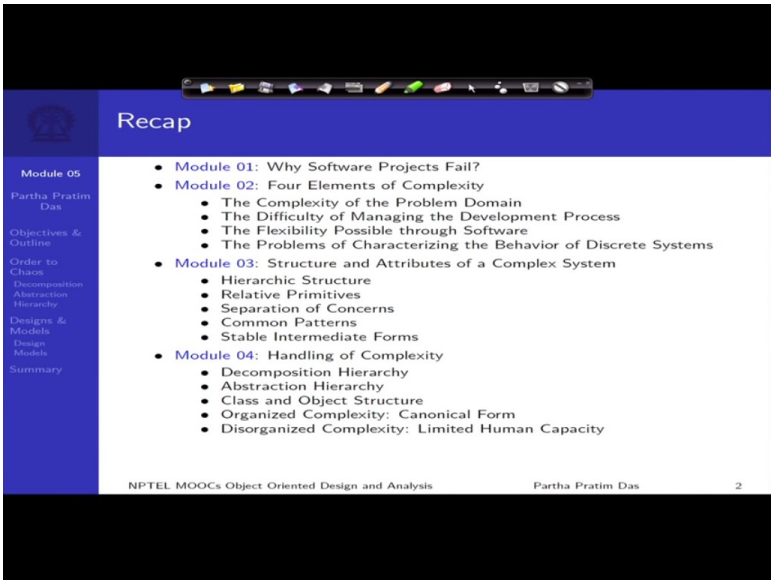
And but our own limitations are fixed we or capacity of being able to cope with this complexity is fixed. So how do you resolve this predicament and that is what will be we will try to outline some of those aspects of how probably we can start handling and behaving this in a way so that we can handle it better in the next module, in the module 5.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 07
Bringing Order to Chaos

Welcome to module 5 of object-oriented analysis and design. Here we will primarily talk about how what should be the broad approach designing object-oriented systems before I get into the specific objectives.

(Refer Slide Time: 00:41)



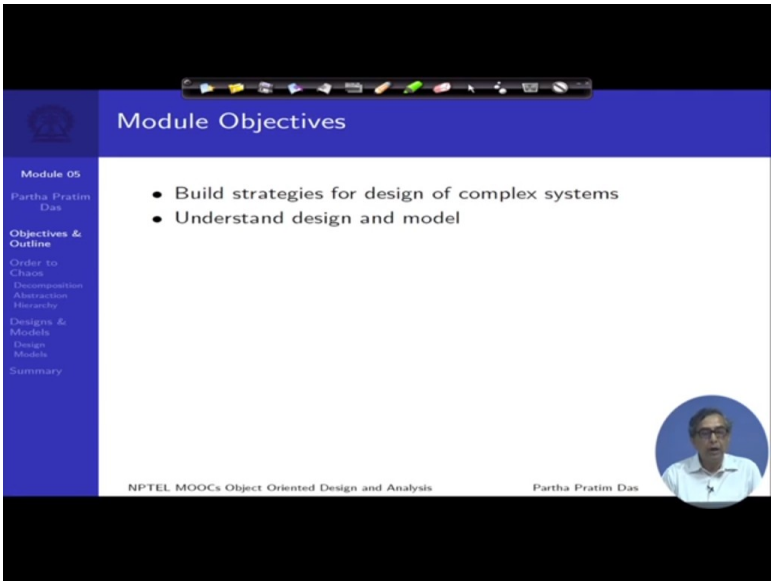
Let us just quickly recap the major points that we have discussed so far in the course. Initially we talked about why software project failed and identified that software development, software projects are inherently complex and there are 4 primary elements of complexity coming from the problem domain, the difficulty of development process managing development process, the flexibility and the problems due to the discrete systems structure of the software systems.

Then we try to study different complex systems, natural, manmade, social and so on to understand if there are commonality between difference complex systems and we identified 5 different attributes for complex systems starting from the hierarchic structure, the primitive relative primitives or the building basic building blocks, separation of concerns, common

patterns and stable intermediate form. Equipped with this, we did discuss in the last module about how we can handle complexity.

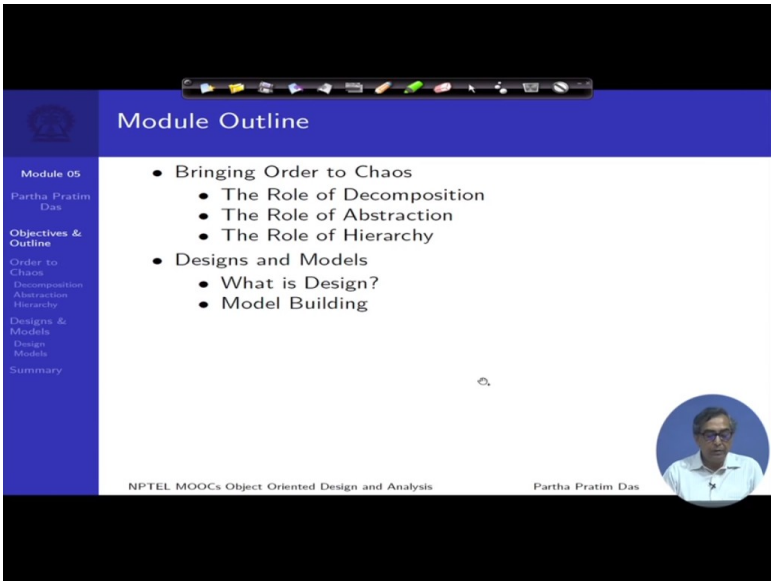
And we observe that the complexity of the design, development process of a complex software system can be divided in terms of organized approach which is primarily driven by a canonical form comprising the object and class hierarchy, the 2 orthogonal dimensions of representation in a complex system and the 5 basic attributes of the complex systems. At the same time, there is a quite a bit of disorganized complexity arising out of the limited human capacity to handle a limited number of information chance at a given point of time.

(Refer Slide Time: 02:47)



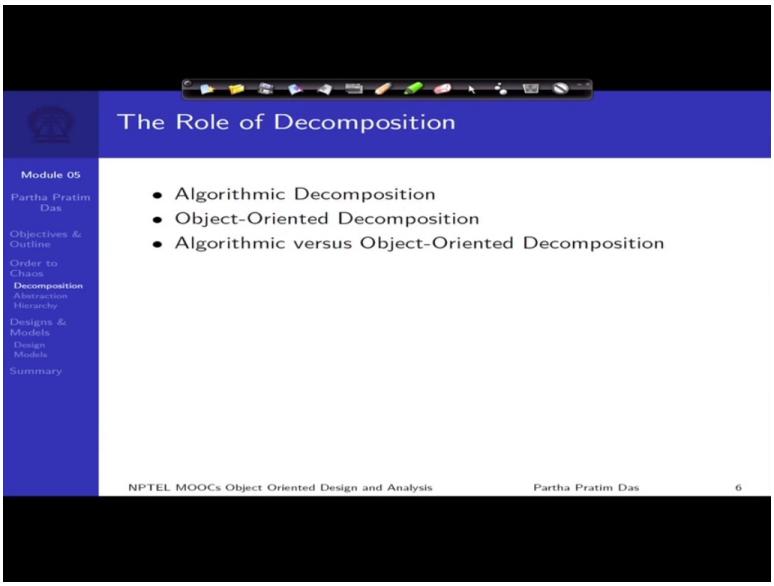
So equipped with all these in the current module, we would like to discuss about some basic strategies for design of complex systems. We will define what is our understanding of a design and that of a model.

(Refer Slide Time: 03:01)



The outline will be primarily about the part to approach the whole design process in terms of the roles of decomposition, abstraction and hierarchy.

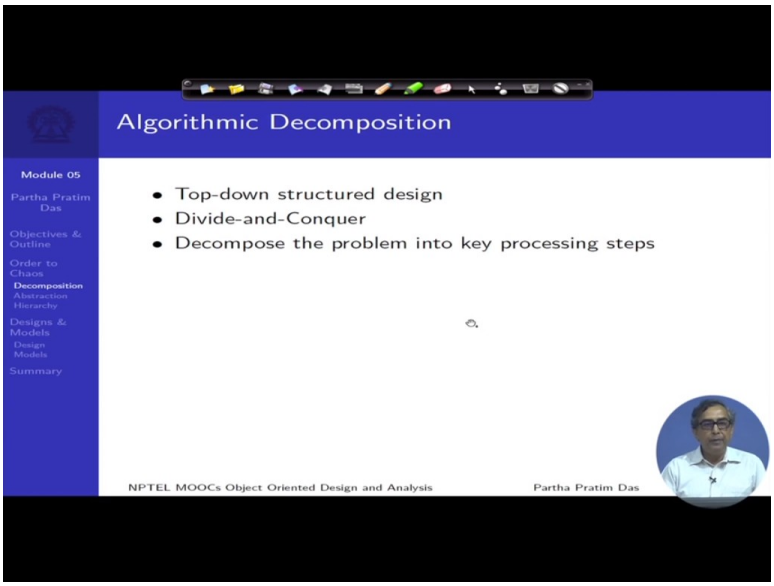
(Refer Slide Time: 03:16)



So we start with analyzing the role of decomposition. As to we have seen that there is decomposition structure, there is possible decomposition of the system in in multiple several instances we have seen that. Now we want to really investigate that given a system how do we go about doing this decomposition and we note that there are primarily 2 approaches that have emerged over the last 50, 60 years that software development has been practiced.

First is the algorithmic decomposition approach and the second is the object-oriented decomposition approach. We will take a look into each one of them and then towards the end, kind of compare them to see what has an advantage over the other.

(Refer Slide Time: 04:10)

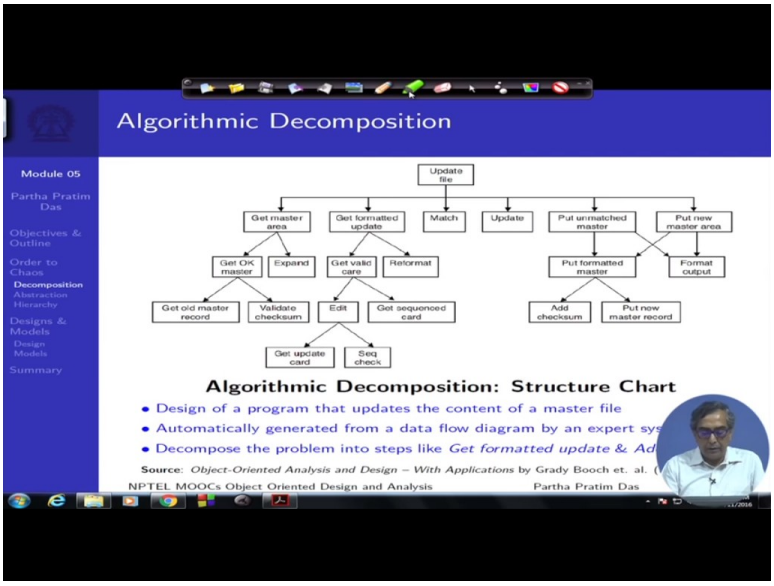


So algorithmic decomposition this is certainly the most fundamental, all these this has been around for a very long period of time and is primarily a top-down structured approach that given a problem, you can we can divide it into a number of 2 or more, usually a number of different sub problems which are simpler in nature and whose solutions together can solve the whole problem.

In very specific terms, we have studied this kind of approach under divide-conquer kind of algorithmic paradigms while we did binary search and match sort and things like that. So in general, what we will look at algorithmic decomposition as is to decompose the problem in terms of key processing steps so the main thing here to note in algorithmic deompostion is you decompose the problem in terms of key processing steps.

So what is inherent of algorithmic decomposition is you identify processing steps then identify the key processing steps and try to decompose the whole problem in terms of it.

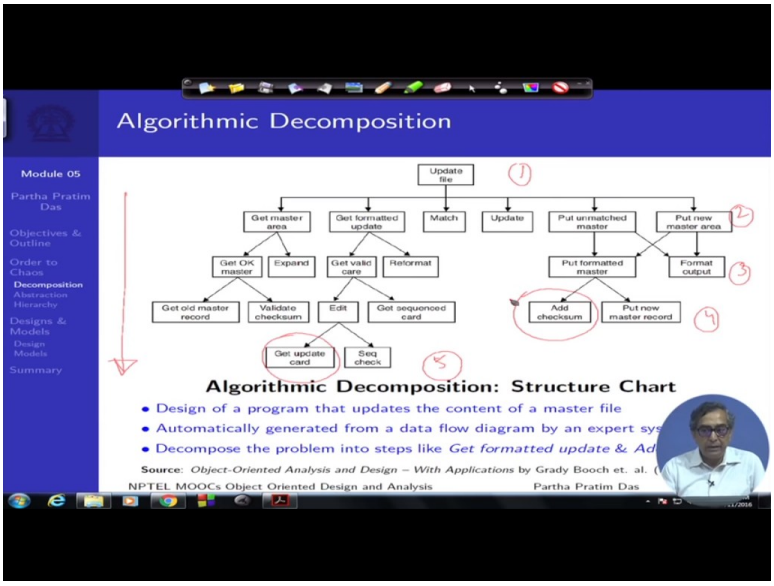
(Refer Slide Time: 05:35)



So here is an example. This example is is from our text book so here the problem is to update a file. So this is the top level problem that we want to actually address, solve. So for doing that, certainly a file will have some master area where the information about the file is written. Then the format of the file then there are several matching and update operations that need to be done. If I am writing the updating the file, the new master area need to be put up and so on.

So based on that the update file, this whole task of update file is divided in terms of a number of subtasks or sub goals which are given in this layer. So this is one level of decomposition where this is the first level in which the tasks are decomposed. Then if you concentrate on any any specific of this tasks they get master area this task has to be accomplished, then this can be done in terms of going to other subtasks of get the ok master and to expand.

(Refer Slide Time: 06:00)



And it really does not matter that much if you do not specifically understand every detail of what these subtasks are in this particular case because what we are trying to explain is a broad idea of how decomposition can be done. So you can see that over this whole tree of this is the root level and 2,3,4 and 5 levels, we try to solve this problem by algorithmic decomposition and whenever we reach a level say get update card, add checksum, tasks which do not have other sub tasks.

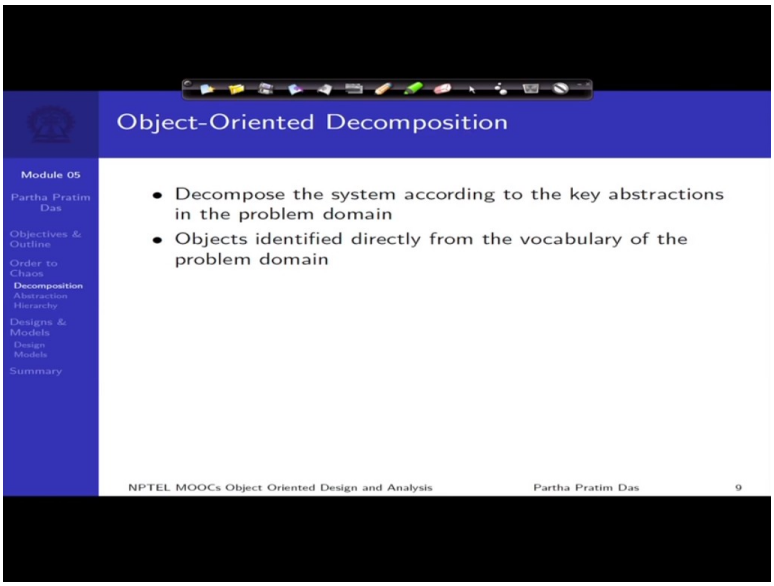
That would mean that is at a level where it can be directly implemented in terms of an algorithm possibly in terms of a specific function. So this is one way to actually try to decompose a problem and start solving it and this particular kind of a chart is known as a structure chart because it shows how the tasks are structured and in this particular case this chart was generated from an expert system tool.

So that is interesting because that means that a quite a bit of automation can be done on this. The input was a data flow diagram. You may not right now be aware of what a data flow diagrams are over the course of these discussions, we will also talk about data flow diagrams. But an expert system tool can take the data flow diagram which basically says how data is going through the system and make this structure chart of algorithmic decomposition

So in this it has divided the problem into very specific different steps like get formatted update as an here or add checksum as an here or some of them are primitive tasks and some are still

composite tasks which need further subdivision. So this is the basic approach of algorithimic decomposition you look at the whole task. We are basically going by the processing logic, try to divide them into subtasks, take up the subtask and try to divide that further into smaller subtasks till you get into primitive tasks that can be directly solved.

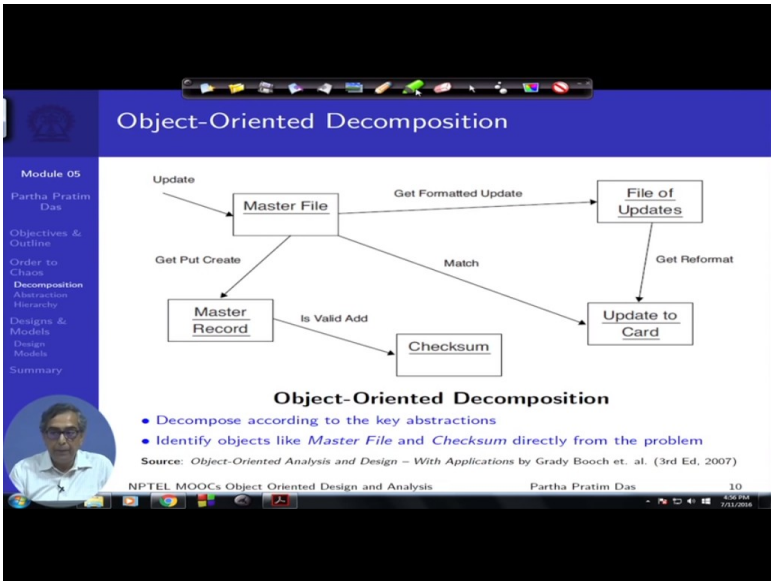
(Refer Slide Time: 09:12)



In contrast, in contrast we can do an object-oriented decomposition. In object-oriented decomposition, we do not look at the specific tasks. Rather we try to decompose the system according to the key abstractions of the problem domain that is we take a look into directly the vocabulary of the problem domain. What is the problem domain talking about? Updating file means what are the basic concepts, what are the basic entities, the basic agents that the problem is talking about.

And we will try to decompose the system in terms of those key concepts, key ideas which we referred to here as key abstractions and then try to model and solve the problem. So it is it is quite different from the way the algorithmic approach will go.

(Refer Slide Time: 10:10)

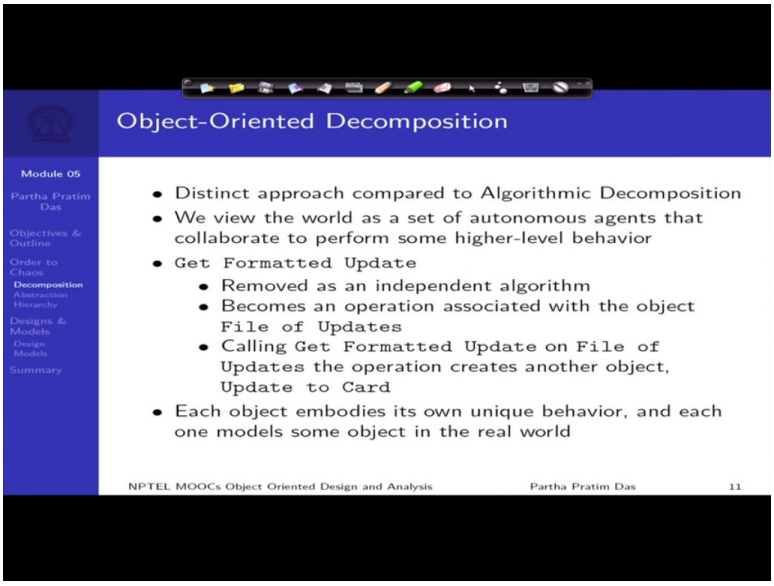


So in here for the same problem of update representation, here we basically divide the problem into number of different agents. One we say is master file, naturally that needs to keep track of what is happening to the other files. This is the file of updates, this is update to card, the checksum and so on. Each one of them is a key concept, is a key abstraction in the whole problem domain and if you look into the arrows,

These arrows mean that file of updates is an object which support this operation that is master file can invoke gets formatted update on this file of update objects or master file can invoke get, put or create operations on the master record object. So the view has become completely different. There is there is no specific get formatted update functionality and the breakdown of that, rather thats gets distributed in terms of a number of different objects

And those objects are identive objects like these are identified directly from the problem domain where we find that these are terms that these are the vocabulary that the problem domain is dealing with.

(Refer Slide Time: 11:50)

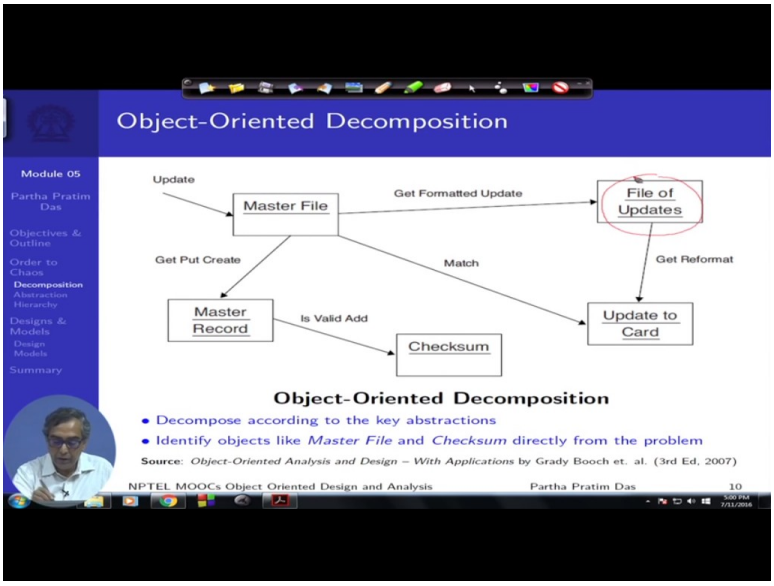


So this object-oriented decomposition approach is quite distinct compared to the algorithmic decomposition approach. The first thing is we view the whole world as a set of autonomous agents. This statement is a simple short statement but this is a very strong statement about object-oriented decomposition in fact about object-oriented analysis and design. That is why we think of that this whole, whole system as if it is a collection of certain autonomous agents.

Autonomous means active entities which can behave on their own kind of if we if we close our eyes and try to think about the system as if there are a number of agents floating around in the in the universe and each one of them can independently act based on what its behavior is, based on what it is requested by others to do and based on what it needs to request others to do. So we will take a look into those aspects but just remember this is the basic view

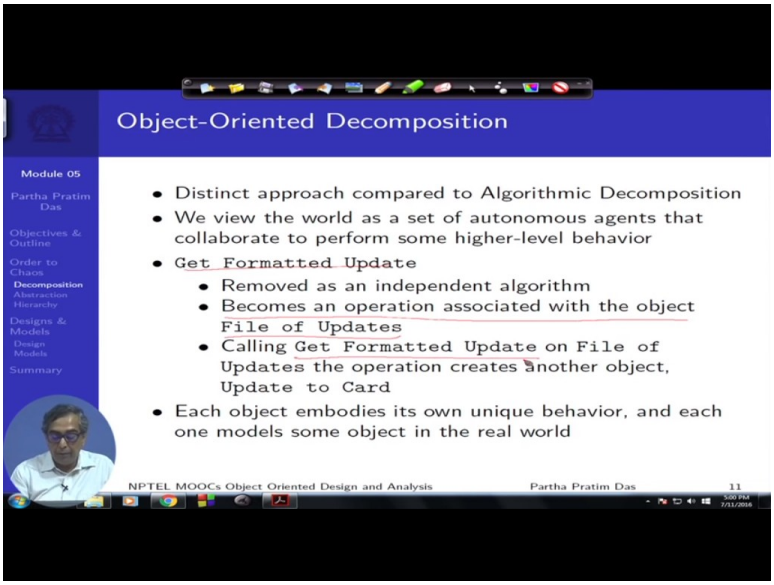
That is a number of autonomous agents collaborate, work together to perform some high level behavior. So let us say how. How does our original functionality of get formatted update how does that happen. This is now removed as an algorithm so we dint have we do not have any specific algorithm on this. It now has become an operation with the object file of updates.

(Refer Slide Time: 13:47)



If you are unclear about this let us just one step go back and if you if you see here now this file of updates, this object has this as a method. This as a request that master file can put to this particular object. So this is the difference, this is the change in structure that is happening between the algorithmic and the object-oriented decomposition.

(Refer Slide Time: 14:14)



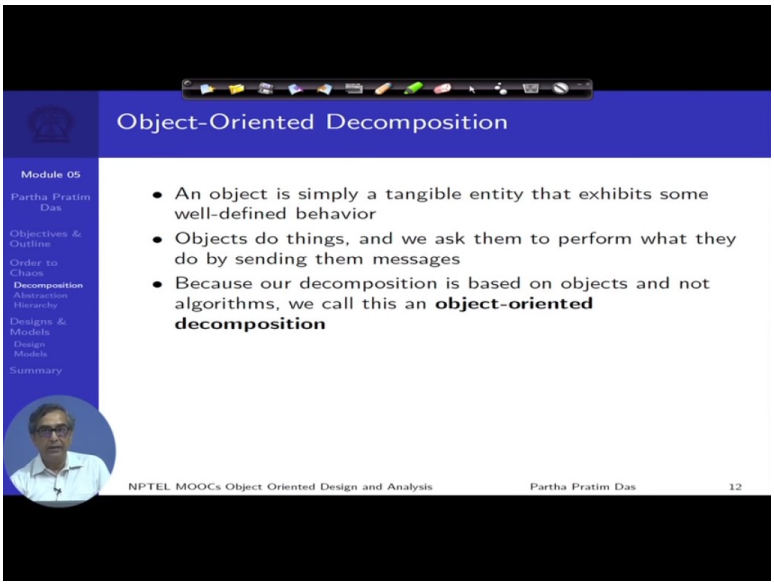
So that is why we are saying that now this get formatted update becomes an operation associated with this object. Similarly when we call this get formatted object on the file of updates, this is an object, the operation creates another object update to card. So it will in turn create an object which is a update to card. As I show you in that diagram, let us say, here as this is operation is

invoked on this object, this in turn invokes a get reformat which generates a new update to card object.

So the way the whole thing is modeled, the way the whole thing is now being decomposed and represented are very different and all are in terms of this concept of autonomous agents which are now this particular file of update object, update to card object, these are collaborating to actually achieve the functionality that earlier which are represented in terms of a complete tasks, subtasks decomposition for algorithmic approach.

So every object has its own unique behavior and each one models something in the real world. Now this is this is this is not easy to see ah at this stage may be. You you may be feeling confused. As you do more and more examples this will become clear. But keep this in mind that these autonomous agents are not arbitrary. Every autonomous agent must correspond to some basic concept, some basic uniq behavior in the real world, in the problem domain that you have. Only then your design would become really effective.

(Refer Slide Time: 16:18)

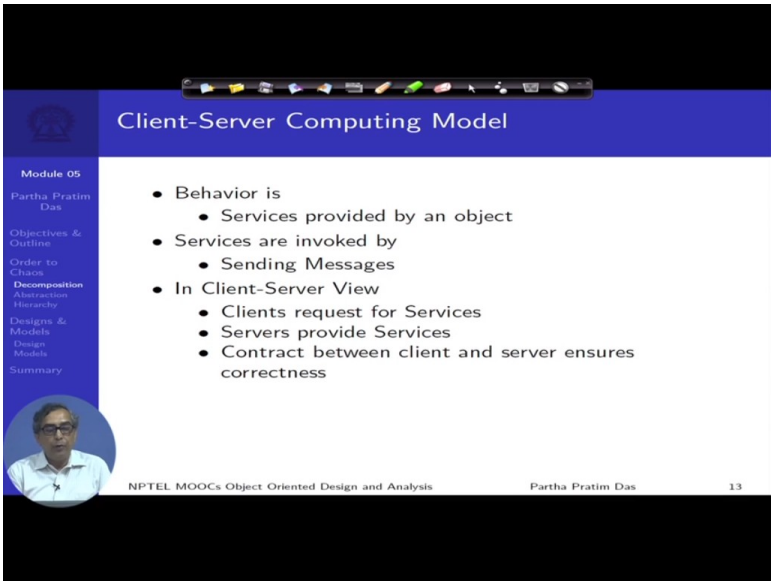


So an object is simply a tangible entity that exhibits some well-defined behavior. Tangible in the sense that in the real world it has some proper existence and it can do things that we ask them to perform and they do this by, this is interesting they do this by sending them messages that is the

main thing that the objects can do they can exist on their own as autonomous agents, they can perform some tasks which we say are operations

And they can send some information, some message to other objects to perform some tasks. This is the whole view and since this whole view is based on the objects rather the autonomous agents and not on the specific algorithms that they perform we call this a object-oriented decomposition.

(Refer Slide Time: 17:10)



So let us take a look at them if if this is how we think about doing our decomposition then how is our problem getting solved. If have done an algorithmic decomposition then we all know how the problem will get solved. We all know how what is the computing model. What is a computing model? Algorithms correspond to basically functions in in some high level languages as you can perceive

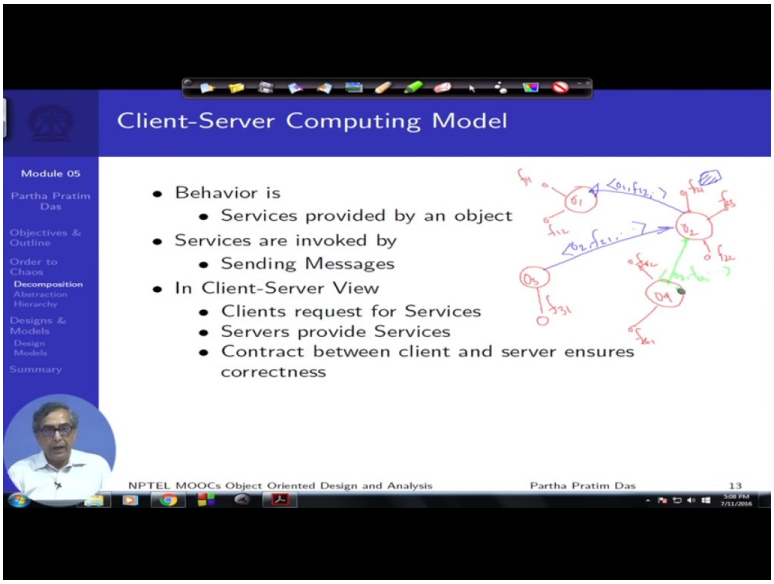
And we always know that functions can be executed in the CPU crossly in a one and one model any algorithm can be executed or if I talk in abstract terms then using turing machines, the algorithms can be executed so we know that it is it always have data in the memory. It will have instructions or operations or primitive tasks that it can do in terms of the instructions. It will take the data, perform the instruction and put the data back. That is the basic computing model.

In a object based system, the computing basic idea of the computing model itself is very different and this is because of 2 reasons. One reason is that when we are talking about object-oriented decomposition which has resulted or object-oriented design has resulted after lot of maturity of software system design. This is not as primitive as a algorithmic decomposition or algorithm based design of systems.

So by the time we started doing object-oriented design and object-oriented decomposition, we are equipped with a variety of different computing models, some of them are purely ---18:50) some of them are follow certain languages like Fortran accept languages like Fortran, Pascal, C, C++ and so on. While others could have multiple processors, they could have different shared memory models, they could do some remote procedure execution and so on a variety of different computing models over the years have emerged.

So the other aspect of object orientation decomposition is also to keep in mind that I decompose what is my computing model and when I decompose I want to make minimum assumption or rather I should say a minimal assumption about my computing model. I do not want to, I yet do not know what will be the best way to design and implement this system. So I do not want to assume that I will run it on a Pentium 3 or I will implement it on a decal processor with 8 cores and so on, I do not want to make such assumptions.

(Refer Slide Time: 20:02)



So I make a simple assumption that it is it is a client server computing model which say that I have a collection of autonomous agents the objects, so object 1, object 2, object 3, object 4 and so on. And on every object I have some possible operations that I can do. So this is known as the services these operations are known as the services offered by the respective objects. So these are the different services that object of.

Now the way you it works is any object can send message to another object. This object say if we just turn over to loop, this mess object can send a message to this object saying that I want to use your method or your operation to 1. So it kind of sends a message like I want to go to o2 and use f21 and so on these are the data on which this f21 should be used and so on. In so what happens is when o2 saves that whatever is this method operation, that method operation will start being performed.

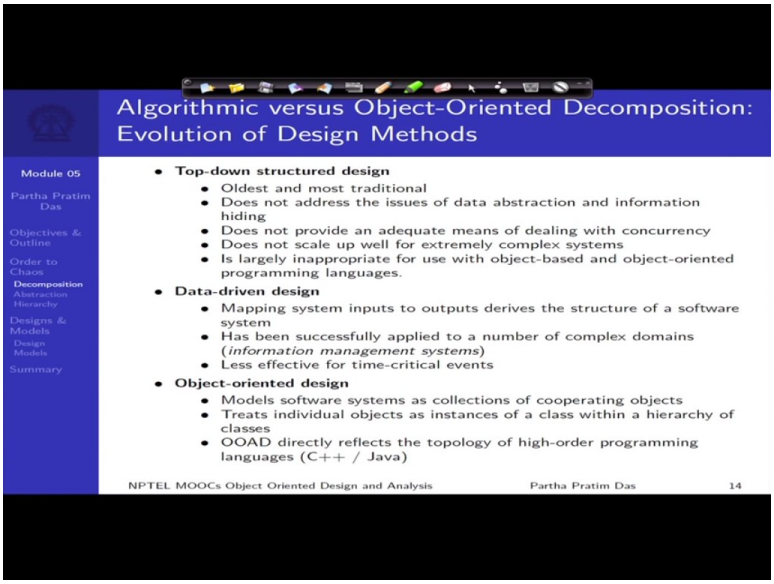
When that is being performed, o2 may need that I would need the help of the operation f12 of object o1 so it sends a message to object o1 with f12 with associated data. And while this is all these are happening simultaneously on a separate note, o4 may have sent another request to o2 for doing some operation f2 and all these can keep on happening at the own speed case at the own speed independence and this is what is known as the client server model.

Why this is anybody who requests for example here if o3 was requesting so we will say here is the client and who was being requested, o2, this is called the server. So as if as as you say that we we say that ah ask for services for the human vendors, it is a similar concept that the client will ask for the service from an object. Now there is no specific object as a client, not a specific object as a server, it depends on who is in that what mode.

In a whenever a message request is sent the sender who is requesting for the services is the client and the receiver who will provide the service is a server and this happens in a completely distributed manner. So you can say understand that as long as you you can have some computing system some realizable system, some implementable system where you can have this autonomous agents or objects and you have some mechanism to pass messages, send messages you will be able to implement a client server model.

So if any concrete terms we talk about say C++, then basically objects are instances of classes and the sending of messages is nothing but nothing other than actually invoking or calling method functions. So that is the basic model like just little bit discussed it so that as we go forward and keep on doing more and more of how the complex systems are designed, we will have this particular model in the background.

(Refer Slide Time: 24:00)



So we have discussed the basic of algorithmic design, algorithmic decomposition, and the object-oriented decomposition and we will next take a look into how do they compare based on that historical evolution.

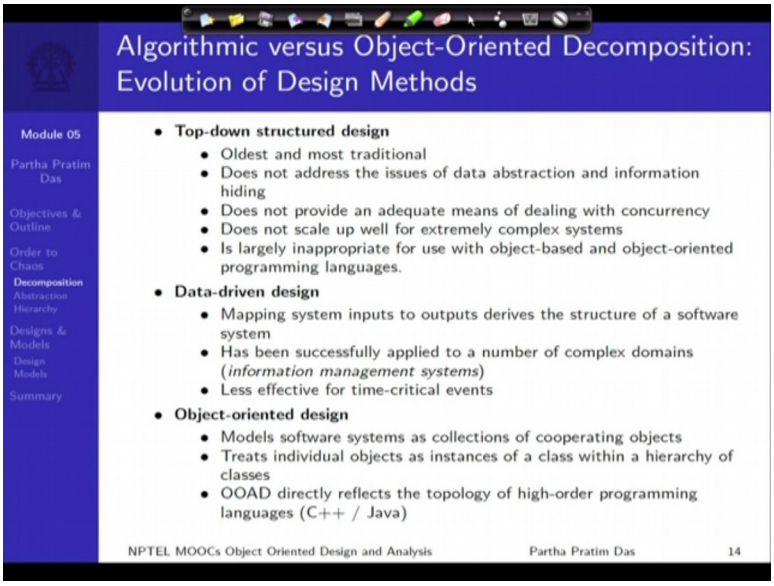
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 08
Bringing Order to Chaos

Welcome back to module 5 of object-oriented analysis and design. We have been discussing about the role of decomposition in modeling and solving complex system design problems and we just briefly took a note of the algorithmic decomposition approach and the object-oriented decomposition approach and in the object-oriented decomposition approach, we have noted that there will be a set of autonomous agents typically called objects which can pass messages between themselves.

So that whenever a object sends a message to another object, the sending object is typically called the client and the receiving object is called the server and this receiving object will provide the service that the sending object has asked for and that's a basic client server model on which the object-oriented decomposition is structured.

(Refer Slide Time: 01:19)



Now let's quickly take a look into how does these two decomposition approaches compare. It's basically this if you look into the history of how what different design methods have been followed they have been naturally tens of them, if not hundreds of them but the most dominant

ones have been the top-down structure design where you start from the most complex system as a whole and then you start breaking it down.

This is being the basic approach of decomposition from a very long time old days certainly the most traditional but certainly this top down approach is closest to the algorithmic decomposition it does not address issues of data extraction, it does not deal with concurrency, it naturally it does not scale up well with the extremely complex system. It's understandable because this is been very very old approach and therefore in those old days the system capacity itself was very limited.

So, the hardware capacity itself was very limited so all that you needed to do is to divide the problem to the extent that the hardware can handle and also there is no question of handling concurrency because there was always just one single processor executing everything. So, this approach primarily went on for quite some time and it's still is predominant in many with many of our designers and many of the design areas and needs to be understood.

But for the broader community of object based and object-oriented languages, programming languages and paradigms this is kind of become absolute. Next what happened is primarily data driven design which kind of dominated the 70s and the 80s where in typical characteristic of data-driven design is there is a mapping between the input system inputs and the system outputs and this mapping can directly derive the structure of the software system.

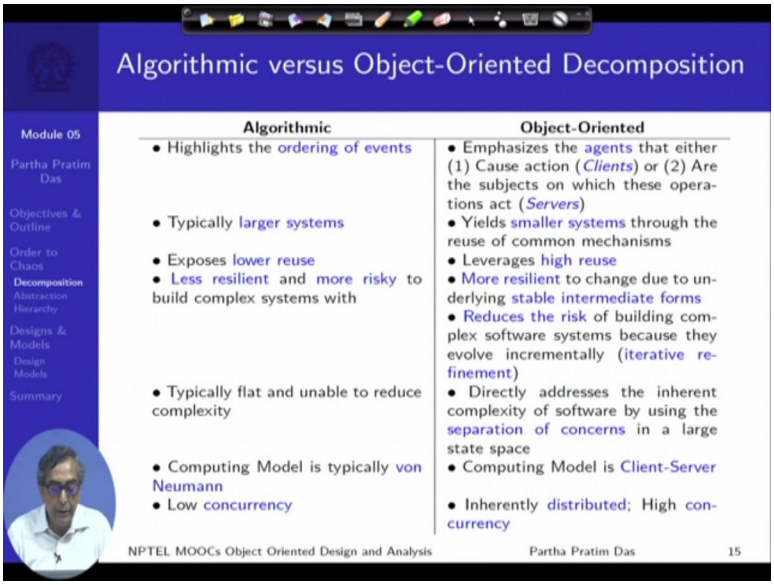
So, data-driven design significantly had a fall out in terms of variety of information management systems those which have been significantly based on variety of database systems and so on but naturally they are not very good in terms of effective in terms of time critical events time critical systems so just to give you glimpses a data-driven design system would still be workable today for say a railway reservation.

Where though you have time bounds there is not much of time criticality and all of that but a data-driven design method would be disastrous if you are designing the breaking system of a car which has very critical time constraints and very complex processes to follow to honor those

time constraints and what is evolved over the last significantly over the last 20 years and emphasized over the last 15 years more our object-oriented design methods.

Where you view the whole world, universe as a collection of cooperating objects or autonomous agents as I mentioned earlier and you treat individual objects and instances of some common behavior classes and naturally that is what is our interest of study in this course and several languages including c plus plus and java support that. So, keeping those in mind we can do a quick comparison between algorithmic and object-oriented decomposition.

(Refer Slide Time: 05:20)



If we look into point wise and algorithmic approach is primarily based on ordering of events because its primarily encoding of algorithm that is the time order whereas object oriented systems are more distributed. They emphasize on the agents that our client and servers naturally algorithmic decomposition leads typically to larger systems, object oriented decomposition gives you smaller systems.

Because you can make better reuse of common mechanisms, better reuse of common properties. Leading from this is algorithmic systems get less resilient or in other terms object-oriented systems are more resilient because since you can follow the stable intermediate forms that we had talked about the stable intermediate forms are transitions of from a simple system to a less simple system to a more complex system to more and more complex system.

So you build the system over iterative refinement and at every stage you have an object-based view of the whole world which is which may not be the exact view that you want is a simplification of that but it's still complete in the in the whole of the staff and therefore can give you a correct system which in which indicates that you have a much better resilient in terms of these stable intermediate forms and can reduce the risk by providing iterative refinement in the design process.

Similarly algorithmic decomposition is flat because you are going task wise whereas object oriented decomposition can clearly take the separation of concerns. I hope you recall what separation of concern is that any subsystem can be viewed as an independent one and with a lot of intrasystem traffic with a lot of intrasytem activity is happening and limited number of inter system activity is happening.

And now you will understand this intersystem activities are basically the messages that are going between different objects and intra system activities are also messages going between the constituent objects of the given subsystem that we are building. So separation of concern basically use a big advantage in terms of building such systems. In terms of computing model we have seen that object-oriented decomposition would use some kind of a client server model.

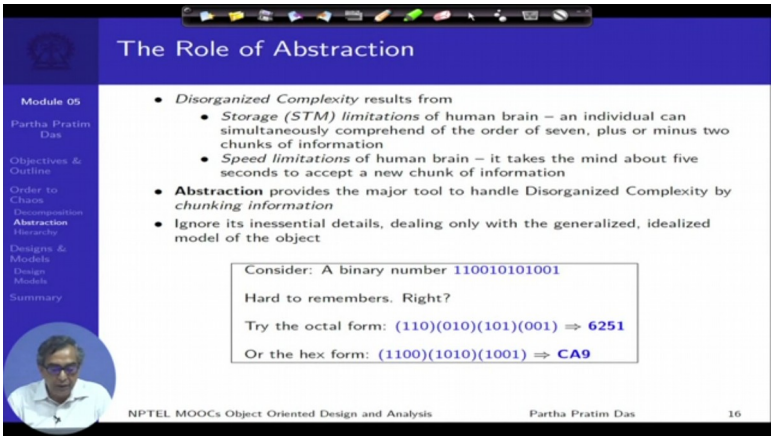
Which is inherently distributed as a high concurrency and therefore in large number of aspects object oriented decomposition has a really good benefit but before we move on just one word of caution please do not think that algorithmic decomposition is useless. After you have come to a certain level of relative primitive in the object-oriented decomposition you will still need algorithmic decomposition to actually implement different logic actually model different logic that your system may have.

There are systems there are complex systems which may be very strictly mathematical formula oriented where even though it is possible to do object-oriented decomposition your algorithmic decomposition may be a much more direct and easily realizable approach. So it is it is gross observation that object oriented decomposition will have a lot of advantages but you should

always remain open in terms of starting with the one decomposition approach and at every stage evaluating, considering as to which approach you should adopt and up to which depth.

In general any design of a complex software system will at different stages have a some bit of hybrid approach, some bit of combination between the algorithmic as well as the object-oriented decomposition approaches.

(Refer Slide Time: 09:23)



Moving on to the other properties of complex systems that we can leverage certainly the next property beyond decomposition is abstraction and we have seen that abstraction is we have seen that there is a huge amount of disorganized complexity in the whole problem domain arising primarily due to limitations of human brain both the limitation of storage, the short-term memory is small, seven typically seven chunks of item even take 12 and the speed is also limited.

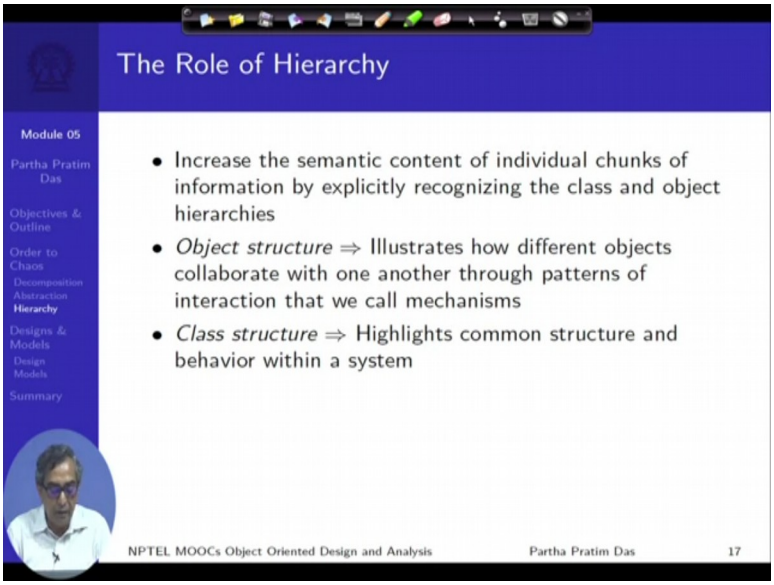
It takes a whole lot of time to process new chunk of information irrespective of what that chunk is. These are all psyhc psychologically proven data that you can you can use and therefore to address the disorganized complexity, only major tools that we have in hand is abstraction which basically what it says that just look into what you need to look into, don't look into the rest. Don't look into the details. Always handle it by chunking, chunking basically means dividing into small pieces.

Small meaningful pieces always handle information by the chunking process so here just a site one I give an example of chunking suppose you are given to remember this binary number and I'm sure many of you are young and have sharp memory but you will still have a little bit of

issue remembering them but suppose I chunk them into three bits together, this is the same, same string I just chunked it into three bits together and then consider these as hexadecimal numbers.

So I got 6251 this is certainly a much easier abstraction to remember or I could actually chunk them into 4 bits together and represent a hexadecimal representation of CA9. So this is is difficult to comprehend this is easy to comprehend. This is easy to remember so abstraction is this is this is one example of chunking and abstraction that is you are trying to reducing the semantic load of the information that that inherently exist and the role of abstraction and in different ways that it will play is to continue to reduce this kind of a semantic load.

(Refer Slide Time: 11:57)

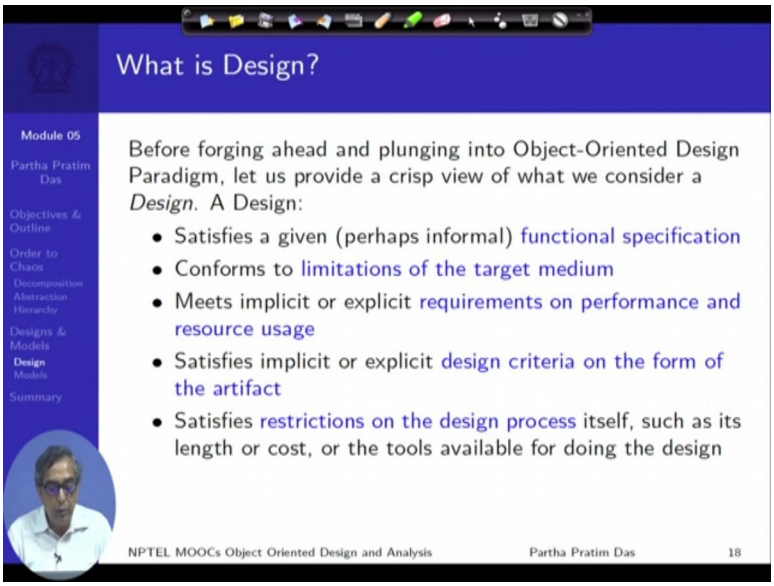


Finally the design process benefit a lot from the role of hierarchy we have already seen different hierarchies in the canonical form and again it can increase the semantic content of individual chunks by explicitly recognizing the hierarchy structures and we have already observed that we have already noted that the 2 major hierarchy structures are part of the canonical form, the object structure which so how different objects collaborate and the class structure which show what is common between different structures.

And behaviors within a system so these three the decomposition, abstraction and hierarchy become the main tools the main weapon so to say to win the war against the complex design of complex systems and from the next module onwards when we start discussing about specific

design concepts, the notions the languages and so on you will see that we will you keep on using a mix of decomposition, abstraction and hierarchy in different ways to actually solve the modeling and design problems.

(Refer Slide Time: 13:17)



Now before I close this module, I would just like to sensitize you about 2 things. One is I have been and will continue to talk about design and so I just wanted to formally define what design will mean in the in the cases that we are dealing with. This is the meaning which is possibly common for almost all engineering designs so for us a design will have these basic properties and or other attributes of concern.

One is it will a design will have a functional specification that is it is for building some system which meet certain functional requirements otherwise we are not doing a design. The second is it will need to confirm to the limitations of the target medium which means that I can take a functional specification and start doing a design assuming infinite amount of computing power or infinite amount of memory or infinite amount of network bandwidth and so on so forth that is not realistic.

What is realistic is on one side I have the functional requirements of what needs to be achieved by building this system. On the other side there is limitations of how what kind of processor, what kind of architecture, what kind of network connectivity, what kind of storage, what kind of

bandwidth and all these are available for realizing that system. So that is the question of target medium. The third it must meet implicit or explicit requirements of performance and resource usage.

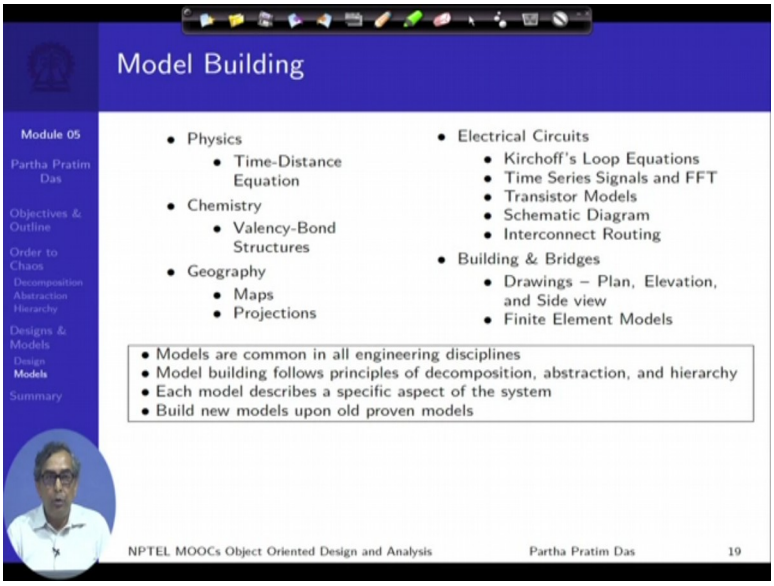
I can say that theoretically any algorithm can be implemented on any system even if it has a couple of bytes of memory but that is not what will give you a working system that is not will give you working design so whenever we talk about designed please try to understand the requirements of performance and resource usage that is performance will be in terms of time, performance will be in terms of may be power and the kind of resource, time, memory, power, you use make it clear that your design need you will need to consider these aspects.

The fourth critical factor for a design is it must implicitly or explicitly satisfy the criteria in the form of artifacts which mean that you whatever functional specification you can think of however sample or have a complex, a system will never meet exactly those functional specification. It will show lot of other fractured behavior in terms of artifacts in the system and gone of the day is when we use to ignore the artifacts and then later on fall flat on our face because we did not consider them at the very beginning at the time of design.

So design must implicitly and explicitly consider the criteria that are acceptable for the artifacts and finally and last but not the least is certainly design must satisfy the restriction on the design process itself. In order to achieve the above 4, I do not have infinite manpower, I do not have infinite money, I do not have infinite computing resources etc and so on. tools and so on to realize that. So it design process itself has to work within the limitation the constraints of the of the tools time, cost, human resource and so on.

So whenever we will talk about design, whenever we will take exercise and design all these five factors, please keep them in mind because missing one or more of them will certainly lead to a design which will not be workable, acceptable in terms of a working system.

(Refer Slide Time: 17:19)



Finally let me close by reminding you that whatever studies you have done so far, you have had a lot of exposure to what is called models. A model is a some miniature of a reality. It is not exactly same as reality, a model is certain aspects of a real situation which is expressed in terms of mathematical form, graphical form may be in textual form and so on. so we have seen a lot of models in different domains so I have just on the left.

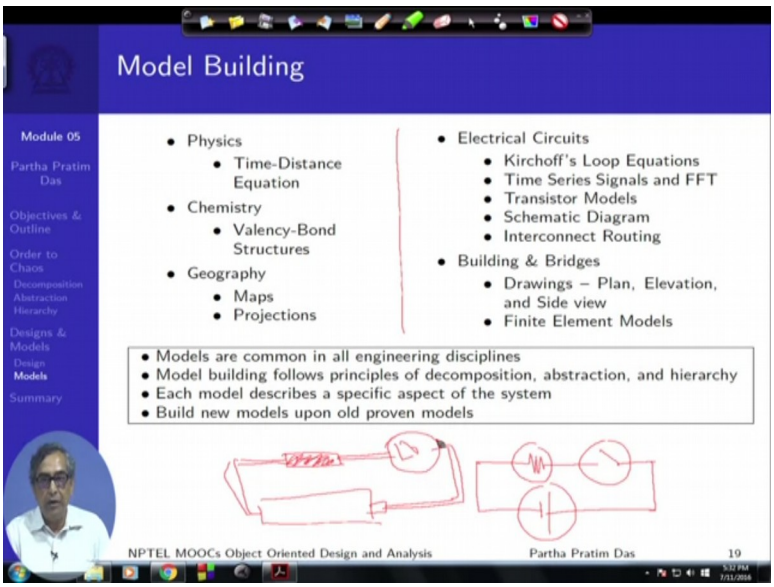
Specifically I have kept some school level generic domains like in physics we have talked about time distance equation that is if I know that a train has started moving at 10 kilometers per second, ab a rather 10 kilometers per per minute and then has been accelerating at a certain rate then our if after a given period of time, how much the train will cover, I do not really need to make the train run and measure and find that out.

I can make a model of the time distance equation we typically say s is equal to UT plus half 80 square or something like this which will tell me given the time the initial velocity the acceleration as to how much distance it has traveled. So that is what is a model which is not but but it will not tell me what kind of noise the train will produce, it will not tell me whether the travel of the train will be risk-free, it will not tell me how much power it will consume in doing this.

Because this model just models a \time and distance aspect and similar models we have seen in variety of domains that we have already studied like in chemistry, we have seen valency bond structure which tell us how the different atoms bind to make molecules and bigger molecules and so on. In geography we have learnt models of maps, projections and so on and if you look into the right-hand side in here these are on the right-hand side are some of the engineering applications.

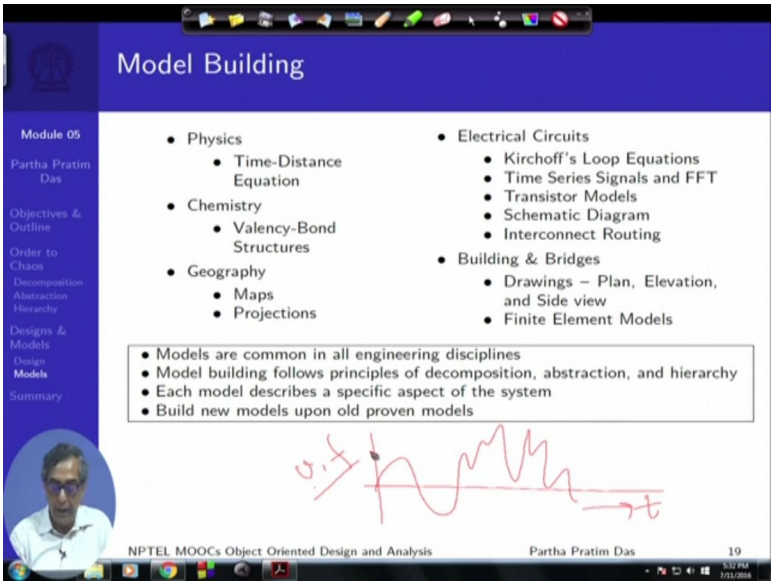
Let's talk about electrical circuits. Now if you want to describe the electrical circuit, you can do it in several different ways. For example you could just do a schematic diagram.

(Refer Slide Time: 20:01)



You will say this is an electrical circuit right this is this we say is an electrical circuit. This is a schematic diagram which there is a schematic of a resistor. This is the schematic of a switch, this is a schematic of a cell and we can use this by for solving different current voltage equations on this in a in a physical reality, if I want to look at this then physical reality this will possibly this resistor will possibly look something like this.

(Refer Slide Time: 21:00)



Where the words will be connected like in here the switch possibly will look like something like this and a cell possibly is is like this. So, the schematic is different from the physical model where you will actually put the cell. If you want to study how the signal will differ based on time, then you will possibly look at the different kinds of time domain signals. So, this is you'll say this is T this is my signal f or this is a voltage v that I have at different times.

All of these are models, whether it's a time series signal, FFT, a transistor model, we can have static transistor model, we can have a dynamic transistor model, we can have interconnect routing, all of these describe electrical circuits but from various different angles. So, the basic idea of a model is that it makes use of the basic principles that we have been talking of so far. They decompose, they abstract and they follow the hierarchy and the strongest element of a model is abstraction.

In terms of talking about electrical circuits all the different examples that I was showing all of them talk of electrical circuit but none of them is complete in itself. If I look at the schematic I do not know really how does the register loop in physical terms. I do not know how does a resistor behave if a 10-kilohertz signal sine wave signal is passed through that resistor and so on.

I have no idea of those whereas when I am looking at the fft of the signal, I have no idea of how the inter connection on the breadboard is. So, every model looks at the limited aspect of the

system, a limited view of the system and that is what makes it manageable by the human mind to work with and that limited view that you focus on primarily gives you the abstraction that you are trying to look at and in terms of arriving at that abstraction we can decompose the models further.

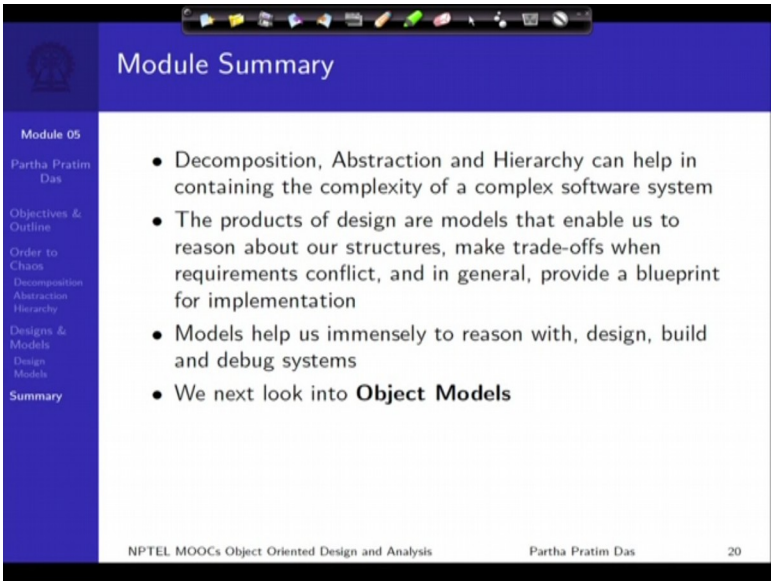
We can put them in hierarchal terms, we have often drawn electrical drawings where you will have different subsystems drawn at different levels of details and as you go deeper into a subsystem, you do more and more detailed models and at the top level you just have this is the input, output of different systems. Similar examples you will find in the building and construction industry construction engineering as well that a different drawing used, plan, elevation side view,

Similarly, on a different model, you will have finite element models to actually checkout if a stress would stand on its feet. so, models are common in all engineering disciplines and this is a big advantage because we all have had a lot of experience with dealing with models of variety of kinds may be only thing that is that was missing is we were not repeatedly told that what we are doing is a common activity, cross domain activity of taking concepts and modeling them,

Solving the problem in the domain of the model and bringing the result back to the real work. That is all that we but we have been doing this in physics we have been doing this in chemistry, certainly whole of mathematics is about that we have been doing this in geography, in electrical engineering, in mechanical engineering everywhere. So, every model will describe a specific aspect of the system and this model based approach has a big advantage that once we have a model which is proven, I can build a system, a new model based on this model.

Because I know this is a proven one so this will not fail. So, the resiliency of the system will increase.

(Refer Slide Time: 24:50)



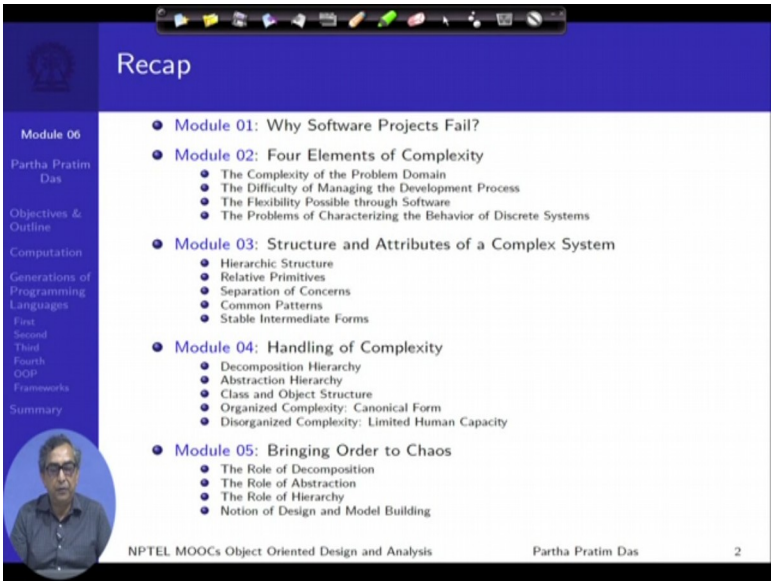
So coupled with the approaches of decomposition, abstraction and hierarchy our next approach in formally going forward in the object-oriented analysis and design would be to take a look into the different object models. Here we have talked about decomposition, abstraction and hierarchy in containing the complexity. We have seen that the product of design or models that will help us reason about given structures make trade-offs particularly when requirements conflict.

And provide become a blueprint for implementation and we will take this forward in terms of actually starting to build models for the complex systems based on the principles we have laid out in the first 5 modules and our next discussion couple of module will focus on how do we make object models and what are the practices to define them and what are the practices to identify and extract them from a real-world system.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture - 09
Evolution of Object Models - Programming Languages and Paradigms

(Refer Slide Time: 00:30)



Welcome to module 6 of object-oriented analysis and design. We have discussed in the last 5 modules, we have discussed various aspects of challenges faced by complex software development that is given a typically complex system, how to analyze the structure of that, what is the commonality between the different structural and behavioral aspects of complex systems particularly even going across domains.

And based on that we have been able to summarize into few common strategies of decomposition of abstraction, of hierarchy extraction and so on and from this module onwards we will slowly start engaging these techniques, these strategies into actually analyzing different real-world situations to develop their software systems, to design their software systems and in this regard, before we actually start doing the analysis and design.

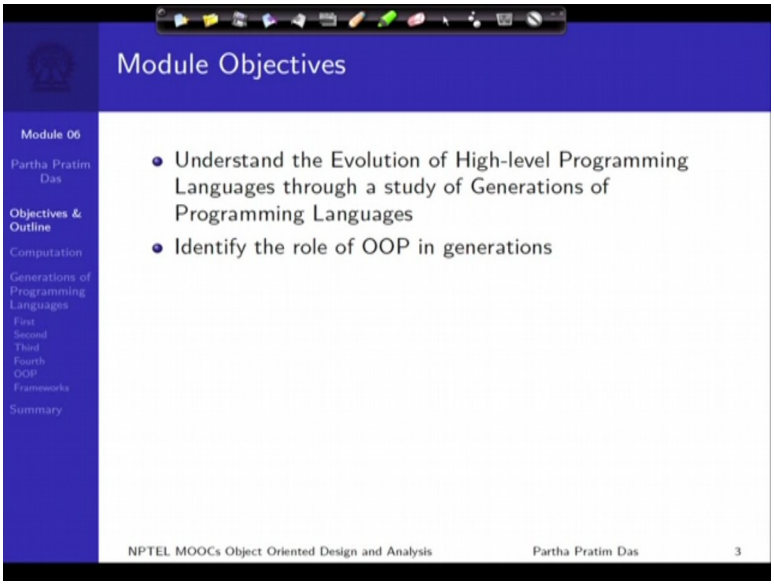
We will take a quick look in the target system on which we can actually implement this software in terms of the programming languages and programming infrastructure that has been available

to us over the period of time when all these different object analysis and design methodologies have evolved. So, once we are able to do an analysis and create a design, certainly a task remains as to how to express that design in terms of a programming a language.

So, the basic definition of a software is kind of an equivalent to in a in a certain way, equivalent to coding my thoughts, coding our thoughts in terms of certain programming languages and therefore on one side, the one part of the problem is to understand the real world, understand the problem domain and realize what needs to be done for providing a solution and the other side is understand the computing system, understand the programming language.

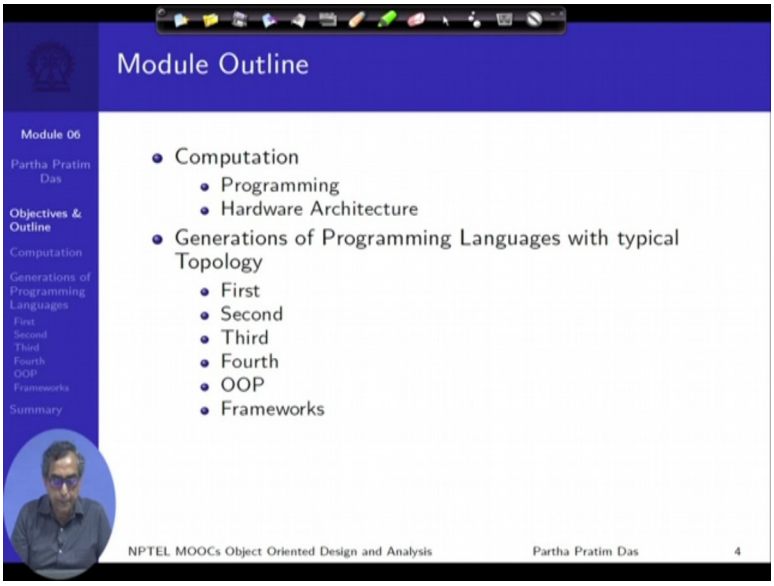
Understand the capability of expression in that language so that we can bridge the gap as software engineers and really implement the designs that we do in terms of the programming systems that we would have at our disposal.

(Refer Slide Time: 03:27)



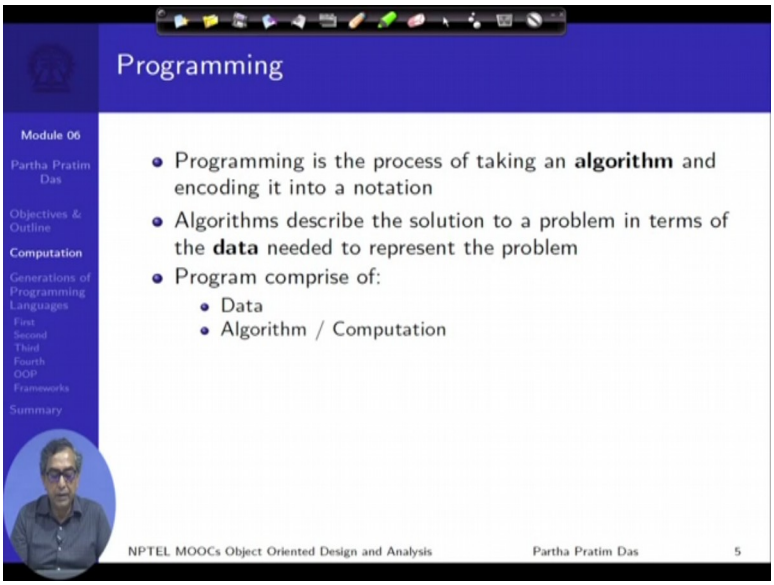
So, to have a good appreciation of what the current situation is we take a quick look into the historical evolution of high level programming languages through the study of generation of programming languages and we will try to see how oop object-oriented programming or object-oriented concepts have evolved over this period of time.

(Refer Slide Time: 03:47)



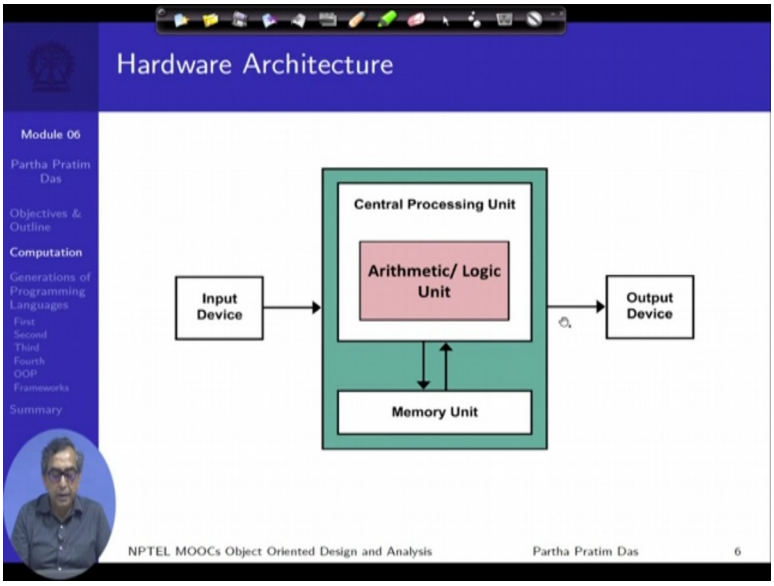
This is a module outline, it will be available on the left of our slides.

(Refer Slide Time: 04:01)



So certainly, the first aspect that we are trying to address is computation which is or commonly known as programming which is a process of taking an algorithm that is a well-defined sequence of steps and encoding it into an notation which we typically say is a program, the notation has to belong to one well defined programming language. The algorithms describe the solution to the problem using or in terms of the data that need to represent the problem.

(Refer Slide Time: 04:35)

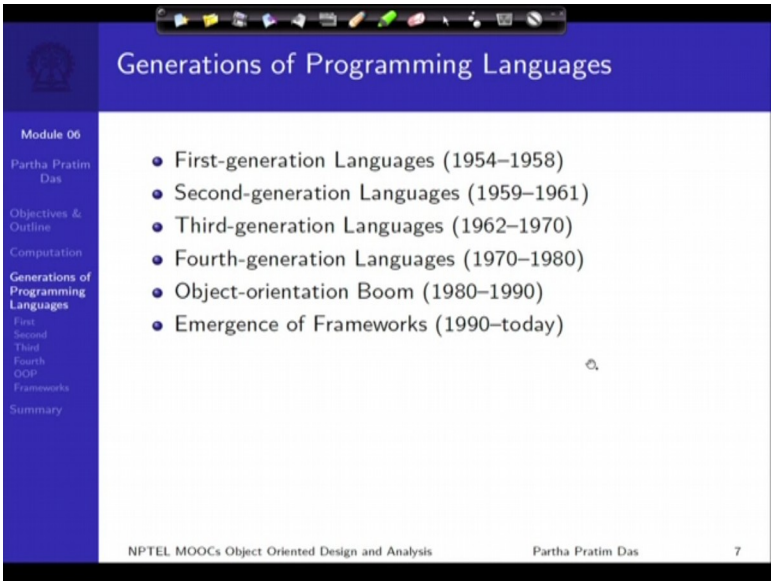


So, we can say that the program typically consists of data and algorithm or computation and though they have been several evolutions of the computing machine, typical hardware architecture or the typical schematic view of the hardware system on which programs execute continue to remain a sample von Neumann architecture where there is a central processing unit comprising an arithmetic and logic unit doing the basic mathematical.

And logical operations and along with that different control units to control the sequence of execution of instructions on CPU memory unit where temporary values, variables can be stored, there could be additionally secondary hardware storage and so on. There will be an input and there is a output device. So, this is the basic computing model and am sure all of you are aware of this model, I just put it up again here.

Just to emphasize that as we evolved into object oriented paradigms our computing model continues to remain the same relatively simple model that have been evolving over the last couple of decades.

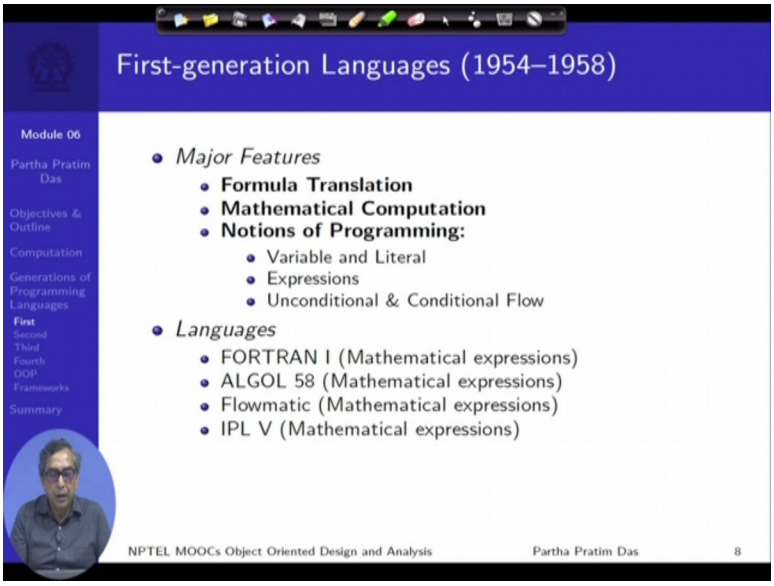
(Refer Slide Time: 05:54)



Now the programming languages all of you am sure know one or more of programming languages and you would be aware that programming languages have evolved and we are talking about high level programming languages not really binary or assembly kind of which is looks more like coded strings. We are not talking about those languages. We are taking about high-level languages.

And there have been several generations programming in high-level language started somewhere in the early 50s and the mark generation started from 1954 and kind of different generations have passed and we will just quickly take a look as one set of languages evolve from one generation to the next what has significantly happened in terms of the language and the computation model.

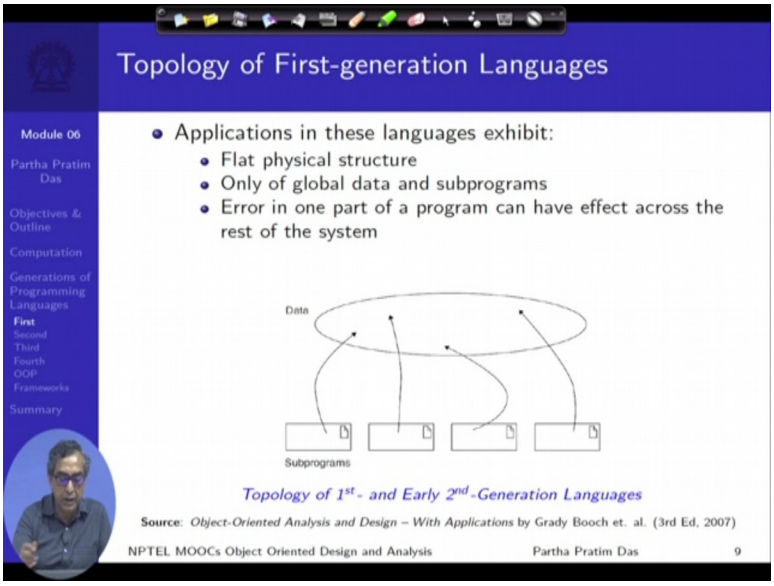
(Refer Slide Time: 06:47)



So, we start with the first generation which certainly is pioneered by Fortran the language known for formula translation. so naturally the major feature of first generation programming languages was to solve arithmetic problems, there you can call them mathematical computation but significantly they are arithmetic problems. So, they translate certain formula so if I know the value of radius of a circle.

Then I can write a program to find the circumference of that circle or the area of that circle using certain formula in Fortran kind of a language and certainly this is the beginning of the high-level language so the basic notions of programming that got introduced during this time include literals that is constants like 5, 10, 2.3 and so on. variables expressions with variables with some basic operators and a conditional and unconditional flow of control though in languages like Fortran or Ipl v these control flow were very archaic rudimentary and was kind of very cryptic in nature.

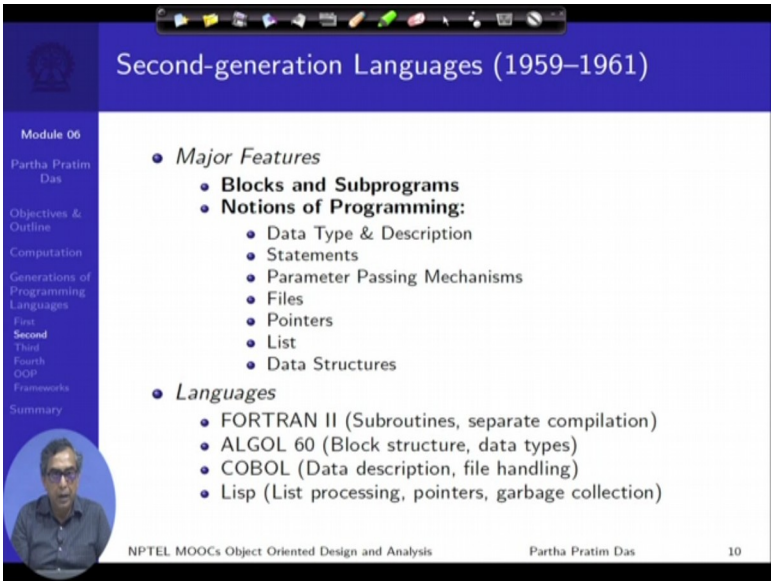
(Refer Slide Time: 08:02)



So, if we look into the topology and if in every generation we would try to look at the topology to see how the computation actually behaves between the data, this is the data and the program or the algorithm, we said that computation is basically an interaction between these 2. So, in the first generation we had in terms of the data, we had a typically flat physical structure, data used to be in a flat structure, possibly one single data only global.

And then you have number of subprograms, these subprograms are what we call these days as functions or subroutines. So, some subprogram would be adding numbers, some subprograms will be printing a value and so on. and any error in one part of the program certainly can affect across rest of the system. so, this was very rudimentary and risky and it continued from the first generation to the early second-generation programming languages.

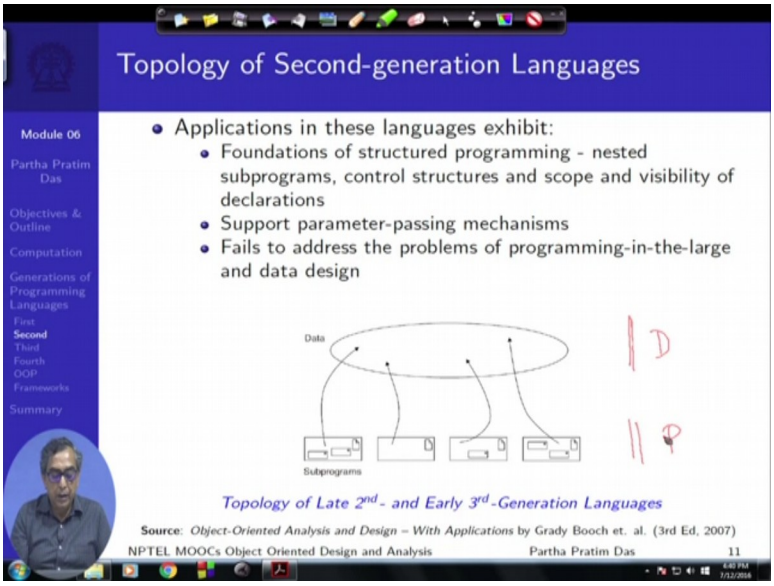
(Refer Slide Time: 09:10)



The second generation which started around the end of 1950s and was relatively a short one saw the emergence of evolution of the Fortran language we had Fortran 2 which is kind of what some of us have had the opportunity to use when we were students like you all, this was the first version of Fortran which had a number of additions to the notions of programming. It had certain well-defined data types, it had a number of different statements, parameter passing mechanisms, files a some not Fortran

But languages like ALGOL started introducing pointers that is address variables so you could create some kind of a list and specific languages like lisp was created to manipulate least kind of structures, data structures started happening. So, these are the though it's a kind of typically marked as a short generation in terms of the lifespan of the programming languages but it was a very significant generation which introduced lot of the fundamental programming concepts in the whole computing module.

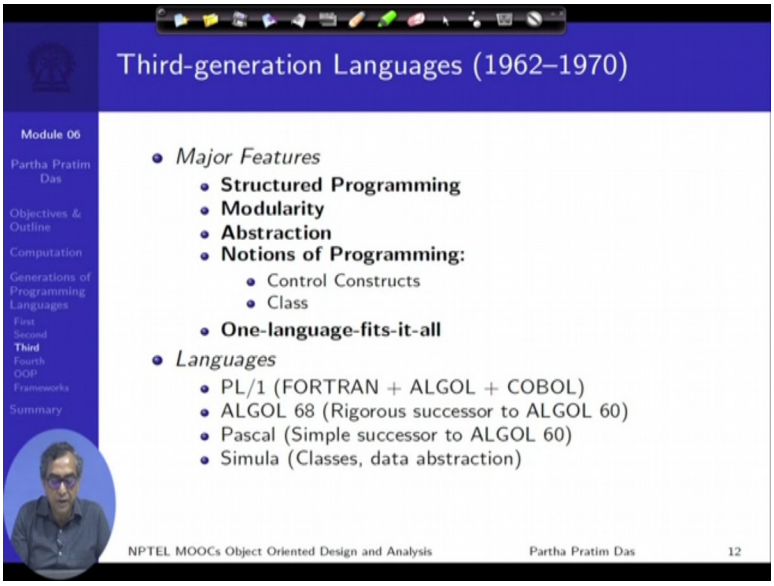
(Refer Slide Time: 10:28)



Looking at the topology again the same we have the data here, we have the program here, but now the basic foundation of structured programming is starting to come in. so earlier if you if you look into the earlier the subprograms were just like this, subprograms are just one single blocks in one single code. Now you have certain structured constructs like loop constructs, condition constructs which are started to get introduced.

And subprograms are divided in terms of those so those kind of you know structured constructs are being depicted here but they continue to work moderately on a flat pool of data supports better parameter passing mechanism but because as the whole collection of data is kind of flat, the second generation of languages fail to address the problem of programming in the large data in case of large data, things naturally became very unmanageable, unwieldy to work with.

(Refer Slide Time: 11:43)

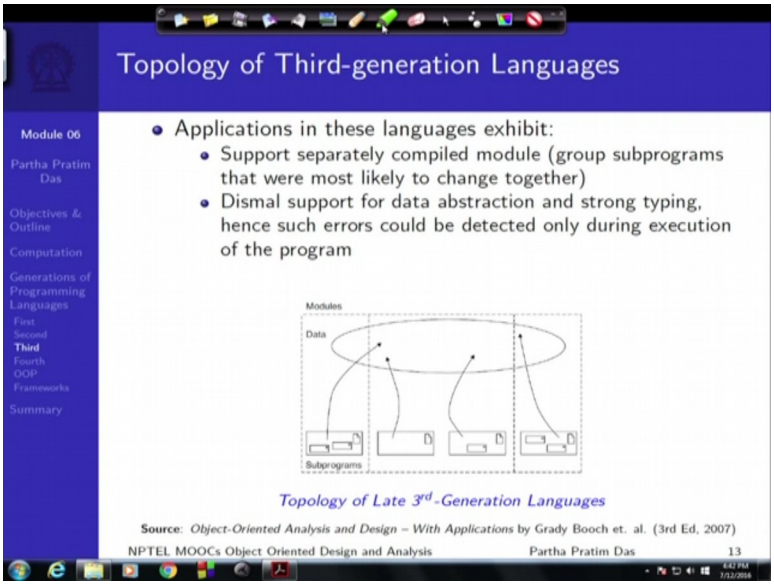


Moving on the third generation spanned relatively longer almost the whole of 60s where several new kinds of path-breaking languages starting started to happen like ALGOL had a next version which is kind of considered one of the precursor to the recent programming style with a with its strong structured programming introduction with its modularity, Pascal the first very simple

Yet robust language for programming was introduced, it was strongly typed and for several years we studied and taught Pascal to our students kind of as we do for c or python these days and this is what saw the emergence also of languages like pl/1 pl stands for programming language 1. So, kind of since as the programming started maturing more and more constructs were getting identified, class got introduced, abstraction got introduced, happened in ALGOL.

So, an idea that started doing the rounds is we should have one programming language which solves all kinds of problems. So, one language fits it all kind of an idea and pl1 is one of the biggest examples of that happening which turned out to be a huge language and could you could almost do anything and everything with that language.

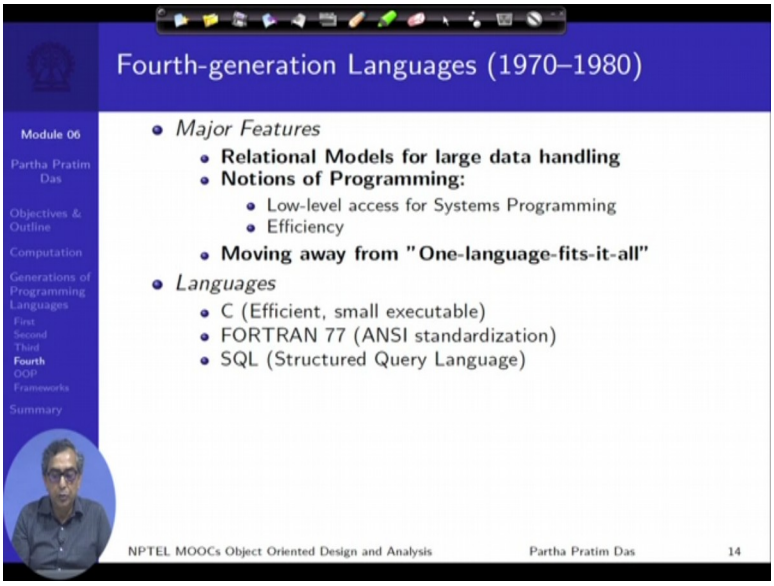
(Refer Slide Time: 13:16)



So that's with that state of affairs the certainly the topology saw some significant changes what is the data is here, the programs, subprograms are here but what you see significantly are demarcations between the subsets of subprograms which was achieved because the first time many of the languages started loving multiple source files which could be separately independently compiled, groups of programs were introduced.

But yet the support for data abstraction or strong typing Pascal as I said had a little bit of strong typing but in general strong typing and abstraction were not widely available in programming languages and therefore handling errors when data starts becoming larger and larger still remained to be a nightmare with the programmers.

(Refer Slide Time: 14:18)



The fourth generation the 70s typically saw the emergence of one of the most popular, 2 of the most popular languages so to say of today. It saw the emergence of c which was created somewhat accidentally by a team of researchers, we were trying to write an operating system from scratch, trying to write Unix and to write Unix certainly it was not fun to write it in assembly so they wanted to create a high-level language for that.

High-level languages could not be used for writing assembly, writing operating systems or writing systems, programs because high-level languages could do only high-level stuff like taking the radius and finding the area and circumference of a circle but they were not equipped to actually go to bits and registers and ports of the hardware and make changes as an operating system or as a systems programs would need to do.

So, coupled with these ideas c was born amongst the people who were developing Unix and pioneers we all know the names of gar Nening Hannon and Richie and there are several others in the team as well and it gave birth to a language which was on one side satisfying the basic requirements of the high-level language at the same time, it was providing the low-level access that is required for systems programming.

The c was also characterized has always been characterized for the efficiency of the programs that that can be written in c. the other significant development during this time is the evolution of

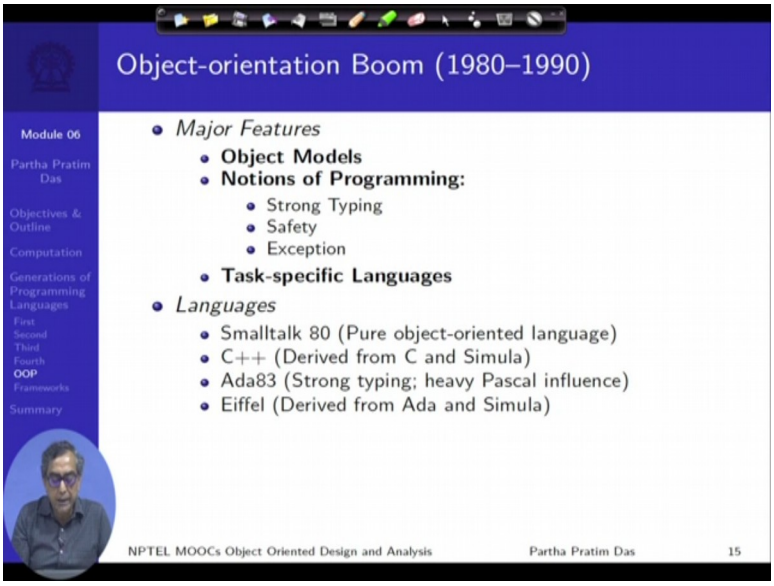
systems specifically dedicated for handling large volumes of data may be not doing complex operations but being able to do bulk operations of certain style and these came to be known as relational database systems or relational models for large data handling

And that gave rise to a popular language known as structured query language or sql which is still predominantly used today in several relational databases to query different large volume data sequences but c and sql remained to continue to remain in independent domains where on one you in c you are looking at more complex logic even systems programming but relatively less volume of data, in sql you are looking at managing large volumes of data but less complex logic.

So, there were certain bridge languages also which were generated in this in this generation which were created in this generation like embedded sql where you could write sql inside a c program or kind of you could call c programs routines from sql and so on and it was dominated by a very strong growth which on one side gave a strong impetus to impetus to generate a I mean designing languages which could solve more complex problems.

And other side could handle bigger volumes of data and as you can see people started realizing that one language fits it all is not going to solve all the problem. So, we see the emergence of different languages for different purpose, c for systems programming, sql for database programming and so on. Fortran 77 also gained a lot of momentum; it is a revised version or Fortran which borrowed lot of other good features from different programming languages.

(Refer Slide Time: 18:19)



The next kind of the next decade, the 80s started seeing the emergence of the boom of object orientation as you would recall from the earlier generations, the basic concept of modularity, abstraction, decomposition, these are slowly getting introduced in languages like ALGOL certain part in Pascal, in pl/1, c had some of those but substantially object-oriented or strongly backed by object-based thoughts, the languages started happening in the 80s.

And saw the emergence and establishment of c plus plus as a dominant language which at present is called a multi-paradigm language on one side, it has strong procedural flavors and you can write very strong algorithms which extremely good efficiency in C++ at the same time, you could do a very strong object based modeling in C++ as well. So, the major feature of this decade has been the boom of object models, strong typing came in in a strong way.

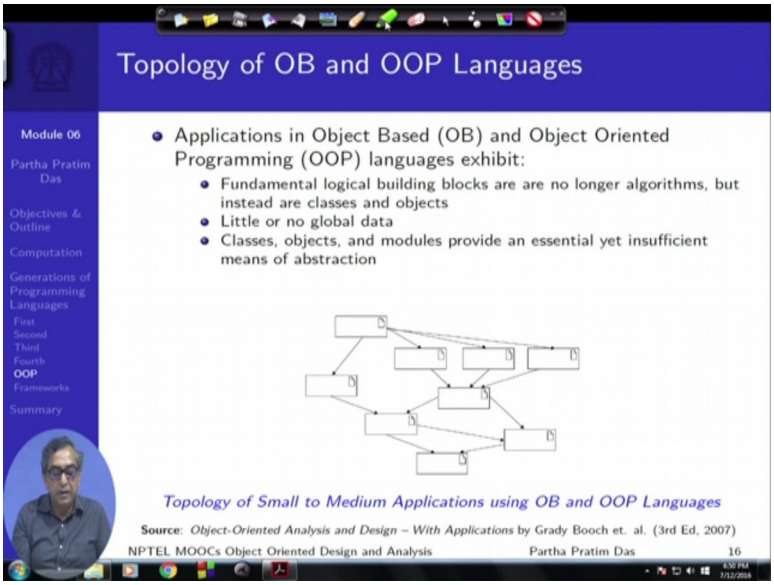
And several new concepts started coming into programming people started talking about safe programs that is programs which can which you can rely on because during this period, several embedded systems started using software as components and for example Boeing started using software components for controlling the aircraft engine. So, the correctness of the software the guarantee that the software will not fail.

The guarantee that the software is safe became critical for human survival, software was heavily put into sending rockets to space and so on. So, safety concepts of safety, concepts of exception

that is taking care of conditions or paths or you know situations in the software which has not been thoroughly thought of became came strongly in terms of different languages, several experimental languages happened.

Beautiful ones like small talk which is very pure object-oriented language unfortunately it has not been very popular in commercial terms ada has had a contribution in terms of strong typing, Eiffel a beautiful language coming from ada on Simula which gives a really powerful modeling in object based terms so these are different possible languages but none of them have besides C++ have made really a mark in terms of large volume of programming.

(Refer Slide Time: 21:09)



Now if you quickly look into the again the emergence of the topology in terms of this object based and object-oriented programming languages, the distinction between object based and object-oriented programming languages are not very high one typical characterization that is done is a language is called object based if it does not if it supports the basic concepts of object orientation like abstraction encapsulation, modularity.

And so, on but if it fails to support inheritance specialization then its typically called object based. You will understand this is difference more as we go forward. Other languages are called object oriented programming languages. So, the significant difference that happens in this is the fundamental logic building blocks are no longer algorithms but they are classes and objects. So,

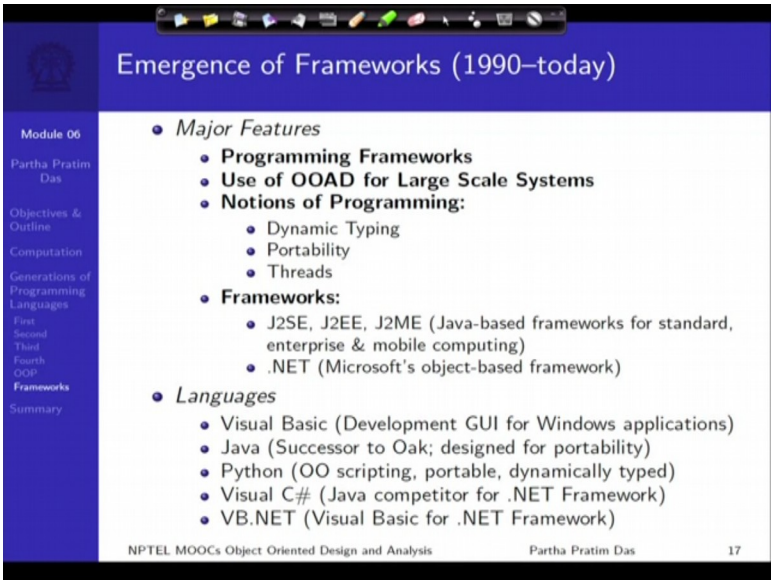
you have had quite a bit of discussions, we have had quite a bit of discussion in terms of classes, structure, object structure.

They become the fundamental concepts in the programming language itself. Instead of the procedural logic that decides given certain inputs what should be the output. There is very little global data, the data is more and more getting encapsulated in terms of objects, they are getting localized, getting inside the object and classes objects, module provide and essentially yet often insufficient I mean it its is not always that classes objects.

And modules can solve all issues of abstraction but a significant amount of higher abstraction got created in terms of these languages. So, you can see that the topological model has changed now. Now you do not have the data separately that has completely disappeared everything is now in terms of objects which has inside it the encapsulated data and it exchanges as we have discussed earlier also. It exchanges which messages between them to decide what computation needs to be done.

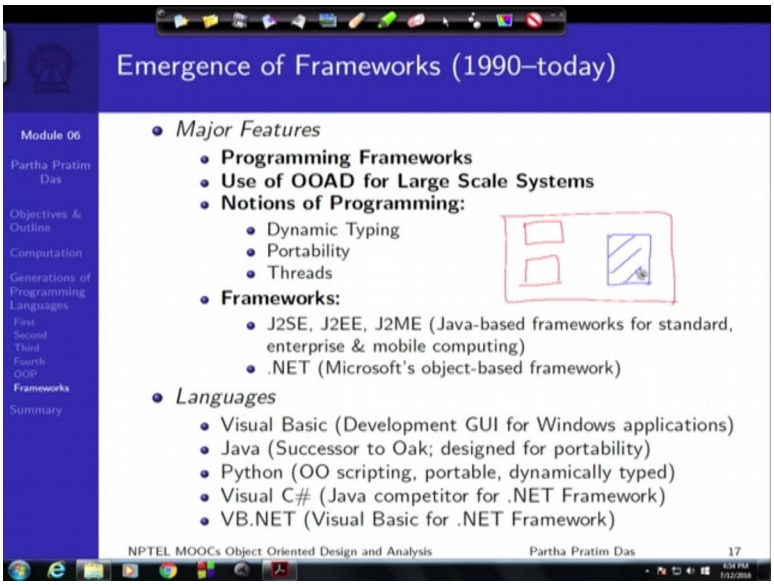
So, client object sends messages to server objects to other objects who will provide that service who has that kind of an interface and the server object will perform that operation and send back a response is kind of the typical topology that turned out for this generation of languages.

(Refer Slide Time: 23:36)



Beyond 1990s slowly the situation started becoming different the trend of designing new programming languages and empowering them with more of object oriented flavor continued but a major trend was being observed to develop programming frameworks that is earlier without a framework you have a programming language, you write a program and then you can compile it with a compiler, you get a user linker to link different sub modules of your program

and you get a get a single executable which you work which will work on the system. with frameworks, the things become somewhat different with framework what it turns out to be there is a framework so there is a certain framework which is given which means that there are some software components which are given and within them you are writing your this is your program. (Refer Slide Time: 24:40)



This is your program but the red boxes are components which are given you not only reuse them but actually they may exist even when you are executing the program all through the process of your lifecycle of variables objects and the program, this framework start helping moving in multiple different ways. So, framework based programming started to become more and more popular even that is what made the use of object-oriented analysis.

And design for large-scale systems more and more possible. so, these are the major features in terms of notions of programming, a whole lot of dynamic typing notions got emphasized that is a dynamic typing is a concept where you do not specify a type for a particular variable, you do not

say in text to mean that x will always be integer but depending on what kind of value you keep on a kept keep in x whether you keep 17 then you will consider x to be integer.

You may keep 13.5 we will consider x to be a floating-point number. You can consider a string path then you will take x to be a string variable and so on. so, some concepts of dynamic typing became popular. Emergence of internet happen so portability became a major question. Portability became a major issue because it became important that you deliver software across different systems.

And certainly, what became very very important is instead of relying on large computing systems which could just work with one very powerful processor solving millions and trillions of computations per second. Instead of that it became a typical to distribute your computation in terms of different threads which started as a trend in parallel programming in larger systems and then percolated as smaller threads.

And smaller concurrent models in terms of even all the small devices that we use today, your cell phone being a big example. So typical frameworks that have emerged include the frameworks on which java that you know the most popular application programming language of today is based. So, when you program in java you typically make use of certain programming software environment like you could use j2SE, j2EE, j2ME.

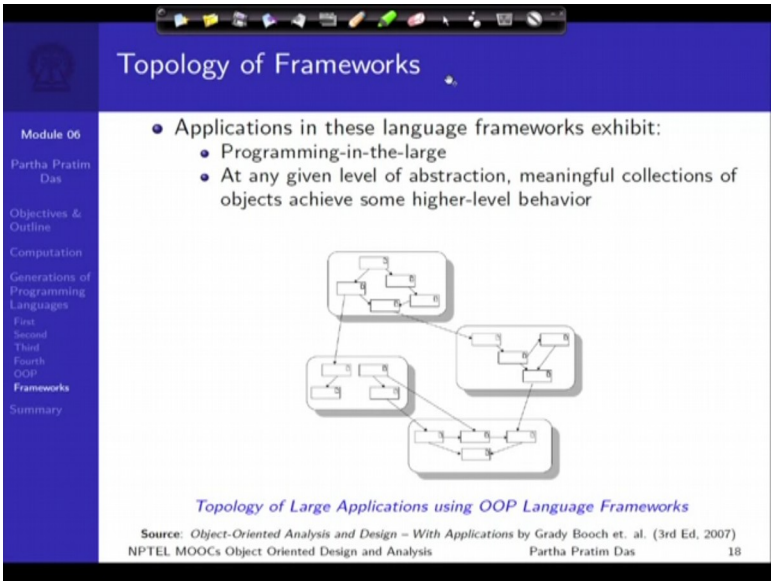
So, these are all different standard and standard environment or the enterprise environment or the mobile environment in which you do certain programming in java and these frameworks help you particularly by providing the support that you need for developing your application. For example, you want to develop and application for an android device, so using j2me it could become very easy.

You do not even physically need a device, the j2me could provide you with a model of the with a simulation model of an android phone and you could write all the development on your desktop computer itself by using that model. So, frameworks started becoming really significant of dot net framework from Microsoft has even very dominantly used in variety of application

development and therefore you will find that several of the languages that happen during this time have significantly been linked to different framework.

So when we talk about visual basic we are certainly talking about GUI programming in on Microsoft or windows applications and when we talk about vb.net we are certainly talking about programming with visual basic under dot net framework, java frameworks we have talked of, even independent languages like python which is one of the excellent object-oriented programming languages of today which is highly portable very easy to learn and its supports dynamic type, in interpreter based also has its own framework to work with.

(Refer Slide Time: 29:00)

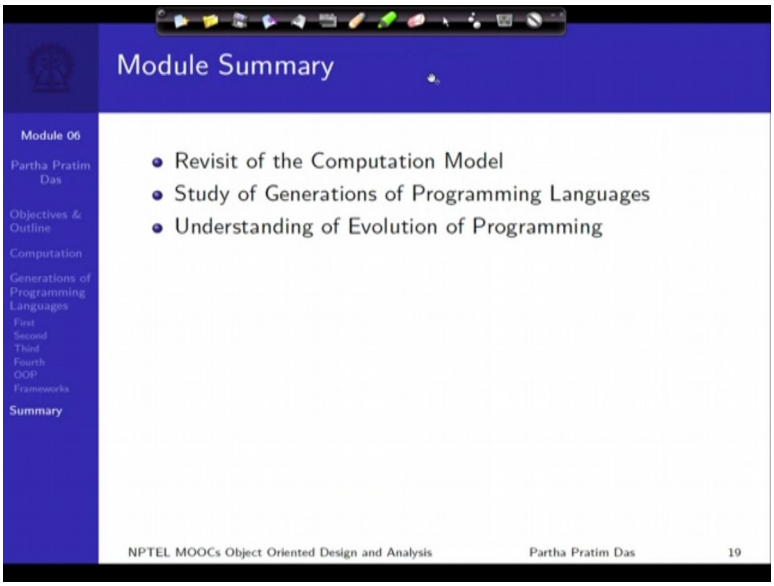


So, this is the these are the different kind of emergence that has happened and the topology for this at least started becoming more and more fragmented and distributed. So, you now have earlier in in terms of the object-oriented systems you had this kind of structure where objects were exchanging messages but they were all together now in terms of the framework they have got grouped in different subsystems.

So, there is a subsystem say if you are programming for android phone as a subsystem which is supporting so your address book, there is another subsystem which is supporting say your messages, there is another subsystem which is supporting your calls and so on. and using those abstraction you can do write a program. So, it is it is migrating beyond just the requirements of

programming languages, it's kind of becomes a framework based programming system on which you typically develop your programs.

(Refer Slide Time: 30:00)



So, to summarize in this module we have revisited the basic computation model we continue to use a different variants of the one moment model though we use it with certain client-server flavor in the whole thing. And we have taken a quick look at the different generations of programming languages and the current generation being the framework based programming.

And we have tried to understand the evolution of programming so that now when once we understand how to do the analysis and design of a system using object oriented methods, we can choose our right programming language or right framework in which we can go ahead and implement them.

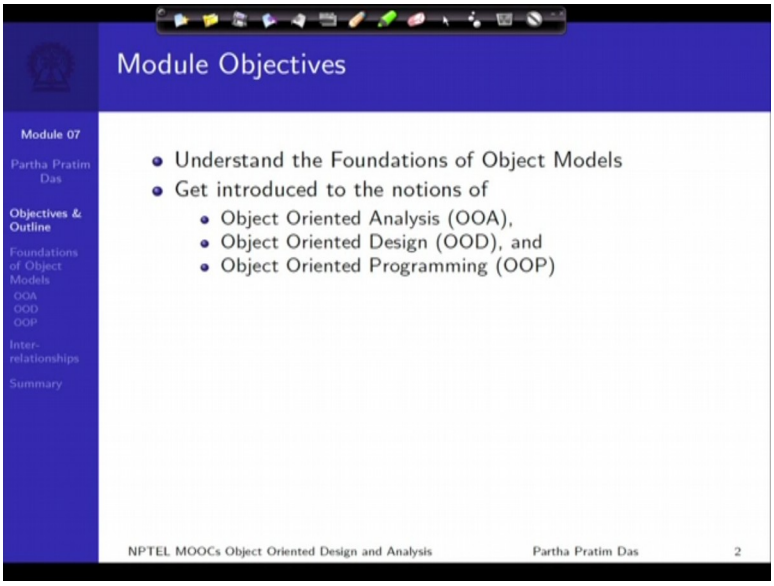
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture - 10
Foundations of the Object Model - OOA, OOD and OOP

Welcome to module 7 of object-oriented analysis and design. In the last module, we took a quick look into the evolution of programming languages, programming paradigms and in the earlier modules we have taken a look into the challenges of software development of modeling complex systems and we analyze to understand a few common properties of complex systems.

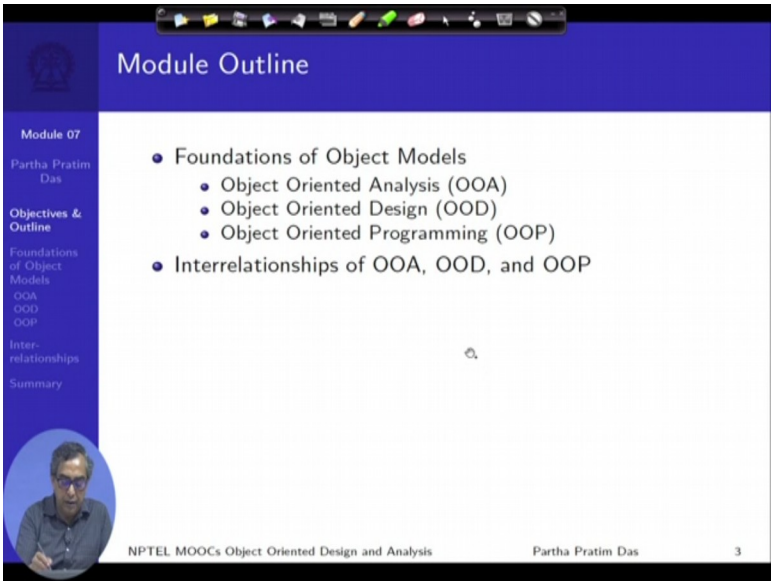
From this module, onwards we would try to use both these backgrounds to slowly start defining our object models which is the fundamental concept of starting to analyze and design something in an object-oriented manner. So, in this context the current module will provide the foundations that talk about the foundations of the object model and are my main intention would be to introduce the free basic terms that we will frequently use.

(Refer Slide Time: 01:20)



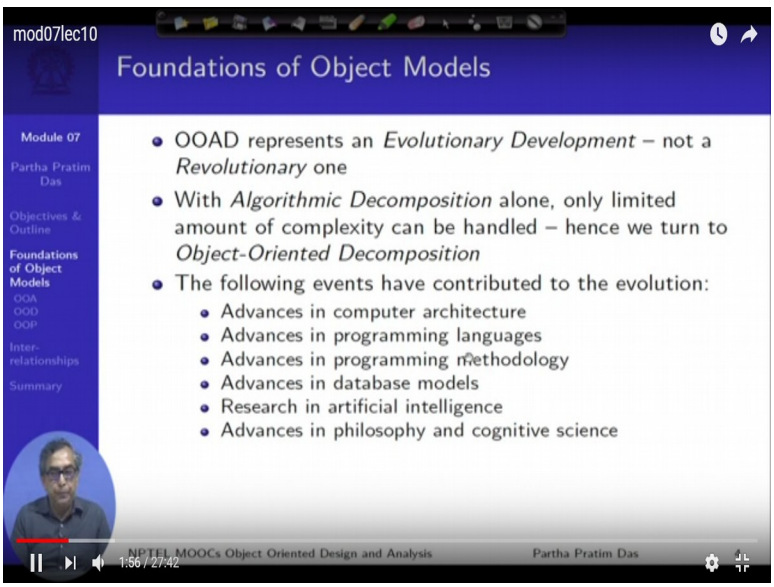
The notions of what is object oriented analysis, what is object oriented design often they are so interrelated that they are referred together as object oriented analysis and design as is the title of this course and we will get introduced to what is object oriented programming.

(Refer Slide Time: 01:53)



The outline is available on the left.

(Refer Slide Time: 01:55)



So, the object models, the foundation of object models certainly is in object oriented in analysis and design. Please note that this whole design, analysis and design paradigm methodology has been a or represents an evolutionary development and not a revolutionary one. What it means is this is not a single path baking part or a path breaking approach by which we will quickly create a completely new solution.

This is not coming and showing you that any set of numbers can be sorted in login time without any extra memory. So, we will solve that problem with a revolutionary solution but object-

oriented analysis and design is typically an evolutionary development which means that in the process itself the approach is to start small, start sample, start elementary and then evolve the system over a series of transitions into the final target complex system.

We have to keep that in mind and that is for the several different reasons that I have highlighted in the earlier modules, going in an iterative evolutionary refinement form to go from simple to little bit more complex to little more complex to further complex and so on and the final system is the only way to keep things under our control, to keep it within the human capacity of handling and manage the complexity to deliver a good working software.

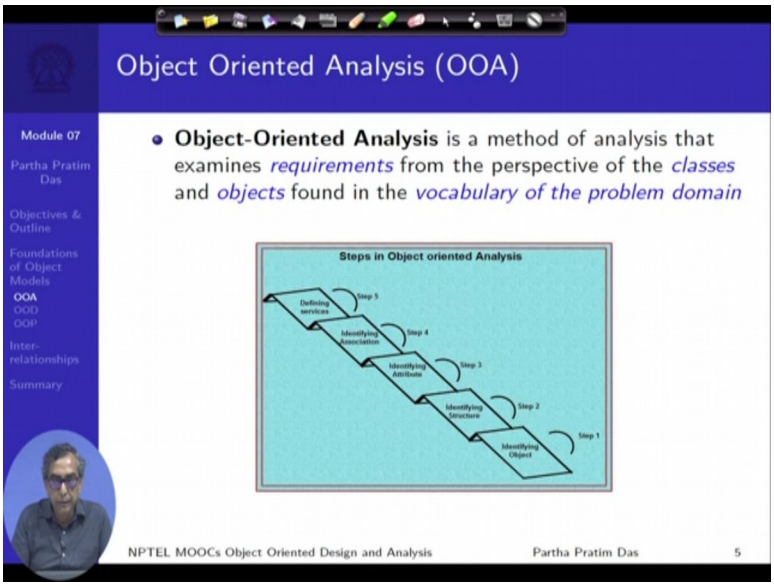
So obviously with algorithmic decomposition alone, only limited amount of complexity can be handled we have discussed this at the end. So, we turn to object oriented decomposition. If you historically look at, we just took one this course in terms of the evolution of programming languages in parallel similar dispose, you could study but the start of time want of time I could not cover all of them in the historical perspective but if you track in a similar way the object-oriented analysis.

And design is the direct fall out of advances in computer architecture which has made it possible that different object based on object oriented features can be brought into the computing system, advances in programming languages which we have already discussed. Advances in programming methodologies like framework based programming which also we have discussed, advances in database models which has given us really new powerful ways to model databases.

And this is growing; this is still getting much bigger than where our current course will stop. We have now databases which are called say no, no sql database, unstructured data database. So that is a regular evolutionary process that is going on even research in artificial intelligence which is shown as that how much the reasoning can be structured, how much the reasoning can be automated given advances in philosophy.

And cognitive sciences, understanding of our own thinking process, all contributed towards the emergence of object oriented paradigms.

(Refer Slide Time: 05:45)



So, in that context we start discussing about the three basic approaches that we need to do. One is the object-oriented analysis. First thing you do is object oriented analysis, the method of analysis that examines requirements, so please this is a very simple statement but please understand it very carefully. It examines requirements from the perspectives of the classes and objects found in the vocabulary of the problem domain.

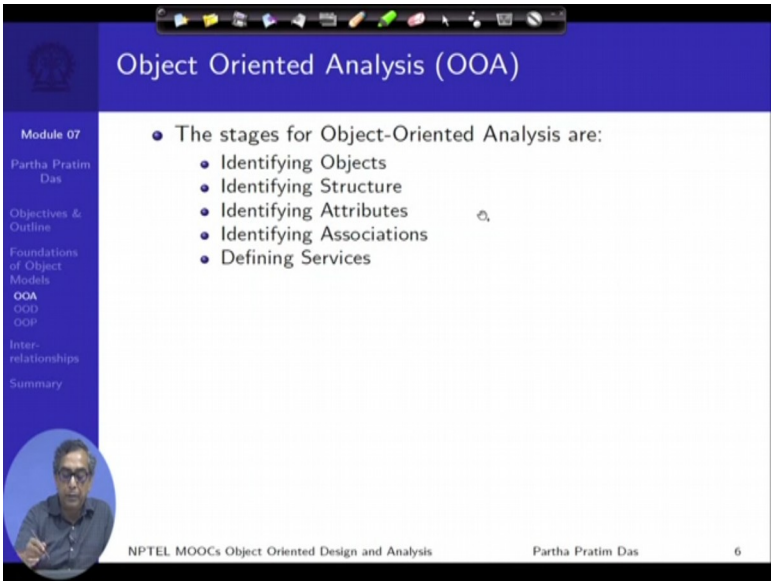
So first we are doing a process of, we are doing an activity of engineering. So, what is the first question you need to ask? You certainly need to ask is to what is my input and what is my output. Am doing an activity, so what I am given and what I had to deliver. So, if I look into object oriented analysis or the analysis phase then what we are given is the requirement. That is somebody my customer needs the system to be built.

And the customer has expressed that need in terms of possibly natural language descriptions to me. Please consider am sure you have had an opportunity to take a look into the tutorial that we created for the first week. The tutorial on leave management system of an organization where you saw two of our volunteers Srijoni and Tanwi behaving as customers and vendor to discuss about what leave management system is required.

That requirement so kind of you can say that the requirement is to specify in terms of this video. So, this itself could be your input. Of course, we will not get you started with such a vague abstract place. We have also provided you a 2 page, 2, 3-page kind of document which gives you some more statements about what the requirements should be. And but that whole thing whether the video or the diagrams or the text are all kind of different vaguely specified, incompletely specified possibly inconsistent.

Requirements that the customer has not the customer herself may not be clear as to whether 100 percent of the requirements are really needed, whether they are really can be done and so and so. So, object oriented analysis is a process is a practice that you will take up in view of this input to actually create a very basic level of identification for classes and objects.

(Refer Slide Time: 08:45)

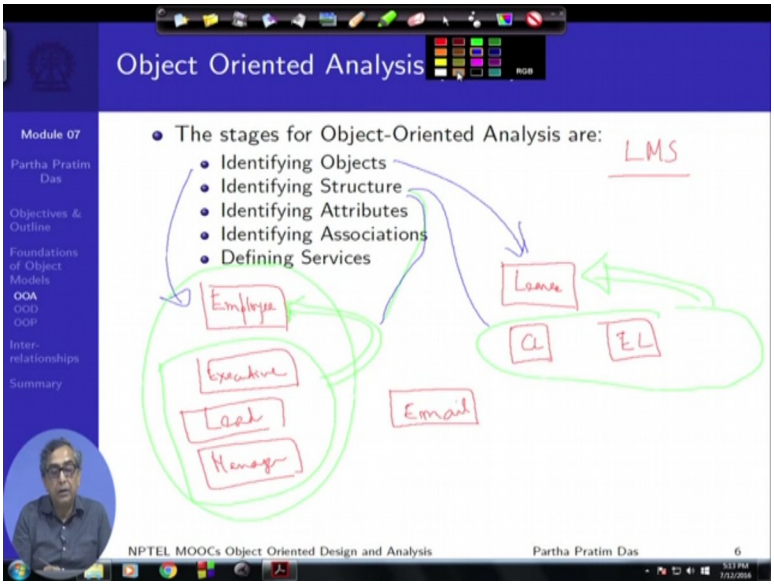


And while you do that, you will try to use the vocabulary of the domain that is the vocabulary of the problem domain is whatever is said in terms of the problem that you are given to solve, whatever you said in terms of the requirements documents, the requirement video, diagrams and so on. so, the typical stages could be this that the first thing you would like to do is to identify your objects.

So, your system leave management system we have heard so what we were told. We were told that the organization has employees and they need a system to manage the leave of their employees. We were told that the organization, the employees in the organization are not of the same kind. There are 3 kinds of employees, executives, leads and managers and we were told that the leave is also not of one kind, the variety of leaves, causal leaves, earn leave.

And so, on and the if you if you go through the 2 pages of the document, you will find several different rules of who can how can who and one apply for a leave and how can a leave be granted, how much leave you are eligible for, what are the conditions for different leaves and so on are documented. Now the first task that we know that we are looking at an object-oriented decomposition of this problem. So, the first things we need to learn are what are the objects.

(Refer Slide Time: 10:15)



So, some of the good cases that we can make is certainly I can say that given this employee is an object. Given this leave could be an object. Like these are good candidates so that when you do identification you saw the employee is an object. You say leave is an object. But what about executive? What about lead? What about manager? These are all possible candidates for objects. Similarly, what about a causal leave, how about an earn leave and so on.

So, this identification process can lead you to identification of objects could lead you to several sides, different possible objects. There could be significantly different object also for example if

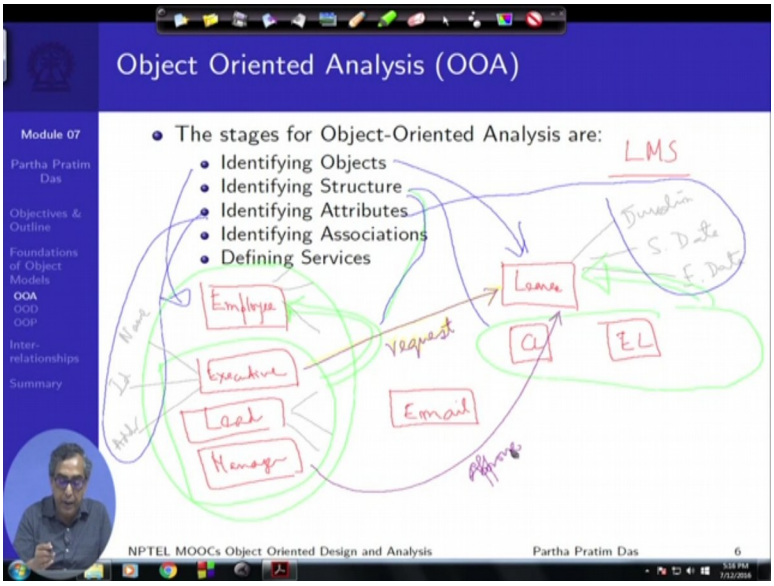
you read in the specification that you learned some places saying that if a leave is approved then an email is sent to whoever has applied. So, email also is a concept, you can say email is an object. So, there could be several objects identified.

And as you identify different objects, keep them try to see if you can have certain structure between them that is of all these different objects that you have here are they really distinct objects or they are kind of interrelated and we can say that one commonality that we can see that these 3 objects are all of have something in common with the object employee. Again at least say an executive is an employee.

So, anything that applies to an employee in general employee has an employee code, get his salary, has a designation, reports to someone all of these will apply to executive to leave manager and so on. Similarly leave different kinds of leave, there could be several others, or leave anyway that is when you get paid possibly and but you do not come to office or you do not work. So, the next to identifying the objects is picking up a wrong color is identifying structures.

You try to see what structures these objects may have. Subsequent to that certainly you will need to look at what are the how are a concept like an employee or a concept like a leave it is an abstract concept, it is an abstraction right. It has a notion but finally the computing system, your final target is computing system so you will not be able to just keep a notion, you will have to get to that notion through a set of properties that the object can have that that concept can have, that abstraction can have.

(Refer Slide Time: 13:42)



So, will start saying that well this executive has a name, an executive has an id, an executive has an address same for the lead, may be same for the employee, leave has a duration, leave has a start date and so on so forth. It must have an end date, etc. so you start to really understand the objects, you start identifying attributes so these are the different attributes. These are the different attributes that you start identifying.

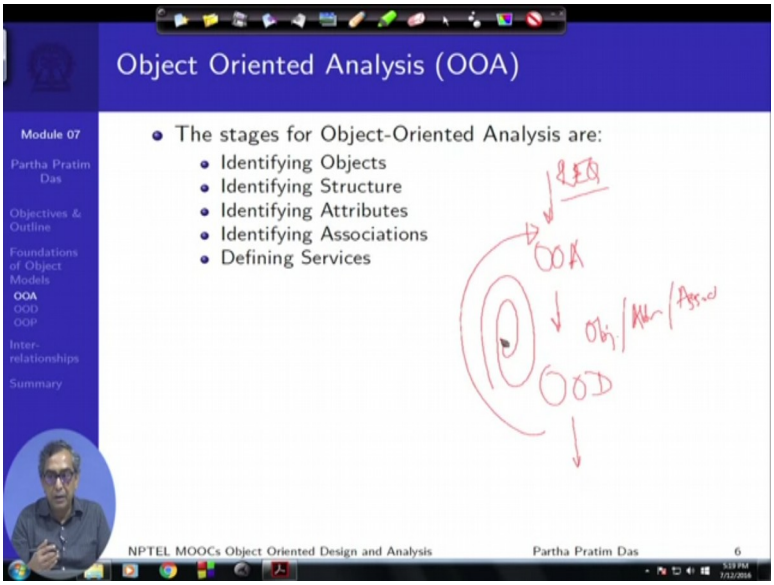
Then you ask that well I know now I know that there are different objects, I know that there are certain structure between them. I know that there are attributes for these objects, but for a system this objecst must be interactive, the objects must be doing certain things together, certain things differently, must be sending messages between them and so on. So, you start working on the associations. So, you say that an executive and request this is barely visible so I will use a different color again.

As an executive can request for a leave. I can say that a manager can approve some leave. So, there are different these are different ways that the objects or structures that you have identified can be interrelated, can be associated. So, these will lead to different, these will lead to different associations that we can have. And finally, to put all things together, you will now have to define what are the possible services that the each and every object has to provide.

What are the services that the system as a whole has to provide for making this useful, for making this confirm to the requirement specification that you have been given with? And in this process, see this is not a flat one-way process and while I just took the LMS example and quickly gave you some dirty I mean back of the envelope kind of reasoning, please do not get scared, we will take you through this in a very structured way in the next couple of modules.

Idea here is just to illustrate that this is the process of analysis that you get started with and as a as I mentioned that whole thing is an evaluation evolutionary process, it is not that you will be able to immediately come to all the objects get all the structures identify all the attributes and have the associations correct, services correctly defined have the whole design. You will do something scratch. Then we will go forward and do certain steps of the object-oriented design.

(Refer Slide Time: 17:27)

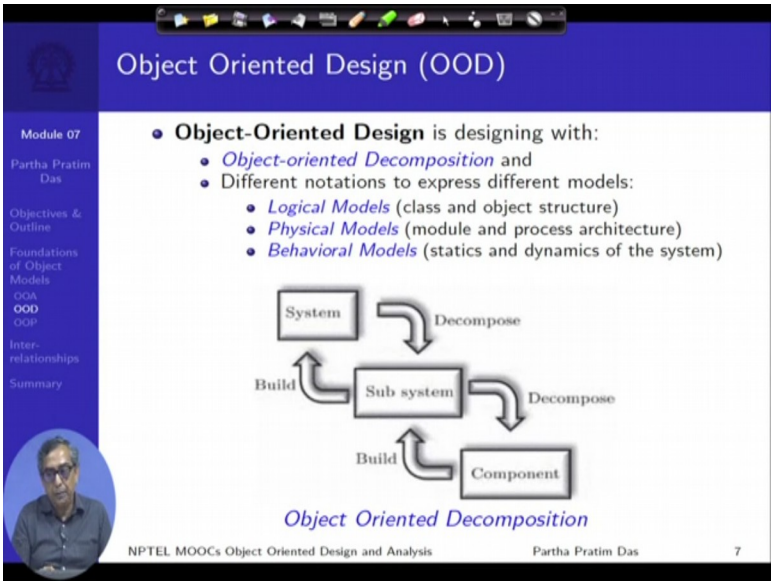


And find that well there are gaps in the design which you cannot fill up so this kind of that is the evolutionary iterative process so you do object oriented analysis. So, you this is where you got the requirement, you have done the analysis. So, we have objects you have attributes, you have associations and so on. And then you do more of design, we will see what design will generate and in the process, we will find that okay things ha have not been completely done, there are contradictions, there are incompleteness.

So, you come back and do the OOA again. You keep on doing this repeatedly and that is the evolutionary iterative process for which you try to perfect your analysis but whenever you are into an analysis what you are trying to do is you are every time you are into analysis, you either go back to the requirements document or to the requirements video of Tanwi and Srijoni or to the diagrams or even when things are not clear in those your setup specific meetings with the clients.

With the customers, with the interface and then ask questions. Well this is specified how do I design in the system. That this process is known as an object-oriented analysis or the analysis process of the exercise.

(Refer Slide Time: 18:39)



This is the first step. Next comes the object-oriented design which builds up on the analysis output that you are generating and it is specifically now talks of object oriented decomposition. So, he will find that it is though for the purpose of sake of presentation and understanding am presenting the analysis and design as if as completely separate phases in practice they will often be very closely related.

You do not just keep on identifying objects and identifying attributes or identifying structures and keep them one side and then you come and do object oriented decomposition. It is kind of you do them together. You identify objects, put them in the decomposition model, identify

attributes put them in a model, you identify services required, put them in the model, those are the messages that is the whole process of regular process of decomposition that will happen.

The second part of the other characteristic activity of object oriented design is you start generating a whole lot of different models of the system. That is, you start looking at the system from multiple angles through multiple glasses and there is no very specific exhaustive list of models that you will provide, that will specify. some of the models are generic, you will be done for all kinds of system and some of the models are specific to certain types of systems.

But it is typical that you will do logical models. Logical models are the basic fundamental object-oriented models which identify, which represent the class and object structures. The class structure if you recall, the reference to the canonical form, the class structure is which gives us the commonality of the behavior, the hierarchy of specializations, the hierarchy of abstraction. And the object structure is the compositional behavior which show that the how objects are built up, of smaller objects in turn are built of even small objects and small.

So, these are basically the logical models that exist in the problem domain. The reason they are called logical models is the fact that these models are completely or significantly dependent only on the problem domain and the requirements that you have given to solve and not so much on the actual target system on which you implemented, actual target system that you built, neither the computer or the architecture on which you are building your system, nor the actual system that you build.

So, the second set of models that we will typically build are called the physical models because they are called physical because that is how your system that you are const or constructing or manifesting that it will show these are the modules. So, in terms of a typical software system that we are doing, these modules are certainly different. collection of may be source files, headers, libraries together independent compilation

You may be dynamic libraries and so on which together does something meaningful as a subsystem. So, in terms of the physical model, a module could be your email subsystem. in terms

of a physical model, your module could be employee management system for the leave part and similarly the process architecture or the software architecture of the whole system which could include the interconnect with the frameworks which could include how your different modules interact between them.

How really your messages are sent? How they responded to? What happens if your situation gets into an exception condition and so on. so, these are the requirements of the physical model and certainly we will talk about several behavioral models which have static and dynamic behaviors that is for example if I talk about the LMS, a dynamic behavior that I need to know is when an employee applies for a leave what happens in terms of the timeline what happens.

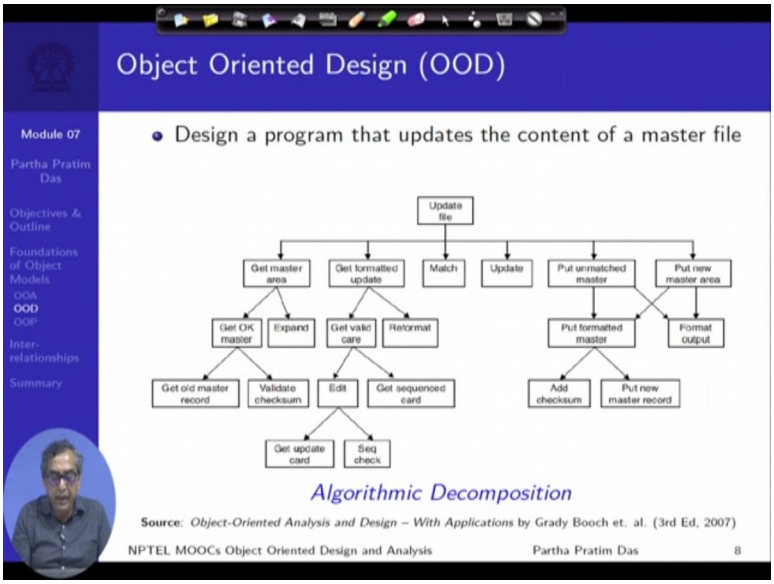
How will the manager of the employee who suppose to look into the leave and to who regret it, how will that that manager gets to know it. And when the manager gets to know it, what is the timeline through which, within which the manager will respond. And once the manager responded how will the original employee who had requested for the leave will get to know. So that is kind of dynamics of the system which says it is not only enough that you have leave as object.

Or executive as object or not enough to have messages to apply for leave or approve leave as services but you also need to have a clear idea about the order in which these things will have to happen the many of them have real time constraints, for example I apply for a leave 3 day down the line say today is a certain day x and I apply for a leave I mean Monday let us say, and am applying for a leave on Thursday.

Now certainly whether I get to proceed on leave or it is regretted, I must get to know it before Thursday otherwise I cannot take a decision. The system becomes inconsistent and if you if you this is a quick example of if you go back to the discussion we had between Tanwi and Srijoni and the document that they subsequently produced will find that this question is not answered. What is the timeline within which a leave must be reacted to?

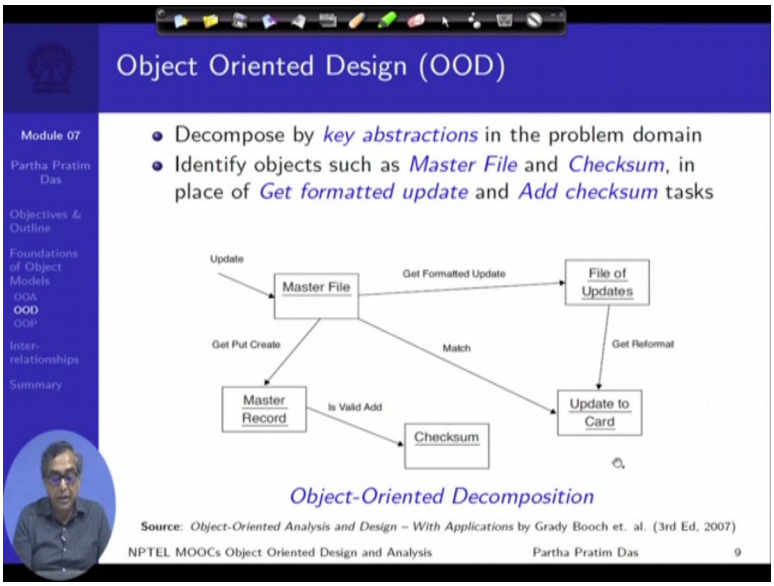
These are a part of the behavioral model that needs to be. So, in terms of the object-oriented design, we basically refine on our findings from object oriented analysis and try to start creating these models. The basic difference between these 2 being that the analysis is repeatedly looking into the requirements and bringing out stock in terms of classes, objects, interactions, messages and object-oriented design is trying to put it in the terms of or actual paradigm.

(Refer Slide Time: 25:25)



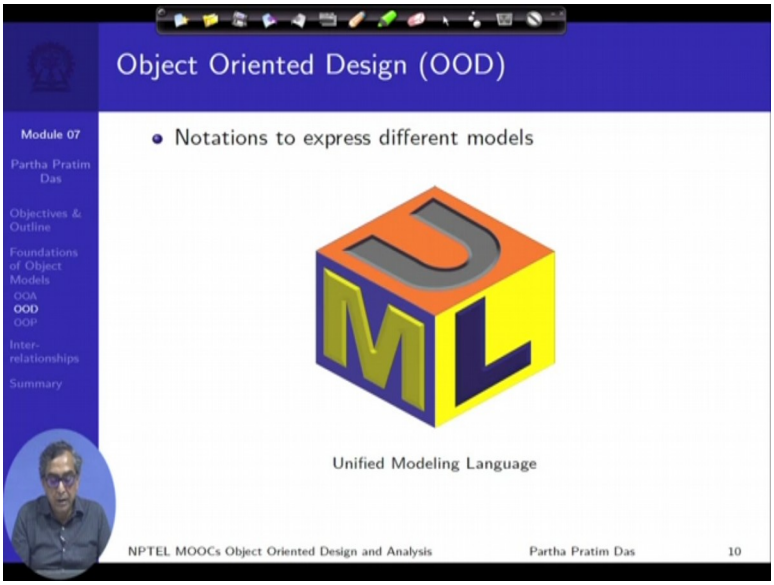
So, this is just for a recap, I will not go over this example once again. We had discussed it earlier as well. So am just making a reference to this that we have talked about updating the contents of a master file and this is what you will get to in algorithmic decomposition.

(Refer Slide Time: 25:45)



And this is what you get to in terms of an object-oriented decomposition and you can see that naturally the process of analysis must have identified these objects, basic objects and because it is somewhat vaguely specified in terms of the problem domain but this come from the vocabulary of the problem domain. The problem domain talks about a master file, talks about a file of updates and so on and based on that this kind of a diagram is getting created in terms of how the objects will interact between themselves and that is the core of the object-oriented design process.

(Refer Slide Time: 26:30)



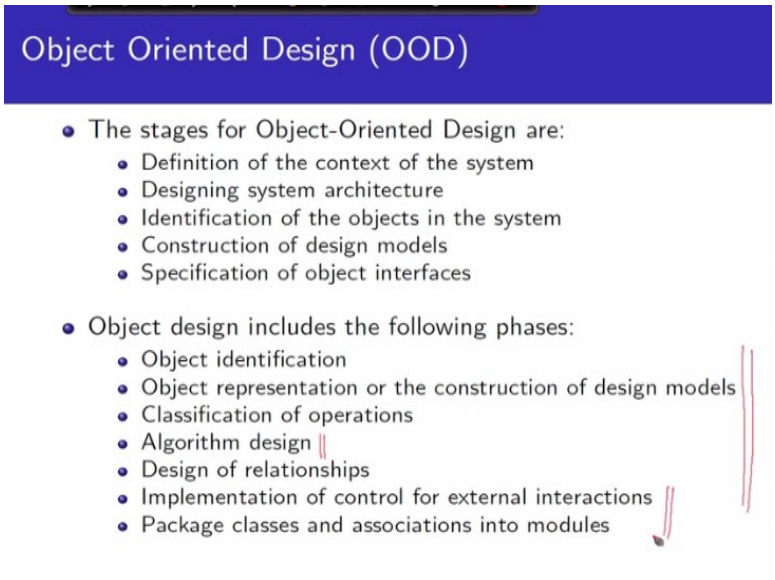
Naturally all those different models that object-oriented design needs to deal with have to be expressed in some concrete terms and we will find that significant part of our course is dedicated to learning for, learning how to express these models which so far even though we are saying that this is a concrete object, this is a concrete attribute, these are messages, services, operations and so on.

We need a mechanism to represent all these logical models, physical models, behavioral models and so on in some well defined, well structured language and we will talk about that language which is called the UML or the unified modeling language. So, with this we will we have taken a look into the object-oriented analysis approach and the basic object-oriented design paradigm. We will now see how these 2 gets engaged into the final system through that object-oriented programming.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture - 11
Foundations of the Object Model – OOA, OOD and OOP

(Refer Slide Time: 00:24)



Welcome back to module 7 of object-oriented analysis and design. We have introduced the basic foundational concepts of object modelling. We have talked about object oriented analysis and what else required to be done for object oriented design. So I will start here with a quick summary of summary and recapitulation of object oriented design. So the object oriented design comprised typically of a couple of stages, so better that we give here is.

The first thing we need to define the contexts of the system that is, you remember when we talked about object oriented analysis, the stage was to start with right away start with identifying objects. Because we are just trying to make something, head or tail out of a video introduction or an arbitrary text and diagrams as well, but now we slowly want to bring it to the usability part of the system, so we try to understand what is the context of the system.

So the context of the system if we talk about the leave management system is the fact that we would like to understand what is the size of the organisation? What is the number of kind of number of employees, whether there are branches? What is the typical kind of leave volume

that is enjoyed and so on, so forth and now that context is not given probably in the requirements, that is the background of the whole thing.

And you will ask, you will wonder is to why I need to know this context? Why is that important to define this context, this context is very significant influence the kind of design that I do. For example, I will just give you in terms of LMS, I will just give you very simple observations in that. Let us assume that we are doing a LMS for a company which works, which is start up works out of a single room and has about 10 to 20 employees.

So certainly what I can assume? I can assume that have a single desktop PC kind of desktop, where there is a single user system, where we were needs to take the leave, will come and log in and put the request and we were needs to approve a review, will again come and log in and do the task, so and as I, you log in, you will get a message screen which shows any updates in terms of the leave that you have requested or you have approved.

And you just get over with it. But in contrast, if I have an organisation which has about couple of 100 employees but all residing within the same physical network within the assembling maybe residing in 5, 6 different rooms. I will need to think about doing a system architecture which is more distributed certainly I cannot assume that there is one system where everybody comes and you know does the leave operations will assume that.

It can be operated access from multiple places and it should be possible to apply from anywhere and approve it from anywhere within the company organisation. Think about another enterprise which is kind of across different locations, may be across geographies. When we do that certainly we cannot just think about having working in a system which is on the single network will typically try to look at some web based kind of internet technology based solution which is completely loosely coupled.

The moment we do that, this is a quiet because if the organisation has offices in Kolkata, in Mumbai, in Ahmedabad and Tokyo and in Rome, then certainly the application that will be the best supported as an internet application and the moment we have that we will have a several rounds of other consequences coming in which were not specified in the requirements document in terms of what is the security of the system.

What if this will be broken through? We will need to take context of organisation in multiple other ways. For example, do people of that organisation travel very frequently, is it required? That it should have a view on mobile phones as well. Now all finally we are talking about a leave management system. Now we will say that, now when certainly, when the organisation give you the specification, the every organisation, every customer thinks that they are the whole world, they are the universe.

So they will think that their situation, their context, their office, their employee, their organisation structure is the only one of the kind that can happen in the world. Therefore, when the requirements are given for the leave management system, certainly the customer did not bother to specify all this context, because to the customer there is only one context in the world and that their organisation.

To you, the vendor, software developer, the engineer, the context could be anything. So it is very critical that the OOD has to talk about defining the context and the success of a good design will depend on that is the second stage will significantly depend on, what is the context? And as I said that it could be just a single note solution, your architecture may not need anything at all.

Just a couple of independent modules which are linked together into one single executable, if it is, for an organisation, which is housed in a single room with 10, 20 employees, couple of different rooms, 100 employees and so on, you will need to have a distributor, more distributor architecture, distributed across geographies, lead to internet based architecture. Depending on the number of employees, you will decide, whether how will the, all these leave record and employee record will be stored.

Would you have a back in data base, how strong will be that back in data base so and so far. So all these are the factors that go into doing your system architecture design and then you will start taking clue from the object oriented analysis steps identify the objects in the system, you already have key abstractions and the structure of those key abstractions given from the object oriented analysis face that you have done.

Using those objects and the details of attributes association and so on, we will start constructing the design models that is we build the logical model, will start building the

logical model, the physical model, the behavioural model and very clearly specify the different object interfaces, so that you can see that under the object interfaces again will go back to references to the architecture.

Because architecture will decide has to if there are 5 different kinds of objects, where do this objects reside. I am just to relate you back to the LMS system in this context. If you are doing the LMS system, certainly we need to know lot of employee information and those employee information, I am not specific to the leave. For example, you need to note the date of birth of the employee.

Now certainly it is quite logical to understand that if an organisation is asking for a leave management system, then certainly that organisation has 5 other systems relating to their employing management, may be in terms of a task management in terms of their payrolls and salary management and so on so far.

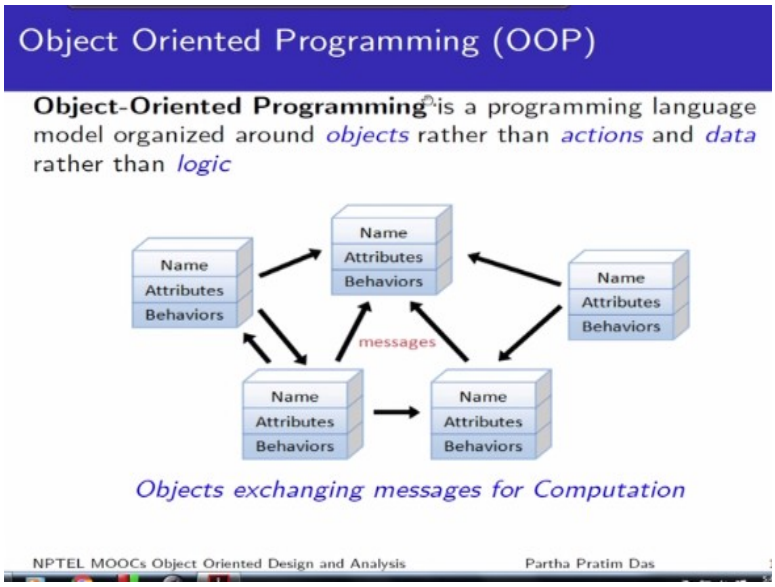
So there are certain parts of the employee related data, employee data which relates to leave, which relates to eligibility of leave, which relates to whether certain leave can be approved, enjoyed by an employee and so on, but a large part of the data is common across other systems that possibly already exist. So those data will not be the exclusive right of your system that you are building, so you can think of modelling employees, but if you model all of these data, which we will typically refer to as master data of employees.

As the soul propriety of your system, your design will not work, because you will not be allowed to arbitrarily make changes or even read some of this data, because at the master data. So it depends architecture has to decide where does this master exists? and what is the interface through which you can actually interact with this master and get extract your related information out and that is the whole exercise that this stages has to achieve.

In terms of specifically designing objects certainly these are closer to what you do in object oriented analysis, you identify the object, you decide on a representation, you classified different operations, we will see more of, understand more of this what does classification mean in terms of, who can use these operations, can any object use that operation or may be the object, define some operations which the object itself has to use and so on.

At which stage, you stop doing object oriented decomposition and start focusing on specific logic that needs to be built up, which will lead to algorithmic decomposition at the lower stages, you do algorithm design and then you decide on the different external interactions, how do you package the whole thing, these are just; I can understand that many of you may not be really being able to follow what these activities will mean?

(Refer Slide Time: 10:57)



But I just wanted to mention to give a glimpse of what are the different tasks that may be involved and these are the tasks that we will reveal to you through subsequent discussions of object models, class models and the UML language. So with this we look into the third and the last of the foundational concepts that is object oriented programming. Certainly analysis and design, analysis is primarily taking out the objects, concepts, notions from the requirements which is weak, arbitrary and inconsistent, contradictory.

And its analysis is giving us certain consistent set of objects, properties, services behaviour all those. Object oriented design is now putting them in terms of the right context, in terms of the architecture, in terms of the different required system models that we have. With these 2, we will finally have to go and build the system and when you build the system we need to actually right code.

So we need programming languages and programming practices and I will emphasise on both of them together repeatedly, what I mean is it is very important to decide on the right kind of programming language because many of the languages support different kinds of features and

depending on what my object oriented or design, analysis and design has revealed what has been the requirements.

The choice of language will be critically dependent on that, just to give you a simple example that If I foresee that my application, my developed system will need to be moved to several different platforms, different operating systems, different kinds of several configurations, system configurations and so on. Then I may tend to use a python kind of language lot more, because it is perfectly portable.

But at the same time I will analyse at their very hard real time constraints, which go below say in the order of below milliseconds, to be met regularly in the system, if it they exist then I cannot use python because python while it is excellent for portability simple easy to code and debug is moderately slow. I should not say moderately slow; it is extremely slow compared to languages like C or C++.

So the choice of object oriented programming platform will depend on the result of analysis, the result of the design that you are making and will also depend on the way you actually are going to design and implement the system. Now it is a common notion that C++ is object oriented language and therefore many of us tend to believe that it is, if I need to do some object oriented programming then I must use C++, while it is good to use C++, it gives you lot of advantages.

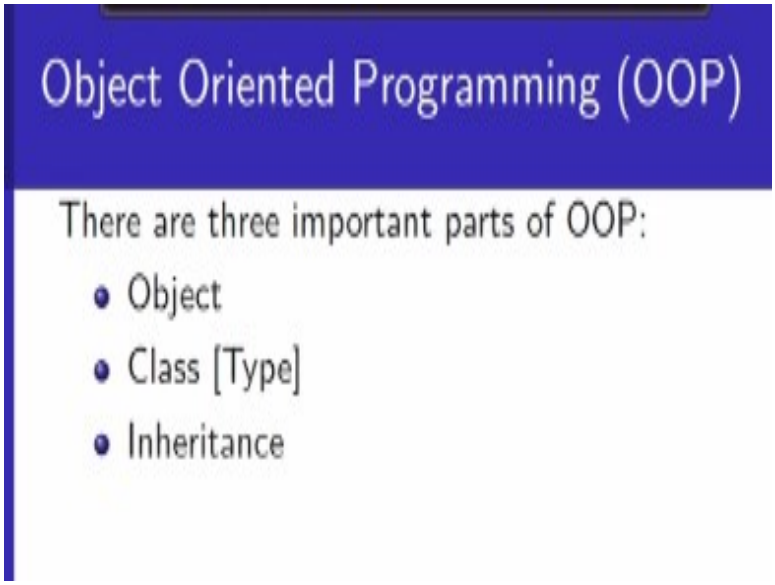
But it is true that many programmers over the years have built many systems with the quietest strict object oriented principle, quite a strict object oriented analysis design paradigms by using simple C language, not using C++.Of course you would, I mean you would wonder and ask us to why should someone do that? There are ceveral reasons for doing that, for example, C is lot more like it.

C is available, C compiler is available for lot many more processes than where C++ compilers available. So when you build small embedded systems, not a big mobile phones, but small systems like a simple voice recorder, a simple answering machine and so on, you may be working with a processor where getting a C++ compiler itself could be an equipment (14:56), I mean the compiler itself, processor is may be so small, then the compiler itself may not run on that processor.

So but it is much easier to get a good C compiler on those, so you program on C compiler, so it is very widely found that several of embedded systems are actually programmed in C, that does not mean that their designs or their implementations practices are not object oriented, object orientation is supported by object oriented programming, but object orientation is more in terms of our thoughts, in terms of our identification, in terms of our analysis and design.

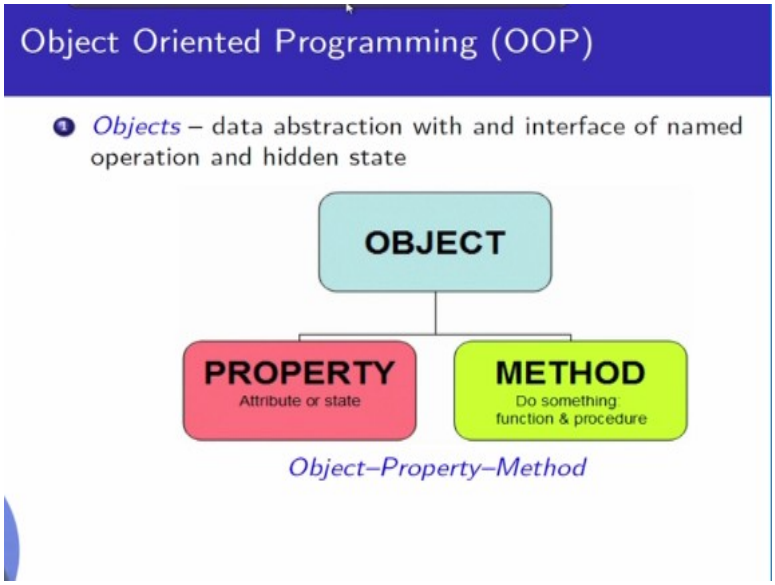
Then the object oriented programming vehicle, the more object oriented, my programming vehicle is; it is easier to program, it is less effort consuming, but we can still program with other platforms as well. So the object oriented programming simply as you can understand, it is organised around objects and rather than looking at actions typically which a more traditional language will do or looking at logic, this looks with objects which basically encapsulate data, we have already seen that.

(Refer Slide Time: 16:32)



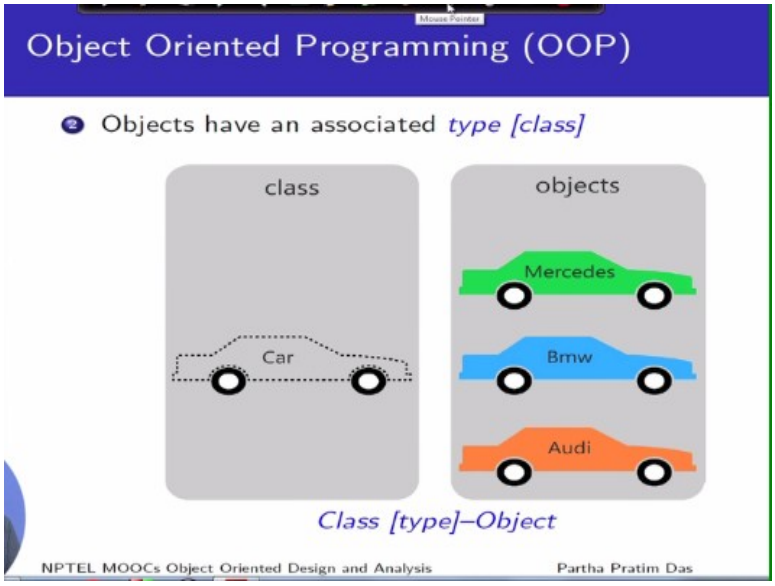
The diagram here is the variant of what we had discussed earlier in terms of the client server model, so these are different objects and they are sending messages between each other to as achieve the different tasks of that what they need to do. There are 3 important parts of object oriented programming and whenever we say that a language supports object oriented programming will typically look for the history.

(Refer Slide Time: 16:51)



Rather we will mandatorily look for the history objects, classes and inheritance. So first is objects, so any object oriented programming language will support the explicit creation definition of objects with 2 components; one is properties or attributes which are basically data and the values of this data will define the state of the object, so if you talk about a leave, then the start date of the leave is a data, the end data of the leave is the data.

(Refer Slide Time: 17:49)

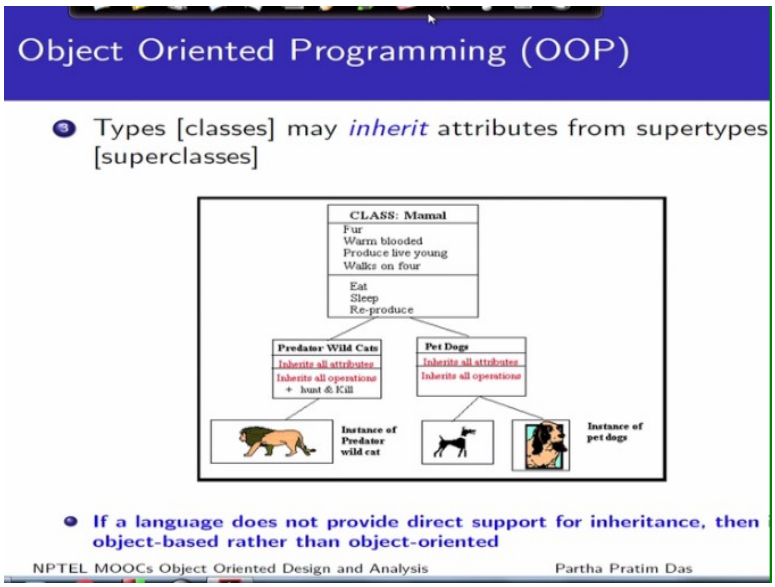


So their values will decide to what is the state of the leave? And the other component is the collection of methods. These are the different services that an object can provide, in terms of a leave object, this will be services like approved leave, revoke leave, enjoy leave and so on. So the objects of the first part that need to be there in an object oriented programming language or system.

The second is objects are associated with type, so this is just an illustration from the automotive industry, right here, you can see variety of different cars; Mercedes, BMW, Audi, we could think about several others but in terms of the basic nature, basic behaviour that they support their say they all are used for transportation and we can say that these are cedars, they are the boutique they can be controlled, their air conditions and all that.

They have a lot of commonality of between what their properties are and that constitutes what we say is a class and object oriented programming languages often will refer to the class as their type, a type like integer is a type, which can add, subtract, multiply. String is a type which can concatenate, which can divide a string at a given point and so on. Similarly, a class will give us all the different properties that it can support.

(Refer Slide Time: 19:11)



It is a fundamental concept of object oriented program. Finally, the third will be the concept of inheritance or specialisation which we will need to explain more, but just if you look into the simple example, here we are talking about a class mammal, sample class. These are its properties, attributes we talked of that is it is far, it is warm blooded, it can produce live young of its kind, it can walk on 4 legs and so on, is a mammal.

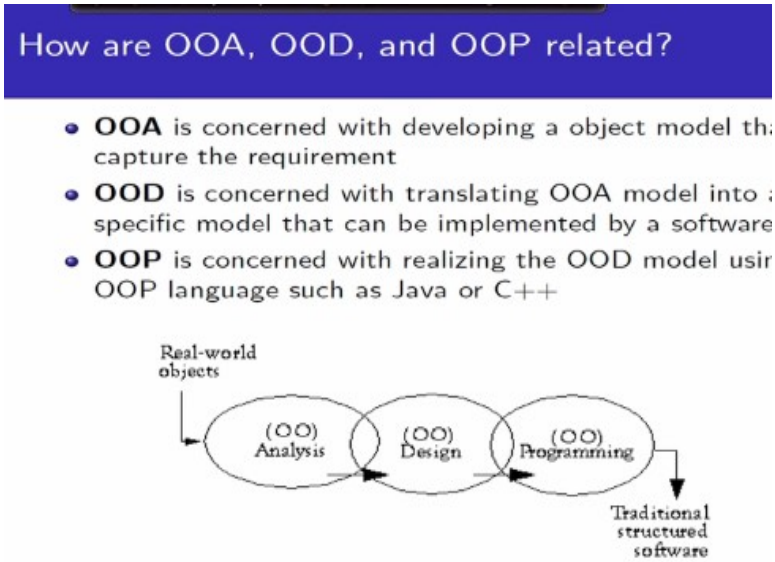
These are the different services, these are the different methods that the class or the object of this type will provide that it can eat, it can sleep, it can reproduce and so on. Now given this basic concept, here we look at one more which you say is a predator wild cats and you say predator wild cats is a mammal, this is an abstraction hierarchy, is a hierarchy which show it has the predator like a lion, inherits all the properties that the mammal has.

The lion will eat, the lion will sleep, a lion can reproduce but it is some more, it has some more, it can hunt, it can kill, so this is what is expressing the concepts interrelated hierarchy where the inheritance can happen that is properties of the super class or the base class which is higher up here, come down to the specialised class the subclass is the basic structure of inheritance, which the object oriented programming languages must support.

For example, if you look into this other side, again the properties come down here, here we are talking about another concept of pet dogs which inherit everything but certainly unlike lamb, this pet dogs will not add its own properties or own methods like hunt and kill, may be you can say you will have a property called cuddle, you can cuddle your pet dog right. So this inheritance is an inherent requirement of an object oriented programming language.

Now there are some programming language, some programming systems which just support objects and classes that is objects and types. But do not support inheritance those languages will typically referred to them as object based language, but not object oriented language, just more a matter of terminology but most of the languages that you will get to use in todays programming world are actually object oriented programming languages and this is what you will be using.

(Refer Slide Time: 22:10)



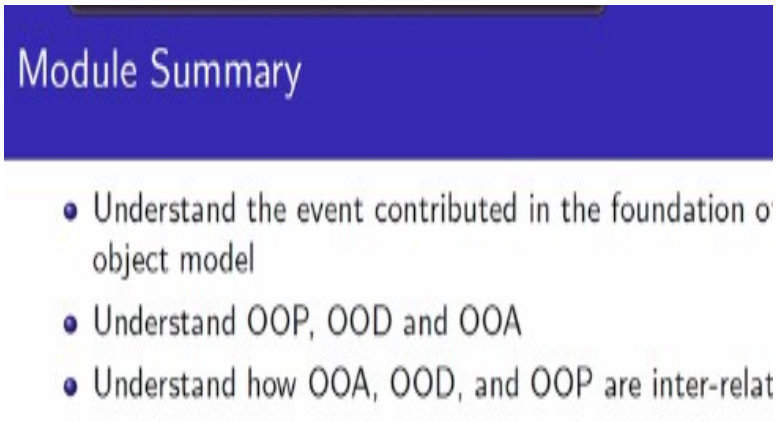
So of all these 3, now we have seen how we start with object oriented analysis to can the requirements, to generate the basic object oriented concepts of abstractions of classes and objects, their structure and inter relationship, how object oriented design basically translates the object oriented analysis into models that are specific and required for the system keeping

the exact contents of the system and several other non-functional parameters like performance and cost and also into mind.

Then finally the whole process of object oriented design also will make choice about what kind of programming language and platform we will use and object oriented programming will be engaged in actually realising these design of classes, objects, interactions and constraints on the actual software system that we are building into, so which will use some language like java, C++ or python, something like that.

While this particular course, we did not really put any prerequisite of knowing this object oriented languages because we appreciate that some of you may be new to the whole concept but I would suggest that in addition to understanding this object notions and analysis and design techniques, please start picking up at least one of the languages, read Java, read C++ or pythons.

(Refer Slide Time: 24:31)



So that you can practice some programming when we come to actually realising the designs in terms of the object oriented programming systems so that you will get a confidence and a more concrete idea in terms of how really all these abstract notions of object class structure hierarchy and all those finally get realised in terms of the concrete software program. So to summarise, we have a here try to understand the events that have contributed in the foundation of the object model.

Specifically taken a look into object oriented analysis, object oriented design and object oriented programming, summarising with their inter relationship to make good object based systems. In the next module onwards, we will start focussing on specific aspects of object models and slowly get into their identification and use in the object oriented system.

Object-Oriented Analysis and Design

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module - 08
Lecture - 12

Elements of Object Model (Major): Abstraction and Encapsulation

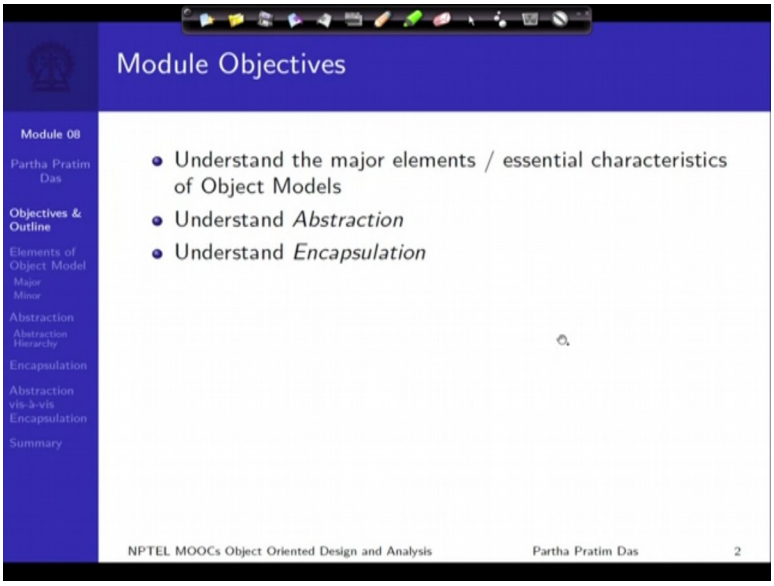
Welcome to module 8 of object-oriented analysis and design. In the last module, we had a brief look on the fundamental approaches in object-oriented paradigm, specifically we talked about the object-oriented analysis that is starting from a requirement and trying to extract may be at times vague concepts, key concepts, key abstractions, objects they are generalization into classes then to perform object-oriented design, that is trying to represent them in terms of the target system on which we would like to realize the system.

And we noted that the analysis and design often is an iterative refinement process where initially you analyze, based on that create a design and based on that design again we go back and look for further details in the refine in the requirements and refine the analysis, have a more refined design, keep on doing this till you reach to a point where you are ready to implement the system or start implementing the system and you get into object oriented programming.

Where you use different language vehicles which either supports object base or object-oriented programming paradigms to make expression of the design easy and in that process, you slowly start realizing the system and once a simple version of the system has been implemented then you go back again possibly to further analysis, refinement of design and then refinement of the system that you have already created.

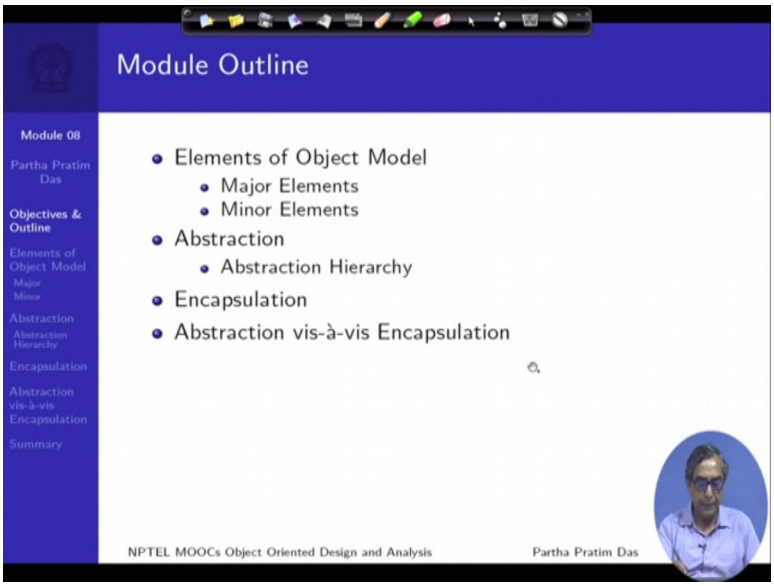
So, this is the overall framework so now we will go little bit deeper into the aspects of analysis and we will start working towards characterization and identification of objects that is key to the whole object-oriented paradigm and the concepts of analysis and design here.

(Refer Slide Time: 02:48)



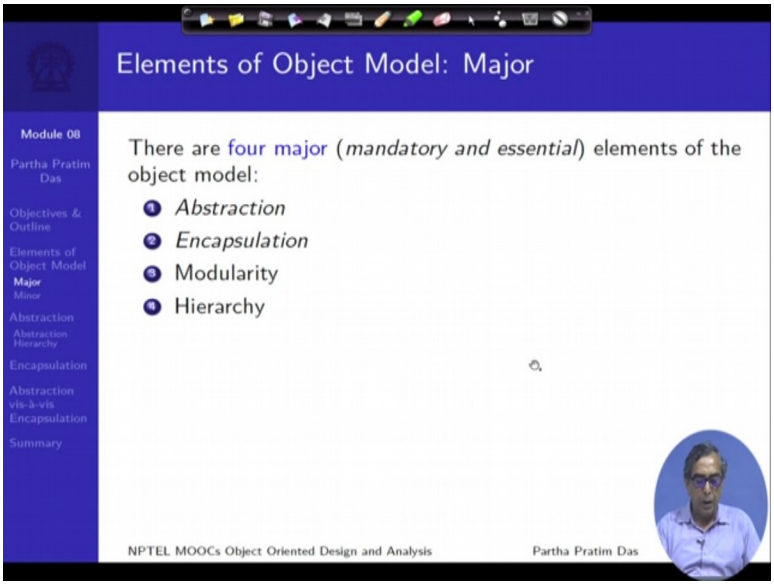
So, in this module, we will start with the understanding of major and essential characteristic elements of object models and specifically talk about 2 such elements.

(Refer Slide Time: 03:02)



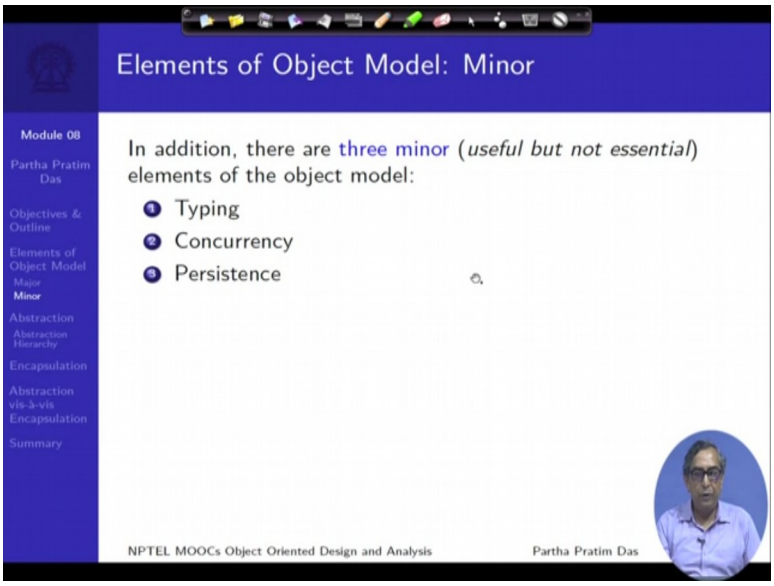
The outline as usual will be available on the left-hand side of every slide.

(Refer Slide Time: 03:09)



Now if we look into object models, then there are four major elements that we need to consider and or endeavored in this module and the next 2 would be to understand this different element of object model. The major elements include abstraction, encapsulation, modularity and hierarchy. You have heard about these terms somewhat vaguely in the earlier modules. Our objective now would be to concretely define these elements. These parameters of object models and see how they play out in different examples.

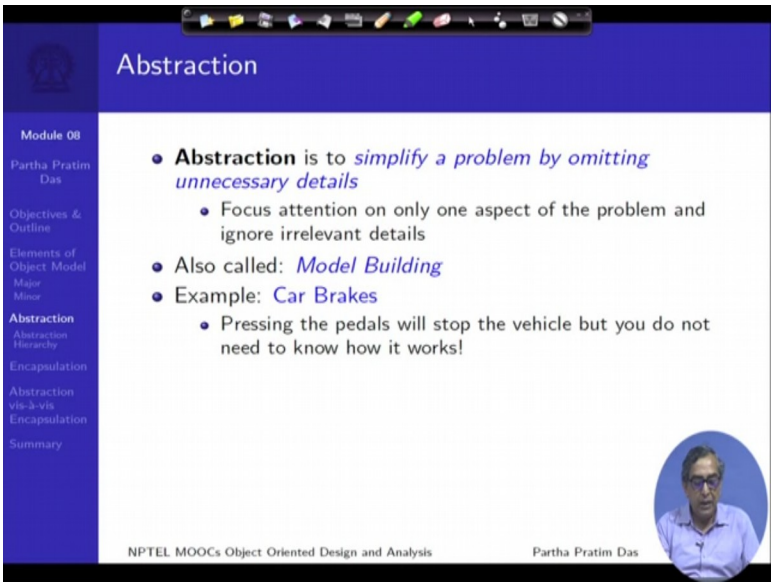
(Refer Slide Time: 04:15)



In addition to the 4 major elements which are essential there are some useful but not exactly essential elements in defining object models. They are known minor elements and they include

typing, concurrency and persistence. So, we will start now with major elements with the abstraction, the core concept of object-oriented programming.

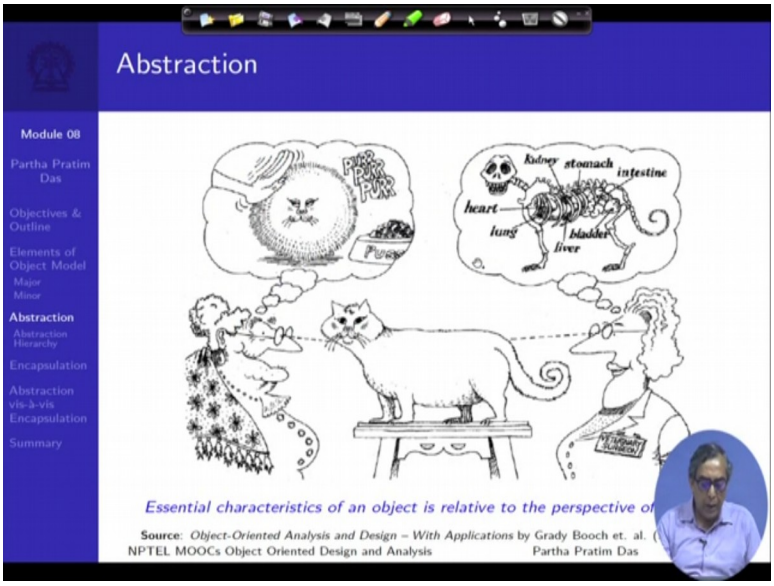
(Refer Slide Time: 04:27)



Abstraction if we as we have already seen and we are reiterating that point. Is abstraction is to simply a problem by omitting unnecessary details. We have seen that human brain is not capable of handling arbitrary amount of semantic complexity. So, things need to be simple at every stage. So, when we talk about abstraction, we first look into what is our intent in creating the description or creating the ba specification that we are trying to do.

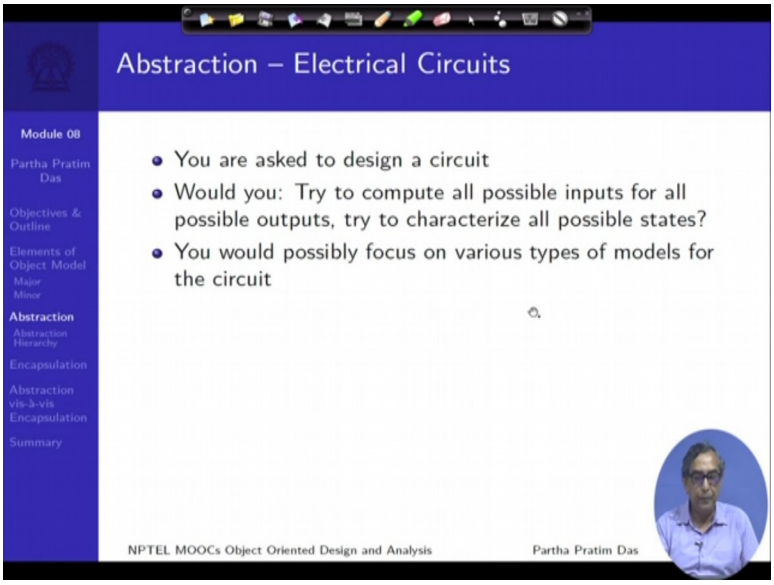
And for that intent we look at only certain specific certain specific view of the system or the subsystem and ignore the rest of the details. This process is called abstraction and as we have mentioned earlier this is also the process of model building. For example, if we are looking into car brakes, as from the perspective of a driver who primarily controls a car, all that we are concerned with in the abstraction is pressing the pedal for the brake should control the speed of the car without much concern in terms of how exactly it happens in the electromechanical or may be even electronic sys subsystem that actually controls the speed.

(Refer Slide Time: 05:52)



So, this is from our OOAD book. This is a kind of a cartoonists' view of abstraction. So, there is a cat and 2 observers on 2 sides. One certainly looks at the cat from certain electronic approaches, the other is a veterinary surgeon looks at whenever he looks at cat, he gets to see all the internal organs. So, it is the same cat, but the 2 observers basically look at 2 different views of the cat as is their interest.

(Refer Slide Time: 06:27)

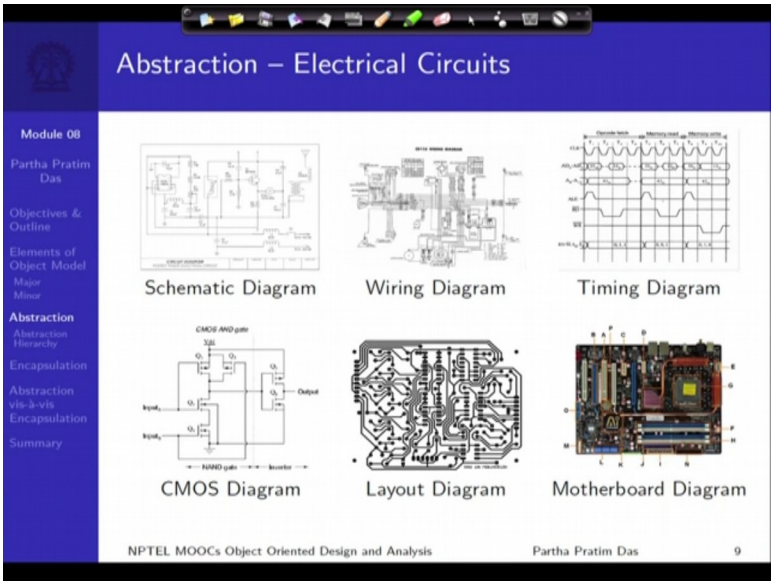


To come to concrete examples, let's start with an electric circuit. Suppose you are asked to design a circuit, it could be any circuit, it could be a staircase lighting circuit, it could be a traffic signaling circuit, it could be more complex digital system circuits and so on. now the question is what should be your approach. An electrical circuit naturally has an input and has outputs and

the they are depends on the time; now would you like to really go over all possible inputs and their corresponding outputs.

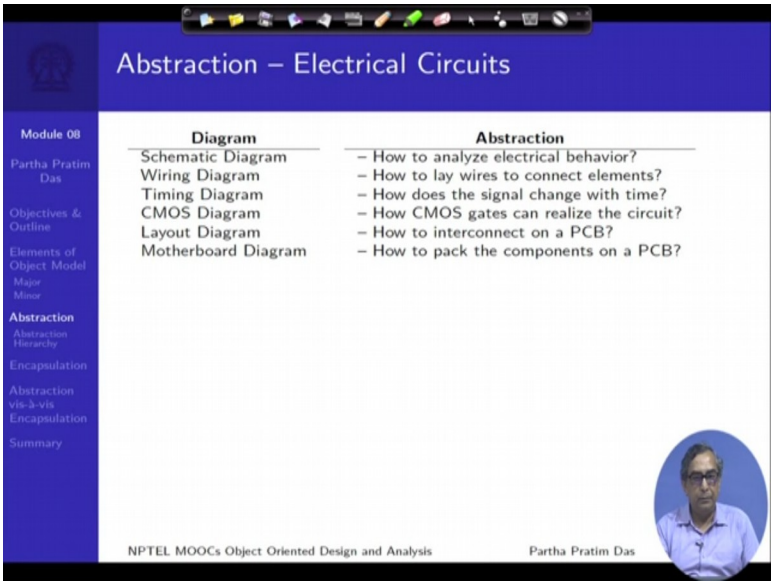
And would you like to characterize all possible states that the system can be in. these are in in a in a mathematical sense, these are possible but in a practical sense, these are not realizable. You cannot look at all possible inputs and know all possible outputs. In in a in a different term, if you knew that then you actually have a system which can do this you don't need to design a new system. So, what you will do, you will possibly look at various types of models for the circuit depending on which aspect you are trying to deal with.

(Refer Slide Time: 07:37)



So please do not try to understand what this specific circuits are, what their functionalities are but this is just to illustrate that the different possible models that are typically used for describing electrical circuits. The schematic diagram, the wiring diagram, the timing diagram, the cmos diagram, the layout diagram, motherboard diagram and there could be several many more that can happen.

(Refer Slide Time: 08:08)



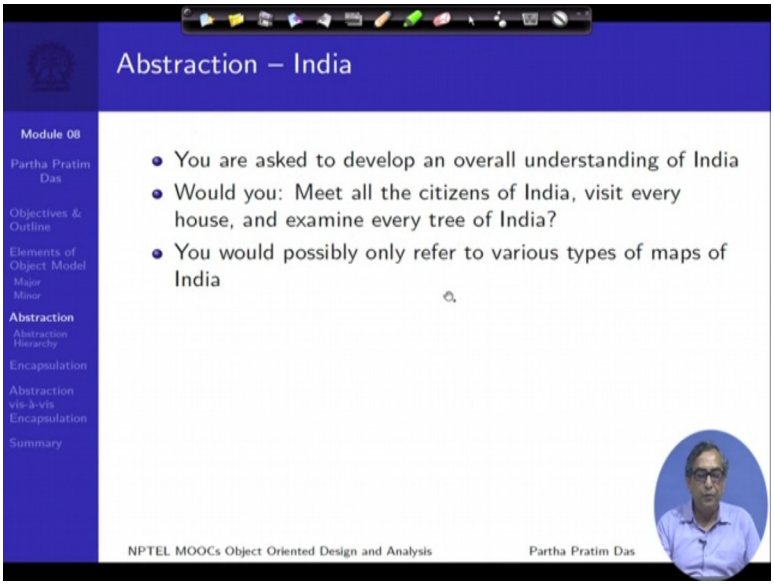
So, if we just take a look when do we use a schematic diagram? When you are interested in primarily analyzing the electrical behavior of the circuit. So, we just want to know how the components are schematically connected without bothering about what kind of wire or what kind of resistor component you are using or what kind of diodes you may be engaging, but you want to use the different kinds of electrical analysis models like Kirchhoff law, Thevenin's theorem, Norton's theorem, super position and all that and you engage into a schematic diagram.

But when you know that you will have to actually start putting that circuit in terms of a board and you know that there are possibly couple of 100 wiring connections that could be made, we will look at a wiring diagram which will show how the connections are being made. If you want to see the the way the output of a circuit will change with time typically you will look at timing diagrams.

If you are looking into semiconductor circuit, you are trying to realize a some kind of a binary logic at a low level, then you will possibly be looking at cmos gates, the smart diagrams. On PCBs, to interconnect component, we will look at layout diagrams, to package components on a physical PCB, we will look at the motherboard diagram and so on. this are this is not exhaustive, this is indicative. So, all the point that am trying to make is an electrical circuit does not have a universal abstraction.

It does not have a universal model, depending on your intent, depending on what you need to do, what purpose you need to achieve, you look at different aspects of the circuit using different models and when you look at one aspect when you are looking at the schematic, you are not concerned about the timing. When you are looking at the timing, you are not concerned about how interconnects happen on the PCB and so on. So that's what is called the basic objective of abstraction.

(Refer Slide Time: 10:18)

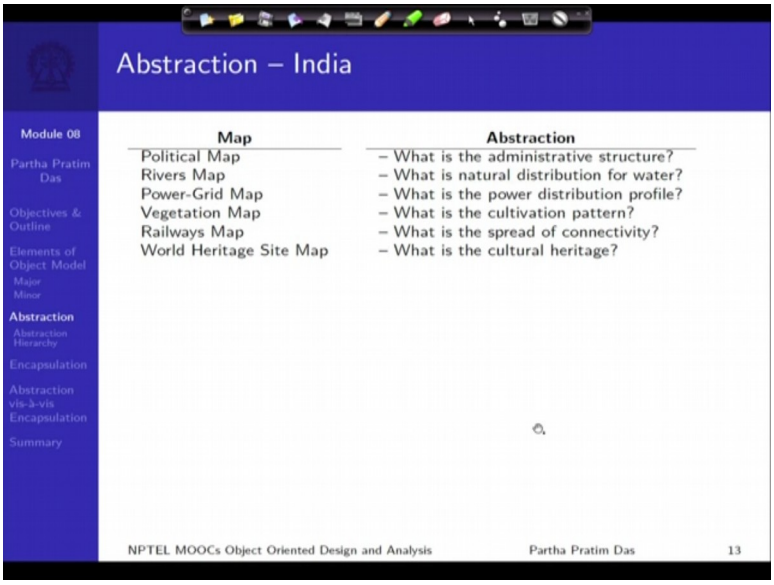


Let us move to a very different example. Let's say we want to talk about our country, India, we want to understand India. You are given that problem, please understand India. How what would you do? You go to every citizen of India and meet and try to understand what the citizen do. We will visit every house, examine every tree, go to every paddy fields and so on. impractical, certainly.

(Refer Slide Time: 10:50)

But what would you possibly do, you will possibly look at multiple different types of abstractions or the models which are typically maps. So, these are just a couple of examples of variety of maps of our country, political, river map, power-grid, vegetation, railways, world-heritage site, if you go to maps of India, you will find that even the list of the different types of maps, different types of models available for India runs into pages.

(Refer Slide Time: 11:18)

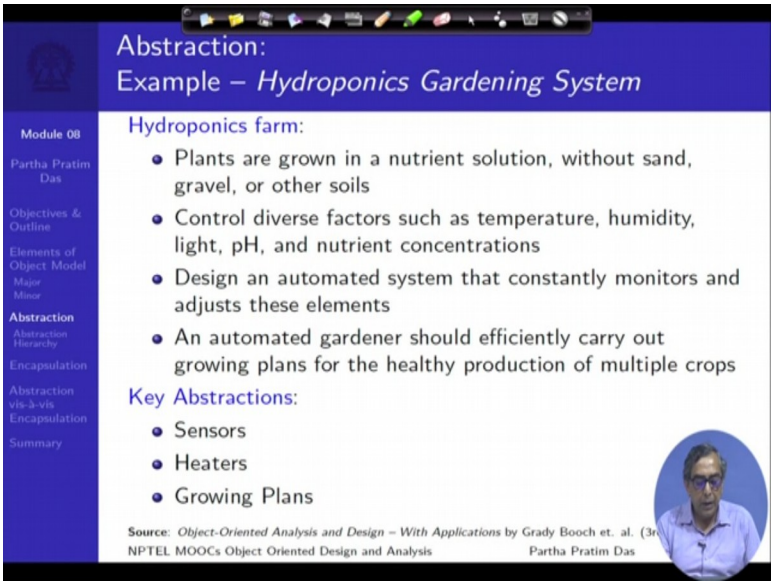


So, if you look into the specific maps a political map we will refer to when we try to study the administrative structure. A river map will tell you about the natural distribution of water resources, the power-grid will talk about how the overall power structure is organized and so on

so forth. So again, when you are looking at a particular aspect, when you are trying to look at possibly you are planning for certain tourism services.

And you want to make a tour of interesting cultural heritage sties and tradition of India, you will certainly look at world heritage site map ignoring how rivers are connected in India. But you may in the next step need to refer to the railways network map because you want to abstract in terms of how those tourists would be transport across places. So, depending on your intent, you look into certain aspects and ignore the rest.

(Refer Slide Time: 12:19)



A third very different example, suppose here we are taking about hydroponic gardening system a hydroponic farm is a soil less sand less, gravel less farm where you just have a nutrient solution to grow plants and the suppose you are trying to model that kind of a system. Now what will you need for a hydroponic farm. You will need to control several greenhouse parameters like temperature, humidity, light, the ph. acidity, nutrient concentrations and so on.

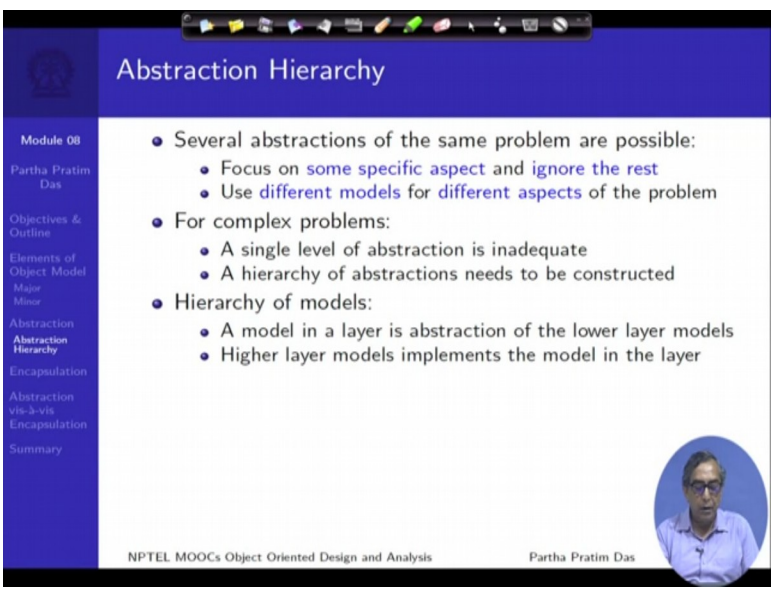
So, if you are intending to design an automated system you would need to carry out the varieties of abstractions or capture varieties of abstractions. For example, in this simple case if I have to control different greenhouse parameters, certainly the first 2 abstractions are quite obvious. 1 is I would need to be able to know what these parameters are at any point of time, so I will need

sensors and since I want to control them, knowing is not enough if this temperature is too low I need to increase that, if the temperature is too high, I will need to decrease that.

So, I will need actuators say for temperature I will need heaters. I will need several other actuators for other parameters. In a little bit, more intangible terms certainly if we are designing a hydroponic farm, then we will need to have an idea about how plants grow and how their growth plan can be controlled by controlling different nutrients and conditions. So, a less little less obvious but very critical key abstraction in this system would be growing plans we which will decide how plants can be made to grow.

So, these are all abstractions but each abstraction looks at a different aspect and when you look at the sensor, you certainly are ignoring the aspects of growing plant. When you are looking at the growing plant, you are ignoring the aspects of the heater and so on but certainly you need all these different abstractions to somehow work together for the whole system to be feasible.

(Refer Slide Time: 14:31)



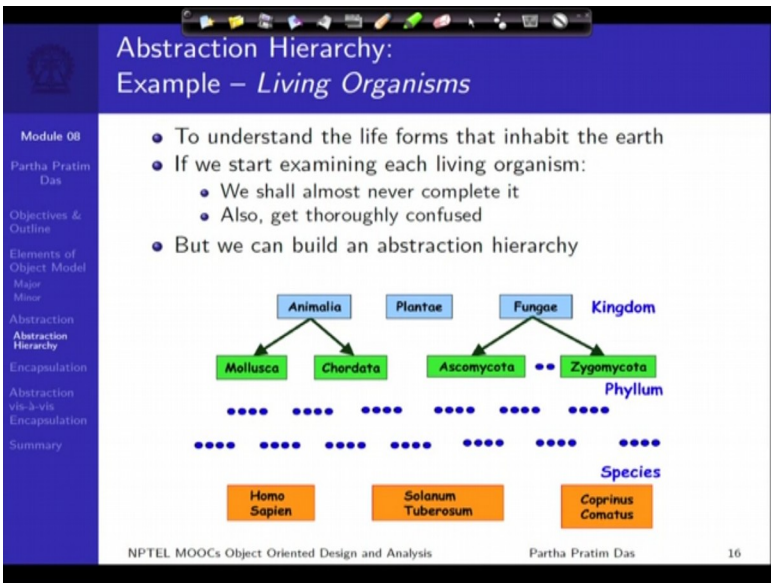
So, this the basic concept of abstraction and we will see that often abstraction is just an isolation; the abstraction is difficult to express. So, what we do is several abstractions of the same problem are often possible and we are focusing on some and ignoring the rest. For each abstraction we have a different model, we have just seen several examples of this. Now when the problem

actually is complex which is the typical target systems that we are looking for, then the single level of abstraction is inadequate.

This is a key point that you will have to remember that if you want that you will take the whole system and you will create a set of key abstractions for the whole system will often find that it is extremely complex. So, what you do is you resort to the concept of hierarchical abstraction that is you try to model a layer in abstraction and you model that using the abstractions of the lower layer combining the abstractions of the lower layer.

Then the layer that you have actually modeled will be used will be implemented in terms of the higher layer of the abstraction. In this process, you create a hierarchy of abstraction for the whole systems.

(Refer Slide Time: 15:59)



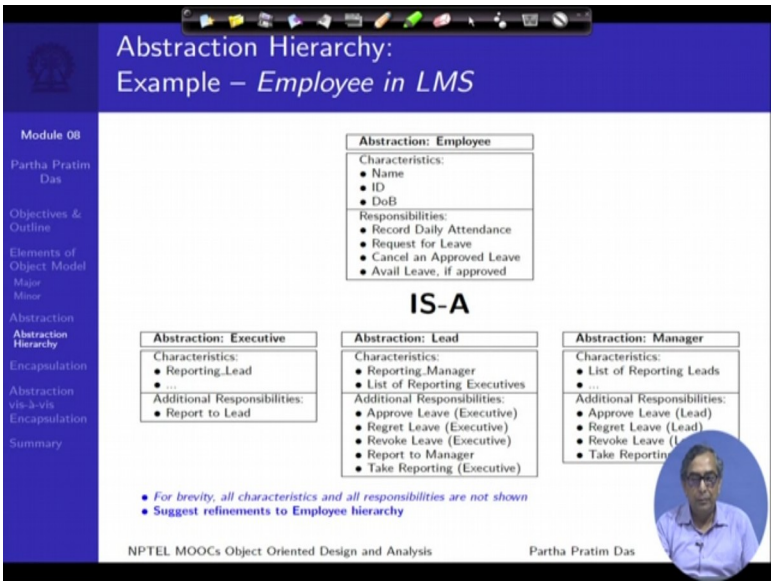
Let us take a look at the examples say we look at we are given to study the life forms that inhabit the earth. Now naturally it is impossible to look at every living organisms so what will you do? You will do a hierarchy of abstraction so if you look into the hierarchy these are more ad towards below are the most specific species. The hierarchy could go even deeper, even within the species we could have.

For example, homo sapiens is us, homo sapiens is human beings so we could have hierarchies go deeper than that where you can say that there are ethnic groups amongst homo sapiens and so on but if you just start species to be the starting point then there are several of them. So instead of looking at those abstractions directly, you group them in terms of their commonality and have multiple layers in this.

You comes to phylum you aggregate phylum's together you put combine them together in terms of their commonality and come to the basic kingdom of living organisms where you talk about animalia, plantae and fungae which say that the basic living organisms of either plants or animals or fungus and there are certain common properties certainly which decide this hierarchy at every stage and from the problem requirements, from the problem specification it is our job to be able to extract this commonality and organize this hierarchy.

But the basic requirement of a hierarchy is very critical because that helps us to manage the complexity of the abstraction in a much better manner.

(Refer Slide Time: 17:46)



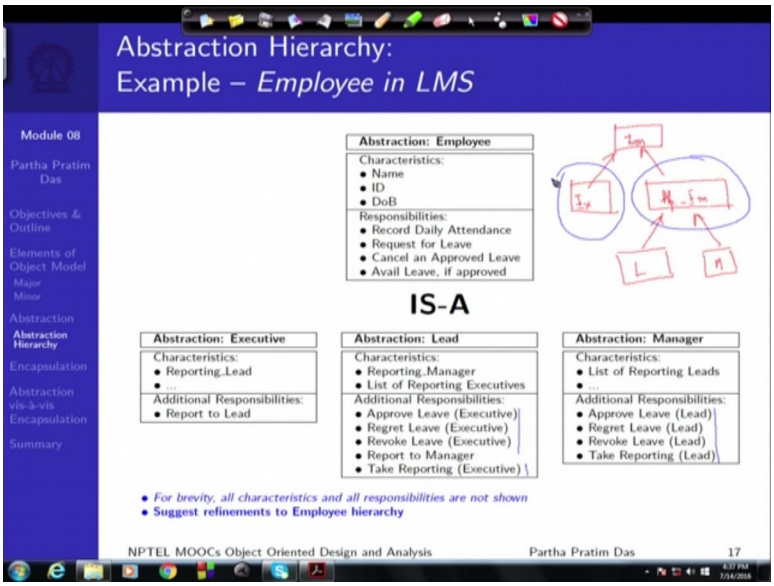
Let us look at little simpler problem but even am referring back to the leave management system we which we are already aware and you know that in this we can have abstractions of employee's executives, there are lead engineers, there are managers and we can express that in

terms of a hierarchy here so what we are saying is an executive is an employee, a lead is an employee, a manager is an employee.

So, in that process if you just look at the specific characteristics then the employee which is more generic in terms of abstraction has a few characteristics and therefore all the specific executives lead and manager will also share the characteristics. But they could have additional characteristics for example an execute, executive will have a characteristic that he or she has a reporting lead.

Similarly, a lead will have additional characteristics to say that a lead has a reporting manager. And in turn has a list of executives who report to his lead, manager has similarly additional characteristics in terms of the behavior of of the responsibilities that this abstraction carry. Few are common at the level of employees which means that all specializations, all lower level abstractions will share those responsibilities but there could be few additional responsibilities which are specific to the lower level abstractions like executives will report to lead, lead can do so many other things, managers can do so many additional responsibilities and so on and in this process, this is ag again again this is not unique.

(Refer Slide Time: 19:58)

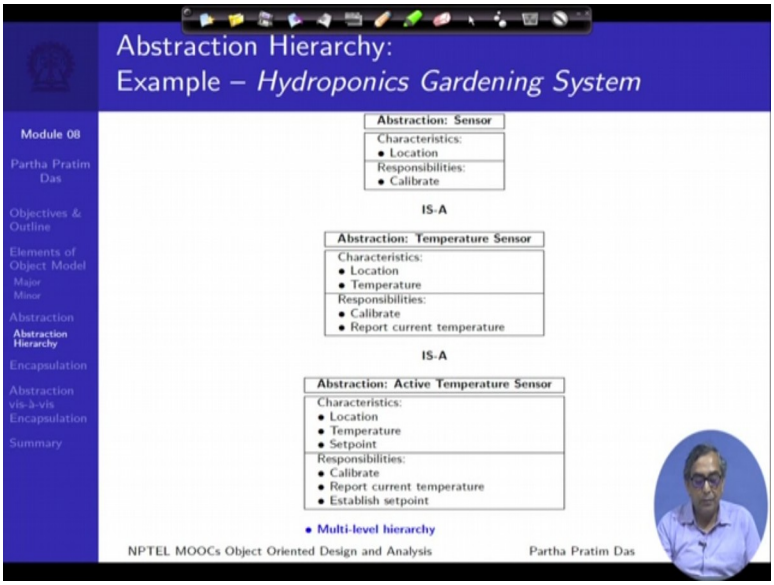


We could for example we could create a different kind of hierarchy saying that this is my employees and this is my executive, this is my approving employees, this is my lead, this is my

manager. I could create a hierarchy of a of this kind also. So, what I do is necessarily I introduce a new abstraction saying approving employee because I observe that between the manager if I take approve leave, regret leave, revoke leave, take reporting these responsibilities I find a commonality between the lead and the manager.

And therefore, I can think of that I have another abstract concept, all those employees who can approve leave for someone else. Executives do not fall in that category because they cannot approve leave for others so they separate out. But here I have another abstraction of approving employees who specialize into the next level of hierarchy in terms of either leads and manager. So, my whole point of illustrating this is to emphasize that abstractions are better expressed when put in hierarchical terms and those hierarchies are not unique.

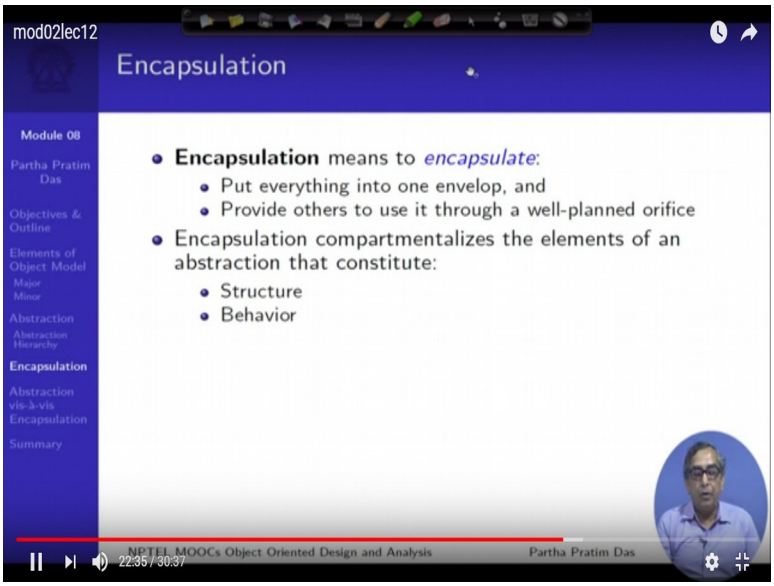
(Refer Slide Time: 21:24)



If you look at look back into the hydroponic gardening system, we have talked about sensors, so we can easily see a multi-level hierarchy that that happens in that you have sensors which generally will have a characteristic of, you need to know where the sensor is and every sensor will need a calibration. So that's the basic property of sensor. Then I have a temperature sensor so in addition to location its characteristic is temperature and should be able to report the current temperature.

Then I may have an active temperature sensor, active temperature sensor which is kind of not just sensor it at certain point of time, it is available every time. It can set a point to function and so on. So, it has more characteristics, more behavior and therefore this is what forms a hierarchy which can also be multi-level in terms of the abstractions.

(Refer Slide Time: 22:23)



The next concept or the next major element that object models will need need is known as encapsulation. Encapsulation means to encapsulate that is in very simple terms like all of us are more familiar with the term capsule and we associate it with the drugs, medicine. We always if we fall sick, then we either have tablets or we have capsules. What is a capsule? Capsule has if you just recall am sure all of us have had the unfortunate situation of having to consume capsules.

We find the capsules have the coating which does not basically taste and but if you have ever ventured into opening a capsule and trying to taste them, actual medicine inside will it will be usually a very irritating taste. So, the purpose of the capsule which has given it that name of the form of administering the drug is to shield is to encapsulate the bitterness of the irritability of the medicine from the acceptability of the tongue and encapsulate is basically that concept applied in terms of object-oriented system.

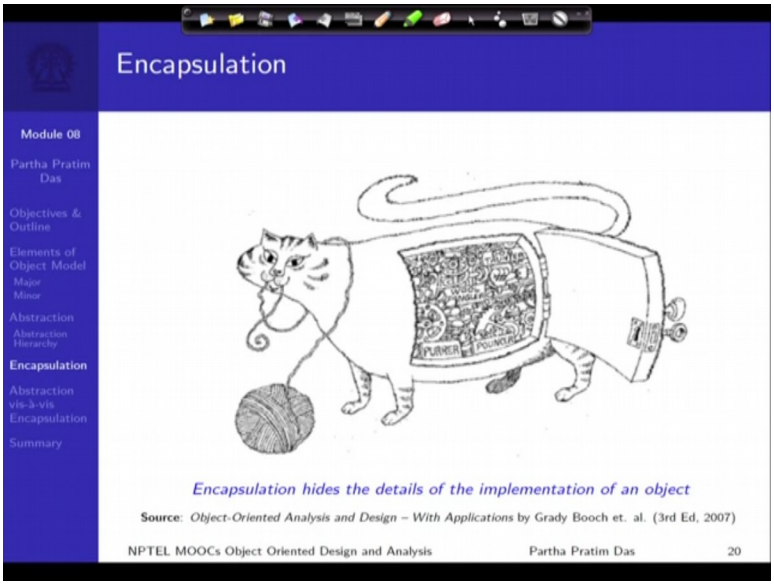
So, it says that you put everything everything in terms of a in terms of I would like to change the color of this, looking into red, okay so put everything into an envelope. Why do we when you put I mean 5 pages of a letter into an envelope and seal and post it, for delivery to the recipient, we are also doing an encapsulation. So, we are basically putting many different things together so that they stay together and to provide another to use it through a well defined, well planned orifice.

So how you use that content of an encapsulated entity has a well-defined plan. So, if it is a medical capsule that plan is to able to swallow and take it to your stomach with the acid will dissolve the capsule coating shield and make the medicine work for you. If it is an envelope then your idea is to retain it for the purpose of expressing the sender and recipient, more importantly the recipient address so that it reaches the right point where the envelope can be torn apart to find the content, letters and so on.

So, there are 2 things in the encapsulation so am just talking about physical examples. In en conceptual terms, there are 2 things that you will need to deal with. 1 is putting things together as one entity and the other is mechanisms to be able to provide access to the contents may be at times selective basis to a with according to a specific plan. So, encapsulation compartmentalizes the element of an abstraction that constitute the structure

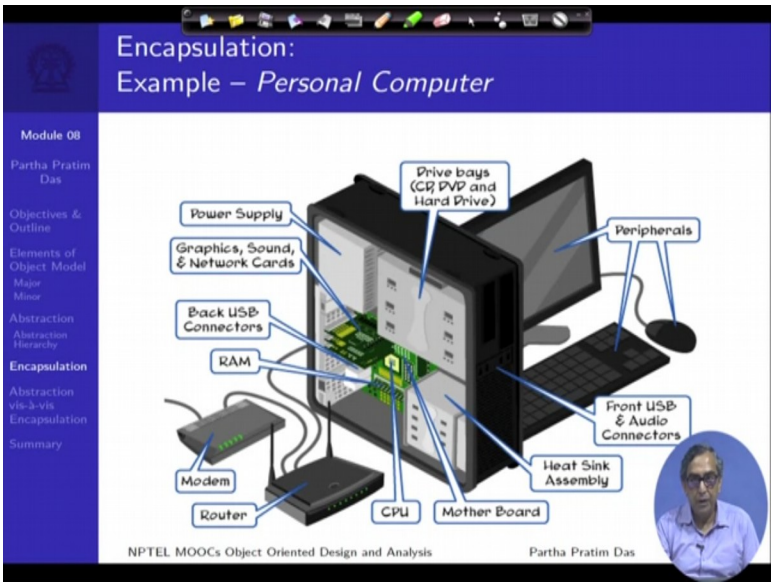
And the behavior which means that we have known that abstraction has a structure, abstraction has characteristics and it has responsibilities that is the behavior. So, we could encapsulate both of them in different separate forms and make them to work together or separately as needed.

(Refer Slide Time: 26:05)



So, this is a cartoon again. I just love this images from the book so it is as if the the whole you know skin and the outer muscular layer of the cat encapsulates what is there inside.

(Refer Slide Time: 26:24)



This is a more concrete example of opening up the personal computer and if you open up am all of us have had experience of looking into that. You find that there is an outer cover which is the outer encapsulation and which exposes only a few of the inner usabilities like you will need to have a power input point. You will have a few usb ports, you will possibly have a serial connector, you will have some leds showing you the if the machine is on or if it's off and so on.

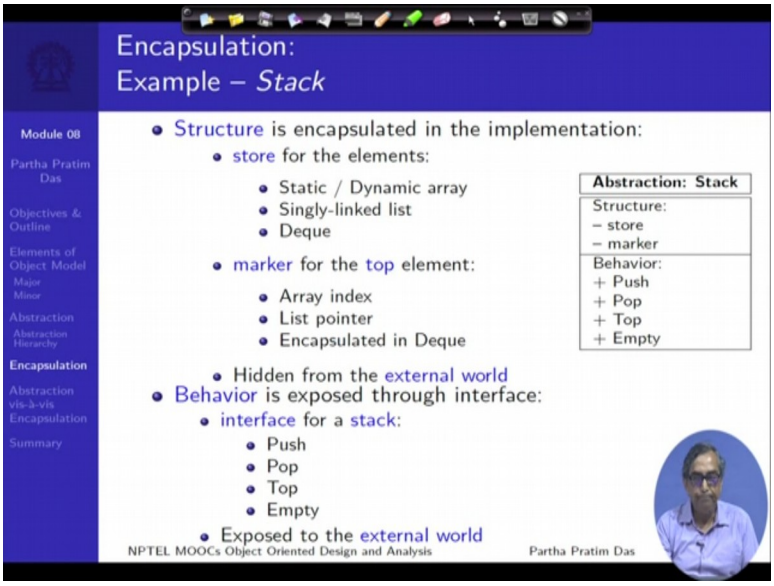
But other than that, it is all encapsulated in terms of this this kind of a big you know big carriage. But if you open inside, you will find it's not that 1000s and millions of transistors, ics roam around, you will find other encapsulations here, they could be the power supply, it could be mother board, it could be the drive which themselves have encapsulated and if you want to take them out, you will be able to do that and in each one you will again find that there is an orifice of use.

If you take out a power supply you will find that there is a power in which takes in the raw power and there is certain power out points which give you different kinds of stabilized DC voltages may be plus minus 5 volts, 12 volts and so on. So, if you look at take out a emm drive say the hard drive, you will find that the hard drive has power input points, hard drive has address and database connections that need to be done, but rest of it is all encapsulated inside a physical cabinet.

Now if you happened to may be lot of unused you know old systems lying around so if you you may want to open up a hard drive and see what is inside, and you will find that there are certain boards, components, pcbs on that and there is actually another encapsulation of a physical drive that exist inside which again if you open up you will find different actual disk platters or whatever different things that exist.

So, the idea is to is to show that encapsulation is a basic process by which you put things together and give very selective usability, selective access to to different people who need to use that.

(Refer Slide Time: 28:55)

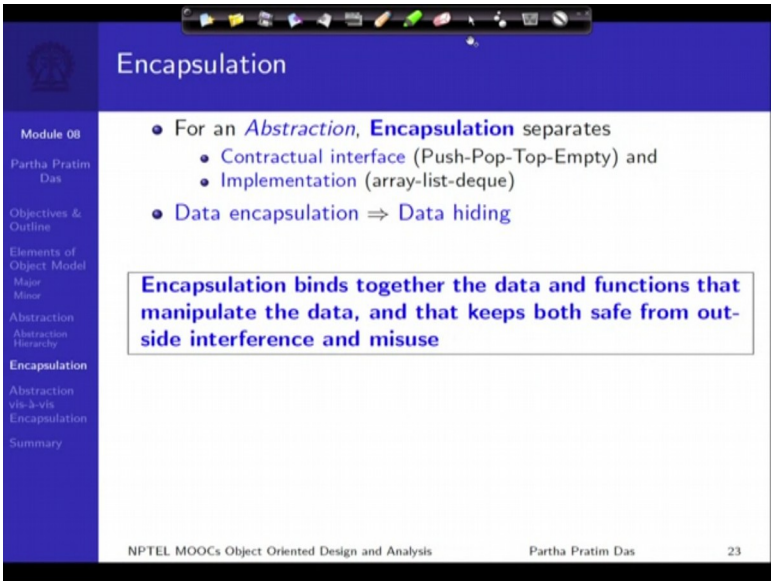


And coming straight to computer science, encapsulation is is something you can understand in terms of if you are trying to create a stack, it has to have a structure and behavior, the structure must contain 2 things, a container to keep the values that I want to stack and the marker to mark the top of the stack and there could be several different ways in which I could actually keep this to us in terms of array, list, dead, li double ended queue and so on.

And accordingly there could be different types of markers and that is what I I illustrate here am am loo I mean am suitably using a little bit of notation which you will over time get adjusted to am using little bit of uml here to write and to demark at that is minus symbol between in front of store and marker basically show that these are these are guarded part of the structure, these are heavily encapsulated and cannot be taken out at all almost and then I have a behavior which basically tell me that these 4 behavior what designates a stack.

So, this part is encapsulated and this is the orifice through which you use a stack. You really do not you may have a stack which is on array, on a or on a list or on a deck, you just really do not care as long as your orifices are same, your use model is the same.

(Refer Slide Time: 30:27)



So that's the basic property of an encapsulation. It hides the certain interesting parts of the implementation from the contractual interface.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 13
Elements of Object Model (Major): Abstraction and Encapsulation (Contd.)

(Refer Slide Time: 00:33)

Welcome back to module 8 of object-oriented analysis and design. We are discussing on elements of object model, we have talked about one major element of abstraction, we have introduced the notion of encapsulation. There is a second major element. We will continue from there, we will little bit in this see how encapsulation and abstraction interplay between themselves. So in encapsulation is for an abstraction and encapsulation does 2 things.

It separates the contractual interface and the implementation. This is a very very key concept to understand and this is the first point from where you should start realizing that the different elements, major and later we will see as well as minor elements of object models are not independent aspects. They are not competitive; they are more cooperative. In that one every major element supports the other major elements in terms of the design so we have seen the abstraction of concepts, the key concepts, different hierarchy of that.

Now encapsulation actually helps abstraction by in terms of this. That is what I want is, I do not want when I implement that abstract concept, I want to kind of as I say, hide that implementation. A concept,

a term you may have heard quite frequently is encapsulation basically hides that. You have put it all together into the packaging of the hard drive, you know that there are rotating disks but it is hidden, you cannot see that but at the same time for a system to, if everything is hidden then the system has no use.

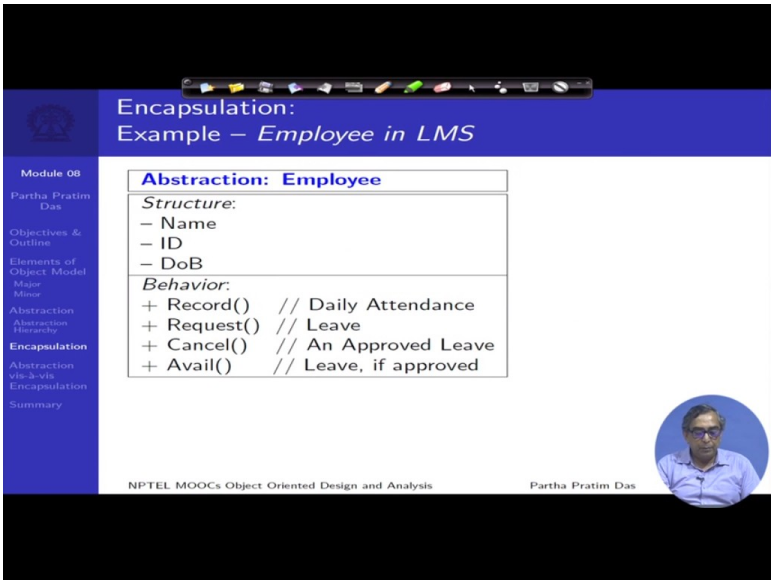
But for the system to be useful, we need to put an interface in which I write here as contractual interface. Why is it called contractual? It is called contractual because my actual implementation inside, my hard disk could be rotating disks, it could be magnetic disks, it could be optical disks, it could be solid state hard disk and so on. But my interface of being able to put an address and get the data out from that address or write a specific data to an address cannot change because then the hard disk as an abstraction fails to provide the services.

A stack may change in its implementation but the basic operations of push pop top and empty of a stack cannot change. So what we say is an interface that others can use to view the system, to model the system has to be invariant with respect to the implementation and that is kind of the concept kind of ease that in a client-server model interface is what your clients get to see, who wants to use your object, who wants to use that object.

And you have to make a commitment to the client objects that I am not going to change that giving rise to the notion that this kind of interfaces are contractual in nature. These are not contractual in the sense that you sign anything on the stamp paper but these are contractual in terms of commitments that you are making and then that process, the data encapsulation basically give rise to data hiding so the implementation is not exactly known to the outsiders but the contractual interface allows you to use that abstraction.

And so if we look, go from abstraction to encapsulation basically we are dealing with the same thing but in a different package. In abstraction, we have not at all talked about the implementation itself. But in encapsulation primarily talks about that implementation and wants to keep it away from the public view.

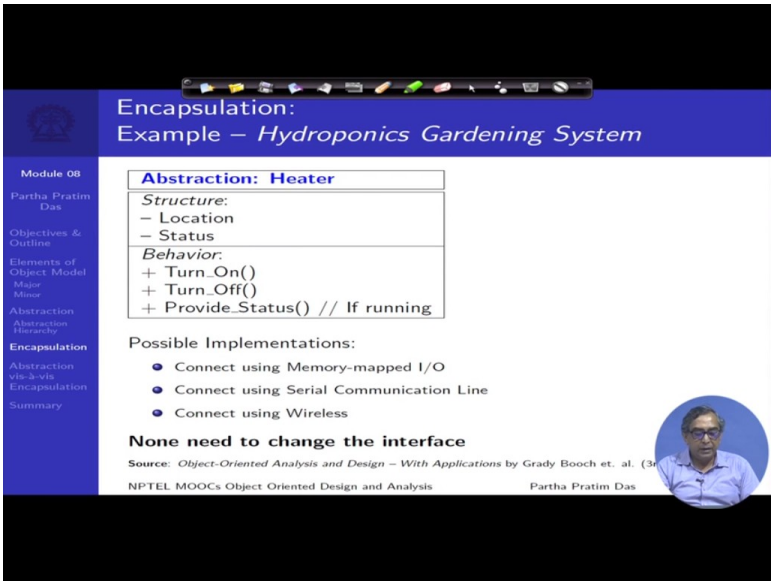
(Refer Slide Time: 04:33)



So there are different could be different example. For example, here going back to the leaf management system, this is the employees, we saw it as an abstraction you can look at the same thing as encapsulated as I mentioned that these symbols will mean that this these are part of the implementation which are hidden, which cannot be seen by others. This show that this is public or this is contractual interface which other can use.

So given an employee, you may not exactly know how it stores the name, the id and date of birth and so on but you will always be able to use that object to record the attendance or request for leave and so on.

(Refer Slide Time: 05:26)



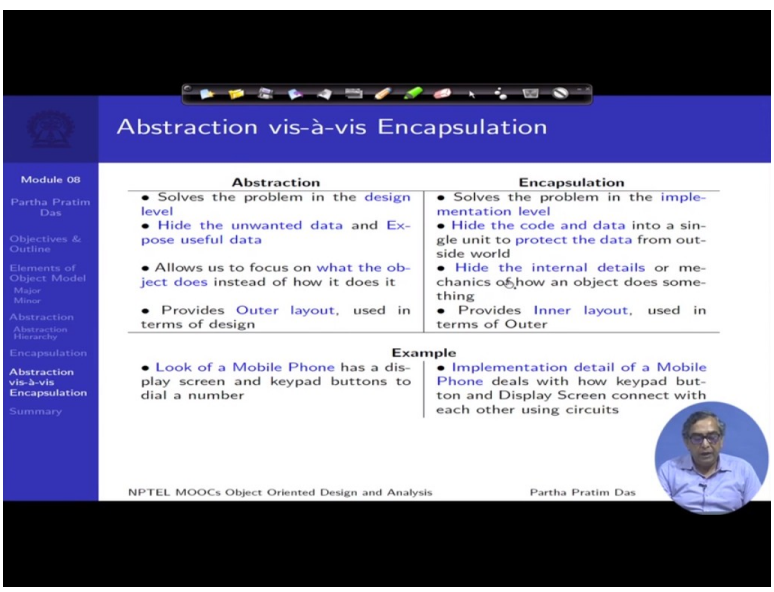
In the hydroponic gardening system, we can think about a heater. It is a abstraction is one of the other

abstractions and it has similarly a structure or implementation of location and status and the functionality or contractual interface of being turned on being turned off or just checking kind of a Boolean operation which checks whether the heater is running or it is off. Now it is quite possible that actually the implementation of this heater.

In terms of the system could vary in terms of the actual make of the heater even in terms of the way the computer system, your control system will control this heater or connect to that heater. You could connect using a memory mapped I/O. I mean if you have some idea about computer architecture or microprocessor will understand what is memory mapped I/O. It does not matter if you do not understand that but it could also connect using a serial communication line.

Or it could be that you just want to connect it to a wireless communication stack so that there is no physical wires that you would like to turn around and there could be various different possible implementations and the key point in the whole thing is that there is no need to change the interface that remains contractually fixed.

(Refer Slide Time: 06:50)



Finally, if you look into abstraction and encapsulation as complementary elements of object model, we will see certain characteristics that abstraction is primarily at a design level whereas encapsulation is talking more in the implementation level. So if you are talking of abstraction you may be putting some hints about implementation but you are necessarily talking about the contractual interface because that is the design.

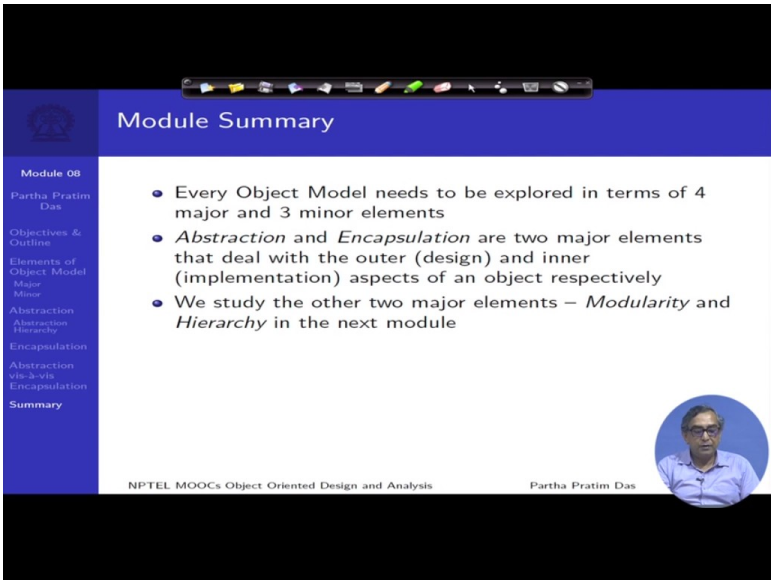
If you are talking of encapsulation, you will certainly have to honor the contractual interface, the abstraction, the concept that the abstraction has given you but you are certainly talking about how to implement that system, how to package it together. Abstraction on one side will hide unwanted data and focus of the on the useful data, we have seen that. In contrast encapsulation will hide the code and the data and protect that data. These are basic encapsulation basic purposes, data hiding.

Whereas abstraction will hide aspects which is not interesting for that concept. Encapsulation does not go back and rip that apart. Encapsulation starts with whatever abstraction has given it in with it works only within that paradigm but now in terms of the code and data it may choose to expose some and hide the rest. Abstraction will allow to focus on what the object does instead of how it does it. So what is again it is kind of the same thing being said in little bit different terms.

Encapsulation looks at the internal details and therefore abstraction is providing the outer layout whereas encapsulation provides the inner layout of any object model. For example, if you look at a example of a mobile phone, it has a display screen, keyboard, buttons and you can use that to dial a number and make a phone call in terms of an abstract view these are the contractual interfaces but how that those are implemented? How that keypad is implemented?

How does the display function, are parts of the encapsulation which is hidden from our common view. So this is so you can see that encapsulation and abstraction are very strongly related and complementary concepts are complementary elements of the object model which we will deal with in terms of our object model design.

(Refer Slide Time: 09:30)



So in summary we had we have seen that every object model needs to be explored in terms of 4 major and 3 minor elements. And we have in this module explored on the 2 elements, abstraction and encapsulation one which deals with the outer or the design aspect of object models and the other encapsulation deals with the inner or the implantation aspect of the object model. In the next module we will study about the other 2 major elements of modularity and hierarchy.

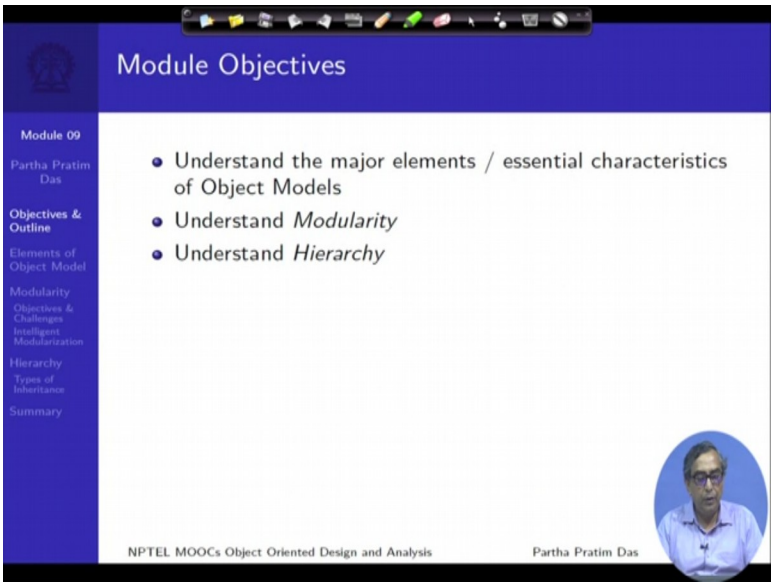
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 14
Elements of the Object Model (Major): Modularity and Hierarchy

Welcome to module 09 of object oriented analysis and design. We have been discussing about elements of object models. we have noted in the last module that there are 4 major elements of object models and 3 minor elements. Of the 4 major elements we have already discussed about abstraction and encapsulation. We have seen that abstraction is widely used to identify the key concepts in the system and it relates to primarily design formalization.

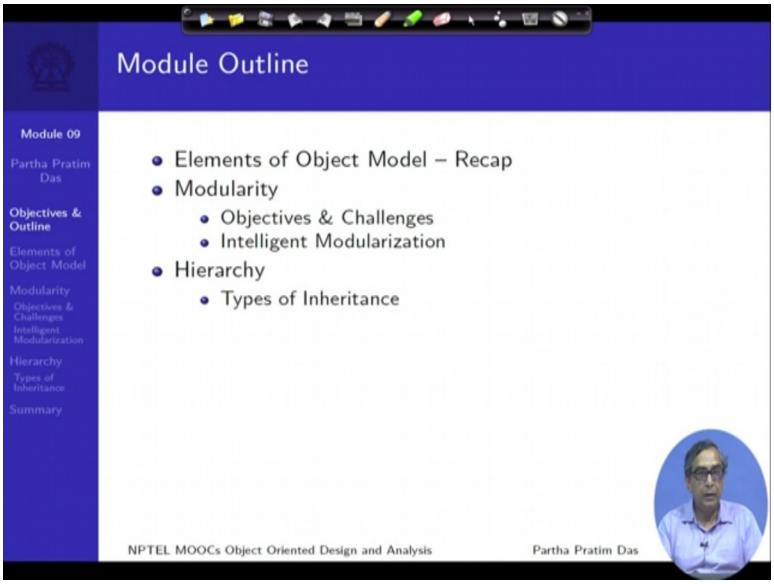
Whereas encapsulation has to deal more with the implementation aspects of the system which encapsulates or packages the implementation and puts it the way from the public view giving only a limited required view of contractual interface to the users and we have seen 2 different examples as to how encapsulation and abstraction interplay between themselves.

(Refer Slide Time: 01:52)



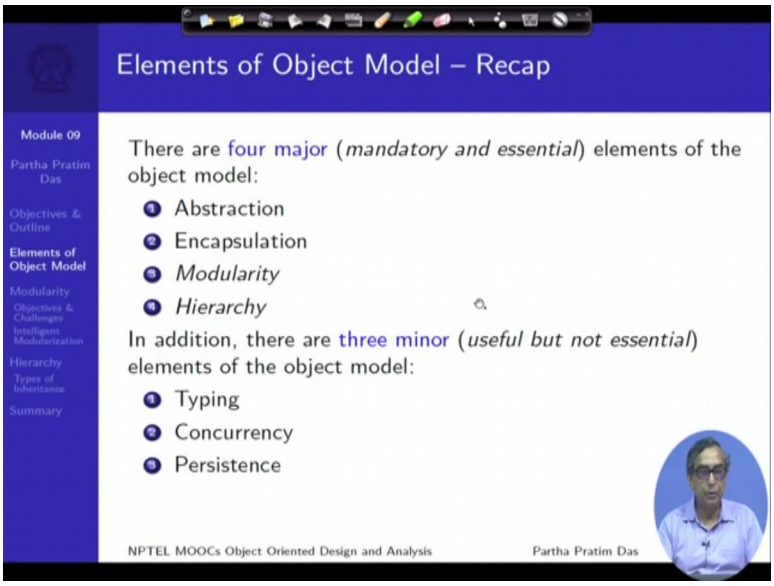
Building on further, we will in this module take a look into the other 2 essential characteristics of object models namely modularity and hierarchy.

(Refer Slide Time: 02:07)



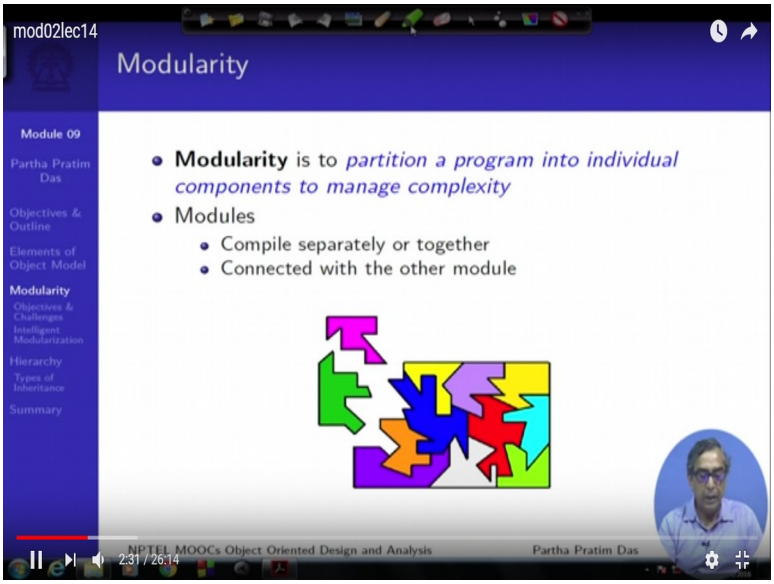
This is the outline; we will present a quick recap and then discuss the specifics of modularity and hierarchy.

(Refer Slide Time: 02:16)



So, this is just to remind you that these are the different elements of object models that we are discussing in this series of models and we will now focus on modularity.

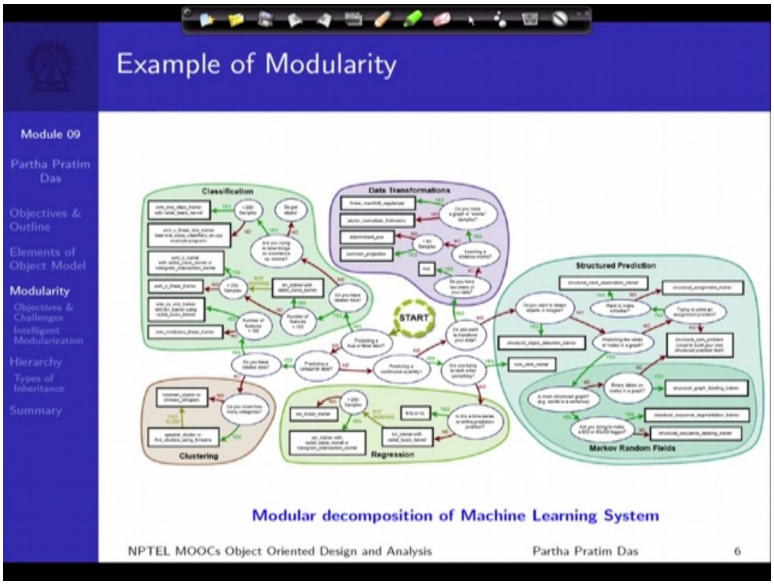
(Refer Slide Time: 02:28)



So, modularity has to do significantly in the way a program or the code of the software, the data of the software, how it is organized. so modularity is about partitioning a program into individual components and we want to do this we want to make modular designs so that we can again manage complexity. So it is yet another aspect of system management which addresses the original set of issues we had raised about software, about the inherent complexity of software.

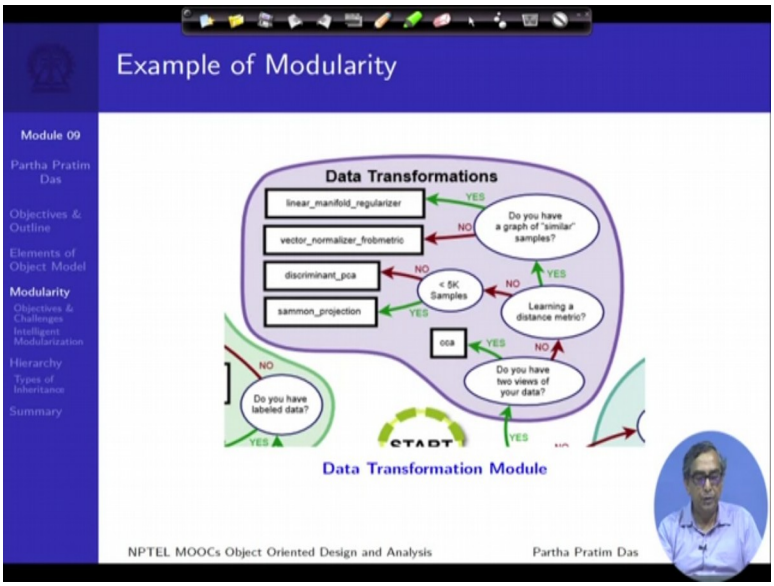
And the inherent inability of human beings to deal with more than a few semantic elements in any design or any implementation. A typical characteristic of a modular designs or module, modules themselves would be that they compile separately doing cases, they could compile together and obviously there is interconnections between the modules.

(Refer Slide Time: 03:50)



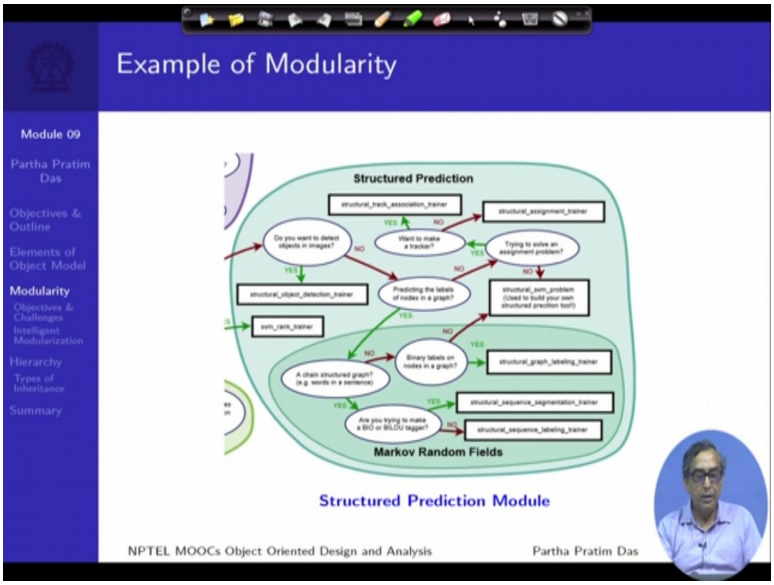
So, will start with an example so this is please do not get worried about the clutter of this diagram, it is intentional. We are just trying to show that if you are dealing with the design of a machine learning system then this is the kind of different components that you need to deal with and these different sets shown in different colors show the different modules. So I will quickly take a walk around the modules.

(Refer Slide Time: 04:25)



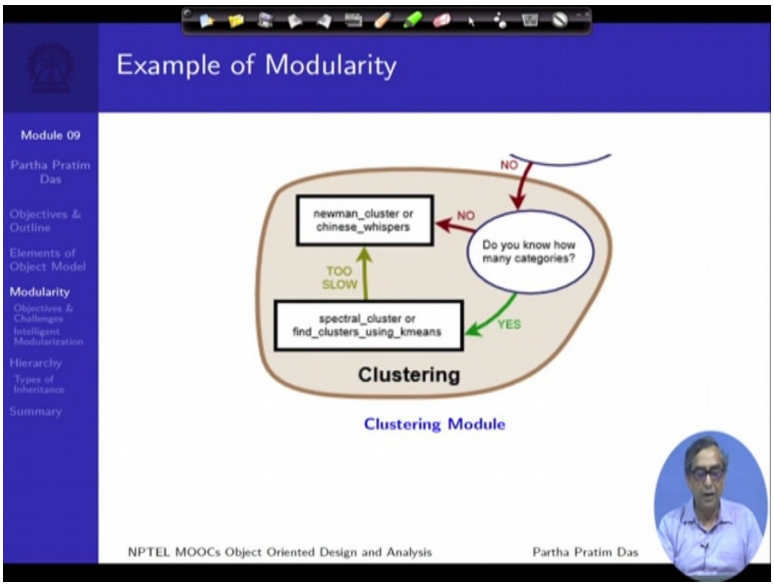
So, one module is data transformation where what we need to do is in whatever form we are getting the data for machine learning, we have to transform them, represent them into different standard forms of a linear manifolds or vector manifolds and so on so all components relating to this transformation tasks is put together in the data transformation module.

(Refer Slide Time: 04:54)



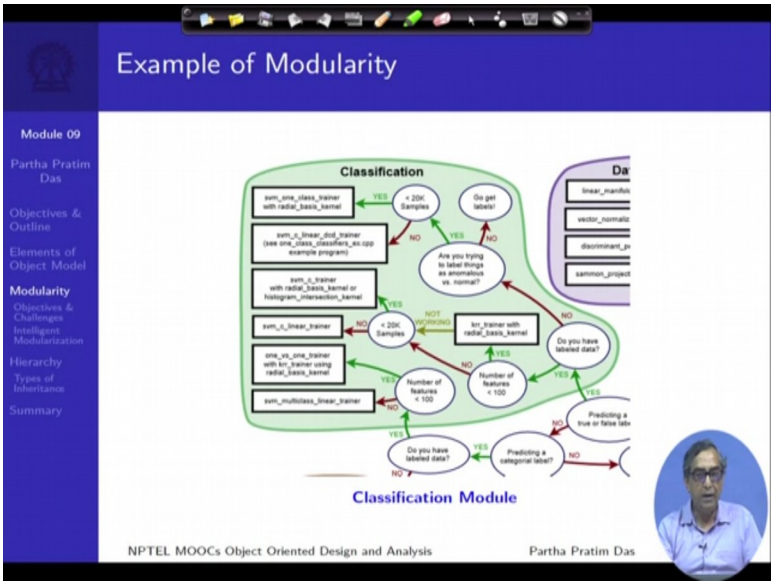
A second module or another module is structured prediction which talks about given a certain pattern of data what can you, how can you infer a predictable model for that, we will often use Markov random fields for that or different other ways of doing structured prediction. Again the specifics of this are in the domain of machine learning so it is not expected that you will understand it that is also not the requirement.

(Refer Slide Time: 05:25)



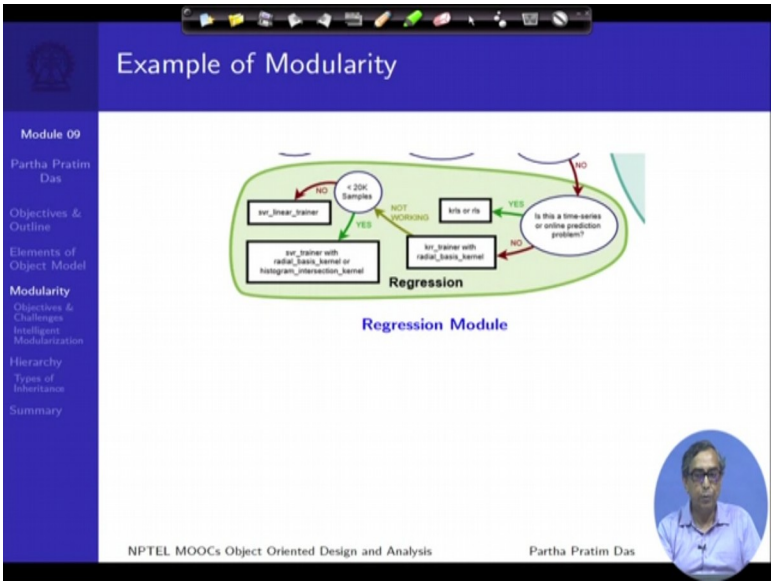
Am just trying to show that these are the different kinds of module that a complex system can have. We will have a separate module for clustering which when we do not know given the data how many different categories or sub categories or classes it belongs to, we perform clustering to decide on what is a good set of classes.

(Refer Slide Time: 05:46)



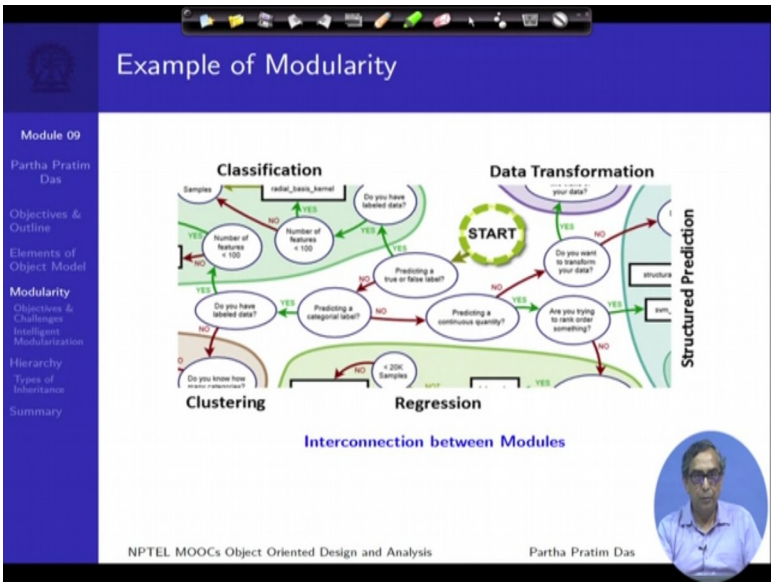
We will have a classification module which given the set of classes and given a test data can decide which of the classes it does belong to using different machine learning algorithms

(Refer Slide Time: 05:59)



And then there is a regression module which checks based on the different test cases as to whether it has been correctly classified or the classification has actually regressed in that there is some errors in the classification.

(Refer Slide Time: 06:14)



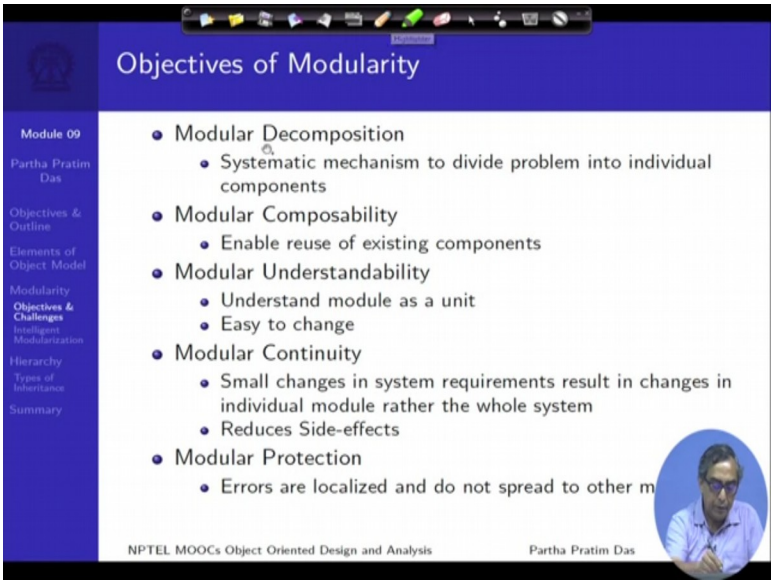
And finally, all these different modules actually interplay between themselves so this particular area if you see on different sides am showing the different modules that you that we have already shown, the data transformation, the structure prediction, the clustering, classification, regression, so all of these are the different modules that you have clubbed your code into but then this together becomes a system

when you have the interconnection between modules.

So, when we talk about modularity one is grouping, partitioning the program code into interrelated semantically interrelated set of a files and data. At the same time, there are certain interconnection parts for example here if you if you look into all these different subcomponents their task is to basically read the interactions between different modules and this will typically be the view of any complex system when it is modularized.

This was just to give you a glimpse of the glimpse of what kind of the complexity a system will actually involved and how modularization can reduce some of this complexity. So you can very well understand that if you think back about the original system we just looked at a clutter, compared to that when you take each and every module, it is much easier to handle that and this can again de go in an hierarchical manner, a module can be again sub modularized into smaller modules and so on. And we will look into some of those principles.

(Refer Slide Time: 08:01)



So, to set the target, there are certain specific objectives of modularization. So let us look into them one by one. One is the decomposition that is it is we want to give the whole set of codes files we want to organize them in terms of modules, in terms of related groups in a systematic manner so that what it meant by systematic manner is we only put together those part of the program, of those part of the code into one module which has some kind of well defined functionality.

We do not just do not just arbitrarily there are 100 program files; we will not arbitrarily put 17 of them

into 1 module. but we will put them together in a way so that together they either represent a set of concrete concepts, or they provide certain algorithmic functionality and so on. So modularity of decomposition is a key area. The second is composablity that is as we divide the system into smaller modules, every module into sub modules and so on.

We should be able to reconstruct it back that is you should be able to take smaller modules and put them together into a bigger module. So the way it will help is if we have similar functionality in two or more areas of the program, then we can have a single sub module which can be reused, composed into other modules and increase the reusability. So modularity significantly increases the reusability if you just want to think of in terms of conventional c programming that many of you may be familiar with.

You will find that we often use standard library like you know that c uses standard library okay. And in standard library what do you do? You have include different standard library headers you say this is we have stdio.h, we have # include<math.h>, we have #include<stdlib.h> and so on. now all of these header files, necessarily have a set of function headers because we are talking about c programming so there is no class but you can still see the explicit use of modularity, very organized use of modularity.

So what are we doing? Why have do we need to have all these different header files? Because I mean what could have been the other alternative, why could I have had a standard library.h include all the function headers, but we do not do that. So what we have done, we have done a specific decomposition and composablity. So when we talk about stdio.h we have put together all those function headers like printf headers, scanf header, fprintf header, fscanf header and so on which relates to some kind of input output mechanism.

But when we will have to deal with different mathematical operations like computing the tangent of an angle, the tan function or the half tan function, or the square root function, we will have a separate header file which keeps them together. So the way it this will easily demonstrate that while decomposing, all different standard header files of all different standard functions that you want to provide to a c programmer, you do not provide them as flag.

You decompose them into functionally congruent these are these actually never interact also between them. Printf does not interact with scanf, scanf does not interact with fprintf and certainly rtan does not interact with sin but you still conceptually put them together because with that it becomes much easier

for the user to use that module. So the composablity becomes easier, user must build up a program all that she needs to think of is am I doing a io here, then I must include stdio.h,

Am I doing something which is relating to say memory allocation and reallocation, I must use standardlib.h and so on so forth. So, this is just a simple example relating to basic programming that shows that modularity is a key requirement, is a key tool in terms of organizing better in terms of this providing good decomposition, good composability. And the consequence of that when you have been able to do this well then the other objective that you make is modular understandability.

So, as I said that if I want to do input output, all that I will need to do is include stdio.h. so this is the understandability that is I understand I have a semantic meaning for that module even though as I said the different components of that module different functions will not interact between them but there is a conceptual boundary which say that all input output operations would be here. So, I can understand the module as a kind of unit which makes it easy to use and which makes it also easy to change in case I need to make some changes.

The fourth objective is modular continuity. Modular continuity is little bit more in depth. What we want to do is we want to divide the system into such modules so that if in future we need to make some changes and changes can be because of different reasons for example the changes could be simple bug fixes that there are some issues in the code and I need to fix that, correct that. The changes could be due to change in specification; the changes could be due to changes in the business process and so on.

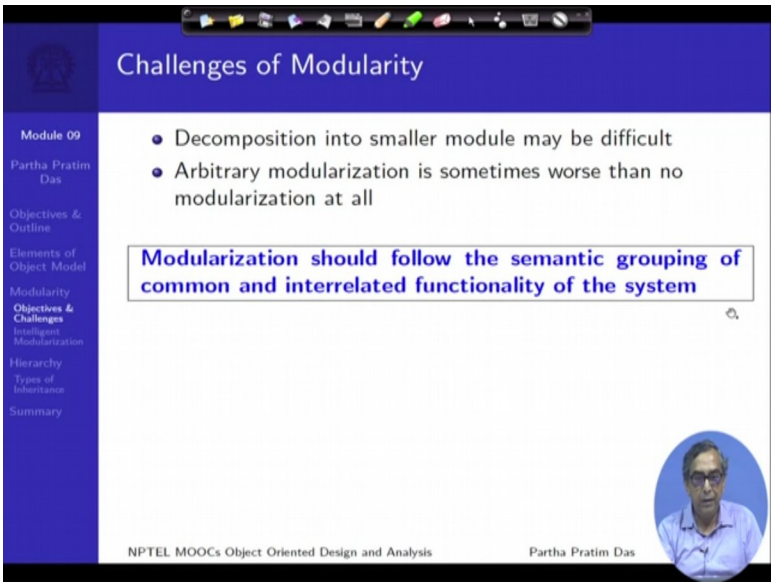
So I would like to we should be creating modules in a way so that when some changes required then that change should not go across multiple different modules, the change should be limited to one module or should be limited to very small number of modules so if the modularization is not very proper, then changes will spill across different modules. For example, if I had split the math.h set of errors into two different modules or if I would have clubbed math.h with stdio.h then any change in the numerical algorithm would compute different trigonometric functions which have spilled across modules.

So this will reduce in a way, this will reduce what is most important is it to reduce the side effect that is change of one module impacting the other and certainly related to that objective is modular protection which say that errors should be localized that is if I come across error say I come across a divide by

zero error then I should be localized to math.h. if I come across a memory allocation failure error, it will be limited to memory.h and so on.

So these are the typical objectives which should be kept in mind in performing a modularization of the code but performing the modularization of the design that you are given to deal with.

(Refer Slide Time: 15:46)



Now certainly in life no lunch is free so when you engage in modularization try to meet these objectives then you often come across challenges. for example, given a general system, it is not often easy to decompose it in terms of smaller modules often it needs quite in depth study of the system, understanding of that experience of design and often the use of other elements of object models like identifying what are the abstract concepts

What are the possible encapsulations even what is the hierarchy if something is existing and so on. so it is not always an easy, trivial task. This is not always just that you decide on which files that you can put in one module or the other. Even deciding which functions we put in different files, which headers you put in header files, which class definitions you put in different files, all these go in depth, in deciding how well the modularization is happening.

And keep in mind that if the modularization is arbitrary then often it turns out to be actually more disastrous it is often worse than not modularizing at all. Because you are drive bounded which are not aligned with the design boundaries of abstraction and encapsulation that you are always be emphasized. So modularization should follow the semantic grouping of common and interrelated

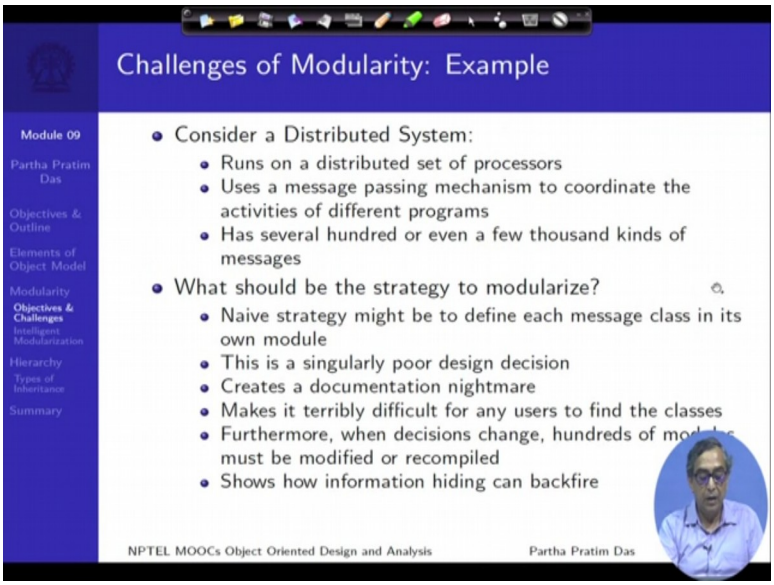
functionality.

So please note this points as it should follow the semantic grouping that meaning wise they should be similar they should be interrelated and what should be interrelated common items common functionalities and interrelated functionalities. Interrelated functionalities will make sure for example I will again go back the example of stdio.h, think about the printf function so it is an out input output function which c programmers would have use. So you have put it in the stdio.h.

Now there are functions which some of you may know some functions like read some functions like write which can also be used for input output but they are at a different level of detail. They do more like binary data read write and so on. now if you put printf into one module, one header file and put read write into a different then you will find that you will often have a lot of interaction across these two modules, because printf will never get implemented by itself.

Printf will regularly use read write operations, write operations will use other kinds of input functions, there are functions called input and so on. there are functions called fwrite and so on. so it is the commonality and interrelationship of functionality is a major tool by which you should be doing the modularization.

(Refer Slide Time: 19:00)



In terms of challenges I have just tried to describe some more systems, you can consider a distributed system which has many processor and you have several messages going across, these are processors, hundreds of different kinds of messages. Another question is what should you modularize.

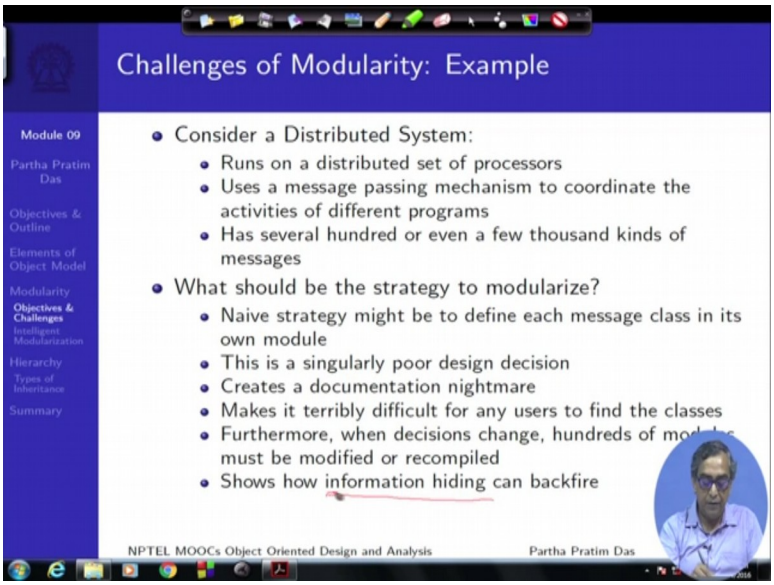
Should you modularize and since every module must be as independent as possible you will put every module required message types, within that module, you can do that.

But if you do that then often you will have quite a poor design because now since every module has its own message structure a user who has to deal with these messages will not be able to deal with them without knowing in which module what structure exist. Documentation would become difficult; it will become really difficult to maintain. So there are certain aspects like when you are putting them all together then certainly you are kind of adhering to better encapsulation because you said that well this kind of message relates to this module

So, let me put it inside that, so you are hiding everything. But too much of hiding could actually impact the modularization lot more, it could backfire because now you do not know what are the different messages that need to go across, what are their formats and so on. so you need to strike a balance possibly strike a balance by doing a different modularization where you could identify that well since there are several kinds of messages, so why not create a module which actually deals with messages

And provide interfaces which different modules can use based on what kind of messages they need. We will see more of that but this is just a glimpse of what could be the possible challenges that could come across.

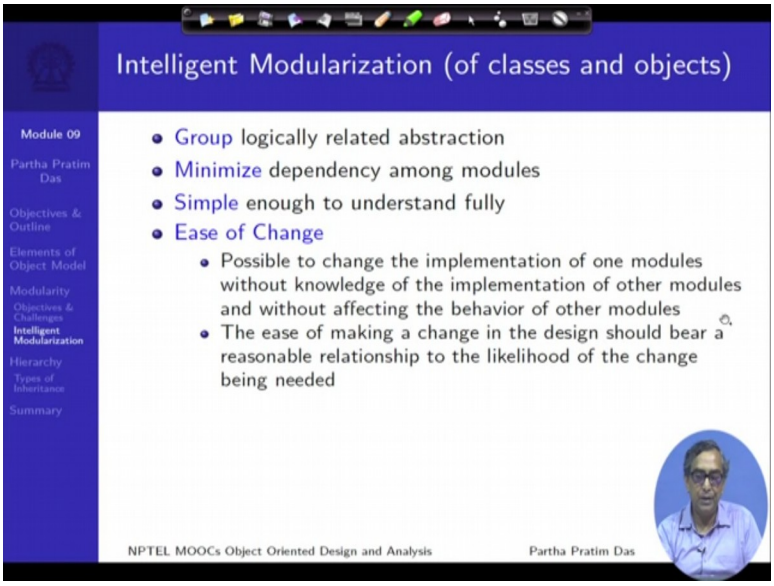
(Refer Slide Time: 20:55)



So, in this background to keep through the objectives of modularization we often talked about intelligent modularization that is these are some of the thumb rules rather or some of the basic

guidelines that you may follow to achieve the objectives better is one is

(Refer Slide Time: 21:13)



That is why I have just highlighted the key words in this, am sorry this got changed. the highlight keywords that is grouped logically related abstraction. You already have abstraction on the system so in a module you should typically now naturally one extreme is every abstraction you put into a different module. That may give you a lot more fractured modularization which will have lot of interactions so your modules will be simple but you will have so much of interactions,

You just recall the example I showed in the machine learning system that the interaction part has to be also manageable complexity. So instead of just putting every abstraction into a separate module you will try to group them into logically related abstractions. So abstractions of input output operations should lead to one module, abstraction of mathematical modeling should lead to another module, abstraction of system related operations should give you stdlib.h and so on.

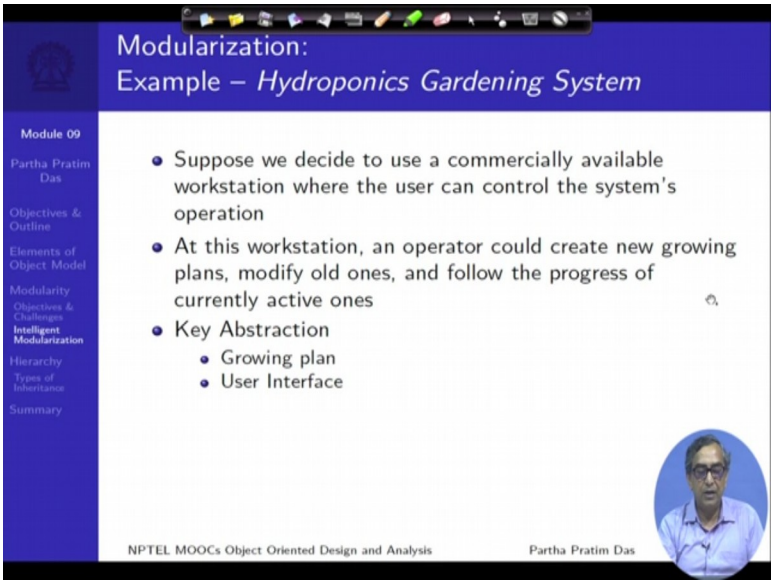
And as you do that then what you try to achieve is minimize dependency amongst modules. So this is should be now clear that if I logically group logical abstractions into module more then certainly I will need to go across modules much less only when I go across modules only when I need to achieve a higher level of functionality. And both of these logical groupings and minimization of dependency across modules will make every module simple.

enough to understand and that would be a key idea simplicity should be a key idea of modularization because recall that we are why are we modularizing because we want to contain the complexity. The

complexity is too high and we want to make all we are doing all these to make things simpler leveraging the notions, the vocabulary of the problem domain I will frequently go back to that leveraging the abstraction that I have already created, leveraging the possible encapsulation that we have.

And finally, the modules must be complete. Final but override of the intelligent modularization principles is ease of change even though whenever we do a design when we start doing a implementations that organizing the code, we are confident that we are doing a correct job but the experience shows that for different limitations of the human mind of the system and of the business scenario change is an inevitable, unavoidable reality of software development of system design and implementation.

So ease of change must be always kept in mind that is we should organize the modules intelligently enough so that some changes in one module should impact least number of other modules and so on. (Refer Slide Time: 24:30)



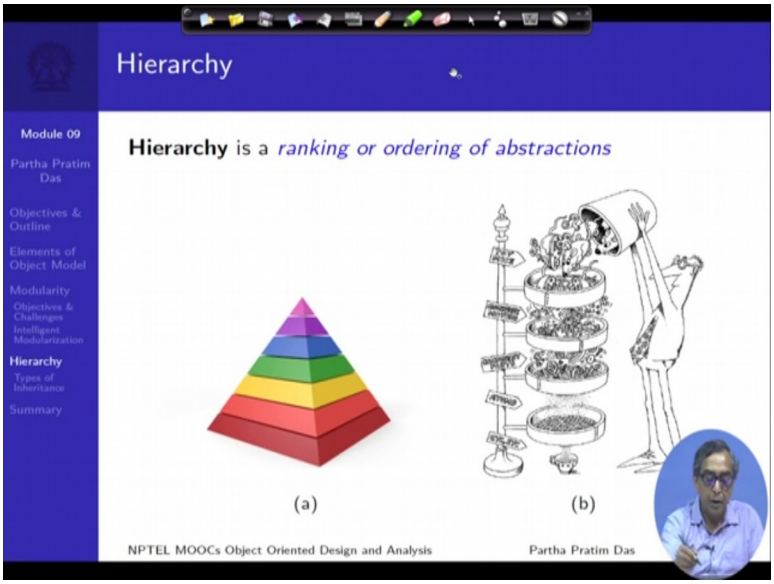
So, with these principles we will regularly go ahead and look at different modules, there are some references to the earlier examples, the hypotonic gardening system so suppose we want to build up an automation system which we started discussing and certainly some of the key abstractions that we have seen include growing plant that is these are different strategies to be imbed by the system to grow different varieties of plants.

The user interface so these are some of the key abstractions that you would already have from the

analysis of the abstraction of the system. So I will be quite logical to create modules for growing plan, so in the growing plan module, you may have a growing plan for fruits, a growing plan for grains and so on. it will be possibly good to put this all together and this is what will relate to have grouping the interrelated abstractions together.

It will relate to simplicity, it will make sure that you have much easier to change because a fruitgrowing plan change in fruit growing plan will impact possibly the general growing plans strategies and so on. on the other hand if all the user interface functionalities that you have it will certainly be better to put them together into a single separate module.

(Refer Slide Time: 25:57)



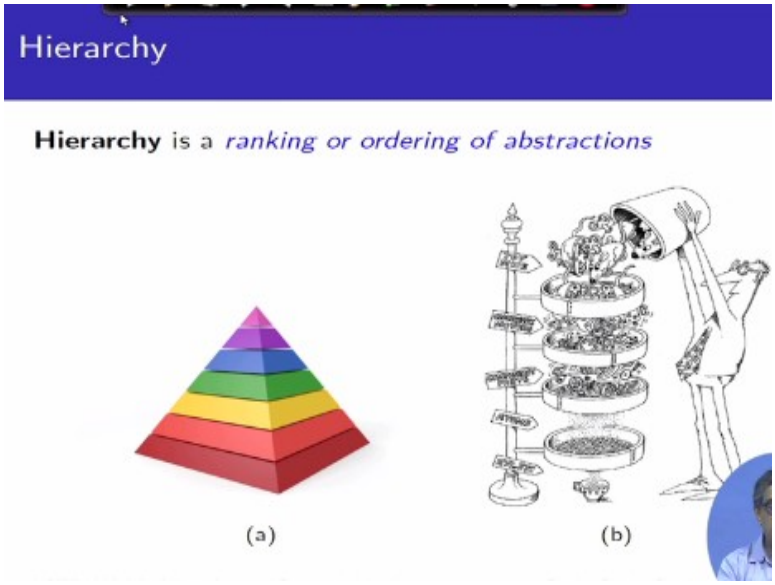
So, with this what we have seen is we have seen the introduction of modularity as a major element for object models. We will stop here and next take up the hierarchy.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture-15
Elements of the Object Model (Major): Modularity and Hierarchy (Contd.)

Welcome back to module 9 of object oriented analysis and design. We have been discussing about elements of object model, last module had discussed about abstraction and encapsulation and in the earlier part of this model, we have talked about modularity and different aspects of what a modularity should try to achieve and what are the basic principles of intelligent modularisation.

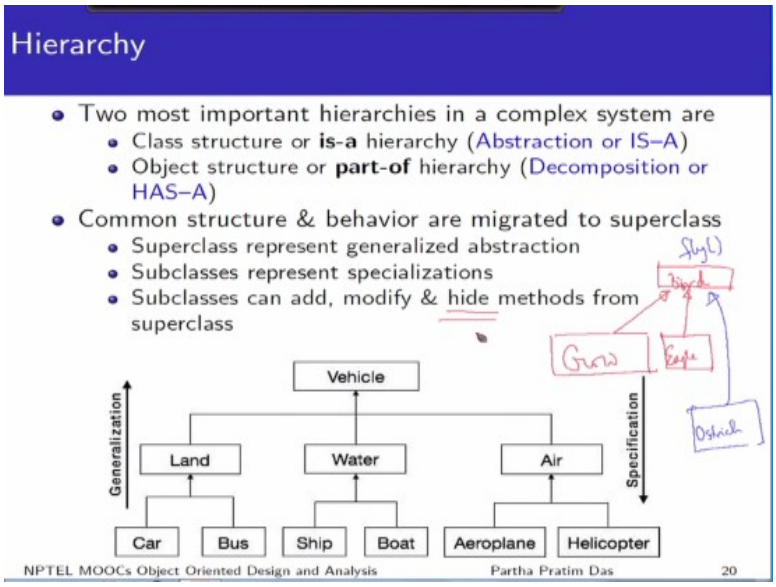
(Refer Slide Time: 00:51)



Next we take up the last of the 4 major elements that of hierarchy. Hierarchy is a ranking or a ordering of abstractions so you would have seen a glimpse of this in module 8 when we talked about abstractions and we said that abstractions could be hierarchal as well, so you got initiated to that, but hierarchy by its own requirement is eligible to be in other important element for object models and is described as such.

So it is a ranking of abstraction so which mean that the different abstraction that the system has that goes from details to less details to, even less details to absolutely top levels so it is kind of in an artistic view, it will give you a pyramid like structure to work with. So if you are at the top of the pyramid, then you get to see only very simple the whole thing in very simple terms, you use the abstractions only at that level.

(Refer Slide Time: 02:09)



Then as you go down, you see more details as you go down you see further details and different kinds of hierarchies play role and let us take a look into those. What we are talking about here again you have already had some glimpses of it, so when we talked about specifically the canonical form of complex systems. We exposed you to 2 work through orthogonal hierarchies that are common most important for complex system.

The hierarchy of class structures which is typically called as abstraction hierarchy or IS-A hierarchy and the hierarchy of object structure which is called the part of or decomposition hierarchy or HAS-A hierarchy. The class structure hierarchy shows you how over the hierarchy the common property is a share between one layer to the next and the object structure show you a hierarchy of containment.

How one object can be thought of having been composed of different components or different component objects and this in turn can be thought of in terms of sub components and so on. So in this context, the common structure and behaviour are migrated to what is known as super class, so what we say is in a hierarchy, a class that occurs higher in the hierarchy, hierarchy is a typically drawn in terms of inverted freeze.

So that classes or the abstractions which occurred at a higher level in the tree and known as super class and the related lower level abstraction is called as sub class. So if we look into the diagram below, this is we say is a root abstraction, so at this root, a system is being defined in

terms of one single concept vehicle, so if you say a vehicle all of us understand something. So that could be lot of different as you say specialisation.

But vehicle we can say is combines all of them together by one unified concepts that it is a mechanised gadget to transport. A vehicle is a mechanised gadget to transport. Therefore, if this is the general concept then we can always say that vehicles at least are of, can be of free types that which transports on land, the land vehicles which transports over water, water vehicles and which transports through air, air vehicles.

There could be other kinds of vehicles also like vehicles going for space, vehicles which can go over more than one medium and so on we will not go to those kind of complexity but when we say this, this is what is a concept hierarchy or abstraction hierarchy, so this is what we say is a super class and let me use the different colour, these are the subclasses, now please understand that the super class and sub class are relative notions.

So vehicle is a superclass, relative to land, water or air vehicle, but if I just to focus specifically on the water vehicles, let me pick up red again, if I specifically focus on the water vehicles, then the ships, the boats, they are the subclasses of this water vehicle. So this now becomes the superclass, this is the subclass. So it is in the hierarchy, if 2 concepts are related in terms of generalisation and specialisation then we will use the term superclass for more abstract or more generalised concept or abstraction.

We will use subclass to represent specialised more specific class or more specific abstraction. Now as we go for this hierarchy, then on 2 sides, there are 2 directions showing that certainly if you go from below to lower level to a higher level, then you are actually generalising that you are losing special properties so as you go from cart to land, you lose the property that cart is far limited transportation.

Transportation of limited number of people or limited volume of goods, whereas bus necessarily is for transportation of large number of people. The commonality that you can out here, as you go up is both of them run on land and similarly you have certain commonality with water and when you go to vehicle you lose even more, you lose even the fact that on which medium the vehicle transports, you just retained the generalisation that the vehicle indeed transports.

So this is what would keep on happening, so when we go the other way, that is when we go from generalisation to specialisation, then a subclass can add, modify or hide methods from the superclass. So vehicle says that I can transport. When I come to water vehicles, then it will add that I can transport of course, I have got that from the super class vehicle but we will add that I can float.

This is not the requirement of vehicles in general but water vehicles will have a requirement of float. I will add different other factors like I can possibly go with much less loss of power because water provides far less resistance than land. So I have more and more properties coming in here and then when we go for the down, then ship has everything that water does, water vehicle has but it has something more.

It says that it is a big water vehicle which is typically motorized and so on. Whereas when I talk about boat, then I will typically have water vehicles which are manually driven with will have oars and all that ships will usually, at least these days will never have oars, they are driven by underwater propellants and so on, so as we go along the specialisation from top to bottom, we can add and modify different methods.

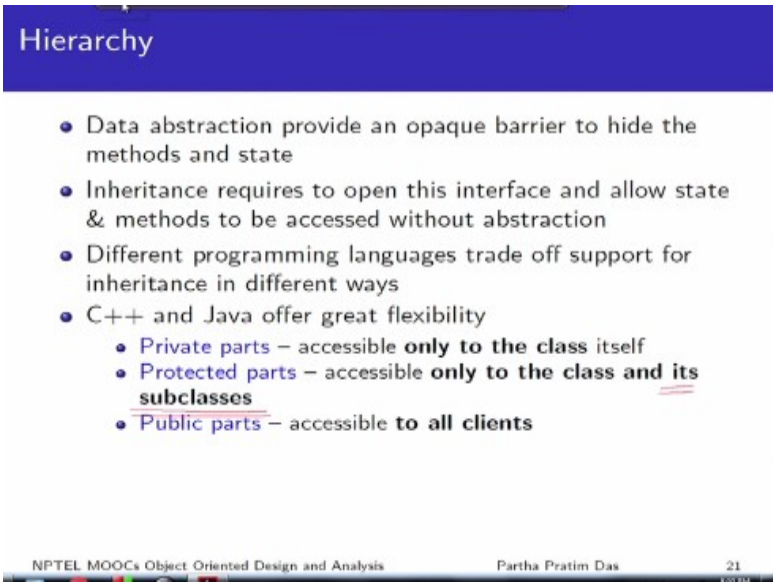
In fact, there could be instances where we main into hide a method. For example, I will just by the side show you an example, suppose I am talking about a concept bird, so I can say that crow is a bird, because anything the bird can do, the crow can do that, may be crow can do something more, for example crow can creates a kind of sound in a certain way. I may have eagle, is a bird, it can do anything that the bird can, but it can do something more.

It can go to really very high heights but let us now look at that I want to put here say ostrich, all of us know ostrich is a bird, we know it is a bird. But we also know that a method or a property after being a bird is to be able to fly, ostrich we know cannot fly so we get into certain kind of contradiction that our hierarchy rule say that a subclass will get all the methods, it can add new methods, it can change certain methods.

But it will get all the methods at the superclass has. The superclass has a method called fly, birds can fly, crow inherits that, crow gets that, crow can fly, eagle gets that, eagle can fly, eagle can add other method that it can glide which crow cannot but I come to ostrich, it

cannot fly, so this is the specific example to show that on a hierarchy, we my often need not only to add new properties to specialise.

(Refer Slide Time: 11:27)



Not only to, may be change the way the method works, change the way the birds fly, they change the way birds reproduce and so on but you may need to actually omit certain methods which has superclass has, so you need to look at all of add, modify and hide methods from super classes. So hierarchy basically provides data abstraction and I mean hierarchy provides on top of the data abstraction it tries to provide a mechanism by which the properties can propagate from one layer to the other.

We can see that once we have done abstraction so we have already done 3 major elements so we know that here is abstraction, so I have abstracted a concept then we know on that abstraction where putting encapsulation, so moment we have put in encapsulation, we have put an opaque barrier to hide the methods in the state, we said implementation is not feasible. Because it is encapsulated. Now I want to say that now I am coming and saying that here, I have vehicle.

It can transport and then I have land vehicle then I have car and so on. Now if I have done my abstraction and encapsulation in the right way then this is encapsulated. The properties of vehicle are encapsulated. Then how does land vehicle define how transport happens? Because vehicles in general cannot say how transport will happen, it is just an abstract concept of transportation, land vehicle has to do transportation using certain physical laws.

All vehicles need to do transportation using certain physical laws, land vehicles use friction or the absence (13:12) of friction, gravity and all that, water vehicles use buoyancy, third law, etc. Now so there is an inherent contradiction in terms of the way we have abstract and encapsulated when we want to create hierarchies, where we want properties methods to propagate from a superclass to the subclass. Now this is something which is specifically handled in different ways by different implementation languages, different programming languages.

Particularly, one of the reasons that C++ or Java gets very popular in terms of object oriented programming is a fact that these languages allow you a great flexibility to provide mechanisms by which you can honoured the abstraction and encapsulation, at the same time, create hierarchies, so though we are not going to discuss in depth about the language constructs in this course.

But it will be good to understand that there are concepts like visibility which say that you can, of your encapsulation you can create different grades of encapsulation, different porosity of encapsulation, certain encapsulation are known as private which are totally encapsulated only if you are inside that class, only if you are inside that abstraction, you will able to use those properties, use those methods.

The other extreme is basically to provide the exposing part of the encapsulation to honour that contractual interfaces that you wanted to provide which is a public part, the public visibility which anybody and everybody can have a look at. Now these 2 do not solve your hierarchy problems, So C++, Java these kind of languages introduce a very interesting kind of intermediate visibility or intermediate encapsulation granularity.

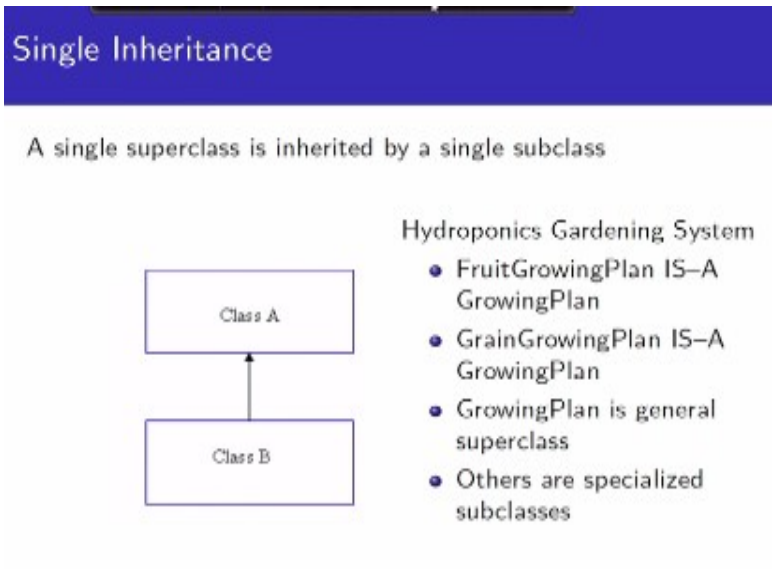
Where you say that you are different implementation parts are protected, by which they are available to the class as the private ones are, as the public ones are as well, but it is specifically available to this is the main part of protected, it is available to the subclasses, but these are not available to public, so protected parts do not become part of the contractual interface.

They are just an extension of the implementation of the structure, but they can be propagated along the hierarchy to make that realisation of hierarchy is possible, so it is kind of, little bit

started you know my great in between the conceptual aspects of object oriented analysis and design and slowly taking you closer to how will it will look in the implementation, so I just included this part to be discussed as a part of the hierarchy to give you a glimpse that

It is not that everything in this whole paradigm of things are fits into together exactly seamlessly, certain concepts which we need to impose like hierarchy, which is very critical to manage complexity contradict some of the other assumptions you made for abstraction and encapsulation, but this is just to highlight that do not get worried because the languages when you finally come to system implementation, the languages will be able to take care of that.

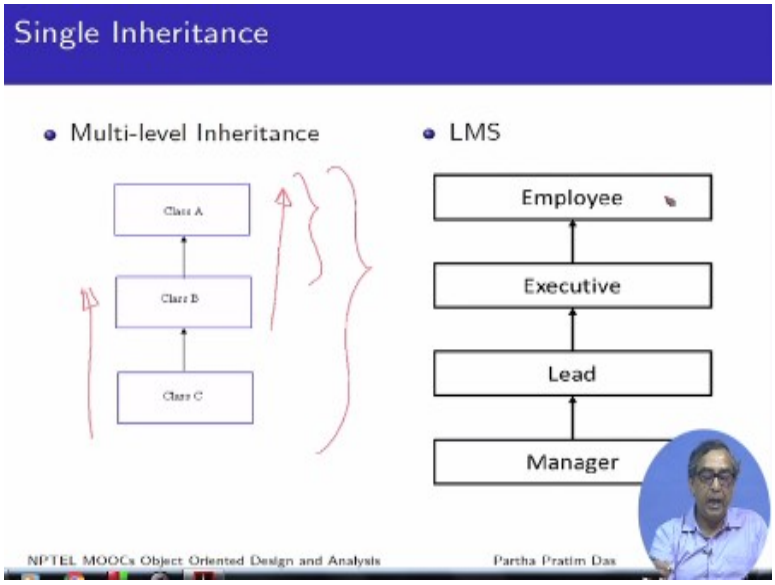
(Refer Slide Time: 17:17)



We will see when we talk about UML that we translate all these different elements of the objects models into UML language, we will able to express this varied kind of visibility and porosity of encapsulation over the UML hierarchies. Now in terms of this there are these are more like of terms you need to understand and remember. On a hierarchy we say the term single inheritance is referred to when there is one superclass and

There is a specialised subclass, so we will say, [No video or audio from 17:41 to 19:47] here in a single inheritance, we will say that this is a single super class and this the above on we will write it as B is a A. So that is the simple presentation so that is called the single inheritance and if we just refer to the Hydroponic gardening system then fruit growing plan IS-A growing plan.

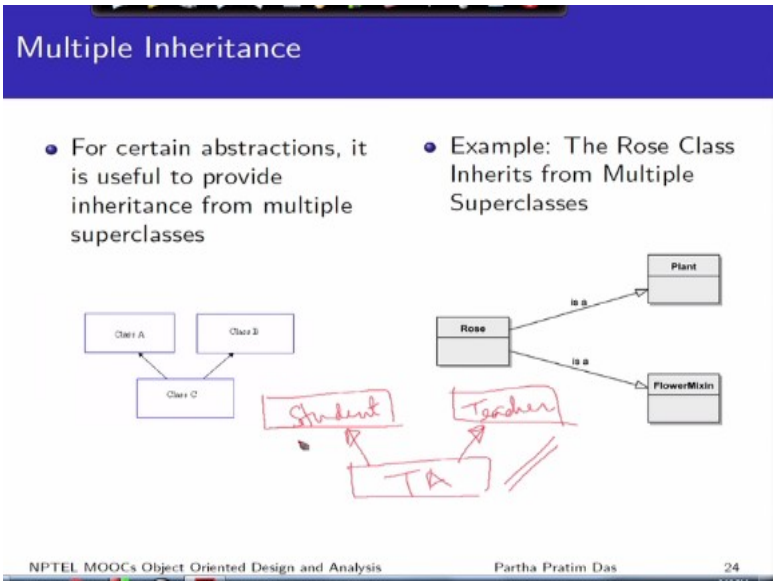
(Refer Slide Time: 20:24)



So these are all specialisations and these are specialisations and these are generalisations and growing plan may be the most general super class that it will have. Single inheritance can also be extended it multiple levels, so here we have the same, we have B is a A and then we have C is a B, we have already seen instances of this in terms of the vehicle example, so when this happens then we say it is multilevel inheritance.

That is inheritance is happening not between just 2 levels but it is more than 2 levels, so a manager is a lead LMS, leave management system a manager is a lead, can do anything that the lead can do, lead is an executive, the same property is be an executive as an employee, so employee happens to be the most generalised super class and manager happens to be the most specialised subclass in the whole hierarchy of multilevel inheritance.

(Refer Slide Time: 21:14)



A different concept come in terms of multiple inheritance, in multiple inheritance, you basically have more than one super classes which is jointly specialised into common subclass so whenever that happens we say that we have multiple inheritance because when this will happen then C, the class C will enjoy the properties, methods of class A and as well as of class B and multiple inheritance is a regular reality of art life.

Multiple inheritance come from the different roles that we keep on playing all the time, for example, let us think about just I have been regularly, I am getting regularly assisted in this course by 3 teaching assistance (TAs) Himadri, Srijoni and Tanwi and so think about the classes of students and teachers and TS and what will happen, we will have students, what is the distinguishing property of a student? Why it is a key abstraction? Because, student study.

We have teachers, what is the distinguishing characteristics of the teachers, they should also study but distinguishing characteristics of the teachers, they should also study, but distinguishing character is a teach. What is a TA? The TA has to teach because she is a teaching assistant but to be a teaching assistant you need to be a student, so a teaching assistant a TA is kind of specialises from, it gets certain attributes certain properties from the students, certain properties from the teacher.

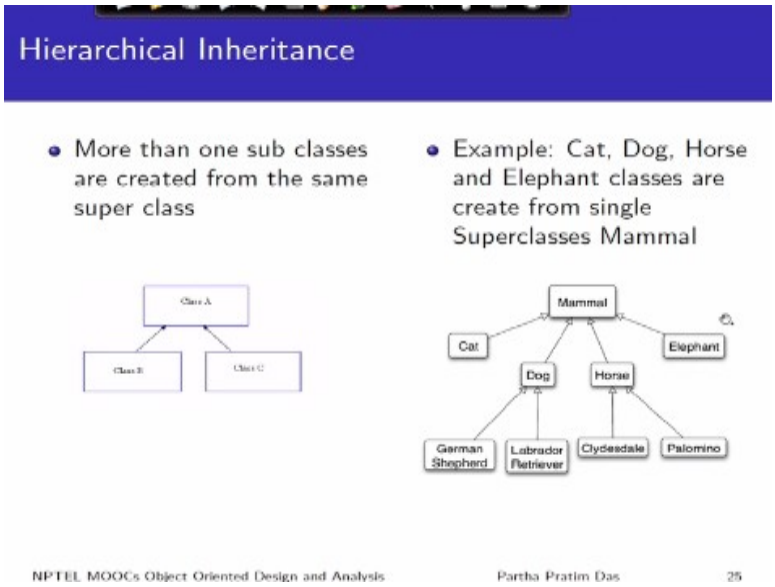
So this is one very direct example of multiple inheritance we will need and if you think little bit more, you will find that you will need all different kinds of properties of rather power of subclass, subclass TA to add, modify and hide methods. For example, TA can assistant the codes, TA can evaluate assignments but TA cannot award the final grade, which teachers can, so if I say TA is a teacher as I am saying here.

Then TA will inherit the method to be able to award grade to students which it has to hide so but at the same time it will from the student it will get a lot of properties which will possibly, it will possibly will be able to enjoy altogether, so this is a basic context of multiple inheritance it is a, I would like to warn you this there is an another different example here which you can make out by yourself.

But I would like to warn you multiple inheritance is one of the very difficult areas of design because is not only easy to identify multiple inheritance there are lot of fundamental theoretical conceptual issues in dealing with multiple inheritance. Some of them relate to that

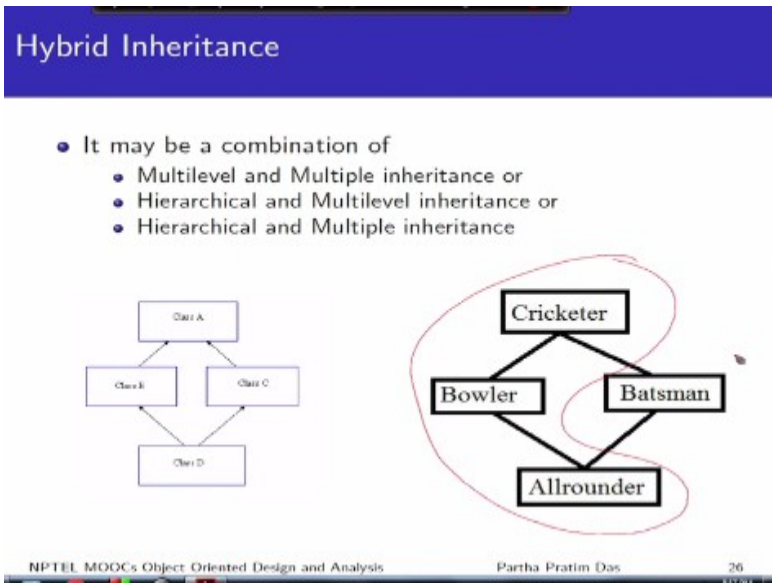
if you inherit the same method the same behaviour from 2 different super classes then how should you behave in the sub class and so on.

(Refer Slide Time: 24:53)



So there are different design principles relating to in terms of dealing with multiple inheritance and we will talk about some of those when the time of actual email modelling comes. Then next is hierarchal inheritance, which basically is when you have more than one subclass specialising from one common super classes which is what we have already seen quite a lot in terms of the vehicle example we just took.

(Refer Slide Time: 25:28)



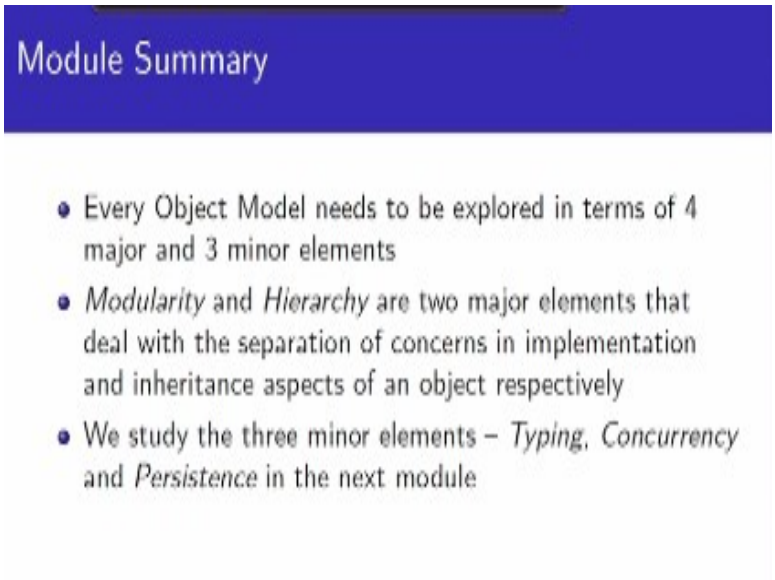
Here I have shown another example based on mammals so this mammal is a most generalised super class then we have cat, dog, horse, elephant all different subclasses, they in turn have other subclasses so when we have this kind of inheritance structure we say we have

hierarchal inheritance, now in reality what happens is, you will not have, often you will not have pure single inheritance or single multilevel inheritance or hierarchal inheritance or multiple inheritance.

You will typically have a mixture of them and therefore often you will have a situation of hybrid inheritance as you can see here, so in this hybrid inheritance, you can see here you have a more pure form of hierarchical inheritance, here you have a more pure form of multiple inheritance and if you look into this part of the structure then you have multilevel inheritance.

So hybrid could mix up different kinds of inheritance structure in this and the reason in object oriented design someone equipped with that needs to understand the hierarchies and these inherited structure is this is one great tool which go a long way to restrict complexity of systems because as you can understand that once a subclass can inherit a whole of a superclass with very minimal changes additions and you know hiding then you have a great opportunity for use of code.

(Refer Slide Time: 27:04)



We have great point as to a how should modulearise you have a great use of the whole notion of abstraction that you have been working with. So a major consequence of hierarchy is these different forms of inheritance that it results. So to summarise in this module, we conclude on the major elements of object models, we had 4 of them, 2 were done earlier and here we have talked about modularity and hierarchy.

Two of the other major elements that deal with modularity certainly has deals with the separation of concern, the better separation of concern, you can do in terms of code organisation, we have better modularity in implementation and we talked about several principles for that and hierarchy deals with the inheritance aspect of an object and these 2 together will go a long way supporting with abstraction and encapsulation in doing good design of object oriented systems.

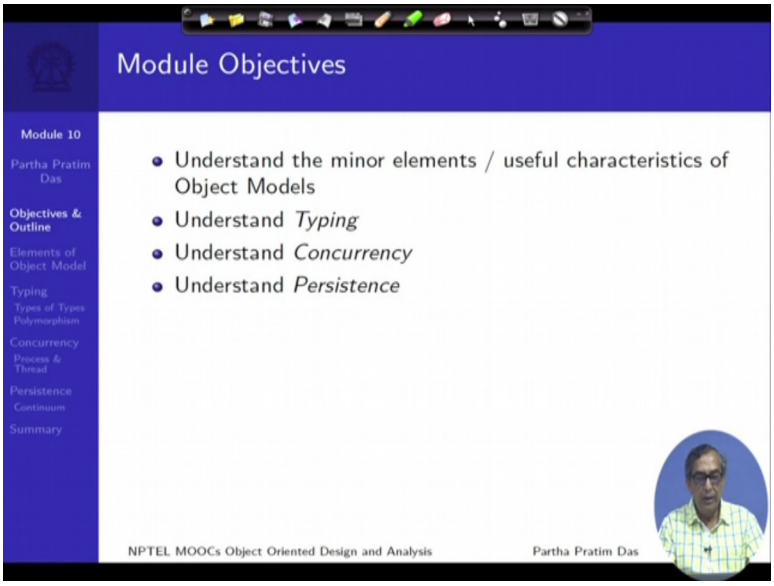
Now in the next module we will take up the minor elements which are not essential but often turn out to be very useful elements to provide the actual related functionality of the system that will be in module 10.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 16
Elements of the Object Model (Minor): Typing, Concurrency and Persistence

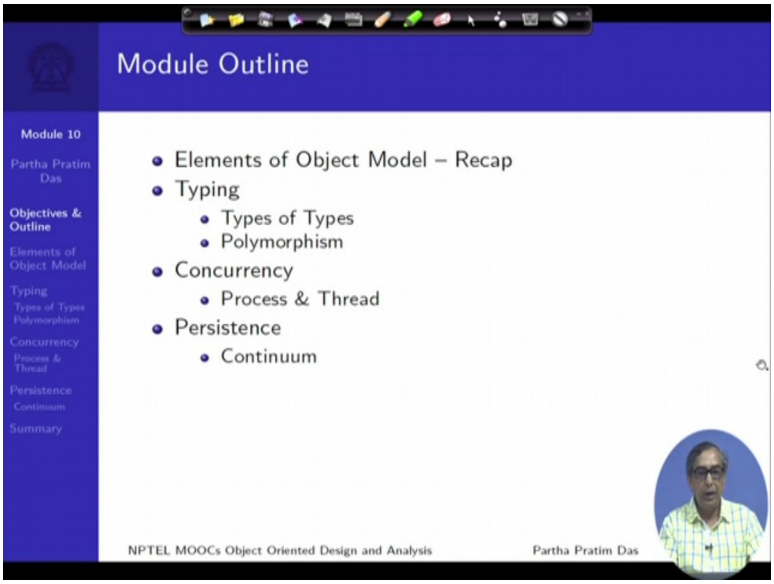
Welcome to module 10 of object-oriented analysis and design. In the last modules 8 and 9 we have been discussing about major elements of object models. We have seen that object models to be designed need to consider 4 major aspects of abstraction, encapsulation, modularity and hierarchy. They relate both to the aspects of design in the system when we talk about abstraction and hierarchy and they also strongly relate to how the implementation is done and how the code and different modules are organized.

(Refer Slide Time: 01:14)



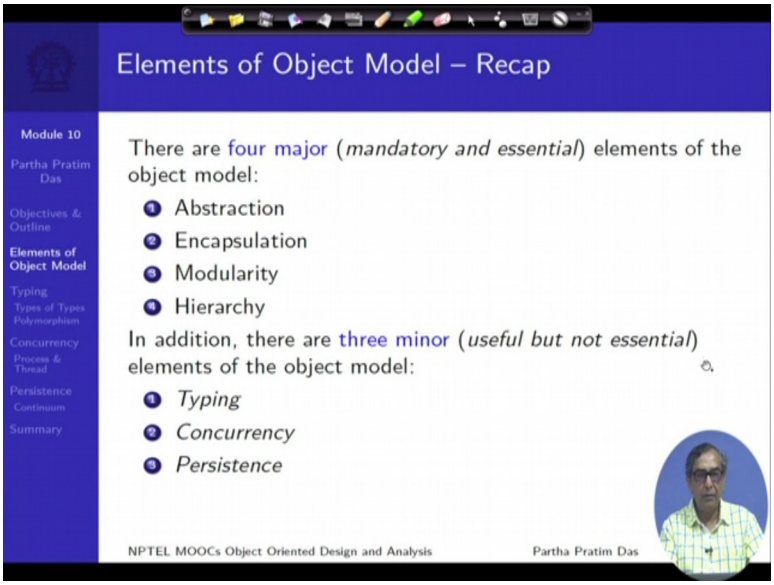
We continue on that and would like to conclude today on the elements of objects models and today we will discuss specifically about 3 minor elements these are useful elements in an object model but these are not essential depending on certain system requirements and the skill of the designer these elements will be considered. This will include typing, concurrency and persistence.

(Refer Slide Time: 01:41)



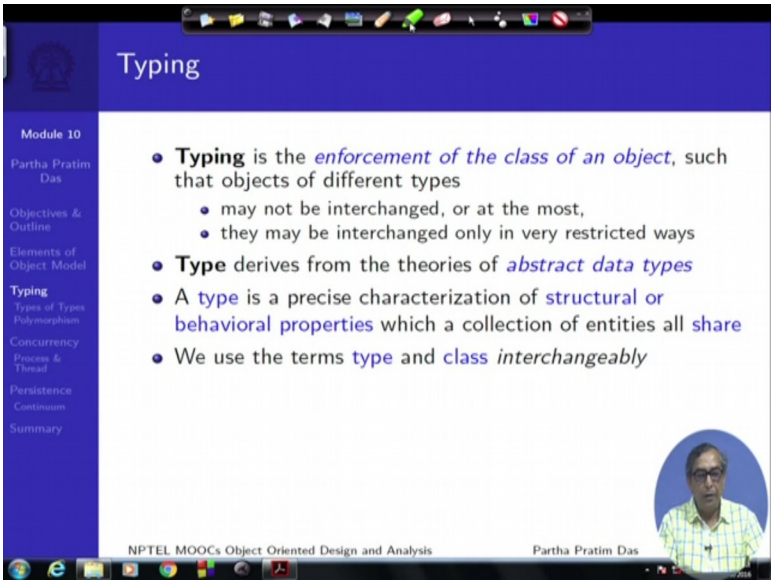
This is the module outline. This will be available on the left of every slide.

(Refer Slide Time: 01:49)



So, we have just this is just the context of the elements of the object model. Just for your recap, we have done the top 4 already. We are doing the bottom 3.

(Refer Slide Time: 02:00)



So, the first element minor element that we deal with is called typing. Am sure all of you you have done some bit of programming in C or C++ or java or python. You would know that whenever you need to use a value or a lateral or whenever you need to use a variable there are certain types associated with it. So, if I say am talking about a variable x then I many not by default know what kind of value does it contain

Or if I talk about 2.3 I need to understand that whether it is a floating-point number, how many bits it is represented in and so on. and things really get complex when we talk about objects which is not simple primitive types of values but they are composition of they are collection of several independent value items which we say are properties. So, type is the core concept which defines how should an object behave. So, in formal terms the typing is a enforcement of the class of an object.

So, typing says that if I know the class of the object or if I know the type of the object then it will make sure that that notion or the conditions or constraints of that class of that type will be enforced whenever the object is used. So, it may not be interchanged for different object types and if it is interchanged occasionally in some restricted ways then it will follow certain pre-defined rules. what in simple terms it means that if I have 2 variables in C say x representing an integer variable which will contain an integer value

And p representing a pointing to character which will probably mean a C string then we cannot use them usually in interchangeable manner. I can use the integer value to add another integer value to it. But if I have a pointer I cannot take another pointer and add these 2 pointers. But again, I can take an

integer and add to a pointer so all these different restrictions of behavior will be derived and imposed based on the type of the object.

So just to give you the background and we will not have enough time to talk about in depth. Type basically derives from the theories of ADTs or abstract data types. A this is a kind of development into programming systems which happen quite before the object-oriented paradigm became dominant which abstract data types have to say that if we are talking about any type then you basically have 3 different components that you need to deal with.

So, for an ADT you talk about a domain of values, you talk about operations and you talk about what is known as axioms. So, domain say the whole set of values that a variable of a certain type can take. So, if am talking about say a 1-byte unsigned integer in the domain is the set of values going from 0 through 255. Operations say what can I do with that value. So, we know if we are talking about integers, then I can add the integer, I can multiply integer and so on.

But when I add the operation has very specific meaning. For example, if I am working only with one-byte unsigned integer then if the result of addition of two such integers happen to be more than 255 then certainly my domain cannot support that. I cannot hold that value and we kind of say that we have an overflow of the values wrap around and if I add 1 with 256, 255 that is the largest possible value in 1 byte unsigned then the result is not 256 but the result becomes 0.

So, this is these are the kind of different operations which are defined and then finally axioms say that how these different operations interplay between themselves. For example, if I add one to an integer such kind of unsigned 1-bit byte integer, if I add 1 to it and then if I subtract 1 form it then I should get that the original result. So, an axiom could be of the form that x + 1 - 1 should give me x. now if you will you will understand that the immediate example I talked about if am adding 255 with 1 that is if the value of x is 1, I is 255, then x + 1 will become 0 by the use of overflow.

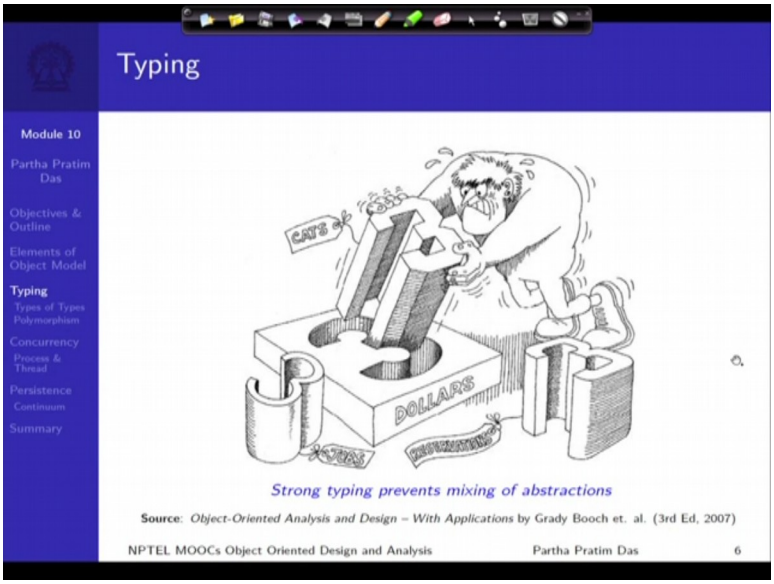
Now when I subtract 1 from 0 the value should become 255 by the underflow rule. So, this is the overflow rule, this will affect the underflow rule unless these rules are symmetric; this action will not work good. It may be that the overflow is allowed but underflow does not change the value. Then this axiom will be failed. So abstract data typing is the notion which tells you more on theoretical formal terms as how should a type operate and over the years this has been used in several designs of several

programming languages.

But it has become less and less directly usable for the programmers because we have evolved the mechanism of object oriented programming where the use of the abstract data types or typing gives an underline principles underline promises on which the all these axioms are ensured and you do not really need to explicitly know or talk about that. So, given that the type is precise characterization of the structural and behavioral properties we have seen about these properties, that means basically the attributes.

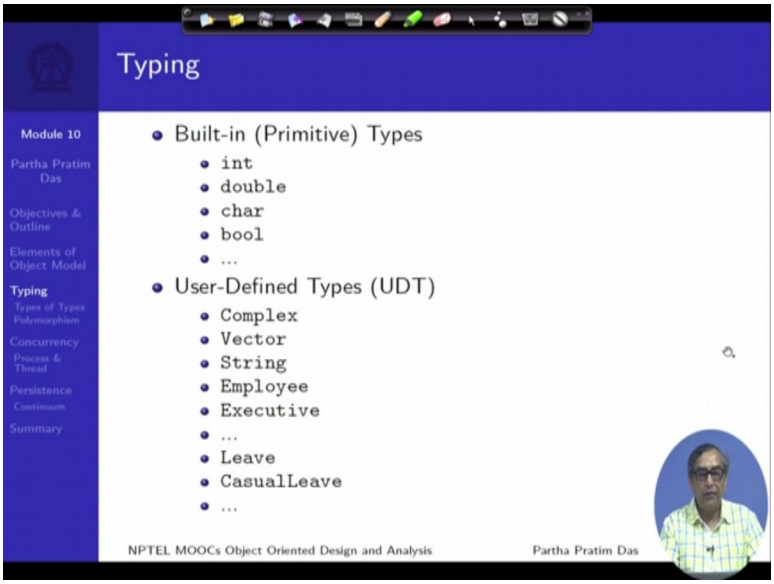
And the methods and it will talk about those properties which all entities of that type will share and just to clarify that we have been using the notion of class in the same context in the current course, we will use the type and class interchangeably though please remain informed that in general in the programming language theory or object-oriented theory, they are similar but not exactly the same but we will treat them as same in this course.

(Refer Slide Time: 09:18)



So, this is that very nice cartoon from our OOAD book, we can see that if type mismatches, both of them are 3 but this person is trying hard to fit one shape of 3 into another. so that's what happens when your types mismatch between variables and when we try to do something with that.

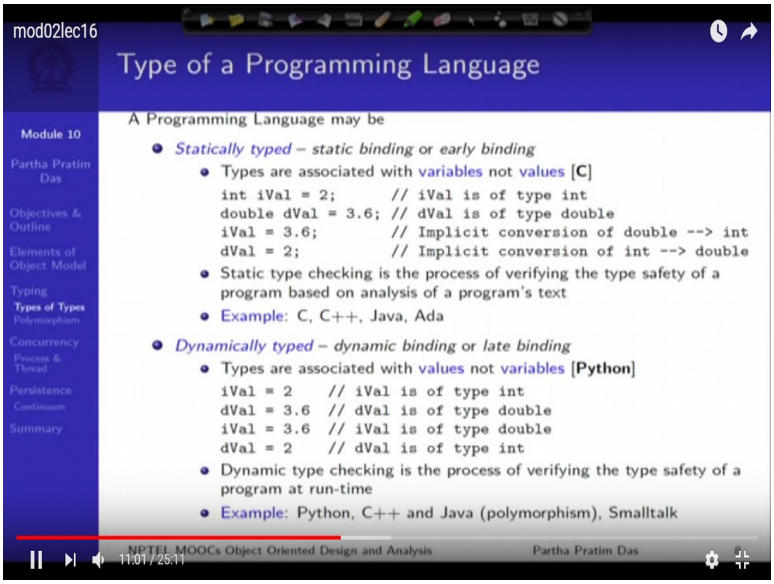
(Refer Slide Time: 09:37)



Certainly, several different kinds of types we have been exposed with significantly if we are coming from C and as you move on to C++ or in java, there are a number of built-in types or primitives' types which are given in the language and then we have certain builders given in the language like struct in C or struct or union in C or in C++ you have struct, union, class these different kinds which will allow us to define new user defined types these are called udts

And these are some examples of udts for example in C++ you can make a complex class of complex numbers which will certainly have different kind of algorithms for addition than the algorithm you use for adding 2 integers. It will have also new operations like taking the modulus of a complex number and so on, we can vector types, we can have string types and if we refer back to our LMS example these are these basically are possible from our LMS example that we could define types or class for executives, for employee, for lead, for leave, for causal leave and so on. so, these are different kinds of types that we will read out.

(Refer Slide Time: 10:56)



Now we need, we would like to take a look into the type of a programming language. This particular heading has a little bit of fun what use of the word type means kind of programming languages or the data type of programming language you can inter predict either way so typically programming languages are classified into 2 categories as strongly statically typed or dynamically typed. A statically typed language use static binding or early binding.

A dynamically typed binding uses dynamic binding or late binding. In exact simple terms, what it means is just focus into this these 2 statements. When we are talking about type, certainly we have the context is of a variable and a value that resides in the variable. If my language is statically typed, then the type is associated with the variable. So, you say that the variable has a type and that you define at the time of writing the program and the compiler while compiling determines THAT this is the type for that variable and in every context will enforce that type.

So, the type is associated with the variable, it happens at the compile time that is the reason you call it the so these are the process of verify types of your program is based on the analysis of the program's text that is the compiler is working, no data is there and that therefore it is statically typed. So, if I look into this simple example of C, you all would know if I define 2 variables ival of integer type, dval of the double type then the type is associated with these variables.

So, when I start doing certain things that I should probably not doing, I take a double value and try to put to ival, the type of the variable is still hold on to be of integer. So, what it does the compiler to make this assignment possible, it could have done 2 things. 1 is it could have disallowed it. Some

language will exactly disallow that or some languages like C or C++ will allow implicit conversion. It will take 3.6 as a double value knowing that ival is int will make a conversion which in this case will be a truncation

3.6 the point 6 part will be just thrown away and 3 will actually get assigned to the ival. Similar thing will happen if I do the other thing 2 will become 2.0 when assign to dval. In contrast if use a dynamically typed language the nature of the code is very similar. Its only that the code is written in python and the important commission is all these preceding predefined types for the variable names. Now we do not provide any. So, what do we say is the type now is associated with the value?

So, it is not determined at the time of compilation. It is determined at the time of execution at the dynamic timing. So, you look at the value the value being 2, you infer that the value is of an int type and since you are assigning that to ival, so you infer that this is of int type and since you are making this assignment you infer this int type on this ival. So, you say ival is int. but at a later point of time, say in here these are these are all sequence of instructions in python.

So, at a later point of time, I choose to assign 3.6, 3.6 naturally has a type which is double or floating point. So, what will happen when I make this assignment this type is put to ival. So ival becomes a double type variable and this is not restricted to primitive types alone. This will also extend to the different user defined types that python allows that different python classes. So, the type of a variable in python is the type of the value that was last assigned to that variable and that type is maintained.

So, the that is the difference in in these 2 statements here in static case, it is associated with the variable; in the dynamic case, it is associated with the values. So that's the basic difference between dynamic and static and that gives rise to a lot of consequences because if you can statically type a language, then the advantage that you get is you can add compile time, you can say that well this probable is wrong this probably is a wrong assignment or this probably is a wrong invocation.

In contrast if its dynamically typed you cannot say that but when we are going to use different kinds of objects moving around, different kinds of objects on different hierarchies if you recall the vehicle hierarchy we talked of earlier, a vehicle could be land, water, air. A land vehicle could be car or bus. Now if I just get an instance of a vehicle I may not exactly know whether it is a car or a ship or it is an aeroplane or tin what it actually is. So, it may be very difficult to conclude at the static time to decide

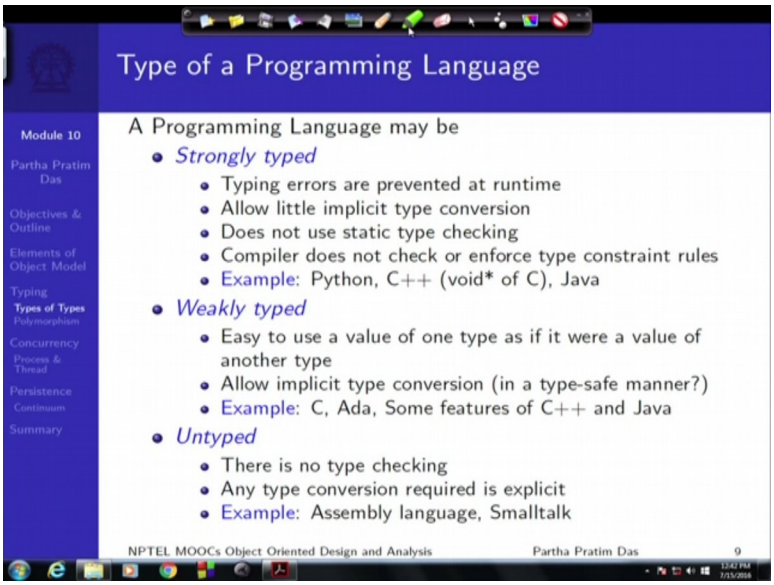
what kind of operations I should allow.

If it is a car, I should ask for how many wheels it has. But if it is a ship I should allow how many different parts does its propeller have and so on. so, there is always a trade of between whether I want to do static typing for better program safety or I want to do dynamic typing for better flexibility and more object orientation. So, what happens if we try to look into the more common languages that we intend to use, C is certainly strong statically typed.

Everything is done as a static time so in C++ and java but C++ and java also support part of dynamic typing through a mechanism called polymorphism and as will see that will become a very strong handle to support object orientation. So, C++ java as a language is crossly statically typed but they also support some dynamic typing whereas if you look at languages like python they are only dynamically typed. There is no static typing in python.

So, this is one way to look at the programming languages and am trying to relate this minor typing element of object models to the language characteristic because when you finally at the end of the design when you choose your programming language, the OOP programming language and want to convert your design your classes, the hierarchies, encapsulation into that language code, you will need to understand what kind of support you can get from the language.

(Refer Slide Time: 18:19)



There is another kind of typing or classification of programming languages that's commonly studied this is between strongly typed, weakly typed and un typed. A strongly typed language say that anything

that you use that must have a type and that type must be a very strict one. So, typing errors will get prevented at runtime because you only allow things to happen if the type is completely known and can be completely reduced. You do not allow or allow very little implicit conversion

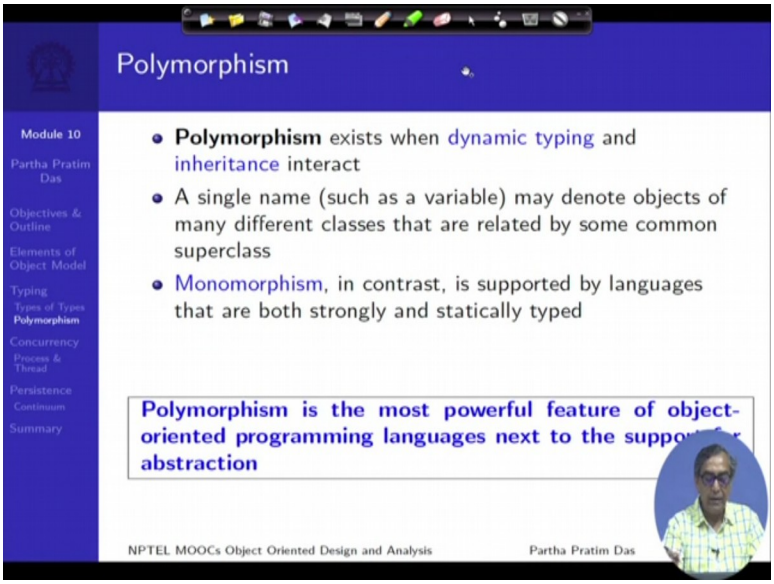
So, you would not end up allowing double to be used in place of an int and so on. thereby making sure that you are always using the value of a correct type in the place of a correct usage and so certainly strong typing often cannot use the static typing mechanisms because statically you may not know the type of a value all the time. So, as it turns out that languages like python have very strong type. C also is a principle after strong typing but as you understand C subsumes am sorry C++ have strong typing and but C++ also subsumes the whole of C which is weakly typed.

It has plenty of scope for implicit conversion, it explicitly has provided a void star kind of pointer that is pointer with no type to take care of different specific programming situations. So those actually make C++ also weakly typed where you can do things that you could do in C. but if you make a guideline if you make a policy at your personal level or at an organizational level that you will only use the pure form of C++, the object oriented and strongly typed part of C++.

You will still be able to do all the programming but of course you will need some skill as to use that and you will have to drop all the weakly typed paradigms that C offers you by default. So, these are the 2 basic different categories and of course there is a third category of programming languages which are untyped that is where you do not need to you do not specify the type of a value ever till you use it, at the time of using it you will say that this is the type

And so, the type significantly does not remain in the program. It remains in the mind of the programmer. So, these are more difficult languages to program in and we will not expect typically an object-oriented system to be coded in this. For example, is any kind of assembly language if for all those who know assembly languages will understand if you are working in assembly languages you are directly dealing with bit patterns and therefore there is no specific type which is known.

(Refer Slide Time: 21:44)



So, but this is am just including these Un typed languages as a for completeness we will typically not make a reference to this. So, given these different kinds of classification of programming languages so just we had strongly typed, weakly typed languages and we have statically typed and dynamically typed languages and there are different combinations of that, the language can be statically typed and strongly typed, statically typed and weakly typed and so on.

And C++ falls in kind of a mixed boundary where as I said mostly it is statically typed, for polymorphism it gets it is dynamically typed. Mostly it is strongly typed but if you use pure C construct within C++ then you will trade into the weakly typed territory of C++. Another concept which you just need to start getting aware of in context of typing is known as polymorphism. polymorphism as the name suggest basically they has two parts here,

Poly is one part one word it says many, morph means change so polymorphism is basically the mechanism of working with change. So, it exists when dynamic typing interplay, interact with inheritance. It talked about inheritance as a part of hierarchy just recall we talked about single inheritance, we talked about multilevel inheritance and so on. so, when you would like to code those kind of object models into language, we will need the inheritance to be present in the object oriented language.

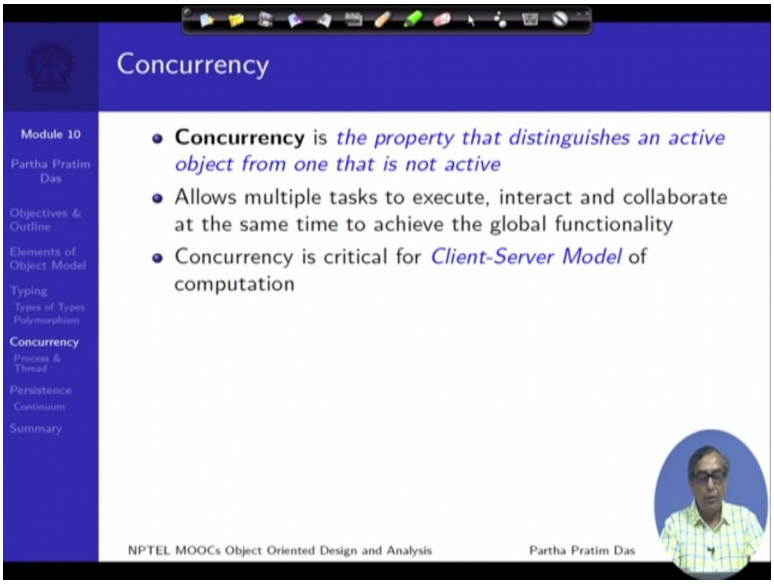
But along with that if you also have dynamic typing that is on that inheritance model if you do not know on the time of compiling what a specific object is, which class does it belong to, then I need to resolve that dynamically then you will need to make use of dynamic typing and when you combine

these 2, you have polymorphism. C++ naturally gives a very strong framework for polymorphism and so does java so that they become very good vehicles for object oriented programming.

There are languages which are monomorphic in nature which does not allow such behavior to be encouraged. They are strongly and statically typed languages. So, to conclude on this part, please remember that polymorphism is the most powerful feature of object oriented programming next to abstraction. So, if we if you are now faced with the question of what are the 2 top features that an object programming language must provide so that you can realize all the object models easily in them.

It will be the number 1, could be abstraction which you have already studied and now you are getting introduced to polymorphism which is typing at the runtime on an inheritance hierarchy. We will all of you may not fully understand what am talking of here. But do not get worried as we go along this course you will talk about these polymorphism, polymorphic behavior and the and its use in the object-oriented analysis and design.

(Refer Slide Time: 25:07)



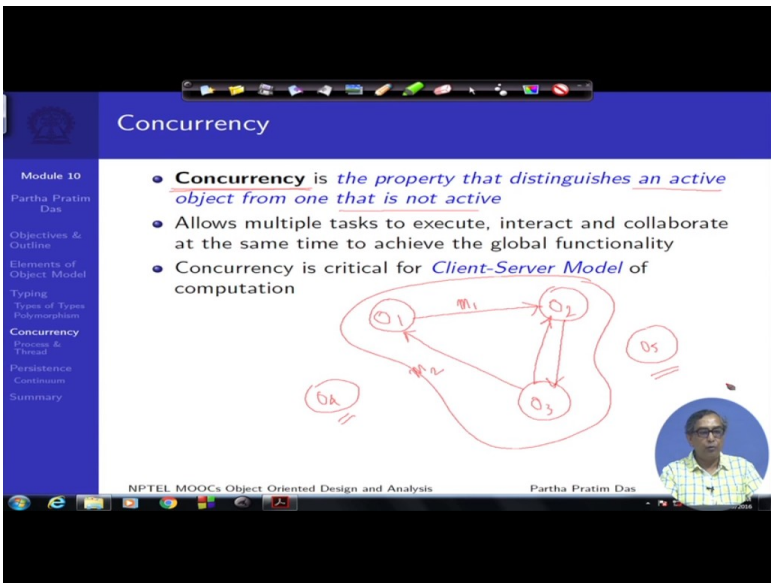
Further you want to talk about this concurrency. we will take a break and talk about that.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 17
Elements of the Object Model (Minor): Typing, Concurrency and Persistence (Contd.)

Welcome back to module 10 of object-oriented analysis and design. We have been talking about minor elements of object models. We have already discussed about the aspects of typing strongly and weakly type languages, statically and dynamically type languages and we have talked about polymorphism. Next we talk about the second element which is concurrency.

(Refer Slide Time: 00:52)

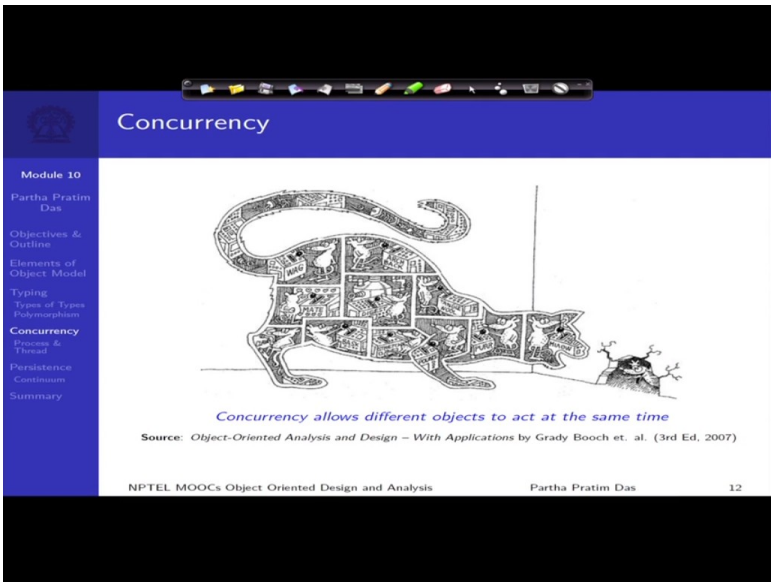


Concurrency is as the name suggests is relates to 2 or more things happening together in the same time. So if I have one object o1, if I have another object o2, I have another object o3 and these are between themselves sending different messages and expecting to get service we have seen in this model. Then certainly you cannot have a sequential or well ordered structure to do this. You want all of these to happen as and when they need to happen.

So while o1 say is sending a message m1 to o2 to get a service at the same time, it is possibly getting message m2 from o3 to provide a service. Both of these have to happen at the same time. Now this is known as a concurrent behavior. And at the same instance it is also possible that I have object like o4, I have objects like o5, what is sitting idle, who are not neither sending messages nor receiving messages. So they are not actually doing anything.

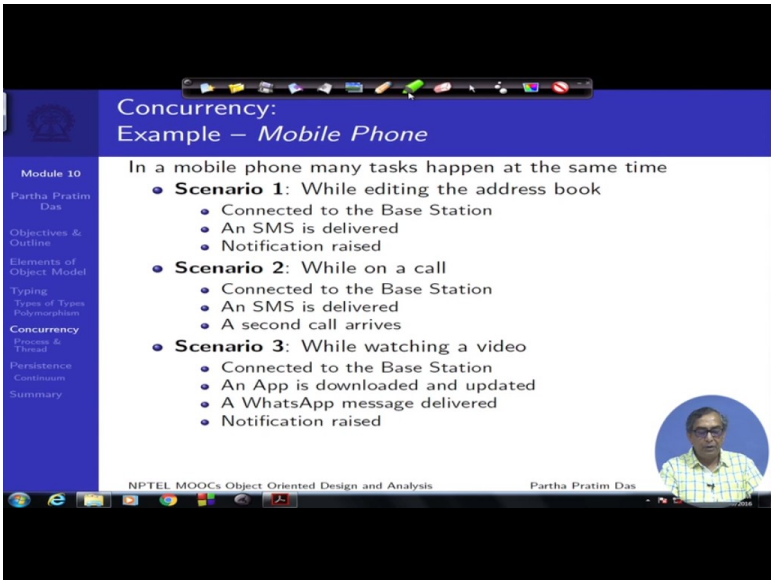
So the concurrency is a property that distinguishes an active object from an inactive one. These are the inactive ones and these are the active objects which are doing something or the other at the same span of time. Certainly you can understand that without concurrency the abstract client server model that I talked of earlier has a basic computing model of object by systems would not at all work. Because I cannot have an ordering of exactly first step 1 has to happen then step 2 has to happen message 1, message 2, message 7, message 6. I will not be able to we will not be able to define this single temporal order for things to happen.

(Refer Slide Time: 03:13)



So concurrency is a very necessary part of the object model of the typical systems that we work with and this is the cartoonist's view of what happens inside the body of a cat is wagging the tail and digesting the food at the same time, trying to hunt something for itself.

(Refer Slide Time: 03:28)



Coming to more mandate example which am sure would be closer to your heart all of you am sure use mobile phones in different aspects. So here am just talking about different scenarios. For example you are in your mobile phone suppose you are editing your address book; you are adding a new friend. Now while you doing this apparently your focus is on editing the address but at the same time, the phone is doing a lot of other things. For example at that same time it has to remain connected to the base station.

I connect being connected to the base station is commonly said and my phone does not have signal, my phone does have signal. That is a kind of visual indication we get. What it means that it is being there are land station call base stations to which my phone is currently communicating and is connected and all the time when a our phone is live, when a our phone has towers, it basically is connected to the base station so it cannot it is not acceptable for us that while I am editing my address book, my phone does not remain connected.

So every regularly it is sending certain information to the base station, it is getting some information from the base station again sending information, getting information. So this task and this task has to happen concurrently and while this is going on, while am addressing changing the address book, I suddenly receive an sms. So the delivery of the sms happens when the phone is connected, when I am editing the address book during that time itself the sms get delivered

and as the sms is getting delivered, it is being put to my sd card a notification pops up by the side of my address book screen to tell me that a sms has come. So that notification raise has to happen. So if you look into all of these, they will need to happen concurrently, there is no specific ordering strictly

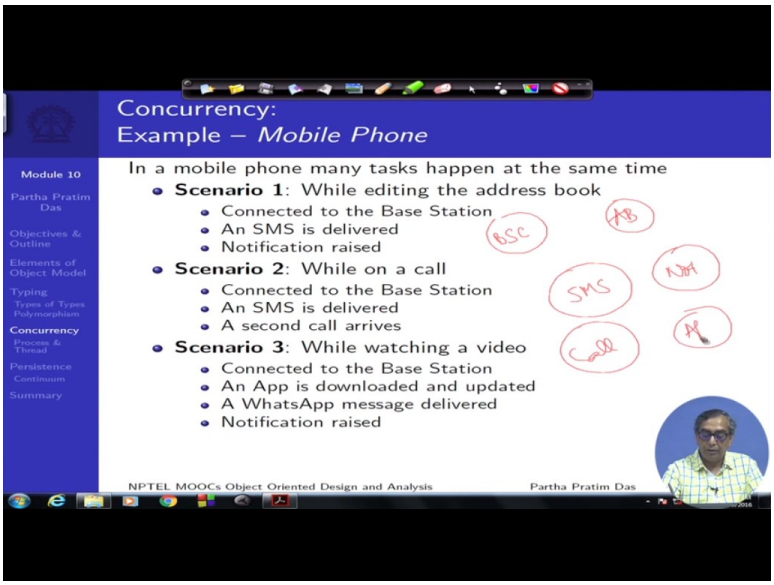
between all of them. Of course sms being delivered will be followed by notification then there is an ordering there. But there is no specific ordering in terms of how much I add it in the address book.

And the sms being delivered and the connection of the phone. In a different scenario failure (05:49) we are in a call the phone certainly needs to remain connected to the base station, so this is the call, this is the task, the sms can get delivered and in fact a second call may arrive. So I get to see that a call waiting is happening. So the ongoing call and the waiting call have to happen in the same time space. So this needs concurrent behavior.

You are watching a video in your mobile phone, you are still connected to the base station, so while you are watching the video, without disturbing you an app is getting downloaded and updated, installed in your phone. Your WhatsApp message is getting delivered and you get notification for that is raised. All of that is happening in the same time space. There could be you if you start just thinking of different instances of your use of the mobile phone.

You will at for every kind of scenario you will identify 4, 5, 10, 20 different things that need to happen, the task that need to happen at the same time and without a predefined order, a strict predefined order for things to work properly.

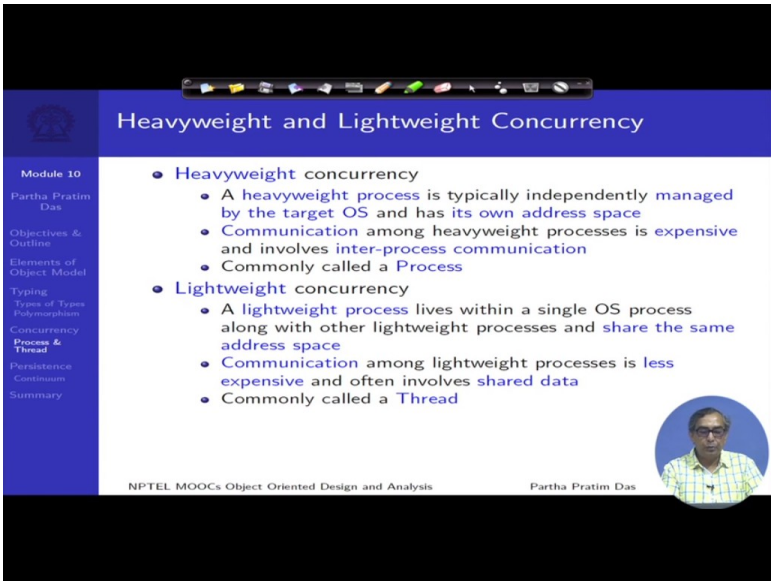
(Refer Slide Time: 07:20)



So that is a basic significance of having concurrency in object model. So if you think about all of these address, if I think about say address book is an object, if I think about base station connected is an object, if I think about sms is an object, if I think about notification is an object, if I think about call as

an object, if I think about app as an object and so on.. They all will need to interact, exist and work at the same time domain so that is the basic notion of concurrency that we need to work with.

(Refer Slide Time: 07:43)



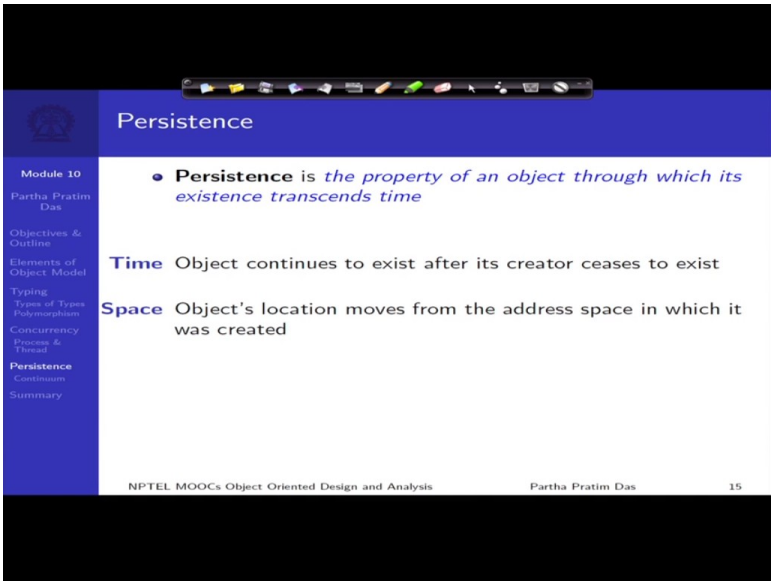
Certainly coming to the technical terms, concurrency basically relates to processes in our computing system and we often talk about two kinds of concurrencies more frequently, one is known as the heavyweight concurrency or which is commonly called a process. The other is known a lightweight concurrency which is commonly a called a thread. A heavyweight concurrency or process is an entity that the os deals with and provide its own address space.

In contrast, a lightweight concurrency happens within the within a single os process which is a heavyweight concurrency item but it shares the address space that the heavyweight process or heavyweight concurrency of the process as it does not get a separate address space. It shares that with other lightweight processes and so on. The communication typically is expensive between processes or heavyweight concurrency items it is usually through a shared memory in terms of lightweight concurrency items or threads.

So these different kinds of concurrency models has to work with the way I want the objects to interact is so naturally objects which belong to the same module, objects which are closely integrated should work concurrently using threads. Whereas very different modules which are completely independent, which have very little functional overlap or very little code overlap would possibly work as independent processes because they need to handle very different independent resources of the same system.

So as you go to the refinement of the design in terms of the object model, you might need to look into these concurrency aspects of the object models.

(Refer Slide Time: 09:46)



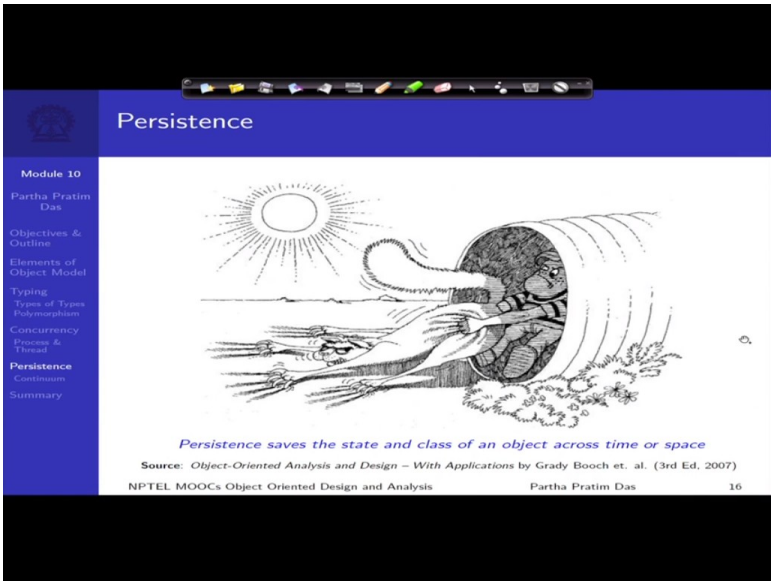
We move on to the last minor element of object model that is called persistence. Persistence basically talked about talks about 2 major access. Access of time and access of space. We have been talking about objects all the time. Now finally when I realize in a programming system what is an object. An object will be represented by a contiguous sequence of memory locations, a collection of bits together. So there is always a beginning of an object, an object gets created by some other object because everything is object.

Once it is created, it occupies certain memory space and its stays for a certain period of time, till it is destroyed. Now in terms of time and space, many things can happen, for example in object may continue to exist even after its creator ceases to exist. Somebody created that object. But that whoever created that creating object has been destroyed but the created object will stay. How long it will remain is an independent factor in the design. Similarly an objects location may move, a object may have been created in this memory space memory space of thread 1.

Here I got an object this is my conceptually this is my object and this is where it is got created in terms of memory. I may move it to some other say thread 3 at a different location and say now this object has I want a blue. Now I say that this is where this object has gone. This is a is a same object but I may change its location so the space and time of an object can go through different kinds of transformation

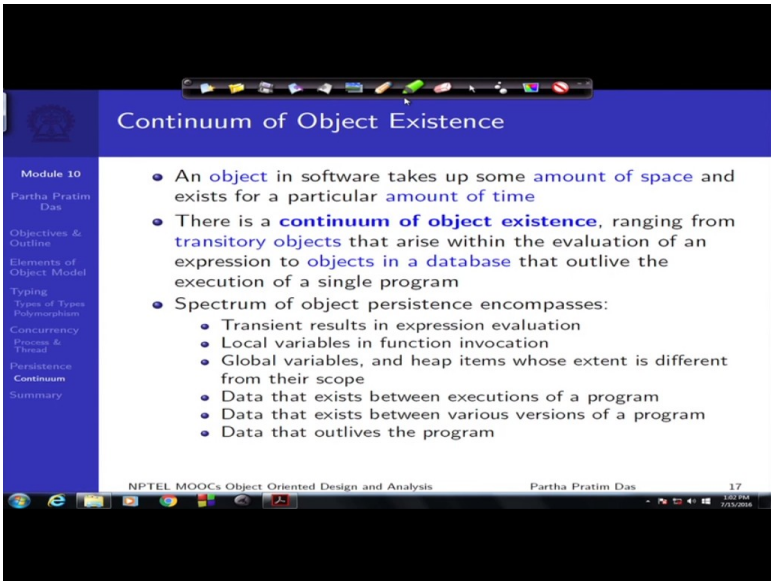
and how those happen is addressed in the issue of persistence.

(Refer Slide Time: 12:07)



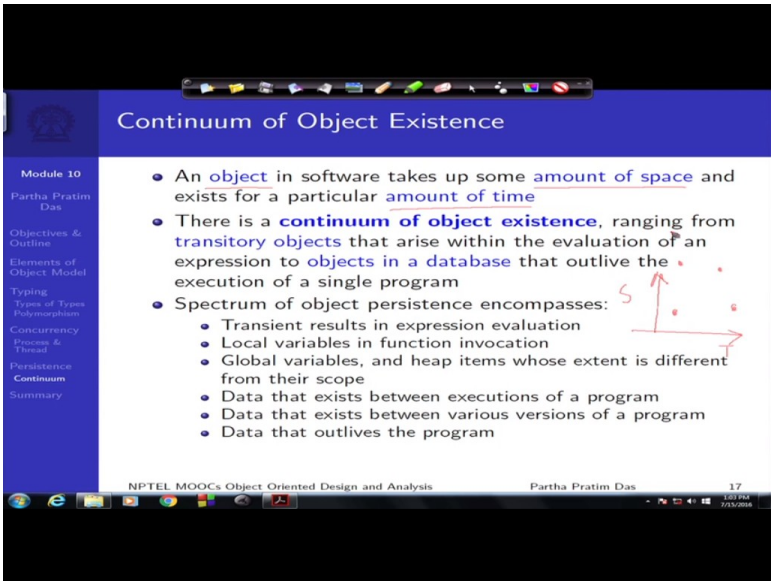
The cartoonist's view for you to enjoy, I will not go into explaining this. You should take your time of and try to understand what is being said in persistence layer.

(Refer Slide Time: 12:20)



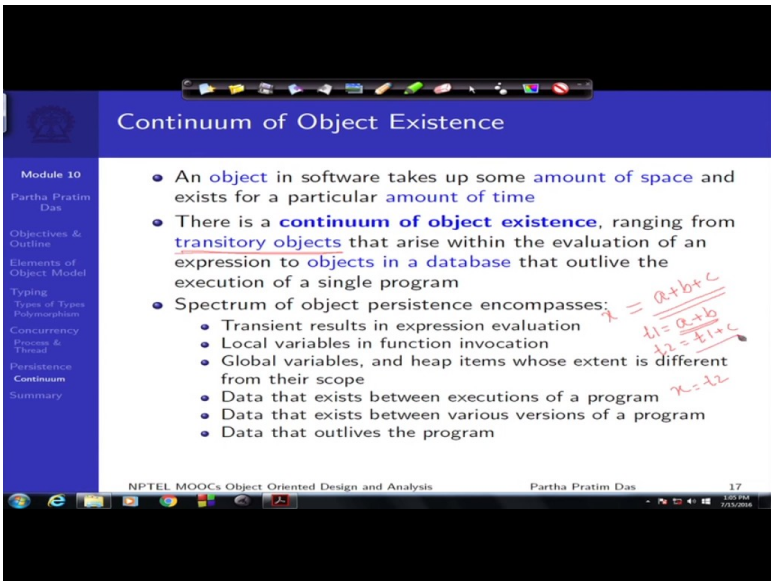
Let me focus on the fact that persistence built a continuum of object existence. As I have already explained, an object let me get back to read that is more prominent. So an object in software takes certain amount of space and exist for a certain amount of time. So how much so if you just look into these two axis, so we say this is time and this is space. So we will have objects in different points in this space. So the further down you go on the right here means these are objects which stays longer from the creation to the destruction is the longer time.

(Refer Slide Time: 13:13)



If I go on this, then it says that these are objects which move between more and number of spaces, address spaces. So if I look into this, then we find that there is a continuum of object existence that is it varies significantly as to how long an object stays and where. For example the shortest living objects usually are the transitory objects, objects that get created because you just wanted to move some value from one place to the another. For example if you know C++the basic of C++

(Refer Slide Time: 14:20)



And you are returning a value by copy then while you return that the actual object that you are trying to return cannot be returned. Because that object has to die as soon as the procedure function call is over what you do you make a transitory object you make a temporary object and return that or for that matter if I am trying to evaluate something like a + b + c then I cannot evaluate all of them together so

what I do I kind of evaluate something like t1 is a + b and then t2 say this has to go to x.
Then we say t2 is t1 + c then will say x is t2 because at every time the object can do only one thing make either add 2 numbers a and b add 2 numbers the temporary result of addition of a and b with c and so on. So these kind of objects are the transitory objects or temporary objects. They naturally have very short time span because t1 gets created here and will get destroyed here start. 2 different objects that we might need to keep in the database think about the leave management system.

We have talked about so many employees of different types and certainly these employees stay in the system for a very long period of time once a an executive has joined and has been created in the system database that employee object representing the executive will remain till that executive has been patched from the system possibly because she has left or she is resigned and so on. So the object that exist in the database often outlive.

So this is a very critical concept outlive the execution of a single program which means that the object representing an executive will remain much longer in the system compared to the method that needs to be performed for requesting a leave by that executive or by approving a leave for that executive and so on. So based on this is this is kind of a continuum hierarchy of most transient to most persistent objects these are expression evaluation I just showed you an example that how transient values happen in expression evaluation.

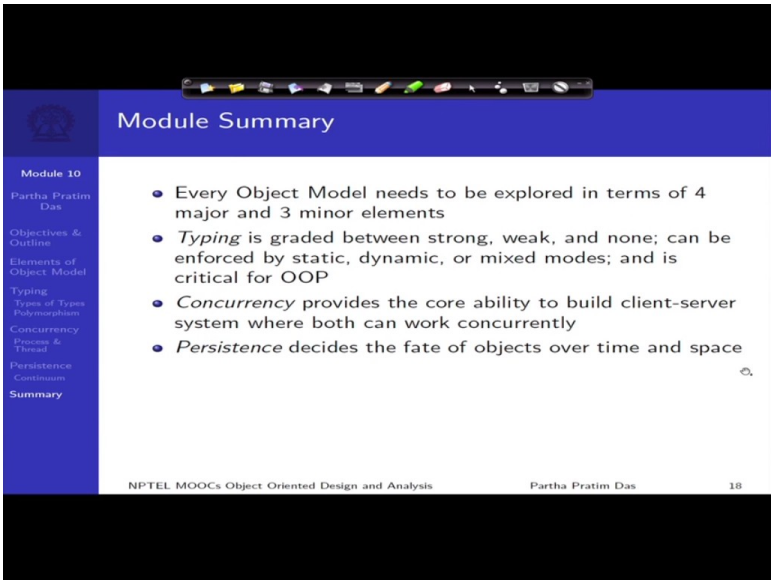
Similar things will happen for local function invocation you know whenever we do call by value the in c the values get copied and those copies get removed when the function invocation is over. there are global variables dynamically heap items that means dynamically allocated with allocated variables which outlive the scope that they related to and so on till you talk about really database persistent objects in the database which will outlive any kind of programs which create, manipulate and destroy them.

So this is it is it is very important to understand this continuum and whenever you are designing the object understand what is the persistence model of that object for very simple things for example if an object is very transitory it occurs and then disappears then we would like to make it very light weight we will try to keep it very light weight because if there are too many transitory objects they are getting created and destroyed, created and destroyed very frequently so system will have a lot of over weight of simply creating and destroying objects.

If an object is very large it is huge it is takes a lot of space we need to measure that it does not have ample transitory behavior it should have more persistent behavior and only a small part of that object which might need changes or which might need relocation will have to be treated separately so that you do not get into performance issues so different aspects of persistence or different levels of persistence in time and in space will guide how you should design different objects.

How you should encapsulate and modularize them so that I can design an efficient system. So these factors this is the third factor that you need to keep in mind.

(Refer Slide Time: 18:36)



So to summarize we have in this module talked about the three minor or useful but not exactly essential aspects of object models but I would show you from my experience that when if you want to do a good object oriented system design or rather if you want to do a good design for a complex system then alongside the major element is very critical to take care of the minor elements because minor elements relate to three major access of implementation.

Which often turn out to be the core nonfunctional requirements of a system leading to performance in terms of time, performance in terms of space and general behavior of how responsive the system is, how robust the system is and so on. So please keep the aspects of typing in mind which creates between strong, week and more type between static and dynamic and is critical for oop you will learn it certainly be conscious about the concurrency to be able to build client server models.

And persistence will immediately tell you what kind of system you want to build. For example a very very simple example for persistence would be you would need to decide whether at all the objects outlive your application or not. If they outlive then your immediate conclusion is there will have to be a database there will have to be a database to store, there has to be a persistence tool to keep the objects because otherwise all variables.

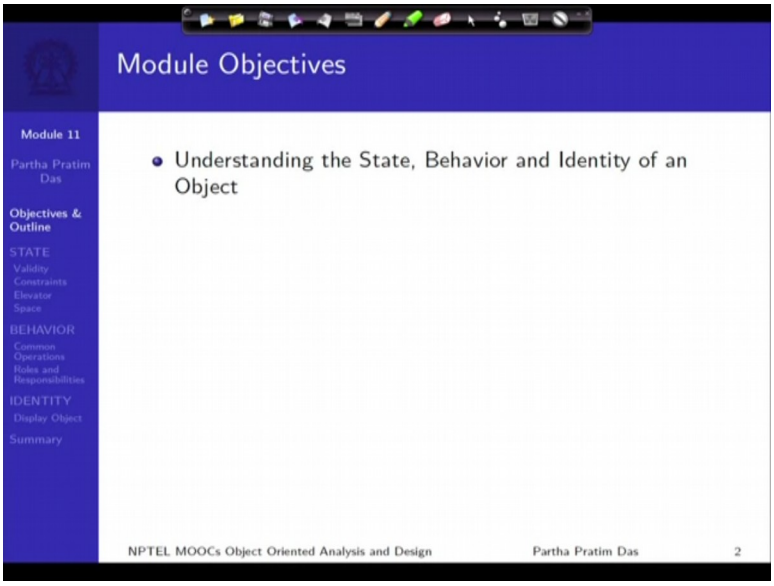
And objects that you create would deal within a program would certainly disappear will get forcibly destroyed as long as your main function can. So you will not be able to work with them next time you run your program. So these are the considerations that you will need to work with. So with this I will conclude on the elements of object models and next we will start talk about the other details of objects models from the next module.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 18
Nature of an object: State, Behavior and Identity

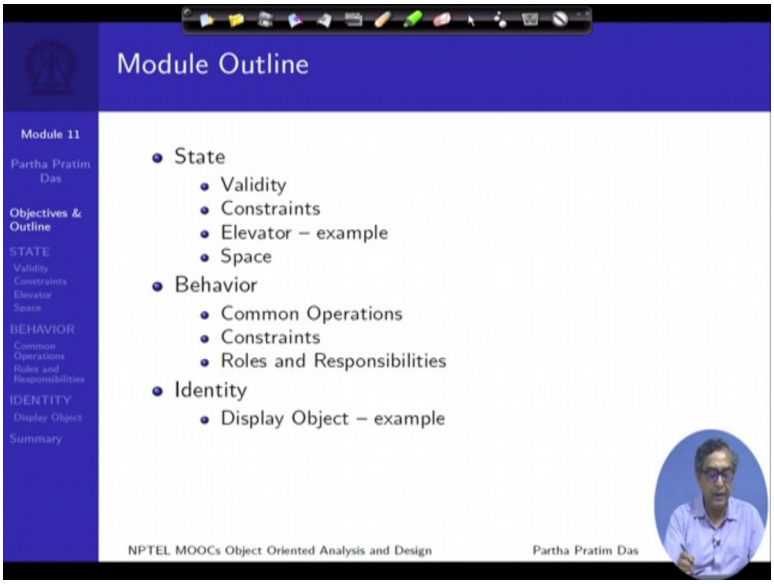
Welcome back to module 11 of object-oriented analysis and design. In the last couple of modules, we have discussed about different elements of object models. We have seen that there are 4 major and 3 minor elements which will guide, which will be the principles on which object models, should be built up. Based on those, now from this module onwards, we will actually get hands on with starting to define the details of objects, their relationships and classes and the relationships of classes and slowly get into actual exercise of identifying objects and classes in an object-oriented system.

(Refer Slide Time: 01:20)



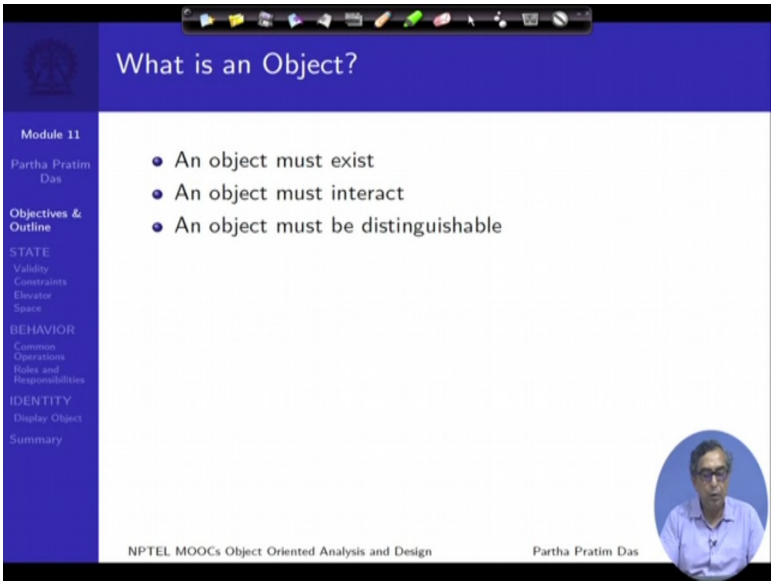
So here the in the current module we start by discussing the nature of an object, the objective of this module is to understand the three aspects known as state, behavior and identity of an object.

(Refer Slide Time: 01:41)



Outline is given here and like in every module will be carried on the left of every slide.

(Refer Slide Time: 01:50)



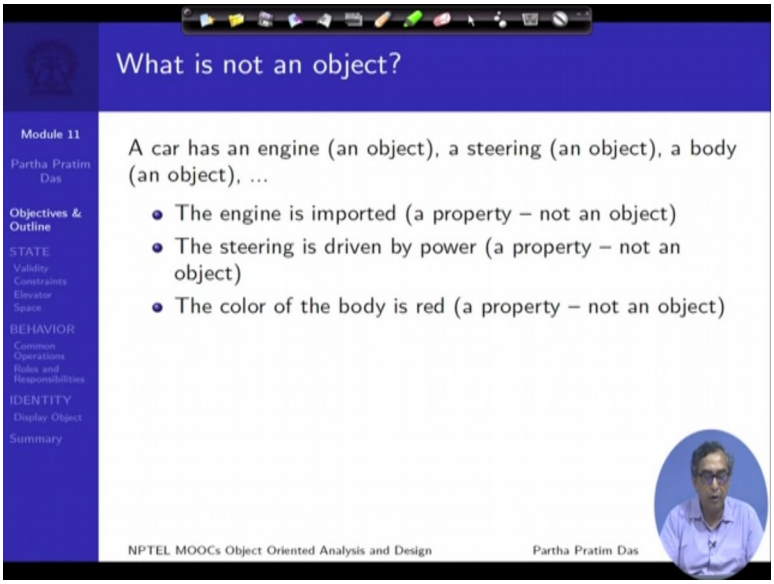
So just to quickly recap what is an object? Now we have talked about objects in terms of abstraction. Now if we finally want to come down to design, then what we say will be an object. The first condition or the first concept that we will try to enforce is that object must exist. There must be some kind of an existence for the object. So, objects could be tangible that is we can physically touch them. It may be somewhat intangible for example a process being followed in a manufacturing company could also be an object.

You certainly cannot touch the process, but it exists. It could be visible; it may be less explicitly visible and so on. different various aspects but fundamental to everything for an object is it must exist. The

second is an object that exist must interact with others that is a system is not interesting if it has objects which just exist by themselves and do not do anything with others. So, if a car has objects like steering, like body like engine, like wheels, like seats and so on.

These objects must interact between themselves and that is how we will get the existence of a bigger abstraction or a bigger object like a car. Third and a very important factor is object must be distinguishable that is it is not like the dust in the air or the water vapor in the cloud. But it must be like people, like chairs, tables, items, processes, practices and so on. so that one object must be distinguishable from another. so, these are the basic premise on which an object will be identified.

(Refer Slide Time: 04:22)

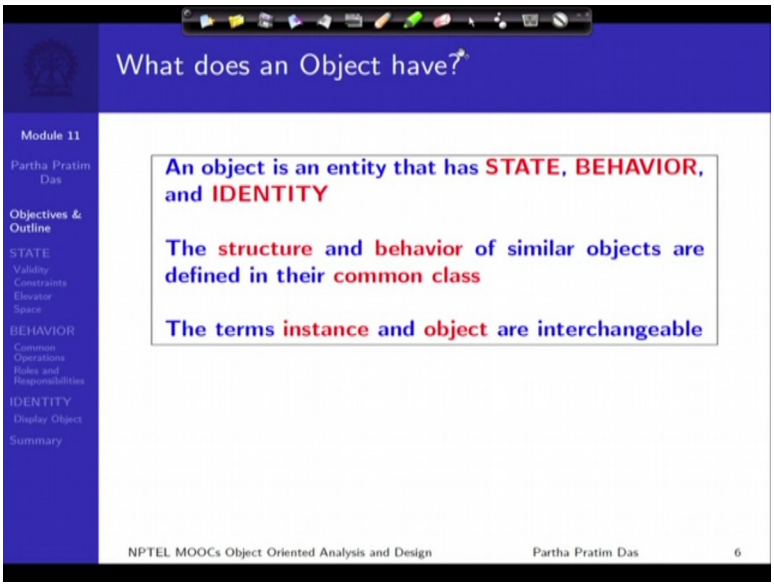


Now certainly it will be in order to also talk about if these are objects then certainly there is a lot which is not an object and what is not an object is certainly a more fuzzy somewhat vaguely defined concept and certainly something that does not satisfy one or more of the aspects that we just talked about will not be an object. For example, suppose a car has an engine, so if we just look at, it has an engine. So naturally an engine is an object.

It has a steering that is an object. It has a body that is an object and so on. now we can say that the engine of the car, I want to put this information in the design that the engine of the car is imported. Now this the fact that the engine of the car is imported is a property and being imported is not an object. Kindly understand that this is different from import which will be an object. But here we are saying that the fact that the engine is important that it is manufactured outside the country and purchase from there is a property and is not an object.

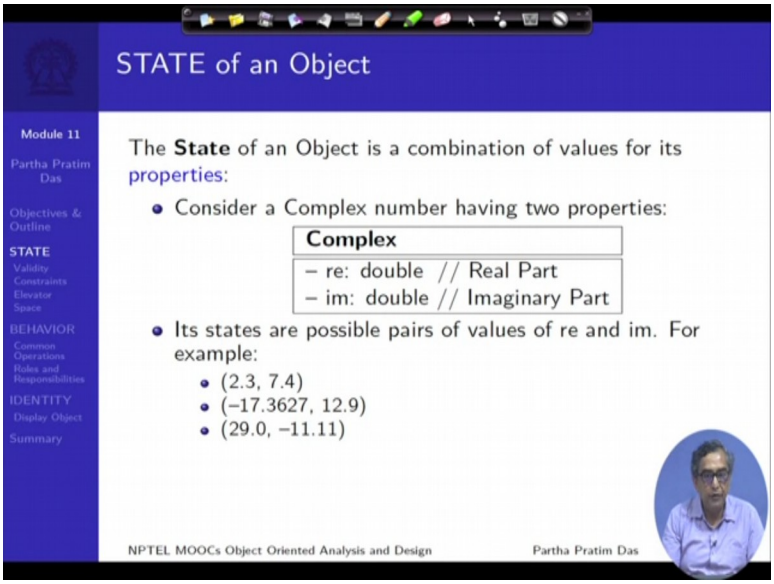
Similarly say that we have a power steering so we say the steering is driven by power. this fact will have to be somewhere put into the model, put into the design but it is again a property and not an object. The color of the body of the car is red. The fact that it is red is a property and that is not an object. So, then you could you could slowly think and expand on this and you will be able to identify, you will be able reason to yourself that what is an object and what is not an object.

(Refer Slide Time: 06:14)



Now if I say this is we now understand what is an object. Then we ask what does an object have. In object-oriented analysis and design from that perspective, an object is an entity which has 3 basic parts, a state, a behavior and an identity. The structure and behavior of similar objects will be defined in terms of their common class about which we'll talk more in the following modules and just for your reference the terms like instance of a class and object, we will be using them interchangeably one in place of the other.

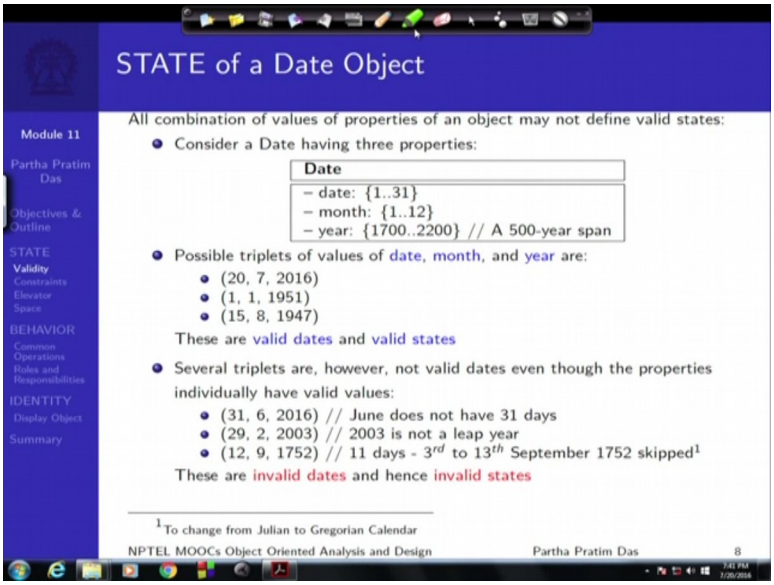
(Refer Slide Time: 07:09)



So now we will take a look into what are these state, behavior, and identity of an object are. So first talk about the state of an object. The state of an object is a combination of the values for its properties. Just note. So, it is dependent on the properties and a state is will be given by the combination of the value that the properties have. So, if I have a complex object, complex number of object, it has these 2 properties, real part and imaginary part, then a state will be since it has 2 components or 2 properties,

A state in this case will be a pair or a doublet giving the real part and the imaginary part and over different states potentially infinite, all different such pairs will define different states of a complex object. that is a simple idea, that is you consider what the properties are, considered what all different values they can take, all of these combinations give you a state.

(Refer Slide Time: 08:10)



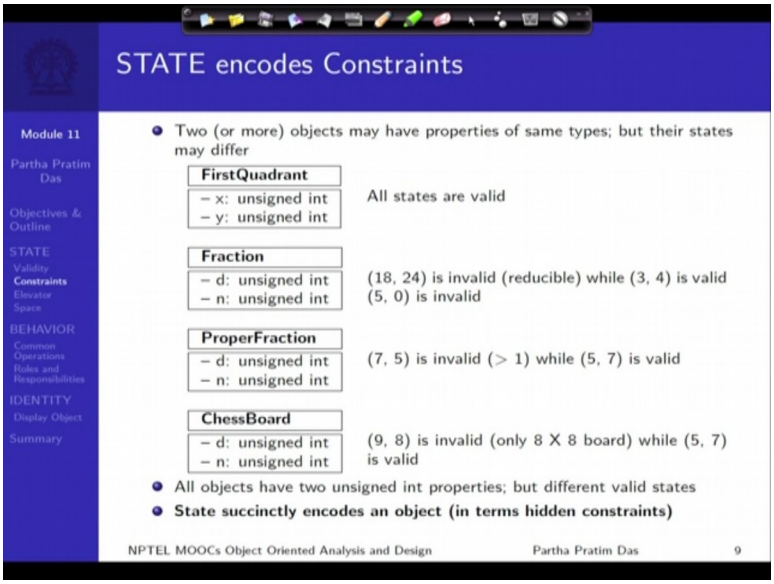
Let us look at a little bit of a different object. Lets us at it date object. So here we have I have just put a date object, simple date object which has a date as in the number of that particular day in a month. So, it can be between 1,2 this this notation means that date is a property which takes a value between 1 up to 31. This is defining a range, it has a month which is 1 to 12and just arbitrarily have taken a 500 year back, 1700 ad to 2200 ad.

So, a date object will have these 3 properties. So, we can we can look at different instances, different objects having these 3 properties like 20th July 2016, 9, 15th august 1947 and so on. these are all valid dates and therefore each one of them define a valid state but we said the state is a combination of possible values that the properties can takes so for a date object, these are also possible values like 31st June 2016. We all know that is not possible, June cannot have more than 30 days.

So, this state for the date object is an invalid state. Similarly, 29th February 2003 is an invalid state because 2003 is not a leap year. Very interestingly in the calendar that we follow if we try to find the date, 12th September 1752, it will actually be an invalid date because it so happened that the calendar moved and 11 days from 3rd to 13th September in 1752 were simply wiped up out of the calendar. So, the reasons could be all different.

In some cases, a month does not have enough number of days, in some cases a leap year has to plug in, in some cases it is a one-time change in the date calendar that happened worldwide. All these lead to the fact that states may or may not be valid that is when we talk about all possible combinations, there is some underlying set of constraints that kick in to define what is a valid state and what is not a valid state.

(Refer Slide Time: 10:42)



So, let us look into little bit more this. Here I have framed and a collection of example objects for you where first you consider the similarity, the similarity is all of these objects have 2 attributes, 2 properties, all four of them and both of these have the same type. So, if I just consider the possible combinations of values that the properties can take, all of these objects is expected to have the same set of states. But now let us think of it semantically with the meaning. My first object is the first quadrant in the cartisian.

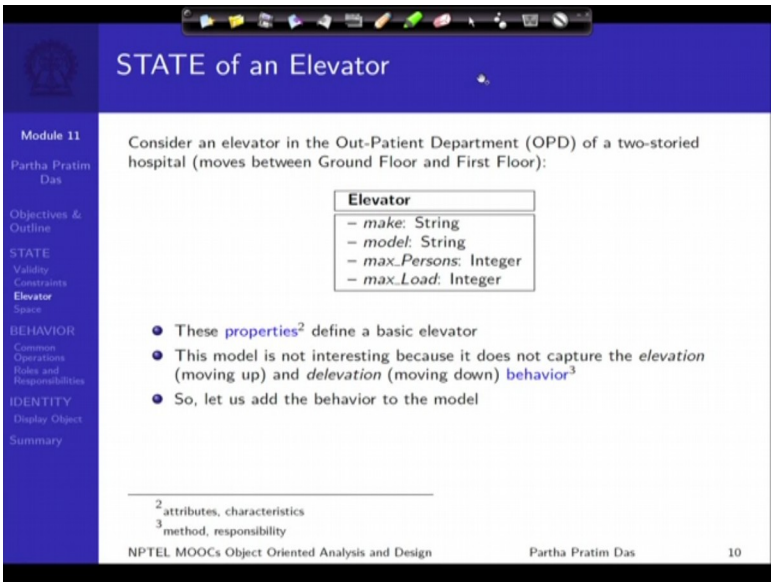
So naturally all possible quadrant values are acceptable, they are valid. So, all state are valid. Now think of a fraction. A fraction naturally is represented in an irreducible form which means that the denominator and the numerator must not have any common divisor. The gcd of these 2 should be 1. So, in terms of the properties, I can write 1824 as a pair where 18 is the denominator this is the numerator is actually invalid because they are reducible whereas when I actually divide them by the common factors it becomes a valid state, a valid fraction object.

Proper fraction similarly would be those which are less than 1. So, 75 would be an invalid proper fraction. If I think about a chess board which is 8 by 8 naturally each component, each property of the chessboard must be a number between 0 to 7. so all other states like 9, 8 will be invalid state. So, what it shows that it is not only the properties and their types but the state is dependent on some kind of hidden constraints or some kind of encoding of objects exist in those hidden constraints which actually define what the state's an object can have.

So, this in every case whenever you design an object, you'll have to implicitly at least understand what

the states are and what the validity of the states are and more and more constraints you actually have which I incidentally often are hidden, they are not explicit in terms of the object design not at least with the way we have seen so far. Then you will have to make sure that you do not do a design which allows invalid states to become possible values in your system if it happens, then we will certainly have different kinds of errors coming from the system.

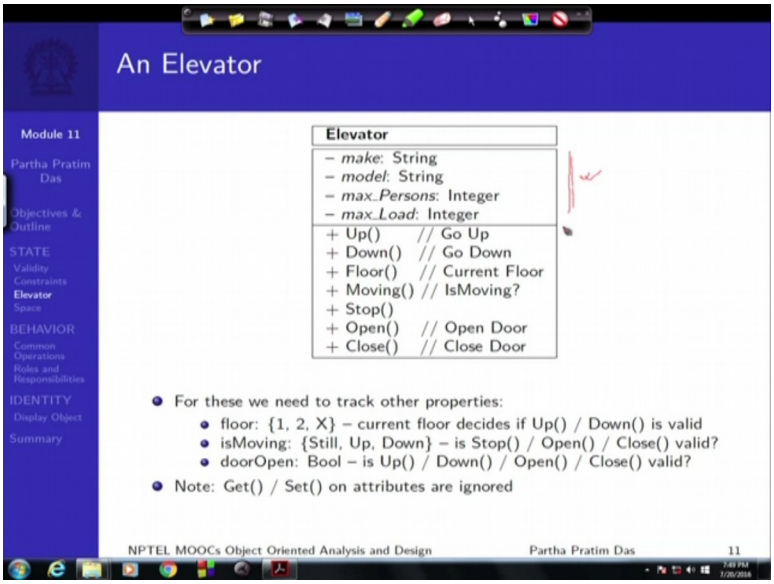
(Refer Slide Time: 14:01)



Let's take a little bit of an extended example to understand some more depth into the state. Let us consider that we are at the outpatient department of a hospital which is 2 storied. It has a ground floor and the first floor or floor number 1 and floor number 2. And there is an elevator which flies between them from ground floor, it goes up to first floor, from first floor it comes down to ground floor and there is an elevator object so this is this is the elevator object

So it talks about different properties like the make of the elevator, the model, the maximum number of persons it can carry, the maximum load it can carry, certainly it defines a basic these properties define a basic elevator but the question is do these properties actually define an elevator when he say it's an elevator, we expect not only information about which company made that or how many persons it can carry and so on but we certainly want the basic elevation and the delegation functionality to be available that it must move up and down.

(Refer Slide Time: 15:26)



So just to explore further, let us see what will be the impact on the state, if we add these operations, add these functionality to the elevator. So just look at what have done, this this remains same, this is exactly what it is, but I have added number of operations here I did a number of behavior to this which say that I have a method up which if pressed will make the elevator go up from floor 1 to floor 2, similarly a method down to go down, a floor method could be there to check which floor the elevator currently is in, it can check whether the elevator is moving

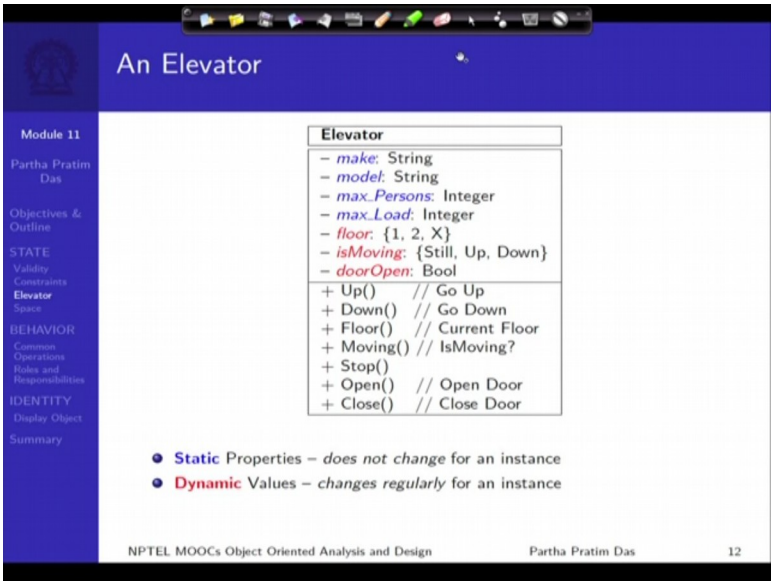
Or it is stand still I could stop the elevator, open the door, close the door and so on. so, these are all, there could be several other operations but am just focusing on some of the typical operations without which an elevator does not become an elevator. Now look into the state information this is your state information which says make, model, maximum persons and load and this is the functionality that we actually wanted in the object. So, we can easily see that the state is inadequately represented to carry the behavior. for example,

If I have to go up I need to be on the ground floor or floor number 1. Otherwise certainly I cannot go up, so to go up I need to know which floor am in. to go up certainly I would not like to go up in an elevator where door is open. So, I would need that the door has been closed before the elevator goes up or it goes down and so on. so, for these operations to be meaningful I will need an extension of the state in terms of a number of additional properties that this object should have.

For example, it should have the property say floor which could take 3 possible values 1,2 and x. 1 when it is at the ground floor, floor 1, 2 when it's in the second floor and x when it is moving that is this floor

is indeterminate. Similarly, it could be still, it could be going up, it could be going down. So is moving is a property which we will say that the door open will be a Boolean, if the door is open or is closed. So, these are these properties will help all these different operations to be actually realized.

(Refer Slide Time: 17:58)

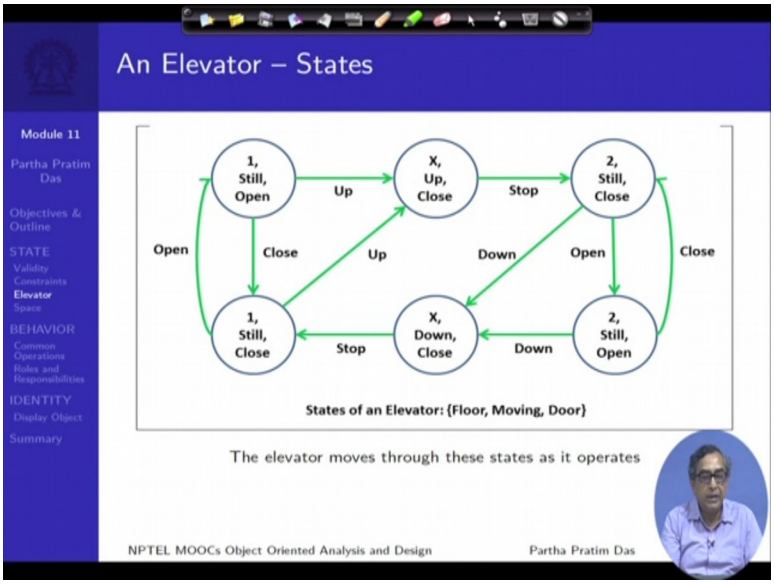


So, with this, naturally the state of the elevator now gets extended so now if you look at, these are the states or properties that we had and these are the properties that we have added, naturally with this addition the possible states of the elevator becomes a lot more. But you can see something the reason I am coloring them blue and red is you can see something about the elevator object, see the blue properties once an elevator has been installed in the hospital, then suddenly its make, model

Or the ability to carry the number of persons or the maximum load it can carry, these properties are not going to change. This will remain same as long as that elevator is in operation whereas these properties, the red properties that we have just added, the floor is moving, door open, these properties will regularly continue to change their values will continue to change as elevator every time is gets into operation. So, I can we can say that in in in terms of defining the state, there is set of static properties which does not change or very rarely change for an instance whereas there are dynamic values for certain properties which changes regularly.

So naturally when I talk about the state of an object it is whenever such situation exists, it is more interesting to talk about this dynamic part of the state instead of talking about all possible values that can take part and define a huge state space of possibilities for the object.

(Refer Slide Time: 19:45)



So this is the, we will talk about these kind of diagrams later on when we talk about uml but this is just a sample representation to show the different states an elevator could be in. for example, initially I could be here that I am in so this representation is like this, within every circle which denotes a state of the elevator object, I write 3 items, the first one designates the floor, the second designates the moving status of the elevator and the third designates what is the status of the door.

So, if I am initially here, then it is one still open which means it is in the ground floor, floor 1, it is still it is not moving and the door is open. So, at this stage what you can do certainly at this in this state you cannot go down because it is already in the ground floor but you can go up. If you go up so this is the action that you can do on this state and then the state changes to something like this where the state becomes indeterminate, am sorry the floor becomes indeterminate

Because it is moving, your direction becomes up because you are going from the floor 1 to floor 2 and your door must have been closed. So, when this this transition of state is happening then this values of these properties must change in the appropriate way as shown. Similarly, when the elevator comes to a stop because a stop operation has been performed naturally the floor of the elevator must have become 2 because it was moving up and it becomes still and the door is still open.

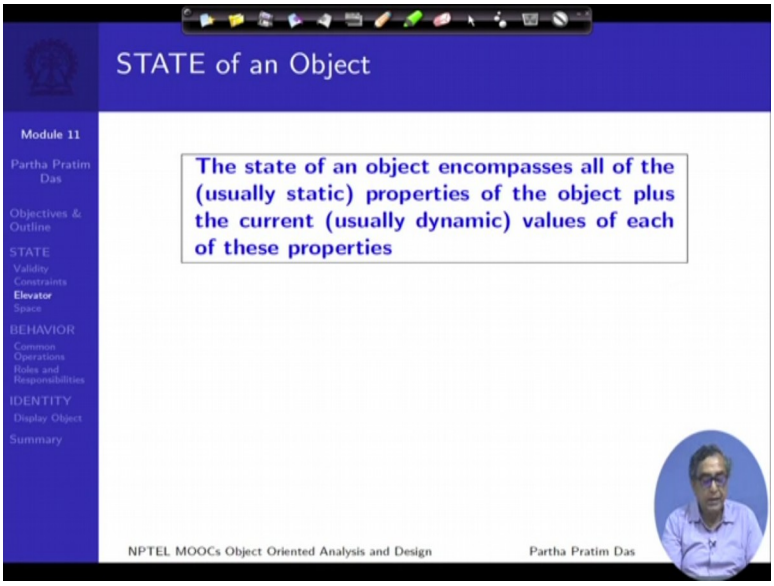
Now if you if people want to go out then certainly they can do an open operation and the state changes to 2 still open. You can switch between these 2 by opening and closing the door or you could from any of these states you could put down keep the elevator moving again but now in the downward direction and finally reach the floor 1 or the ground floor and become stationary. So, this is a we will say this is

called a state diagram or state subsequently will see this will be called a state chart in uml.

These are very powerful tool to represent how the state of an object could impact the behavior of the object in a very second manner how the constraints will apply. Here you can say find that though we are though we are having 7 properties in total the 4, make, model, maximum persons and maximum load are ignored in considering the state because they are kind of static properties and the 3 that we have which are dynamic values even there, we do not take into considerations all of the possibilities because all of the possibilities simply cannot happen.

For example, I cannot have a state where I have x, I have still and I have open or even close that is if my floor is interminable then my elevator cannot be in stationary mode. It must be moving. So, this will tell you that this kind of representation of states implicitly encodes the constraint that exists that an elevator must be at a well-defined floor if it is stationary if it is moving it must have an indeterminate floor, if it is moving the door must be closed and so on so forth. So that is that is the kind of a state information that an object would carry.

(Refer Slide Time: 23:48)

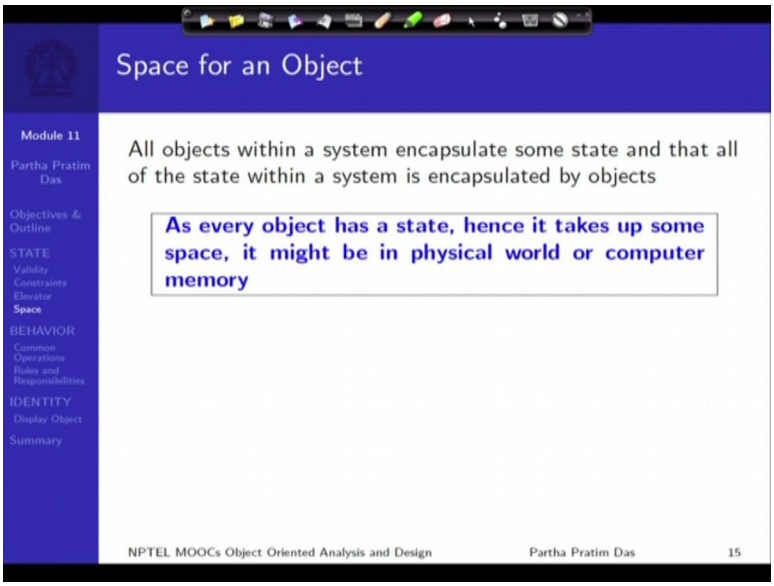


So to summarize on the state, we will say the state of an object encompasses all of the usually static properties we have seen usually static, the kind of properties we say for the elevator of the object, plus the current or usually dynamic values that is the floor is moving kind of properties and that together will define the state of an object and the state of an object may be valid or it may be invalid based on the constraints that are encoded by the state and our endeavor would be to design and define systems and their operations in a manner so that we always move from one valid state of an object to the next

and so on

And we do not ever get into an invalid state, getting into an invalid state is generating an exception in the system because if an object is in an invalid state that is if an elevator is moving with its door open then certainly the system would not know what could be the valid operations for that, what could be a consistent way of moving forward with that system.

(Refer Slide Time: 25:12)



Now along with this we should also note that the state actually satisfies the first condition that we talked about the object that an object mut must exist. That is all objects within a system encapsulate some state and all the state within the system is encapsulated by objects. This is this statement is very simple couple of words, but this is a very deep statement and please read it over and over again, try to go through examples to see that.

All that is being said that if I a system comprises a number of objects then every object will encapsulate some state in the system and on the other hand, the total state of the system if I think about the whole system as an object the total state of the system will be encapsulated by one or more objects in different parts and that is the basic notion of states in the system. So now therefore every object that I have in the system must have a state. Now if it is a state then it is to take up space.

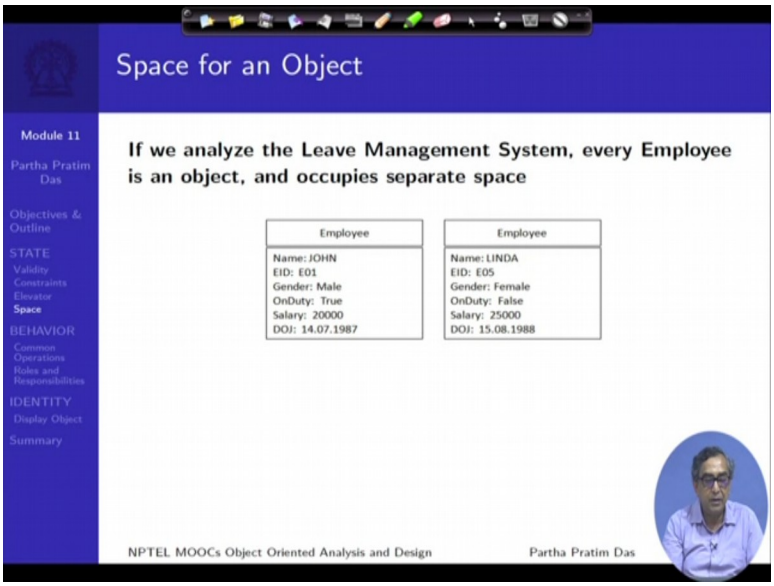
The state is memory, state is remembering, state is knowing in what condition I am. So, a state will always need some space, takes up some space. Very important that space could be in the physical world or it could be in the computer memory, the elevator in the opd of the hospital physically exists.

So, it has states, it is flying people between the 2 floors, opening, closing door, moving up and down. It has a physical existence; the states have resulted in a physical existence.

So, it exists in the physical world. On the other hand, when I would like to design and implement a system for the whole hospital taking that elevator as a part, this elevator object will become a and will get an existence in the memory where in the computer memory I will have binary bit representation for all these different properties and we will do something about the methods so that it will occupy certain bits in the memory and it will have an existence.

So critical to to take back from this is states imply that objects finally have to have space, finally have to have a physical existence however abstraction we actually talked about in terms of their characterization, in terms of their design.

(Refer Slide Time: 28:07)



So, if we just look at a very simple some simple kind of example, talk about a leave management system, then every employee is an object and will occupy separate space. Now naturally physical employees of course will occupy space but their object representations as in here, I have one employee here, I have another employee, john and Linda are 2 employees, their objects will be represented in computer memory with all these values and they will take up physical space for that.

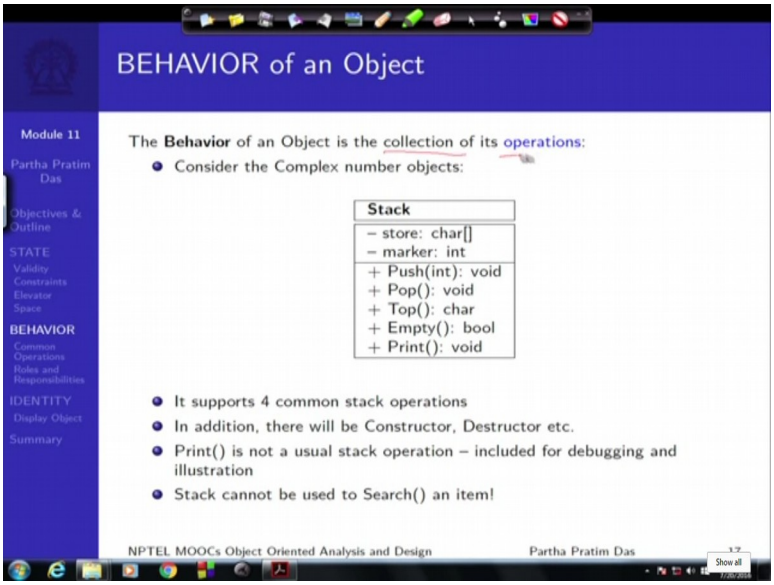
So, with this we come to the end of this first part of nature of object objects and here we have just discussed about the state. Next, we will talk about the other 2 aspects of objects.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 19
Nature of an object: State, Behavior and Identity (Contd.)

Welcome back to module 11 of object-oriented analysis and design. We have been discussing about nature of objects, we have noted that an object must exist. It must interact and it must be distinguishable and taking from there, we have noted that every object will have state, behavior and identity. We have talked about state already, talked about state charts, how certain parts of the property could be static and certain part of the properties could have dynamic values. Now we move on to discuss the behavior of an object.

(Refer Slide Time: 01:05)



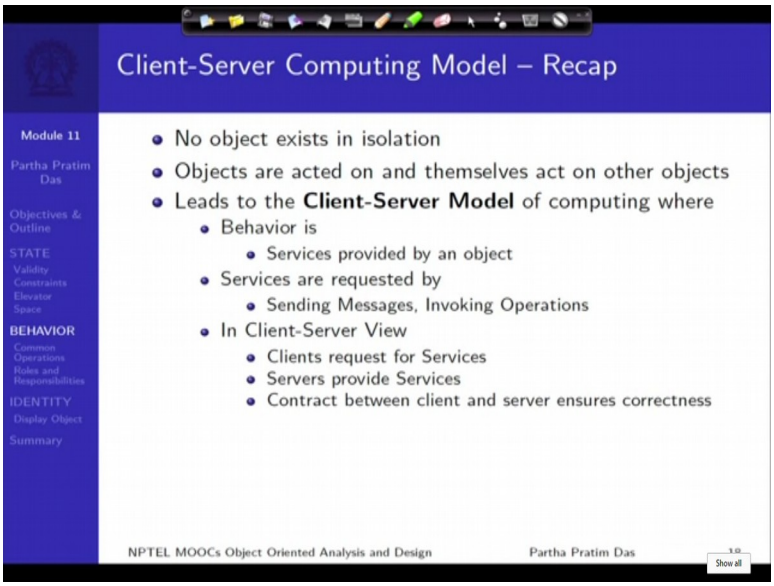
The behavior of an object is a collection of its operations. So, I have a stack object, it has some properties of course to store the items being stack, to mark up what is the top of the stack. But what we have more interested now is a fact that it has different operations. These operations together define the behavior of the stack. So, we can if we just look at specifically then these 2 behaviors, these 2 operations define the LIFO behavior of the stack, Last in first out.

These 4 together these 4 together basically defines a stack. But in addition, I have also provided another operation to extend the behavior. this operation of print is primarily for the purpose of debugging and for the purpose of illustration. For example, if I am creating the stack object

implementing the stack object, and then I need to understand at every stage if my stack operations are working correctly and I might need some help to print the state of the stack at any point of time and therefore add a print operation.

So, behavior will talk about all of these together as a collection and I know that search is not a part of this behavior therefore I cannot search an item in the stack. That's the basic notion of the behavior.

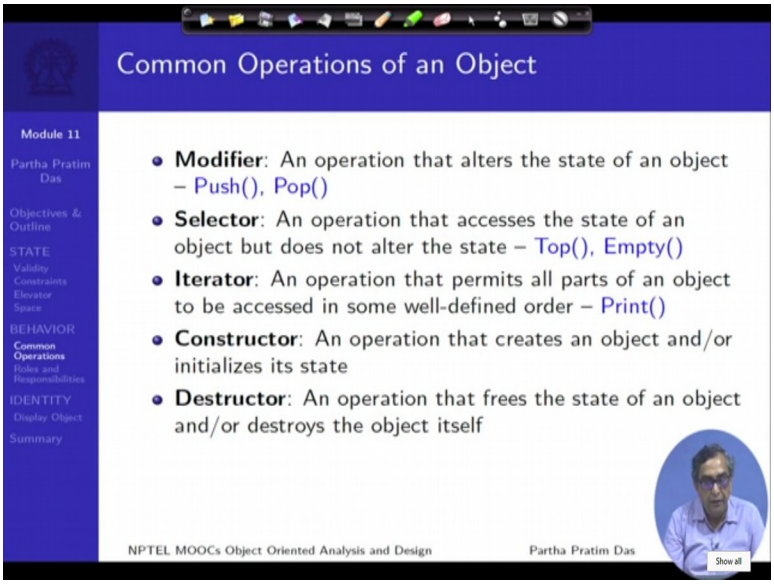
(Refer Slide Time: 02:57)



At this point I would like to draw your attention to the client-server model of object based interaction and computation that we are introduced earlier that is the fact that no object can exist in isolation. An object in isolation has no interesting aspect to talk about. So, objects are acted on and they themselves act on other objects. So, if I have object o1, I have object o2, so o1 acts on o2. So, this is acted on and o1 acts on o2. So, either you are acted on or you act on other objects.

And this is what we have seen leads to the basic client server model the behavior is a set of services operations provided by the object. Services are requested by these are just variance of the terminology, services are requested by either sending messages or invoking operations and in summary the client server view, clients request for services, the servers naturally provide services. Client request for service, the server provides the service and what carries it through is the message which has to be done according to a contract, according to a protocol which is agreed between the client and the server. So, this set of messages or operations basically define the behavior of an object.

(Refer Slide Time: 04:50)



Now of course the objects could have variety of operations in its behavior. but several years of experience of different people in designing objects, in modeling systems have laid to the observation that there are 5 basic types of common operations that usually happen for an object. So why we are designing the object, it will be good to understand what kind of operation we are adding to the whole behavior set. So, these behaviors include the first is all the modifiers.

A modifier is an operation that will change the state of the object, now we will know what is the state of the object. So, if I talk about the stack, then push is a modifier because push what will happen if you push, naturally a new item will get into the store and the marker will move to the top position. Both these properties that the state have will change their values therefore the state of the stack object will change. Similarly, if I do pop again some value will be removed even if I assume that removal is not a physical rewrite of the store.

Certainly, the marker has to move to mark that the top position has changed, so the state of the object will change. So, these are modifier kind of operations. Then I could have selector kind of operations which in which accesses the state but does not change it. When I checked the top element or I check if a stack is empty or not, am not changing the store of the marker but am just checking out something based on their values.

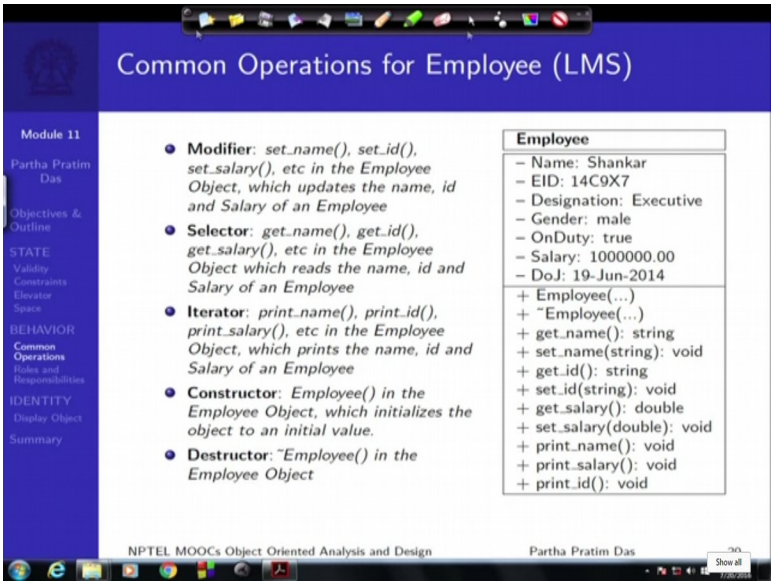
A third class of operations are called iterators and these are very powerful type of operations which basically goes over all parts of an object and does something, accesses, performs something in a well-behaved manner, in a well-defined manner. So, if you just look at consider the print function in the

stack object, we will find that it is an iterator because what does it do? It iterates, goes over each and every element of the store that the stack has and prints it with the output.

So different kinds of prints and different kinds of ways to go over the different parts of the object in an organized manner is the behavior of the common iterator class of operations. So, these are basically what is the what comes from the functionality of an object certainly what needs to be there in addition to these, or the constructor operation which actually creates the object and the destructor operation by which an object annihilates itself.

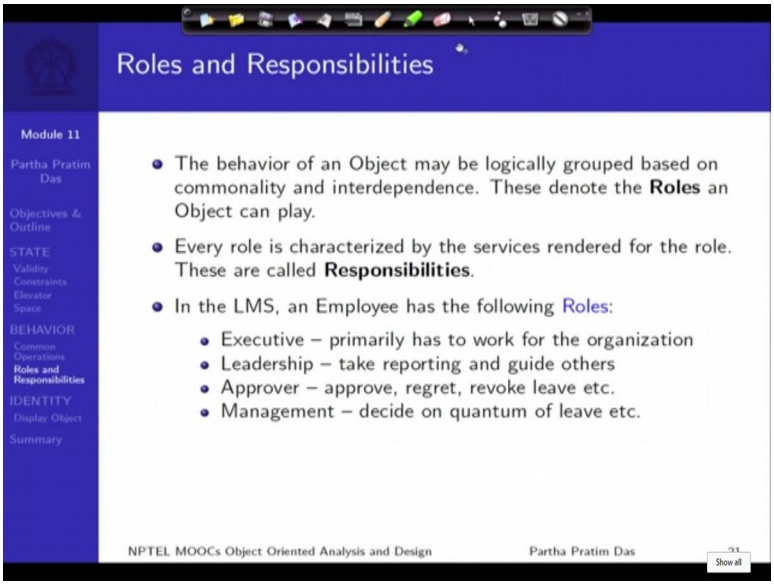
So, though I have not explicitly shown a stack will need a constructor to create a stack and it will need a destructor so that it can destroy itself when the job of the stack is over. So, whenever you design objects and you define its behavior try to be clear in terms of which category your operation falls in. it is usually a good practice not to write very complex operations which could do multiple of these activities in the same operations that is it modifies as well as iterates and while it is doing a construction. So, it is better to avoid these kind of complex operations for the ease and clarity of design.

(Refer Slide Time: 09:02)



So, this is another example for primarily for your self-study I have taken an object instance from the lms example, an employee and I have shown that for that employee all that different kind of some of the operations that you can have, how those operations are classified according to being a modifier, being a selector, being an iterator or being a constructor or being a destructor. We can see the constructor is here, the destructor is here and so on.

(Refer Slide Time: 09:43)



we have roles and responsibilities of behavior. an object which is not very simple will certainly be having a large number of operations and as it happens that often these operations are kind of groupable. I can put similar operations interrelated operations together. So, in a total behavior if I group the operations according to their semantic commonality according to their interdependence, then I will get the different roles that the object play as a part of the behavior and it is possible that roles may overlap that these 2 different roles may have certain things which are common.

So just to take an example is if we if we look at the different activities that the employees do in our leave management system we have seen that there are some employees perform the role of being an executive. There is their main task is to work, primarily is to work for the organization but some employees whom the document says the lead employees certainly take reporting of other employees, executives and guide them, providing leadership. So those who work as executive do not have the role of leadership.

Those who are lead or manager, the lms specification say can actually approve the leave for others, can revoke an approved leave, can regret an applied leave and so on. so, they have yet another role which is what I am calling here is an approval role. Then managers are certain employees who have management roles, for example they can decide how much leave an employee should get and so on so forth. So, this is a context of having roles and behavior it usually becomes a cleaner design

if we have a clear understanding about the different roles that the behavior of an object will have. And

every role will then be associated with the responsibilities or the set of operations that the particular role is expected to perform.

(Refer Slide Time: 12:27)

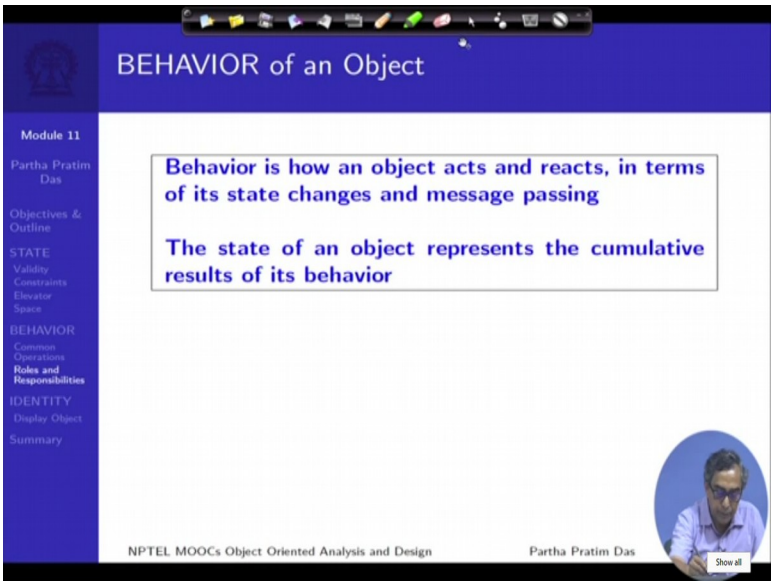
So, this is again for your self-understanding. These are 3 objects been shown from the lms system and the left most one is for an executive so to illustrate that I have introduced a property called designation. So, the left most one is for an executive, next is for a lead, next is for a manger and here you can see what are the roles that they these objects will play. Now if we look into the operations, we will find that this role of an executive is responsible for these operations

that is recording his or her own attendance, applying for leave, cancelling a leave, availing a leave that has been approved and so on. so, the executive has say if the role is of an executive, then these are the roles, these are the responsibilities. Now if we look into an employee object which is for a lead, then certainly that employee has an executive role and associated responsibilities but the employee also has other roles like leadership and approval and associated responsibilities of approving leave, revoking leave, taking reporting and so on.

if we look into the manager, we will have yet one more role and the associated responsibility for that role. So, as we can see here the different objects all are of the all our employee objects, their behavior has different roles and the roles in this case are overlapping, for example a lead employee actually performs 2 roles, the role of an executive and the collective role of leadership and approver, a manager performs all the 4 roles and correspondingly they have overlapping responsibilities to perform.

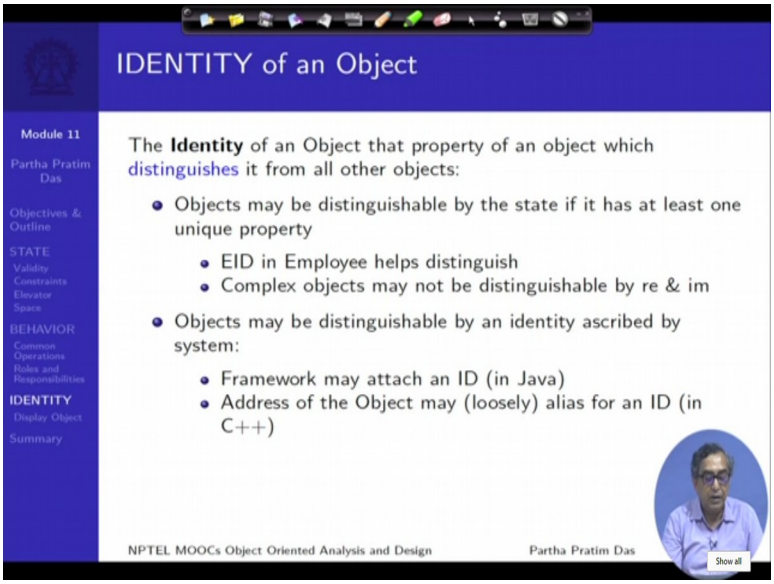
So, it will usually of course it is not mandatory that you will have to define or you will have to demark it what are the different roles of a behavior and the corresponding responsibility but as it turns out it usually gives you a better design if you are clear in terms of the different roles in the behavior of an object that you define and you are able to associate the operations as responsibilities of different roles.

(Refer Slide Time: 15:19)



So, to summarize the behavior of an object is how an object acts, how an object acts and reacts. Act means proactively, react means in response to what somebody else has done in terms of state changes and message passing. So, if the state changes naturally the object will have and if the message has been received the object will react. The state of an object represents the cumulative result of its behavior at any point of time all these operations has performed we have seen whether particularly as we have seen that if we are any kind of modifier operation then the state will keep on changing and that is the result of the behavior that we will get to see.

(Refer Slide Time: 16:12)



Next, we have talked about 2 aspects of state and behavior. next let's talk about the identity of an object. The identity of an object is as we started saying that every object must be distinguishable. So that's the simple thing, the identity of an object is a that property of an object which distinguishes it from all other objects. So that's something very critical for the objects to exist and behave, without the identity, we will not be able to actually do any reasoning in the system.

Now how do I put the identity, how does an object get that identity, it could be grossly, there are grossly 2 major ways the identity can be thought of. 1 is objects may be distinguishable by the state, that they can say that if I know the state of the object, I know which object it is. How do I do that? It's very difficult to do that. But I will be able to do that if the object has at least one unique property. I mean those of you who are familiar with some bit of database systems particularly of the relational kind will identify this unique property as a key.

Now it may be a single property, it may be a collection of properties that uniqueness here mean that a property is unique or a key if for any 2 objects that I can have if that particular property will always be taking different values. So, consider that the employees in the leave management system the employee object has an employee id naturally there cannot be 2 employee objects with the same id so therefore this is a distinguishable property

and the object no matter what else happen just by the state of the object which comprise of various different properties like the name, the date of joining and whether the employees on duty and so on will also have the eid and we will be able to distinguish the employee object by the eid part. So that's one

way of distinguishing objects but you can just extend that if we are talking about complex objects then we may not be possible to distinguish to complex objects by their state

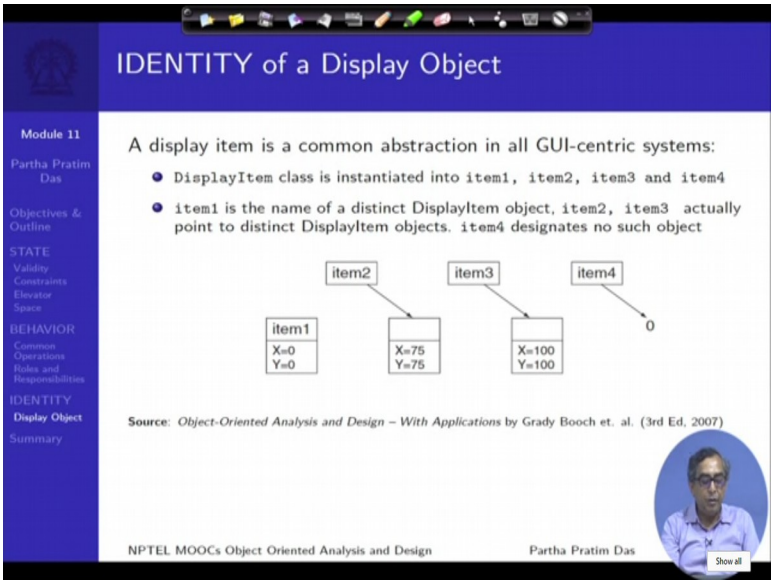
or by the value of re and im, I can have 2 complex objects which have the same real part and the same imaginary part and they are just 2 different objects. So, if so it is not necessary that all objects will have a unique distinguishable property so the objects may be distinguished by the identity that is described on an object by the system. That is a system says that well to distinguish an object based on or to identify an object or to setup identity for an object based on a distinguishable property kind of I open up the object,

look into the state and say well this is the object but in stack the system that manages these objects could put an id on top of that object and this is what is done typically in a lot of programming systems for example those programming systems which are based on framework I would just remind you that we have talked about the fact that in the evolution of programming languages we talked about that all a major part of the recent programming systems are based on framework

whereas there is a certain runtime that runs like jvm for java or dot net for c sharp and so on which manage the objects that exist at the time of execution so if there is the framework that goes with the programming system then the framework may attach an id to the object? So, it does not need to look into what is inside state of the object but based on the id that it attaches separately you can figure out what is the identity of an object but systems which do not ha or the programming system or the programming languages which do not use a framework that is do not use a separate runtime system would not be able to do this.

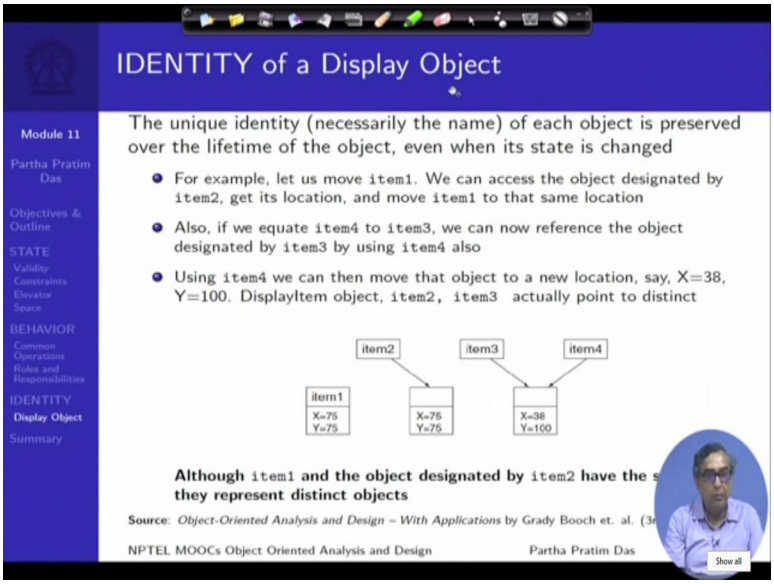
So, for them it's typically the address of the object gives loosely or works loosely as an alias for the id. So, you say what is the identity of this object is wherever it exists. Now this kind of an identity is somewhat difficult to work with because you will understand that without changing the object in anyway if I just relocate the object from one memory address to another then potentially I would be impacting the identity of the object so you will have to be careful in those systems in terms of dealing with the identity of the object.

(Refer Slide Time: 21:41)



So, to look little bit more into this let us just take consider a simple example, this whole example is very well developed in the book in the OOAD book so it is just taken from there, it talks about display item is a common abstraction in gui centric system. So, the idea is you have a kind of a canvas you have kind of a canvas the gui canvas and you have different objects on that canvas. And where they occur in the canvas is defined by a pair of values x and y coordinates of the center of that display object. So here in this instance it shows the 4 different 4 objects this is item 1 is 1 display item 1 display object and item 2, 3 and 4 are pointers to display items. of that item 2 points to this particular display item which is unnamed but you can still identify it by the pointer, item 3 points to another and item 4 is kind of a dangling reference which currently does not point to any display item. So, you can see that in in this case we have different identities, some are maintained directly by the object as an item 1 or indirectly to the pointer if there is no direct identity of the object.

(Refer Slide Time: 23:25)

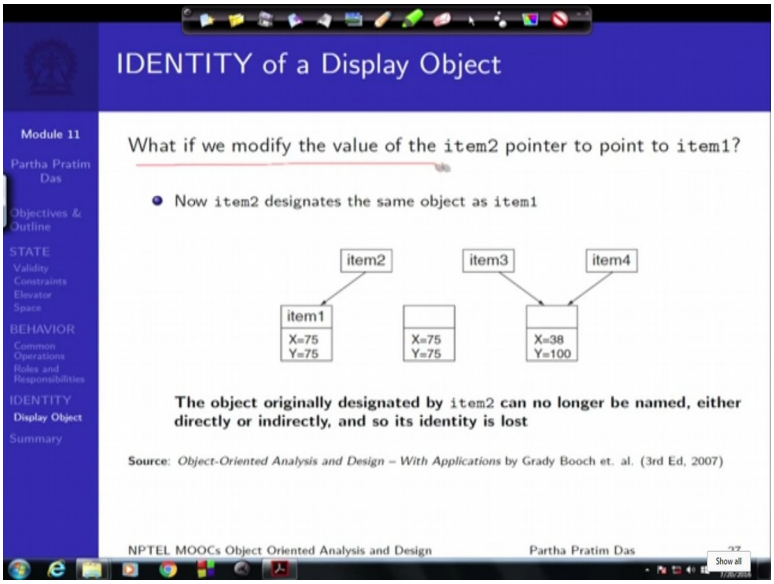


Now let us see what happens when some operations start happening with this that the unique property that each object will preserve through the over the lifetime is the identity even when its state will be changing so let's say on that on that canvas on that gui display system suppose item 1 has been moved item 1has been moved so earlier it was in 0, 0 and now it has been moved to a location 7 5 7 5, 75, 75. How did you move that?

We used item 2 got the location of this unnamed object and we have moved item1also to that same location. Now suppose we equate item 3 and item 4, item4 you remember, item4 earlier was pointing to null, it was not pointing anywhere. Now we have equated 3 and 4 so both refer to the same object. The 2 basically 2 different references but the same object and using item4 we have actually edited the x coordinate that is we have actually moved that object therefore if you try to access it through item3 also we will get it in a new location.

So, what we can see that item1 and item2 have the same state, both are located in the same 75, 75 locations but they necessarily represent distinct object this is the cruces of the identity that a state may not actually define it. Here where is the identity coming from? The identity here is coming from the identity that is written on the item1 object and the identity of the pointer which holds this are named object which holds this unidentified object.

(Refer Slide Time: 25:54)

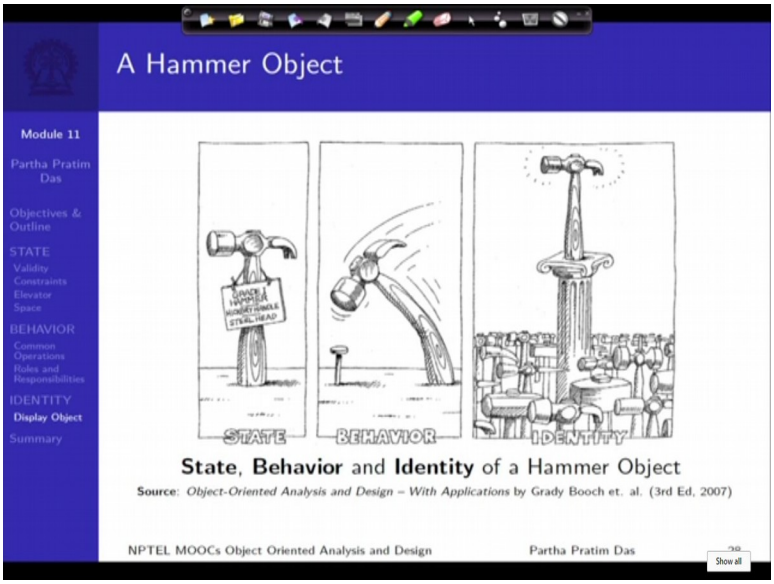


Let us go further let us see what if we modify so we are just asking this question what if we modify the value of the item2pointer to point to item1? So earlier it was pointing here, we change and we make it point to item1. So now what will happen item1 and item2 basically both of these actually mean the same object, the 2 identities but actually they mean the same object and the worst part is this particular object has lost its identity?

It did not have it was not named, it was not designated anywhere, it was identified by the pointer which was holding this address of this object, now that has moved on this side, the pointer has move on and therefore this has become kind of lost its identity and therefore it cannot be accessed it cannot be managed directly or indirectly. So, when you can see that identity of an object is not only, it is it is critical in the sense of abstraction because unless objects have identities how do they interact with each other,

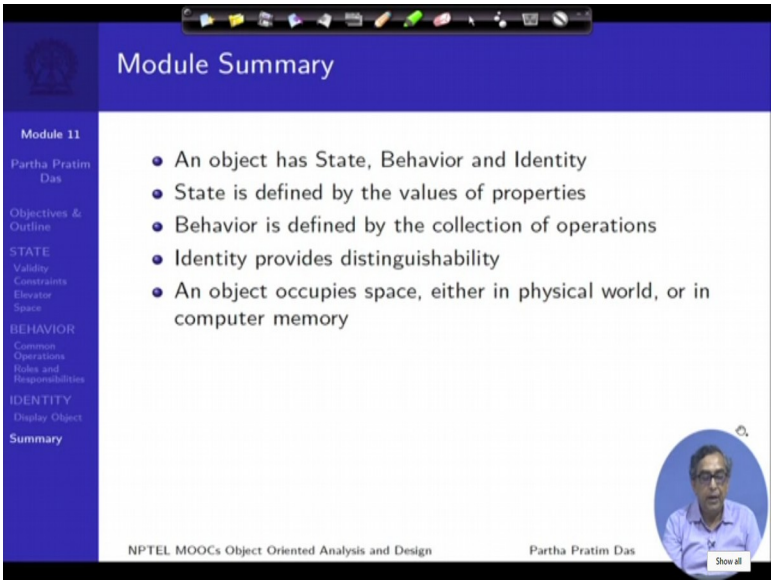
how do they act and react which the behavior need but we note that their identity may not be always derivable from the state there in or certainly the kind of behavior that they demonstrate but identity is third dimension to an object where you will need to somehow be able to distinguish objects and 2 objects having the same state can still have different identities and then object who has lost its identity is lost for the system and so in the design all these aspects of identity will need to be born

(Refer Slide Time: 28:06)



And now we come to the final conclusion so this is I would like to conclude with this nice kind of cartoon from the OOAD book it says what is a state and what is a behavior, the state basically says all these grade 1 kind of handle, this steel head and so on to the to those of the values, the properties who defines the state of a hammer, this is certainly the behavior the behavior of the hammer is to hammer in the nail and the identity is this is this is my hammer which stand still amongst the pool of different hammers. A symbolic diagram a representation but very clearly thinks the core point of the nature of an object.

(Refer Slide Time: 28:59)



To summarize on the module, we have seen that an object has state, behavior and identity. A state is defined by the values of its properties and we have seen different variations on that in terms of static property and property that have dynamic values and so on. the behavior is defined by the collection of

operations, we have seen what are the different common operations that a behavior should include and we have also talked about how the role and rules of a behavior should be properly identified to designate the responsibilities of the associated operations that every role must have

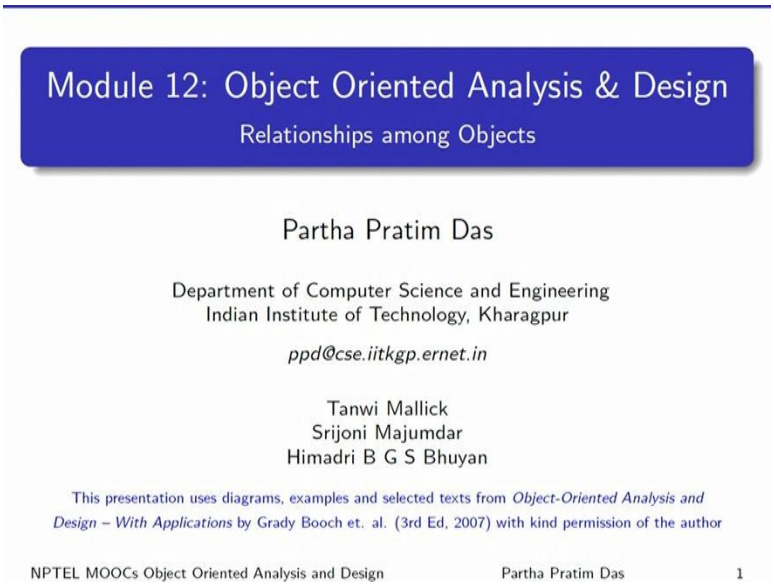
And finally we have seen that identity provides a distinguishability and it could be inherent to an object if there is one or more distinguishable properties but in many cases but in absence of such key properties, an object may need to carry an identity which is either provided by a dynamic framework supporting the whole object based implementation or merely by the location of the object or the address of the object and there are potential risks in terms of those wham of which we have already shown.

So finally, the objects occupy space either in the physical world or the computer memory and it is very critical to keep in mind that an object must exist, must interact and must be distinguishable.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology -- Kharagpur

Lecture – 20
Relationships among objects

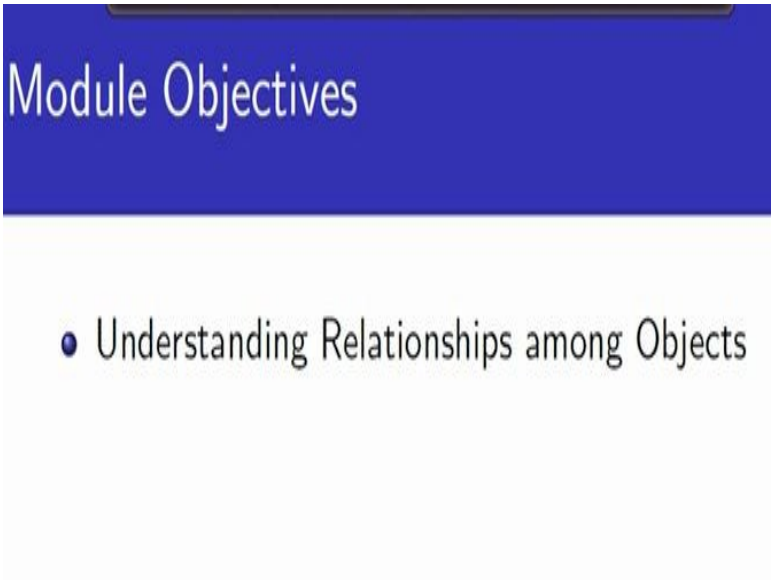
(Refer Slide Time: 00:29)



Welcome to module 12 of object-oriented analysis and design. In the last module we have discussed about the nature of objects. We have noted that an object is typically characterized by its state, its behavior and its identity. So with that we are now ready that in our system we will have identifiable objects as many of different objects as we need to and each object will have its states which can be in terms of static properties and dynamic values.

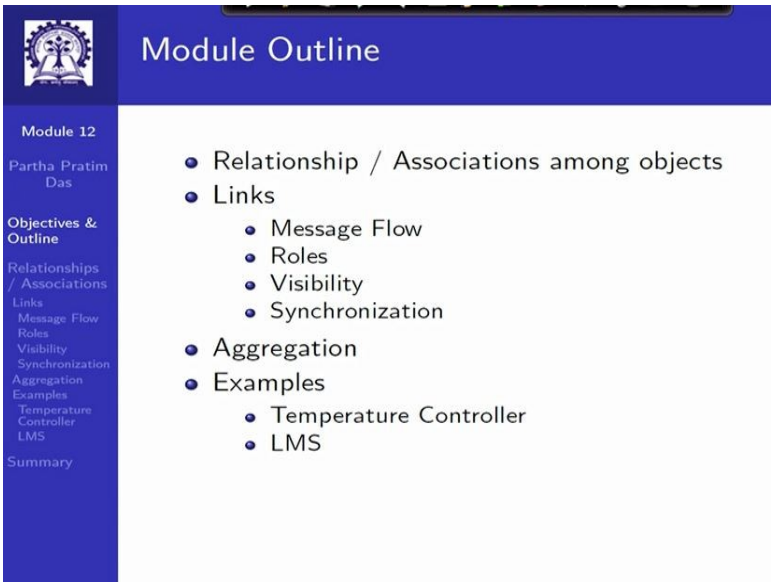
And each object would manifest the different behavior for providing services to other objects and they will interact to build up the system.

(Refer Slide Time: 01:39)



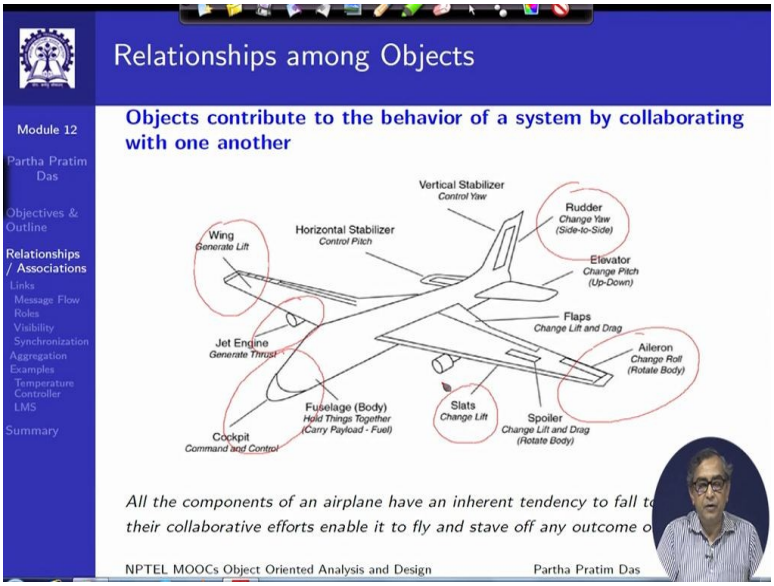
In the current module we will talk specifically about relationship between objects. So our objective in this module understands the relationship amongst objects.

(Refer Slide Time: 01:43)



This is the module outline as usual this will be visible on the left hand side of every slide so we will proceed according to that.

(Refer Slide Time: 01:54)

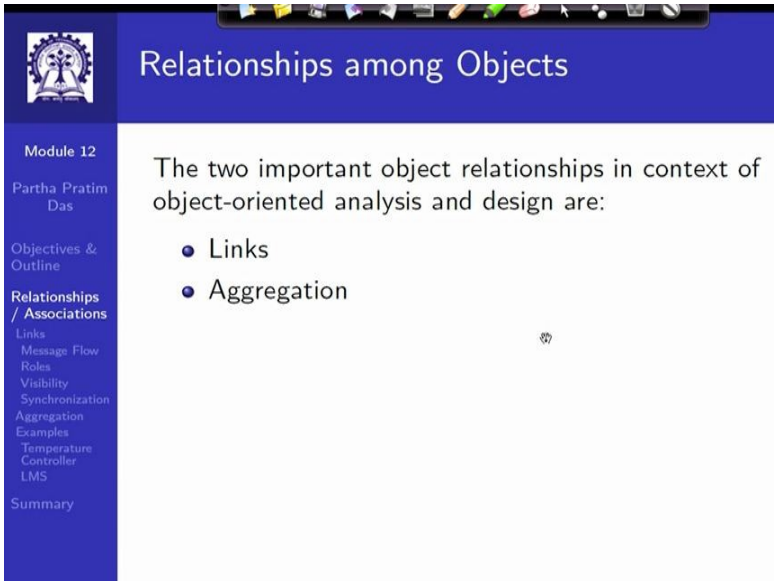


In terms of relationships of objects, the point that we note that in the whole system we will have several different objects that would exist. But the object one or more objects in isolation does not provide any interesting system. This is not an interesting situation to analysis or design. But when the objects come together they get interrelated they get associated link in different ways then they start showing really interesting properties.

They start giving capabilities which are critical for the total complex system that we are designing. Say for an example, here I just show an aircraft. This is as you can see this is the whole of the aircraft and it has a different component object say the cockpit, the engine, the wings, two sides, the fludders, and the slates and so on. The several different objects exist in that airplane.

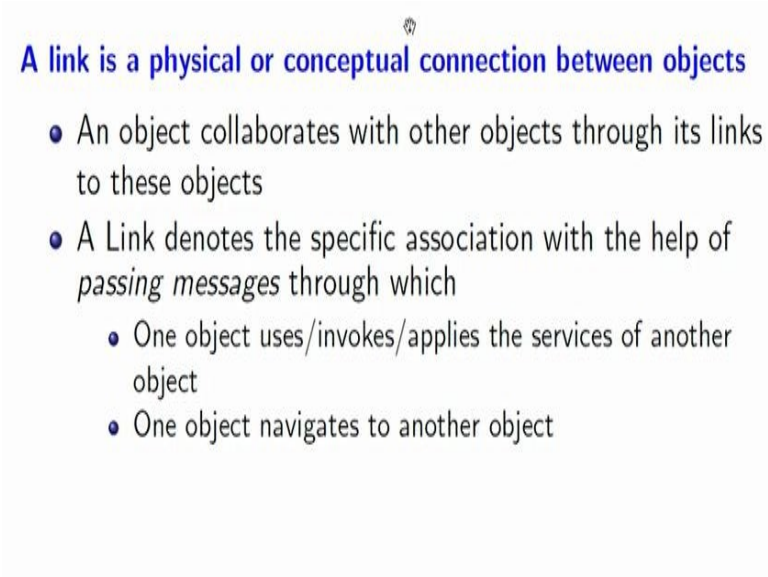
Now if we look into this object individually they themselves have some complex behavior but with each by themselves do not really achieve anything till we put them all together connect them all together into a single airplane which can fly and carry people. So the relationships primarily has to talk about how the objects can get interconnected and behave between themselves.

(Refer Slide Time: 03:47)



In terms of object-oriented paradigm, we talk of two important relationships for objects one that is called links and often they are also referred to as associations and the other is aggregation.

(Refer Slide Time: 04:08)



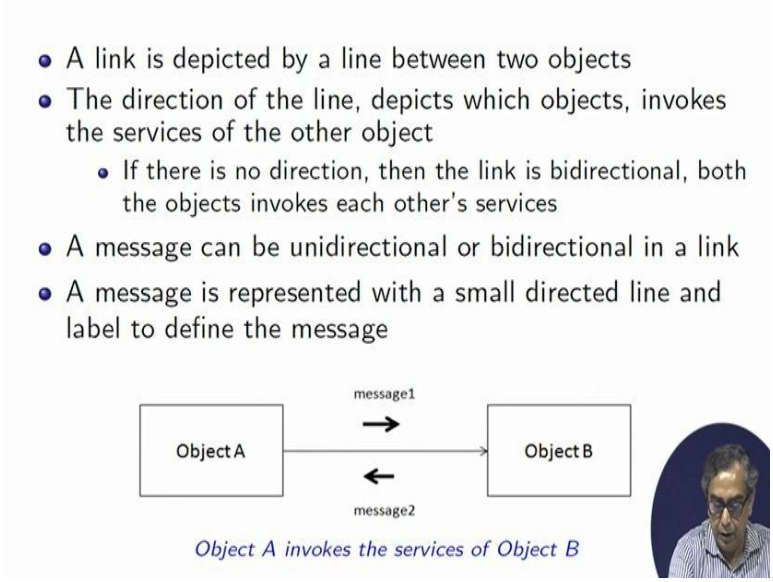
So first let us talk about links. The link is a physical or conceptual connection between objects. So I may have two objects which are physically connected like I have just talked about the airplane. The cockpit, the body, and the engine of that aircraft are physically connected so they are linked or there could be conceptual connection between two objects. I am an individual so I may be described in terms of an object which has a name, date of birth, designation, affiliation and so on.

I have a bank account with some bank where my account has the account number the type of account, the transactions, the different account balance and so on. So there is a relationship between me and my account this is also a kind of link but this link is kind of a conceptual image. There is no physical link between me and my account but certainly I am related to my account. My account related to me. So that is a basic notation of links.

An object collaborates with other objects so it says this in formal terms an object collaborates with other object through its links and this links if we refer back recall a basic model of object-oriented systems. The model is the client-server model where two objects one needs a service and the other which will provide the service. The one which we call the client which needs the service and the provider the server object will pass messages between themselves to achieve what is required.

So links necessarily show that where the messages can pass from one object to the other. It could be a use kind of a message invocation kind of a message or application for service kind of a message or it could be just navigation from one object to the other. So link basically is a representation of the interrelationships between two or more objects.

(Refer Slide Time: 06:34)



In this links are more often depicted in some graphical notation and we will start introducing this and slowly we will see at a later point that refinements of this notation will result in to our

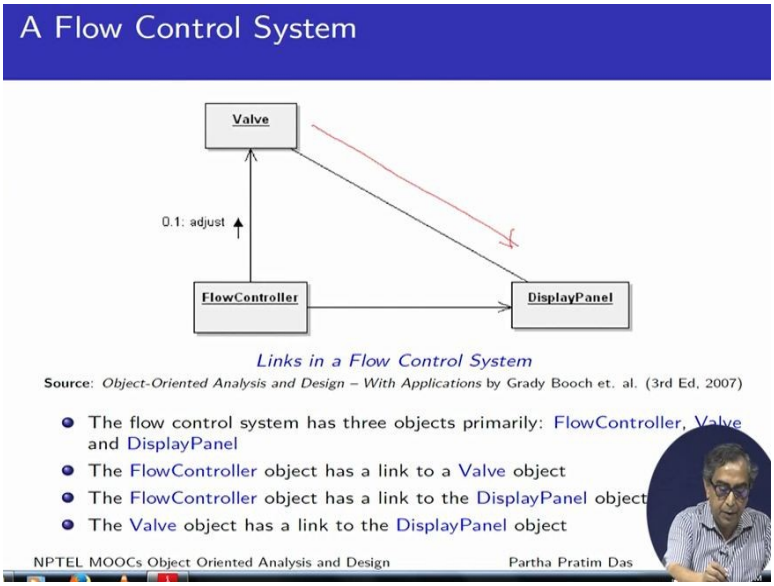
unified modeling language diagrams. The UML diagrams. So here we are just starting with every simple notation where I have one object which is just written shown as a rectangular box. There is another object and there is a link connecting them.

So this says that there is a link there is an association between object A and object B and in this case please note that there is a direction given so it is the link is direct one. It says that object A can ask object B to provide a service. So the object A can pass a message or invoke services that object B provides. So that is done in terms of sending the request message from object A to object B.

And so there is two kinds of directionality that is involved here one is the direction of the link which shows who is requesting this object is requesting and this object is responding. And then on top of that we are showing the direction in which the message goes certainly the client object who is requesting has to send a message to the server object so that direction is same as the direction of the link. Now in some cases the receiver object or the server object will need to send back some result or some response.

So there could optionally be a message which goes in the reverse direction of the link as well. So on one side the link shows who is the client? And who is the server? Who is requesting and who is serving? The message direction on that could be in the direction of the link that must be there because that is how the request is made. But there could also be bidirectional flow of messages over a particular link.

(Refer Slide Time: 09:00)



So with this let us look at a very simple example so this is we are talking about flow control systems has three objects. There is a valve. So it is basically the application is like this possibly there is a pipe through which water is flowing and that flow needs to be controled that you can increase the amount of flow. You can decrease the amount of flow and so on. So the object through which you do that is called a valve.

And certainly there is an object which we model here as flow controller who decides as to how much flow will happen so that controls the valve so it can adjust the flow that is happening and the third we are also talking about a kind of object say like display panel where we can see what is the current flow that is happening. That is kind of a simple model so you can think of I am specifying this in terms of a flow control system.

The similar thing you find in terms of the electrical meters that we have at home and so on. So if we look in to the specifics of this so it has three objects as I have already mentioned. The flow control object has a link to the valve object. I am talking about this particular link and the direction of this link says that the flow control object can sent messages to the valve object. So this will be messages like adjust.

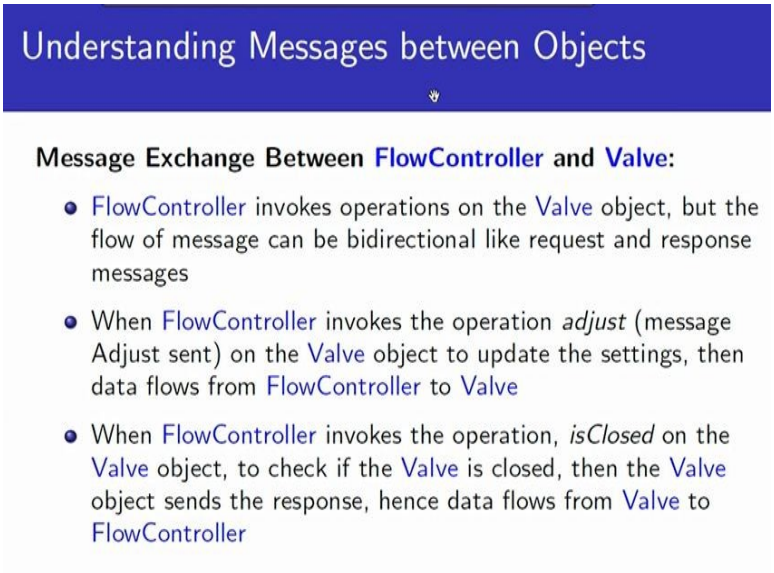
So the flow control will say that you adjust you increase the volume decrease the volume. You can increase the volume by certain number of units and so on. So the link from the flow control

object to the valve object shows that this can work as a client and ask while this works as a server ask it to adjust the flow. In a different way there is a link from the flow control object to the display panel object.

When the flow control object has adjusted the flow the flow control object would sent the message to the display panel say that to display that this is going to be the flow that is going to happen. Or it could also be that the flow control sends message to the valve saying that this is the adjustment and then valve as it gets adjusted to the new flow sends a message to the display panel and shows that message.

So that this link shows that there could be a message from the valve to the display panel as well. So this is the basic way to show how different objects can be linked between themselves.

(Refer Slide Time: 11:56)

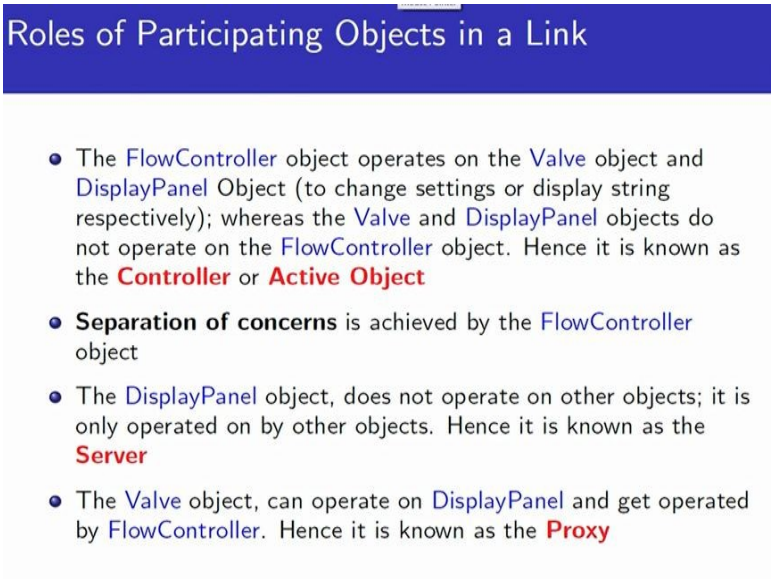


So if we look in to the messages that exchange the flow controller invokes the operation on the valve object here again please note that the message can be bidirectional because it is requesting the valve to do something. So say if it is adjusted then possibly the flow controller will simply say okay adjust increase it by two units, decrease it by three units something like that. But suppose it wants to send a different message to the valve object saying it is closed.

So it is trying to check if the valve at all is working or it has been closed down. So naturally the flow controller need now in this case expects a response back from the valve object as to what is the state of the valve object is it flowing or is it closed it is stopped. So even though there is a single direction which is sending the message is closed it will expect a message that which will say may be say true or false something like this.

So the message flow in this case could be bidirectional. On a single link of one direction there could be multiple directions of messages that can fly between them.

(Refer Slide Time: 13:16)



Now we also characterize so this is we just took a look at how the messages can flow. How that will depict in terms of connecting the objects through links now from this link diagram of message flow we can talk about what is a role that this participating objects play in terms of a link. So some objects like the flow controller basically sends out messages to others to get some service whereas it by itself the flow controller does not serve anyone so such objects are known as controller objects or active objects.

They kind of control the system; they kind of drive the system. We will see more specifically as to what is the definition of active objects what characterizes flow control in this case is a fact that other objects do not operate on it. But the flow controller operates on the valve and the display

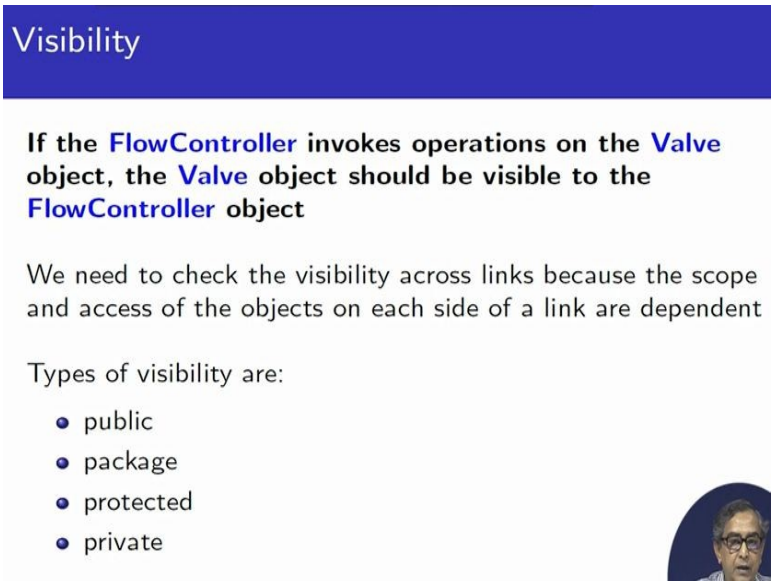
panel. The display panel on the other hand does not operate on any other object. It does not ask for a service from anyone else.

It will just provide service to others that is whether the flow controller asks it to display some information whether the valve asks it to display the flow rate it will comply and display that. So such objects which only provide service are known as the server objects and if you look at the valve object, the valve object actually does kind of both. It provides service to the flow control because it can receive message from the flow control to adjust the flow to respond to flow control as to whether it is closed or open.

At the same time, it could send request to the display panel to display the state of the valve to display the amount of flow that is happening in the valve. So these are kind of objects which at the same time can operate on some other object and can be operated by other objects such objects are known as proxy objects. So we have seen that we have primarily three roles controller or active object flow who only asks for services.

These are commonly called clients. We have server objects who would predominantly provide services to others without asking from service from others. These are known as server objects and the proxy objects are kind of mixture of them. The identification role we will see later on would help us in actually finding out how this different objects should be implemented.

(Refer Slide Time: 16:25)



The next parameter that we need to know is that the objects also has visibility factors. For example, the flow controller has to send a message request to the object valve. Naturally that means that the flow controller must be able to see the valve object. Flow controller must know where the valve object exists and it should be able to send that message. Visibility basically means that whether it can actually send the message whether it can actually send the request.

So it is important that between the objects the links alone will just say whether there is a possibility of sending a message but will also need to make sure that in the whole pool of objects that exist in a system there is proper visibility between them. And there are four types of visibility known by public visibility, package visibility, protected visibility and private visibility. We will not elaborate on this a lot here.

But we will come back to this in a later module after we have discussed about the classes of different modules.

(Refer Slide Time: 17:56)

Synchronization

Whenever one object passes a message to another across a link, the two objects are said to be synchronized – they are *Active* and *Passive* Objects.

Object Type	*Illustrative Behavior*
Active	• An object which instigates an interaction in a thread • It can change its state by itself and send a message to other object if necessary • It is responsible for handling control to other objects • It is referred to as a **Client**
Passive	• An objects which passively waits for the message to be processed • It is activated when it receives a message from other object • It waits for another object that requires its services • It is referred to as a **Server**

The next concept in connection to relationship amongst the object is the concept of synchronization. And this is a concept which you need to understand very carefully because this is little different from the typical programming concept where we know that we have written a code and we start executing it if you have written it C or Java kind of language then we know that there is main function from where the execution starts.

Different functions are called and when the task is over this particular main function terminates. But in terms of an object-oriented system when that comes in to be please remember that we have multiple objects. So if we try to kind of look at the model we have multiple objects I have drawn similar figures very often. We have multiple objects and the objects are sending messages they are sending post back response and so on.

So what we need is when object one who is the client is sending a message to an object O2 then both object O1 and O2 must be in execution. That is unlike the typical programs that you may have written and executed earlier which are called sequential programs or single threaded programs. Here we need different executions happening associated with different objects. Because in this time when O1 executes O1 is doing its operations at the same time O2 will have to do its operations as well.

So these are typically called distributed systems or concurrent systems to be more precise. And whenever you have that there is a need for synchronization because if two threads or more threads are trying to work on the same objects or the same set of objects because there are different objects running on them then there is always a question of issue of whether the operation will be correct whether the operation is performed in the right way or not.

So we will soon discuss an example to illustrate this further but before that let me introduce two different types of objects in the context of synchronization. They are called active objects and passive objects. An active object is this is the definition and an active object is which instigates an interaction in a thread. That is what it simply means that an active object by itself will decide that something needs to be done and we will start doing that.

So the thread on which the active object runs will by itself manage change its state and for doing whatever it is to do send message to other objects. Naturally they are responsible for handling control to other objects because they are deciding what needs to be done. In other terms we will call them as clients. Like may be some of you may have had the situation to deal with the client situation where the client is subscribing to a project pays you for the project and you have to provide service to the client.

Naturally you know that you don't decide when things will happen how things will happen? The client will decide that and based on that you will have to react. So concept of active object is very similar to that. In contrast passive objects are passively waits for messages to be processed. They by themselves do not start doing anything they by themselves do not change their state. They are just in threads which are just listening if there is some message from a client that is arriving so they get activated when they receive a message from another object.

So in this process once they receive the message they know what service needs to be provided to the client they just provide that service and accordingly the states of the passive objects get changed by what the active objects send messages to them. So these are known as the server objects.

(Refer Slide Time: 22:46)

Synchronization

For objects with multiple threads of control, synchronization needs to ensure mutual exclusion to resolve issues of concurrency

Methods of concurrency used for a link between an active and a passive object:

Sequential
- Passive object linked to a single active object at a time
- Active objects must coordinate outside the object so that only one flow is in the object at a time (*function invocation*)
- For multiple flows of control, the integrity is not guaranteed
- One **Printer** (passive) – one **Print Task** (active)

Guarded
- The invoking object (Client) coordinates for mutual exclusion
- Integrity is guaranteed for multiple flows of control – serialize all calls to all of the object's guarded operations
- Exactly one operation at a time can execute on the object, reducing this to sequential semantics (Beware of deadlock!)
- One **Printer** – one **Print Task** uses a queue to serialize

Concurrent
- The invoked object (Server) coordinates for mutual exclusion
- Integrity is guaranteed for multiple flows of control – multiple flows access disjoint sets of data or only read data (Crafty Design)
- One **Printer** uses a Spooler – multiple **Print Tasks**

So given that there are different situation that need to be synchronized now I just mentioned that we need to understand what is synchronization all about? And what could be the issues? Let us take a situation which I am sure many of you if not all of you would be familiar with suppose you are working in an organization or in your college there is a printer which is connected to the network. So I have what I have is a printer which is connected to the network. This is a network.

And different computers are connected C1 is one computer, C2 is another computer; C3 is another computer which are all connected to the network. These days we always have computers connected on the LAN or Wi Fi and so on. Now in these computers different independent people are working so they decide to send files for printing to the same printer. There is only one printer. So let us assume that C1 has requested to print file 1 and C2 has requested to print file 2 at the same time.

Let us say file 1 has 10 pages this has say 20 pages so both of these request arrive on the object print. Now you will understand that certainly every page of the file that needs to be printed must be printed in a consistent matter. Certainly you do not want that you print two pages of file 1 and then three pages of file 2 or if you are doing back to back printing the situation can get even worse. You on one page you print a page of file 1 and in the back you print the page of file 2.

You would like to avoid that. But the request has come together and the request will need to be seareed by the printed at the same time. So we need to make sure that even in this situation where C1 the computer 1 requesting for the printer to print file 1 conceives as if the printer is exclusively available to C1. C2 at the same time requesting the printer object p to print file F2 also thinks that the print belongs only to C2.

C3 for now is not printing but C3 also thinks the same way. So we need to make sure that in this context things can happen correctly. That is all this request should not get mixed up. So the process by which we make sure that these printing tasks or printing file 1 and file 2 would be done in some way. The request can come in any arbitrary order but they must be serviced in a certain order so that result is consistent which means that either I print file 1 complete it.

Then I print file 2 or I print file 2 complete it then I print file 1 or if the request to print file 2 comes when I am already printing file 1 that request will somehow have to be put aside till I am finished with printing file 1. So what we need to do this is known as in terms of concurrency studies or in terms synchronization this is known as mutual exclusion. Mutual exclusion means that if there is more than one request which need to be service by same object at the same time then they will be serviced in certain serial order.

That is the two request will have to be mutually exclusive certainly also please note that not all request need to mutual exclusively. For example, if an object has a value and two other object send request to read that value that can happen simultaneously because it does not matter how many times you read a value from the same variable location you get the same value. But the situation will require mutual exclusion when you need to write that value at that place.

So that needs synchronization that needs mutual exclusion to be put in place and there are three basic strategies to do that for concurrency. One is sequential where there is a passive object that is providing the service and it is linked to only one active object. So the object decides when it is to send the request to the passive object and therefore passive object has nobody else the passive object will service.

So in the context of the example we were talking of that means that you have a PC and you have a printer. Your printer is directly connected to your PC there is no one else. So the print task the active object can use the printer object the passive one directly which this situation is like a function invocation as if the calling function and this is the called function. This is the simplest situation of synchronization which is sequential.

The second could be what is known as the guarded concurrency. What it does is the requesting, the active object the print task will know that well there are multiple files to print. So what it means that the computers C1 and C2 will not send the request directly to the printer but it will send it another object say a print task or print manager kind of object which having received two request can make some kind of queue between them based on the arrival and then send them one by one to the printer.

So looking from the passive object printer it gets tasks one after the other whereas the print task which is an active object which actually received the request is managing how to put multiple request in a serial order so this print task is basically providing the guard that two request will not lay in the printer at the same time. So this is the situation a guarded concurrent is a situation where the invoking object or the client object manages the mutual exclusion.

And in that last case it could be that if you are aware most of the printers these days have an internal mechanism to queue up requests. These are called printer spoolers. So it could be that your printer is advanced enough to have a spooler so it does not matter the passive object printer could be given two request at the same time but it will internally do a spooling. It will internally do a queuing and manage the sequencing of the two request.

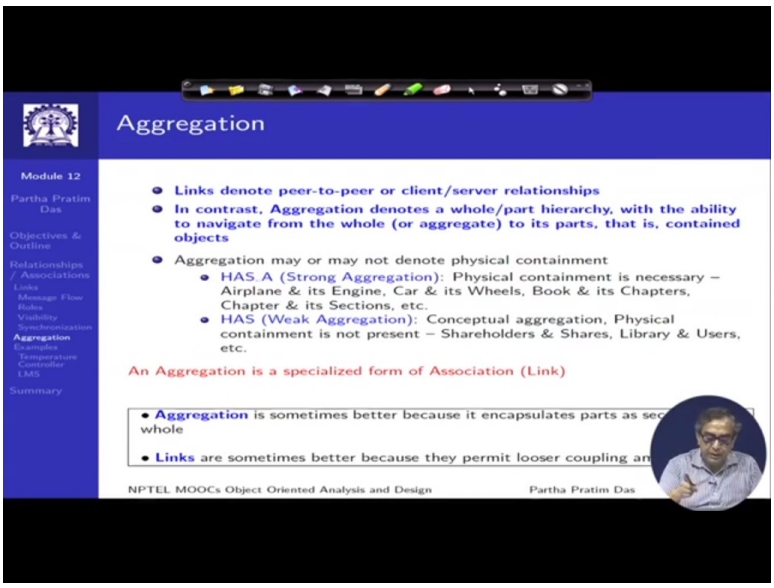
So this is a situation where the invoked object in this case the printer or the server object actually manages the mutual exclusion and the multiple print tasks would just blindly shoot print request to the printer. If this situation exists, then we say it is a situation of concurrent access. So we have seen three kinds of concurrency sequential, guarded and concurrent.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 21
Relationships among objects (Contd.)

Welcome back to module 12 of object oriented analysis and design. We are discussing relationships among objects and in the earlier part, we have discussed the linked relationship between objects and talked about the links to for message flow. We have discussed about different rules that objects can have, controller, server and proxy. We have discussed about visibility and three kinds of mutual exclusion control for synchronization sequential control, guarded control and concurrent control.

(Refer Slide Time: 01:10)



Next we will talk about the relationship of aggregation. So aggregation denotes basically denotes a whole part hierarchy. Some kind of a whole/part hierarchy which gives us the ability to navigate from the whole or the aggregate to its parts that is the content objects. So the aggregation is a relationship where one object necessary in some way is a collection of other objects or one object is a some way a component of other objects and so on.

But the interesting point for in aggregation or the relationship aggregation is that it may or may not be denote physical containment. What does it say that aggregation basically can be of 2 types? One is where the physical containment is necessary. I talk about an airplane object, I talk about its engine object naturally the airplane has a physical containment of the engine. Car has wheel object, book has chapters, chapters has sections, sections have paragraphs, paragraphs have sentences, sentences have

words so on so forth.

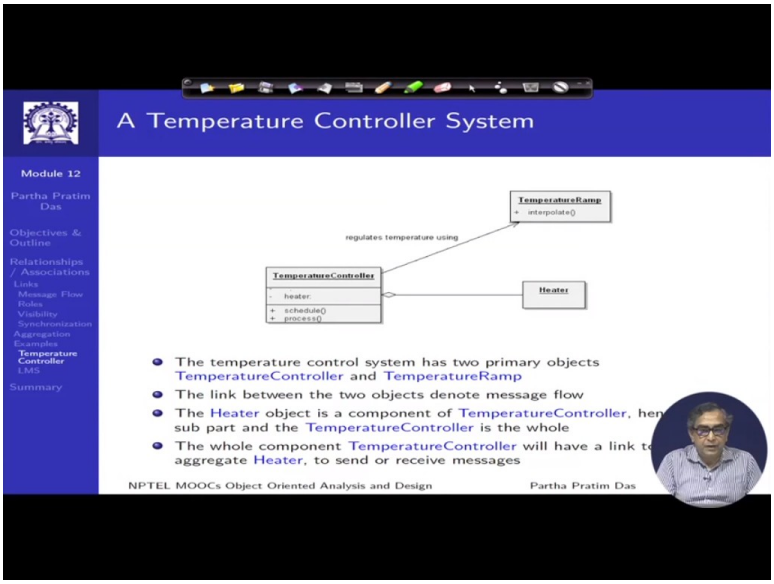
So these are all physical containment and the other could be conceptual aggregation of the kind that I talked about in the last lecture of this module is where the physical containment is not present but there is a conceptual containment, conceptual aggregation in the sense that the relationship between shareholders and the share. Both are objects, they are related but share is not a part of the shareholder and vice versa. Library has users but users are not part of the library. They are not components of the library.

But when you talk about library and books then there is a physical containment. So both of these are known as aggregation and since physical containment is a is a stronger sense so an alternate name for physical agree containment is strong aggregation and the conceptual one is more commonly referred to as weak aggregation. You can realize that we have several times talked about has-a relationship. So strong aggregation necessarily has a relationship and many of us refer to the weak aggregation as simple has relationship.

So now we have aggregation and links so if you ask us to what what should be the way you should model that is should you model in terms of the strong one or the weak one whether containment is better, physical containment is better, whether just associations are better and so on. The answer is both of them exist because there are different situations for their use. It is in some cases aggregation is better because it keeps the objects within itself as physically contained parts.

So there by the principles of encapsulation they are encapsulated secrets of the object that is all of us have seen aircrafts but we have not seen their its engine but links may be better in some other cases because links do not physically contain an encapsulate. So they provided a more flexible loose coupling between objects. So this is the basic notion of aggregation to work with.

(Refer Slide Time: 04:52)

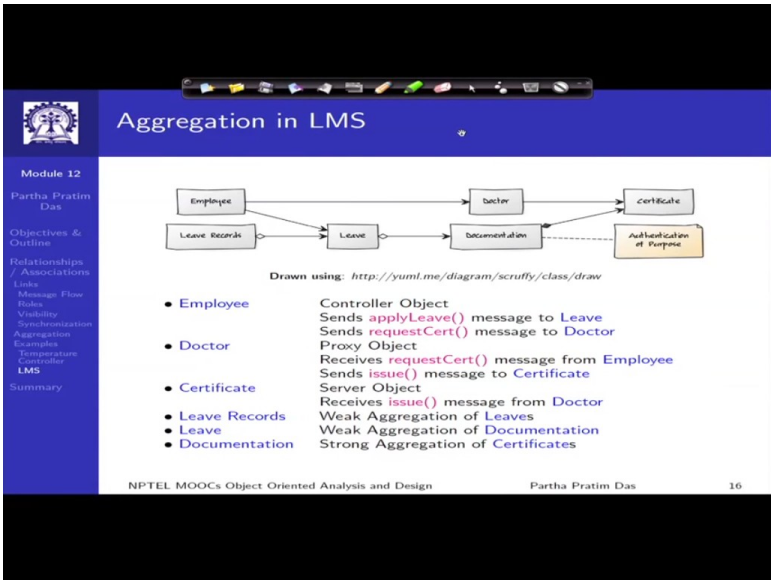


So we will just take a look into a couple of simple examples. Here is one, example of a temperature controller which has three objects certainly this one; the first one is the controller which decides to maintain a certain temperature profile for an industrial process. The temperature profile that needs to be maintained use an object itself which we say is a temperature ramp that is how the temperature should increase, how should it decrease and so on.

And certainly to actuate this whole process, we have a heater which can be turned high, turned low and so on. So if we look into these, then here we see a link where the temperature controller sends message to the ramp saying that this is the kind of profile that you need to schedule, this is a time when the temperature should start increasing, this much it should increase and so on so that information can be sent in terms of request from the temperature controller to the temperature ramp.

But if you look at the heater, heater necessarily is a part of the temperature controller; it is a component of the temperature controller. So there is an aggregation in terms of this heater which exists in the temperature controller system. So this is we just took we are just taking very simple examples to illustrate what different relationships could be in a real system.

(Refer Slide Time: 06:32)



Let us look at look back at our leave management system so this is a simple, a partial model of the leave management system so you are already aware of many of these objects like certainly I will have leave management system so I will have leave object. The leave is for the employees so we have employee object and am just depicting a situation where as you have studied the there are number of leaves like the sick leave and maternity leave and so on where we need separate documentation.

We need certificates from doctor, we need medical reports and so on for that leave to be approved or for that leave to be placed in the system. So if we look into this that part of the LMS system only a part of that which has some of the objects then we expect that there will be a there will be certainly an employee object. Now certainly the employee in this case will decide whether he or she wants to apply for leave or if the employee is an, is a lead or a manager then he or she would decide whether some leave should be approved or not.

Which means in essence it means that these employee object is kind of the driver in the system? So we will immediately say that this is a controller object. Now it sends messages along this link, it sends messages along this link so the message along the employee to leave link could be apply leave message where an executive or a lead is applying for a leave. This link from employee to a doctor could be request certificate link that me employee has not been keeping good health.

So goes to the doctor and request for a certificate. If you look at the certificate, the doctor will issue the certificate in this link based on the status of the employee, the health status of the employee I mean. So if you look at the certificate object here then the certificate object receives a request form the doctor

that a certain type of certificate will have to be issued. So this will have to be printed or electronically signed and so on. So it just serves that request, so we say it is a server object which receives the issue message from the doctor and serves it.

So what is the doctor object? The doctor object is on one side, it receives the request certificate, the one that we saw here from the employee which comes on this link and on the other side it will send a request for issue to the certificate object on the other link. So it is in both so we identify that the doctor object is a proxy object in this situation. Now if we look into the leaf so certainly an employee has applied for a leave that is not the only leave object in the system, surely there will be several other requests from the employees.

So we can conceive of a collection of leaves that exist in the system and we say this is a leave record. So the leave record will not contain will not encapsulate all the leaves, rather it will be a list with appropriate pointers or references to different leave objects much in the way that a shareholder talks about his share, I talk about my bank account in a similar manner. So you can say leave records as a weak aggregation of the leaves and you can see that in terms of this diagram,

We are showing this aggregation with a certain type of drawing where there is a diamond at the aggregate end which is empty and this emptiness represents weak aggregation. So now if we look at the leave itself then we know that the leave in this case, we need to have the documentation, the documentation of the doctors certificate, the documentation of the different medical reports, may be the x-ray report, the ECG report and so on so forth. So the leave has weak aggregation of documentation.

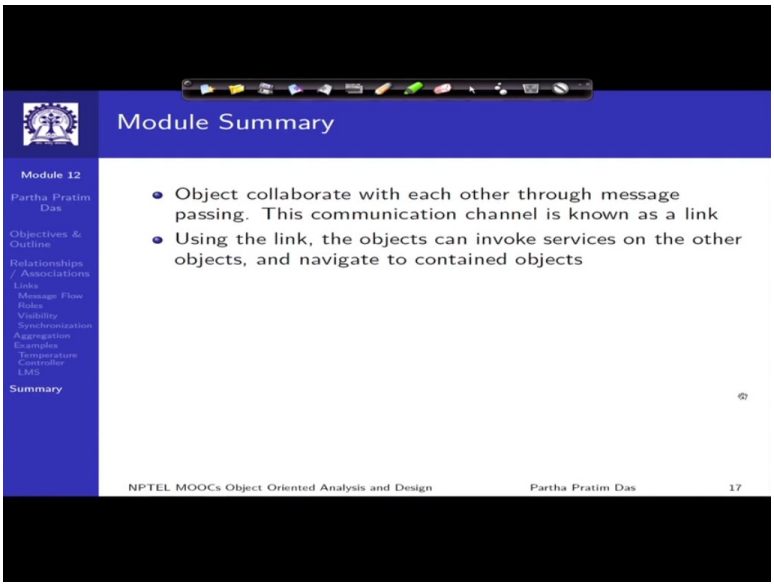
Now leave will not contain the documentation, it is not a part of that but leave will make references to the documentation so this is another situation of weak aggregation as depicted here. Now let us look at the documentation object itself. The documentation is the bundle of the certificate and the different reports. So the documentation is has certificate as a component object because your documentation file which in terms of if you think in terms of the physical situation then it is a documentation file which has papers showing the doctors certificate, papers showing the medical report and so on.

So all of that the reports and all those am not showing here. I just shown the certificate becomes a strong aggregation or a strong or a component of the documentation therefore if you just look at the diamond here this diamond is a filled up one, unlike these diamonds which are empty ones depicting

weak aggregation here. It is a filled up diamond depicting strong aggregation. So this was just referring back to our leave management system to show that the concepts of relationship amongst objects

That we have just understood in this module can be found out found plenty in terms of a leave management system. We are just taken a look into one part of the system you can I would encourage you to look into the whole specification and listen to the tutorial that we have put up on the leave management system and identify more and more object relationships in the leave management system.

(Refer Slide Time: 12:54)

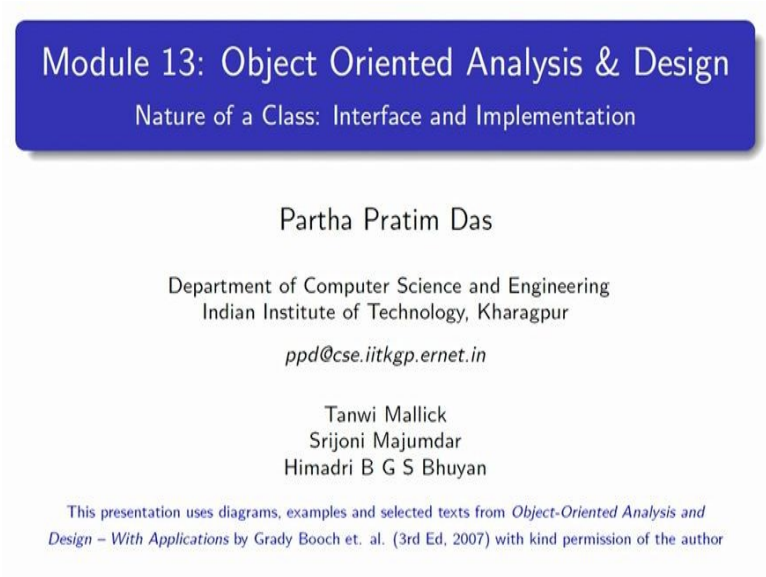


So in this module, we have discussed to summarize, in this module we have discussed the object relationship is how does the objects relate between themselves, how do they collaborate with each other through messages and we have noted that such communication links between object such communication channels between objects is known as links and using the link the object can provide services or invoke services or the navigate from one object to another.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

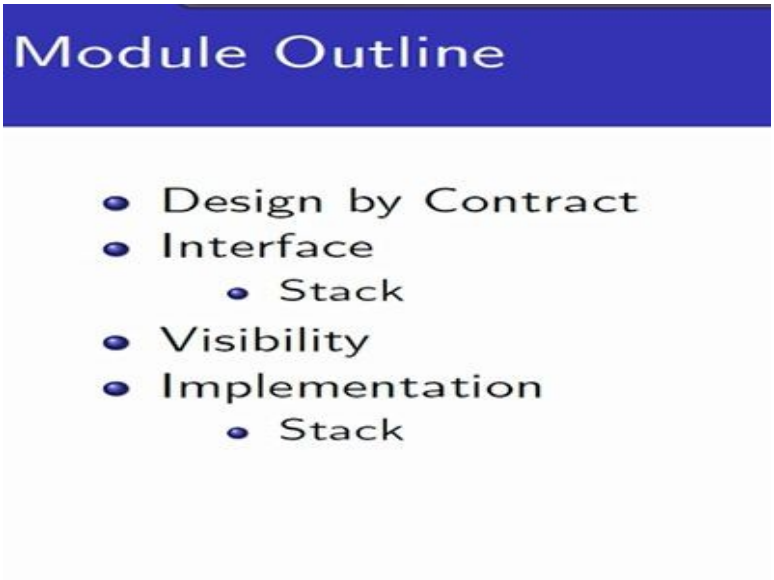
Lecture – 22
Nature of a class: Interface and Implementation

(Refer Slide Time: 00:25)



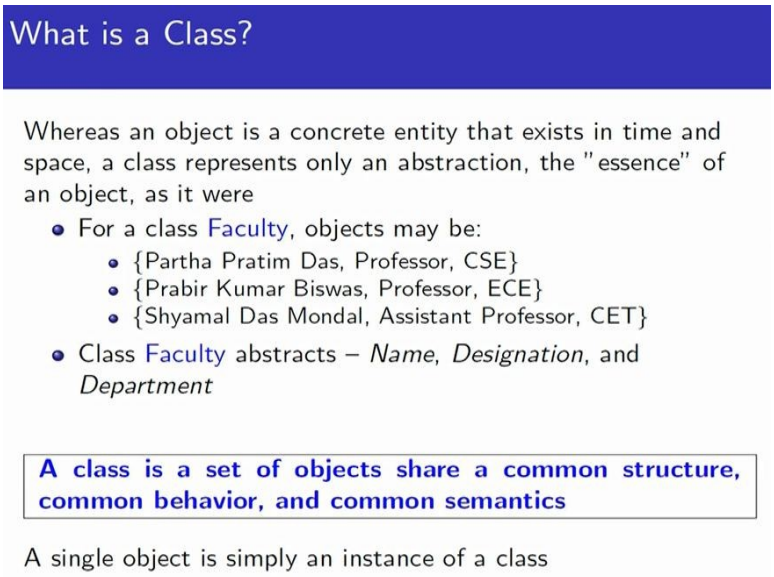
Welcome to module 13 of object oriented analysis and design. In the last two modules we have discussed about the nature of objects and their interrelationships between them. In the current module we will start discussing about the nature of classes. What is an interface and what is an implementation will be the major items to talk about so we will try to understand those from the perspective of a class.

(Refer Slide Time: 00:55)



Here is the outline which will be available on the left hand side of the presentation all through.

(Refer Slide Time: 01:06)



Now I am sure most of you have certain strong idea about what is a class? Still we would like to more formally define that. We have since we are talking about object-oriented systems so certainly we know that the core of the whole system are objects and we have seen we have defined that what is an object and we have seen the object has state, behavior and identity. So object necessarily is a concrete entity that exists in time and in space we have seen that.

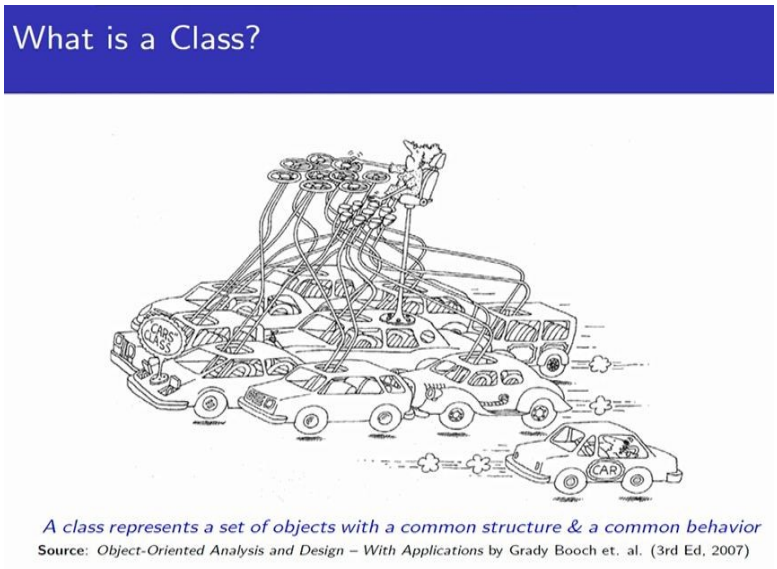
Whereas a class which is closely related to objects is somewhat an abstraction somewhat of an abstract concept which tries to capture the essence of a set of objects which have strong

similarities strong commonality in terms of their structure behavior and semantics. So for example if we consider say a class faculty then. We will say that it has let say for example three different objects three different names of professors, their designation and the department that they are associated with.

So each one of this are objects because they have a concrete existence whereas this class faculty abstracts these objects in terms of the name, designation, department and of course a lot of other behavior lot of other operations that will be common to all these objects. So to put in place a class is a set of objects sharing a common structure, common behavior and common semantics. So when we talk about the commonality of the objects in a coherent manner we refer to the class.

So it is a kind of like a template according to which every object of that class have to correspond to have to adhere to have to confirmed. So we will say that given any class, an instances of the class is an object okay.

(Refer Slide Time: 03:37)



Now of course this is a very nice cartoonist view of what is a class? You can look at it enjoy that.

(Refer Slide Time: 03:51)

What is a not Class?

- An object is not a class
- Objects that share no common structure and behavior cannot be grouped in a class because, by definition, they are unrelated except by their general nature as objects
  - Today's newspaper, Bajrangi Bhaijaan movie, State of West Bengal – are all objects that cannot be put together in a class
- While anything can be a class, the attributes of a complex system should be carefully crafted in designing a class:
  - Hierarchic Structure
  - Relative Primitives
  - Separation of Concerns
  - Common Patterns
  - Stable Intermediate Forms

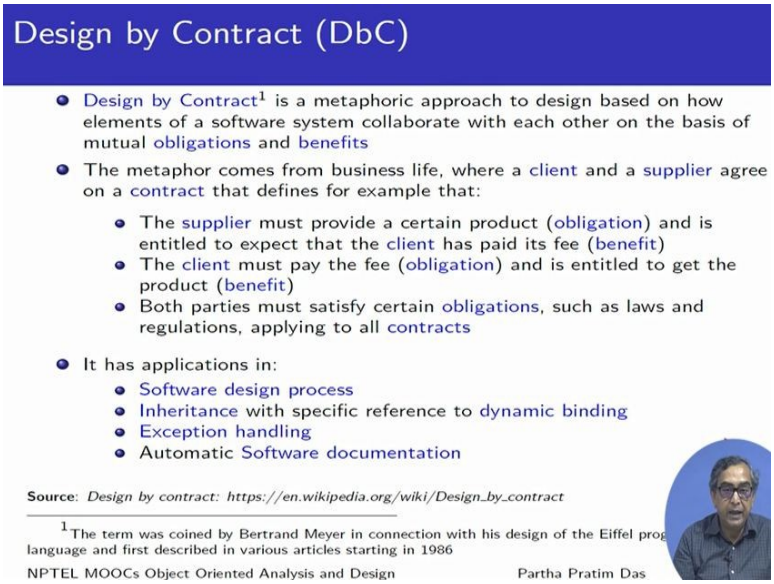
Now if you want to also ask what is not a class? Then certainly we will observe that an object is not a class or the objects a collection objects which share no common structure, common behavior they cannot be grouped in terms of a class. Because the class by definition has to have the common structure and semantics and behavior for example if you just look at the say todays new paper is an object, "Bajrangi Bhaijaan" movie is an object, state of West Bengal is an object.

These all are objects but they cannot be put together in a class. So the class will specifically be in terms of that sharing and beyond that what a class can be is a matter of experience and expertise of the designer who is designing the class. Of course it is possible that if we are talking about an organization as a whole we could theoretically at least represent the whole organization object as one class.

Such a class would be so complex it will have so many different types of interfaces catering to employees, to customers, to vendors, to regulatory authorities, to the management and so on that it will practically become very difficult to deal with to understand to analyze such design of classes. So the point that I would like to remind you about that we will be talked about the analysis of complex systems we learnt that a complex system should typically be analyzed be designed in terms of its attributes.

Five typical attributes are hierarchic structure, relative primitives, and separation of concerns, common patterns and stable intermediate forms. I would just like to remind you and if needed you can go back to that corresponding module and look at that video. Now it is the time that we use this attributes in design of the classes and that will decide what is and what should not be a class.

(Refer Slide Time: 06:22)



Now a class typically has an interface and an implementation. I am sure most of you are familiar with these two terms. But to introduce these I would like to take a little tour of what is known as design by contract? Design by contract is a kind of a metaphoric approach where the concepts are borrowed primarily from the business domain where their clients and suppliers have to work together for getting certain products and to be able to work together there they have to agree on a contract where the client has certain obligations.

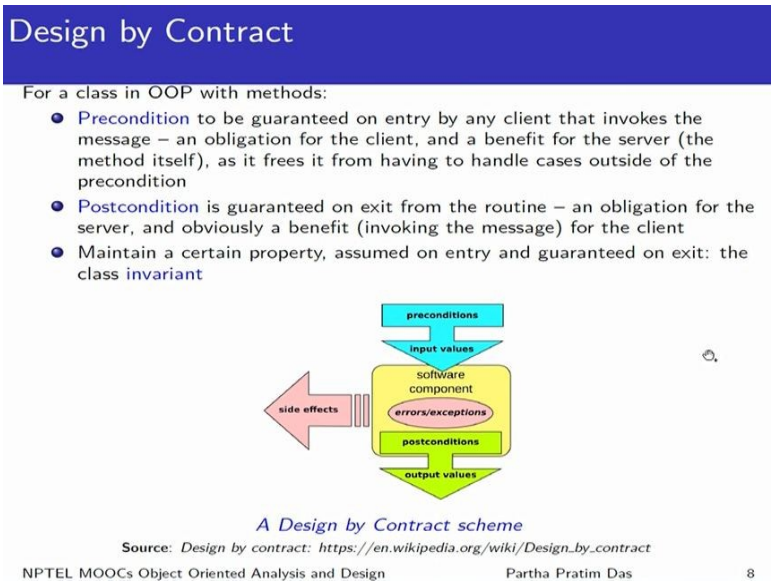
The supplier has certain obligations and the client gets certain benefits, the suppliers get certain benefits and both of them together comply with the contract and keep within the law of the land. Now the reason I refer to design by contract is a fact that when the methods of object oriented analysis and design were evolving and was getting strong grounds then it is mean the middle of the 80's were they design by contract was originally proposed by Bertrand Meyer.

And that has proved to be a very effective approach in solving a wide variety of software design problems OOAD problems and so on. So if we just look at it in terms of the terminology or points that they are between the client and the supply they will be mutual obligations and benefits. And in terms of the contract the supplier must provide a certain product which is the obligation of the supplier to the client.

And it will expect to have a benefit from the client which is the fees that the client fees. Similarly, the client must pay a fees which is the obligation of the client and is entitled to get the product which is the benefit that the supplier provides and both of them have to go by the obligations of the laws and regulations which is a part of that contract. So this is the basic concept that design by contract says if this still is completely in terms of the business domain.

But please note that it designed by a contract or DbC as this is commonly called has formed wide applications in software design, in software automated software documentation particularly in designing, inheritance, polymorphism, exceptions and so on. So we will come to those specifics later on.

(Refer Slide Time: 09:19)



Let us see how do apply design by contract more directly into a software design or a class design. We say that please take a look in to this diagram here we are talking about a software component which could be thought of as an object. It could also be a simple procedure or a

function so we say that this component this object or this particular method of an object is requested with a message with certain input values.

So what will happen if a message arrives to the object for a particular method then that method will take those input values and start executing so that is what is the software component is trying to do and then it will produce output. Now in the process it can also create some side effects certain other thing could happen like there could be some errors there could be some exceptions thrown and so on.

Now what we are saying that in a good object based design the input values must satisfy a set of preconditions that is the client you will remember this our basic model is client-server so the client we sending the message to the server to invoke a message must make sure that the message is good. Let us make sure that the set of preconditions are satisfied by the input values at the same time there is a set of post conditions that must be satisfied for the output value.

So in response to the client's message the output that the server generates or the response message that the server generates must satisfy a set of post conditions. Now if these two are guaranteed then certainly the client and the server can work in a lot more safe and segregable manner. So precondition is to be guaranteed at the entry and the client must do that because this is where the message is coming from so it is coming from the client.

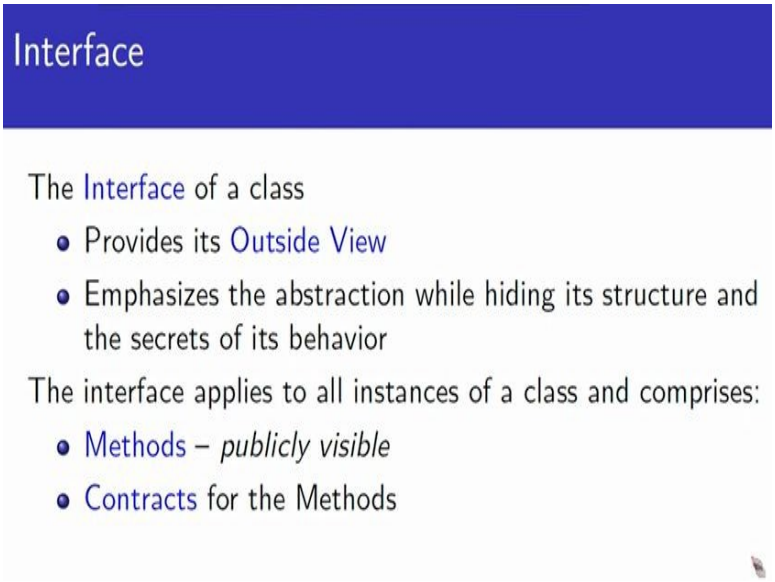
So it is an obligation of the client to make sure that the precondition is satisfied. Against this obligation what benefits does the server get? The server gets the benefit that it now knows that the preconditions are satisfied so while it is implementing the software component while this is implementing the method it does not have to think about any situation which is not protected by the preconditions it can limit itself to adjust the preconditions.

Similarly post condition is guaranteed on exit from the message on exit from the routine, and it is the obligation of the server to ensure that and certainly that is the benefit that approve to the client because he has been able to invoke the message and get a successful execution. And in this

whole process the client and the server must maintain that the class has certain invariant certain things should not change should not be violated in the class all those must be maintained.

So if you see the style of specifying this design or building up this frame work of design by contract you will immediately be able to relate it to the typical business scenario of obligations and benefits catering to precondition, post conditions and the invariance in the class. Okay if this is somewhat sounding difficult to you sounding somewhat vague to you we will soon come up with an example and explain this design by contract method.

(Refer Slide Time: 13:24)



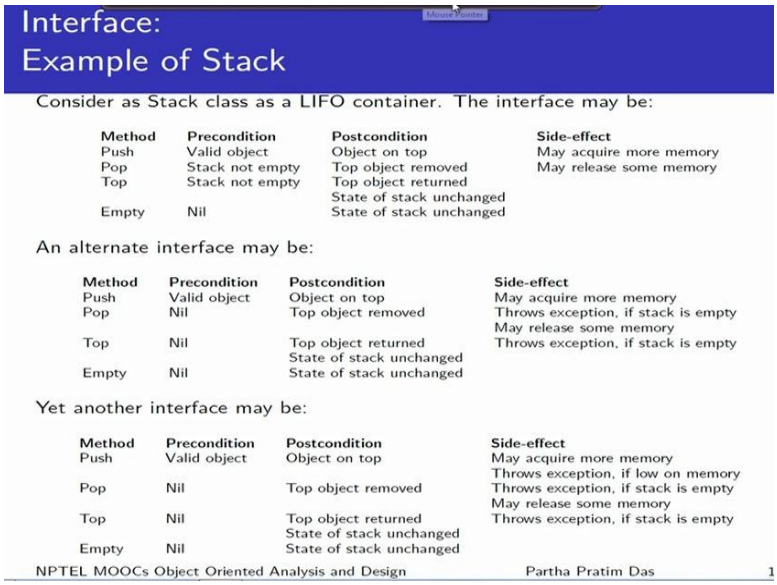
But before that let me introduce what is an interface? We have talked about interfaces even in earlier in number of times so you understand that an interface basically provides an outside view. That is whatever an interface of a class and which will by instantiation will become kind of an orifice through which an object can be operated on is necessarily an outsider's view. But someone from outside that object someone from outside that class can see.

And the abstraction and encapsulation will ensure that all the details the structural details and the secrets of the behavior will be hidden behind this interface view. Now interface naturally has to apply to all instances of the object all instances of the class. So if I say that a class has certain interface then every object that I create of that class, every instance that I do that must satisfy the interface. So what does the interface has?

Certainly this is what you clearly know that it must have the methods. The methods might be publicly visible we have talked about visibility in terms of objects. We will see what does it specifically mean in terms of classes? But in addition to the methods what is important and this is the concept we are borrowing from design by contract is the interface has contracts for methods has certain precondition post-condition guarantee certain side effects of whatever that interface will do.

So interface will simply be the methods publicly visible methods and the contracts of these methods.

(Refer Slide Time: 15:27)



Let us take an example. Let me consider a stack. I am sure all of you know stack. You know it is LIFO container last in first out. So you can add members to a stack by push, remove members from the stack by pop. If you push, they keep on getting stacked and when you pop it is the last pushed element which we will get extracted out of the stack so the order in which the elements enter and they leave the stack or opposite of each other.

Besides so that gives us the two common methods that a stack has push and pop. In addition to that typical we will have another method another part of the interface which is top to get know what is the current top element of the stack and we also have a method empty to check if the

stack at all has anything. So we understand this so given that if we are designing a stack class we will say our interface the method part of the interface at this four methods.

There must be a way to push, pop, top and empty. There could be several other interface functions that could be several other methods also but this is minimum for defining early four. This is minimum for defining a stack. Now let us look at for each one of this let us look at the preconditions and the post conditions and the side effect. So this is the method part and what you see in the rest of the table is a contract part.

This is a contract part. So contract ,so the contract what we have? In the contract we will lead to a precondition so if we look at push let us say let me pick up another color. If I look at push then what is the precondition that is when this push method will be invoked on the stack what the stack object the particular stack object can assumed. The stack object would like to is assume that the object that we are trying to push.

If you are doing a push, then you have some object that you want to push for example it could be a stack of integer when you are trying to push an integer. It could be a stack of strings when you are trying to push a string. So of course say a stack of integer object while push is invoked on that we will expect that you give it always an integer object you don't give it a string object or an employee object or something like that.

So the fact that the object will have to be valid is a precondition for the push. Now what is the post condition what must be satisfied when the output is generated by push or when push message actually completes execution natural the object that we have given to push that must exist on the top otherwise the LIFO will not get realized. So it is very critical that this post condition be satisfied. What is the side effect?

Now it may or may not have a side effect but it is possible that whatever way we are internally implementing the stack we will talk about this more when we come to the implementation part of the class but whatever way we do that we need memory to keep the elements so if we repeatedly

keep on doing push at some stage we will need more memory and that requirement will come up to the system so that is the kind of side effect that we can have.

So if we similarly look at pop we need the fact that I can pop an element provided. There is some element in the stack so the precondition of pop could be thought of as the stack not being empty natural post condition is top objet has been removed and certainly pop may release some of the memory that was used by the last object. Top again will need a precondition that the stack is not empty only then you can get the top most element.

Certainly the post condition is it must return the top object that is what I am trying to find out and the other post condition is in the effect of doing a top should not change the state of the top stack object. The state of the stack object must remain unchanged so there are two post conditions and there may not be any side effect.

If you look at empty, we can say that there is not precondition. I can send empty message to any stack whatever it states and the post condition is the state of the stack does not change there is no side effect. So this is the view of the methods and the contract which together form the interface of the stack class. I would like to warn you particularly those who have already been writing programs in various OOP languages like C++, Java or Python for that matter.

You would tend to often believe or follow that is only the methods only this part is what is the interface? But please understand that if I just know the methods just know the messages that I can send to the object to invoke these methods then I will not be able to do that task properly unless I am careful about the precondition which is specifying the properties on the input the post condition which specifies the properties on the output and the side effect which can happen during this process.

Just to understand that let us say let us take for the stack itself an alternative interface in this alternate interface again the methods are the same. I am not changing any of the methods but what I have changed is I have changed the contract now I say that well in the contract I do not expect that when pop is invoked when a message is send to pop then the client has guaranteed

that the stack is not empty. I might ask for doing a pop or a top on a stack which may not be empty.

I will say that there is no precondition so which means that the stack could be empty and I may be asking for a top. So what will happen now certainly if the stack is not empty then the top object is removed so that is same as what I had. If it is not empty, then some memory may get released but what if it is empty naturally the output cannot be generated or the output cannot be satisfied because there no top object so what I will do I will throw an exception.

Throwing an exception is we will explain that more later on. It is kind of telling the system telling the client-server system that well I was asked to process a message which I was not able to do so in turn I send an exception message to some exception or error handler/error controller. So it will since just consider that in the previous interface the precondition said that stack is not empty. So it was not necessary to consider a side effect where pop could throw an exception of stack being empty.

But here I have changed the precondition and therefore I have a possibility of a side effect where an exception may be thrown for the stack being empty. Similar think will happen for top it could also throw an exception if the stack actually is empty is when you are trying to invoke the top message for that stack. So you can see that just messages being the same does not actually give us the same interface because depending on the contract here and the contract here.

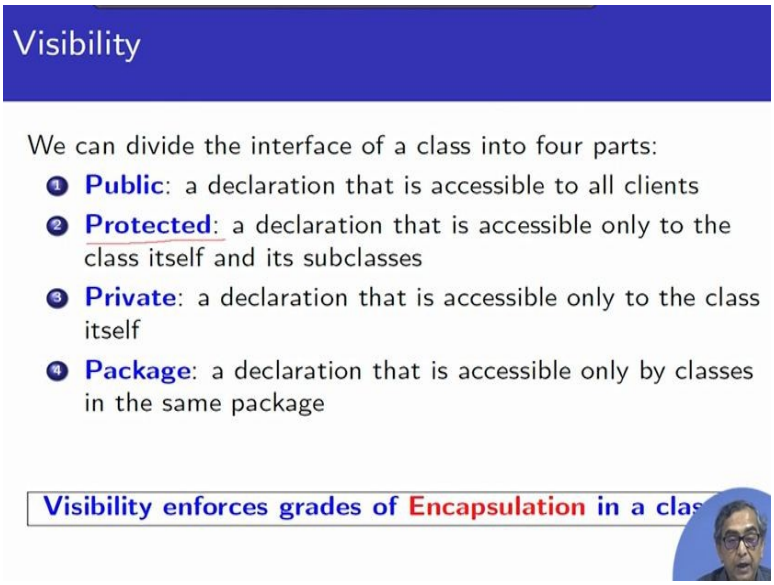
The way the programmer has to implement this method would be very different. You could think of yet another interface so in this interface again we have the same set of methods we have the same kind of preconditions and what we change is we have added in this interface. We have added a specific side effect for push. Here we said that push may create a side effect that I need memory. Push may create a side effect that more memory is needed.

Push here as well will create a side effect that more memory is required. Now stack theoretically is unbounded. I can keep on stacking as long as I want but whatever system I actually use to

implement the stack object will finally have a limited memory. So at some point it will run out of memory. It will become low memory. What will happen to that?

So compare to this interface this particular one the later one is a more, well-specified interface because it takes care of a potential situation that can happen in the system when it actually has very low memory to make a push request/push message actually successful. So this example I am sure have shown you how you really apply the design by contract principles in terms of your interface design and how really different interfaces having the same set of methods but having different kinds of contract will impact the whole design process.

(Refer Slide Time: 25:50)



The next that we would like to draw your attention to is we mention that methods are publicly visible. I would also remind you of the fact that we have repeatedly been saying that we will enforce encapsulation of the objects and certainly since objects are instances of their classes the classes must have a mechanism to encapsulate. Encapsulation basically means hiding certain parts of a class or certain parts of the object that we will get instantiated from other objects from instances of other classes.

So we may note that there are typically four types of visibility or we can say that the interface can be classified divided in to four different parts one that is called public. This is the interface which is accessible to all clients anybody and can use that. And typically when we talk about an

interface for use like we were talking about stack this is based on only the public visibility because anybody should be able to use the push, pop, top, empty kind of interface.

Another is a private one which is accessible only to the class itself not to the outsiders not to anyone else. So in the stack example we could have a method which actually checks if the stack has enough memory when push operation is happening. Now this method will not be exposed to the external clients because they know do not need to be bothered about whether what the stack class or the stack object should do when there is not enough memory to push another element.

This is an internal matter of the stack classes an internal matter of the stack object but stack object actually by itself need to know whether there is enough memory available or not so I could have a full kind of a method which the stack object will send to itself to invoke and check whether there is enough memory such a method would be a private method because that others should not be able to see this.

In between we will also see that there is a class of visibility factors where a declaration of a method is accessible only to the class itself or to its subclasses. We will come to more of subclasses you know we have talked about subclass in relation to abstraction hierarchy so if something is a subclass of another class then protected visibility is for that subclass. So if something is protected then it is accessible to the class but it is not accessible to the outsiders but along with the class the subclasses also can access the protected members.

Finally, some of the object systems introduce another few other kinds of visibility like package is one such visibility where all classes that belong to the same package which is kind of saying that if I put the classes together under one folder package is similar in concept to a folder that all the classes but it is not exactly a folder it is somewhat different. It says that if you put all this classes into a single package then the classes would be able to see each other.

The classes would be able to will be visible across each other. So which say that visibility enforces different grades of encapsulation from very a hard encapsulation of private visibility to

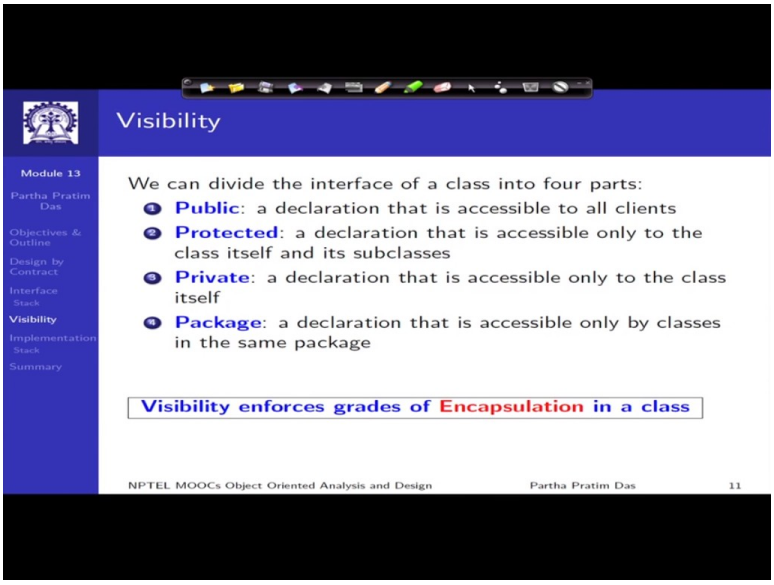
somewhat limited encapsulation of protected package visibility to complete openness of public visibility in to the whole system of objects and classes.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 23
Nature of a Class: Interface and Implementation (Contd.)

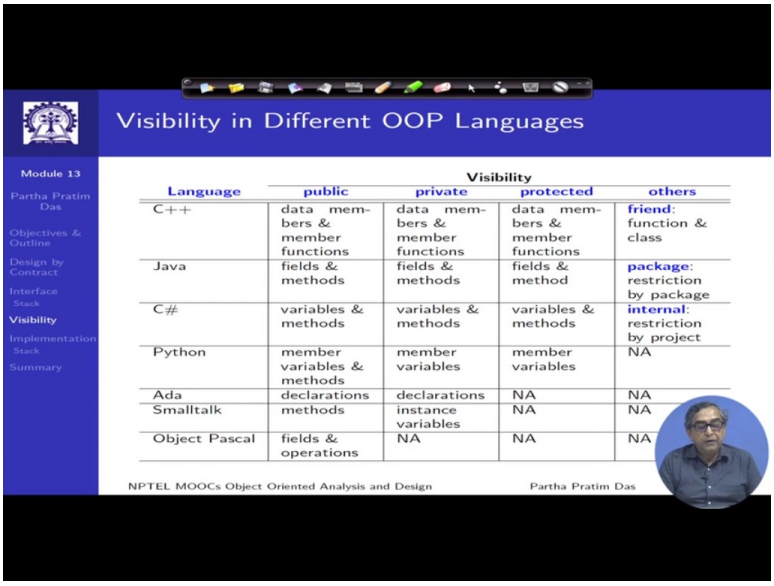
Welcome back to module 13 of object-oriented analysis and design. We have been discussing about nature of classes. We talked about design by contract an how that can be used in designing interfaces of classes and took an example of the stack class, of a stack class to explain the whole idea. Then we also took a look into the visibility aspects of different components of a class, different declarations and noted that primarily there are 4 kinds of visibility commonly used in object-oriented systems the public, protected, private and package.

(Refer Slide Time: 01:09)



Continuing on this I would just like to draw your attention to the visibility as is available in different object-oriented programming languages.

(Refer Slide Time: 01:12)



Language	Visibility			
	public	private	protected	others
C++	data members & member functions	data members & member functions	data members & member functions	friend: function & class
Java	fields & methods	fields & methods	fields & method	package: restriction by package
C#	variables & methods	variables & methods	variables & methods	internal: restriction by project
Python	member variables & methods	member variables	member variables	NA
Ada	declarations	declarations	NA	NA
Smalltalk	methods	instance variables	NA	NA
Object Pascal	fields & operations	NA	NA	NA

Now certainly this course is not directly dealing with object-oriented programming as I had mentioned earlier. But certainly you all are trying to learn object-oriented analysis and design to be able to at the end do programming of the system on some of the common object-oriented languages.

So here I have compiled for some of the common languages which are mostly people use. So if you are using C++, C++ has primarily 3 different visibilities available to public, private and protected both data members as well as member functions. Data members means basically the state variables and the member functions or methods can be into any one of this visibilities. But in addition C++ has another kind of visibility which comes through friend.

It can declare some function to be a friend or some class to be a friend. A friend will have a private like visibility into the inner of the class and therefore it is very important to choose the friends very carefully. What is if I if we move to java, we again have the public, private and protected visibility for the fields and methods but java does not have a friend but it has something similar which is a package where if you put a set of classes within the same package, they can they are visible to each other.

Many of you may be programming in c sharp which has a very similar visibility structure, public, private, protected for variables as well as methods but it has an interesting actually it has 2 more

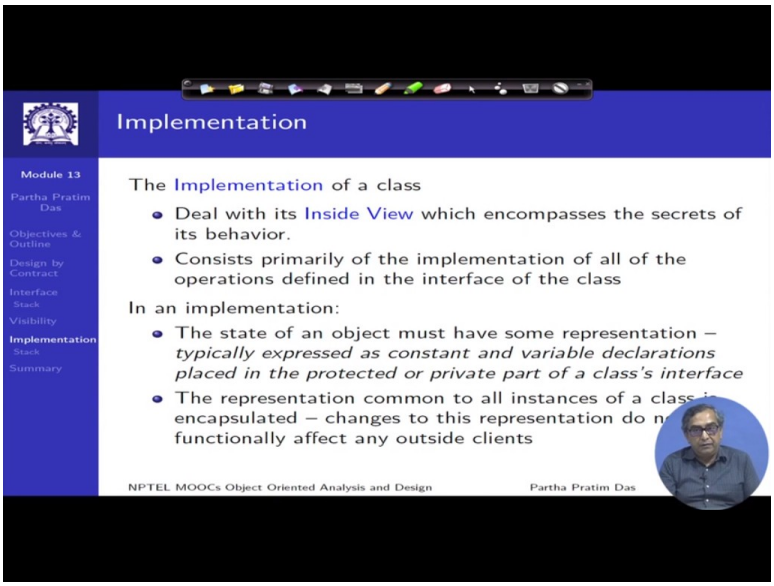
interesting visibility options one is known as internal, another is known as protected internal. These relate to these are restricted to the projects in C++ c sharp that is c sharp as you know is it is managed by dot net.

So any C sharp application you want to develop, you have to define a project so all that all the classes you put inside one project has an internal visibility or protected internal visibility. Python on the other hand has only public visibility for its methods. Its methods cannot be encapsulated out from others. It is always public but the member variables can be public, private or protected. There is no other visibility in python.

And some of the other languages just for the sake of completeness I have mentioned ada, small talk and object Pascal for example object Pascal is an interesting case where all fields and operations in object Pascal are public. So it is another extreme of an object-oriented language which has no encapsulation at all. So there is a lot of different grades so it is depending on which language you are using it will be good to first quickly study what is a visibility

Or encapsulation grades that the language support and then understand how should you design your class, how should you design your interface using this visibility information.

(Refer Slide Time: 04:54)



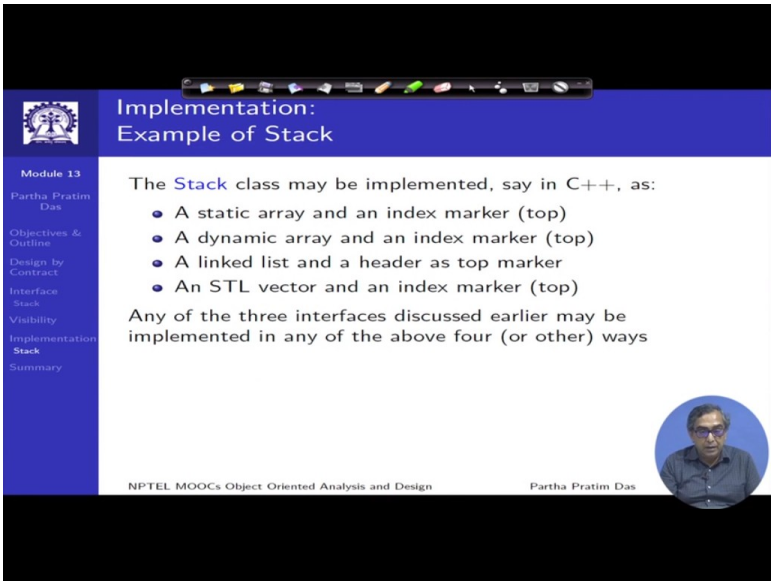
Let us move on to the other part of the class. We have been looking at the outer view now let us consider the inside view, the inner view. The inside view that is where all our secrets are very carefully protected, kept is known as the implementation of a class.

So I can say that the implementation of a class is basically the implementation of all the operations of all the methods that the class supports. Now naturally if we want to implement something we have known that the class will give rise to instances as objects. Objects will have state behavior and identity. So behavior is the exposed methods which is basically what we are carrying in the interface.

So state is what the class will need to represent in terms of the implementation. So state of an object must have some representation in terms of the implementation of the class that will typically have inter occur in terms of constants in terms of variable declarations and typically occur in the protected or private part of classes interface because if it if this representation of the state is put in the public part then you can understand that outsiders can come in any time.

And change those values and your state will become indeterminable. So even though some object oriented languages might allow your variable declarations or data members or fields to be put in the public view, you should be very careful in terms of doing that while you are designing an interface and the corresponding implementation of the class.

(Refer Slide Time: 07:01)



So given that let us look at go back to our stack class. We have already seen different interfaces for it. But then suppose you are given to implement this stack class in say C++, all languages will have similar examples. But if you are given to implement this stack class in C++, you could do that using a static array. You could statically define an array and you can use an index marker which will keep an index of the array to show how much the array has been filled up by the stack operation.

So what you simply do you start with a marker being placed at minus 1 which is outside of the array and then every time you push, you increment the marker and put the element in that array location. Every time you have to pop, you simply decrement the marker. So that could be one way to implement a stack but this will certainly mean that your stack will be able to take only a fixed only up to a fixed number of elements at a time because the array is static,

It will have a fixed size so the alternate could be that you could have a different implementation which will use a dynamic array so that you could if require have a bigger array when you have more elements being pushed in and of course an index marker. Alternately you could use a linked-list if you use a linked-list and you will need a header to hold that list and that header will act also as a top marker.

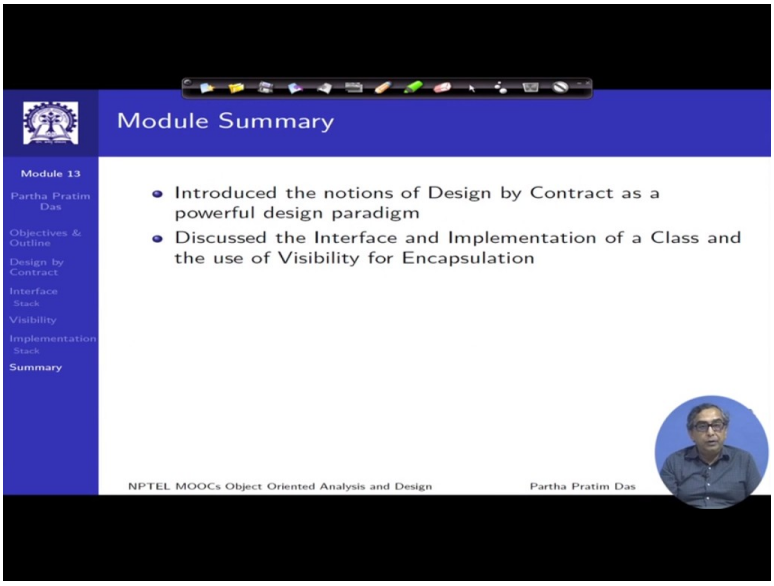
If you are more advanced C++ programmer, you could use a stl vector and an index as a top marker and achieve the same results. There could be other, several other alternate implementation designs in terms of the variables, in terms of the data members as well as in terms of the details of the algorithm that we use but any of the three interfaces that we discussed earlier, remember we discussed three interfaces based on three different contract schemes.

That the stack object may have then any of them can be implemented in any of the above 4 methods or may be even other methods. So you can see that on both terms, you have a wider set of choice and but when you go to implement, you will have to make sure that your interface is completely honored in terms of the methods that you are in implementing as well as the contract that has already been given.

But the contract will tell the implementation as to what needs to be done to fulfill the contract and the implementation will need to be completely hidden in terms of the private or protected part of the class so that if in future from the time that you have implemented the class for the first time sometime later, you want to make a different change in the implementation for some reason or other, interface will not get affected, the methods will not get affected, the contract will not get violated.

And in that process, the user would be able to continue using your object without any difficulty even though the implementation is not maintained to be the same.

(Refer Slide Time: 10:33)



So in this module, we have introduced the basic notion of design by contract as a powerful design paradigm we will see that at some certain later points will again keep on referring back to this and we have discussed the basic concept, nature of a class in terms of its interface and implementation and we have reiterated the fact that the implemented chill and interface needs to be segregated, it needs to be separate.

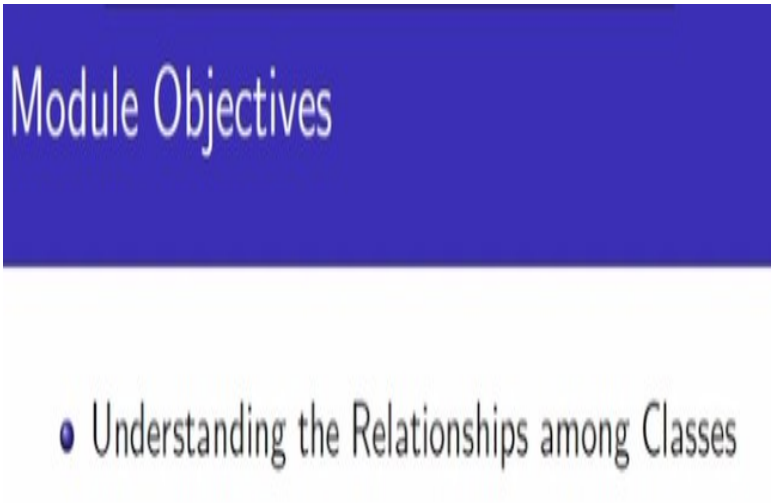
And this process of separation can be achieved in various graded manner by using various kinds of visibility that we may use and certainly we will hope to discuss more about this and get more inside when we have introduced to the other different relationships of classes and particularly the inheritance and specialization, we will see how visibility will control the encapsulation at multiple level to provide you, further refinements on the interface and finer control on the implementation.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture - 24
Relationships among classes

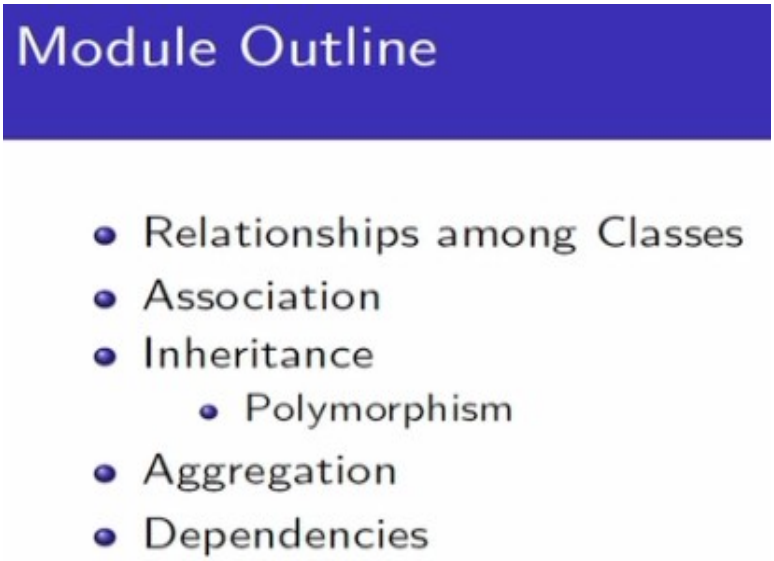
Welcome to module 14 of object-oriented analysis and design. For the last couple of modules, we have been discussing about the nature of objects and the nature of classes, we have discussed the relationship amongst objects and in this current module we will discuss about the relationship among classes.

(Refer Slide Time: 00:46)



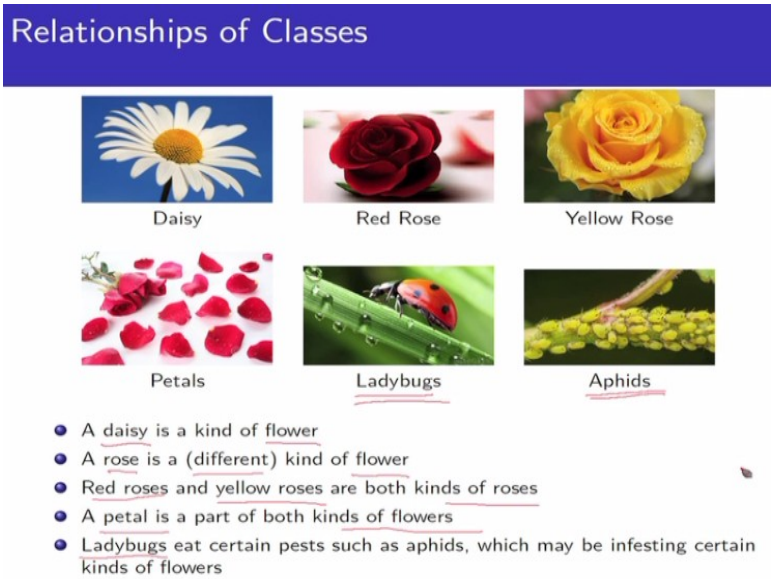
So, the single objective of this module would be to understand a variety of relationships that may exist between different classes.

(Refer Slide Time: 00:58)



So, we will briefly talk about what we mean by relationship among classes and we will discuss about 4 different kinds of common relationships that are frequently found amongst different classes of an object-oriented system, the relationship of association, inheritance, aggregation and finally dependence.

(Refer Slide Time: 01:29)



So, let us start with a illustrative example. So, here the pictures are for your reference. The actual statements are below, what we are trying to say is daisy is a kind of flower. This is a statement that we are making that is what we know. Rose is a kind of flower but it is different from daisy. Red roses and yellow roses are both kinds of roses. So we can just look at the pictures for reference as to how these similarities or relationships exist.

Petal is a part of both kinds of flowers. Be it daisy or be it rose, petals are part of them. Ladybugs, these are ladybugs, eat certain pests, this is the most typical one, aphids and save the plant. So, this is a kind of real world scenario that we are trying to model, that we are trying to represent.

(Refer Slide Time: 02:39)

Relationships of Classes

Class	Relationship
• A daisy is a kind of flower	Sharing connection – daisies and roses are both kinds of flowers – bright colored petals, fragrance, etc. Daisy IS_A Flower
• A rose is a (different) kind of flower	Sharing connection – daisies and roses are both kinds of flowers ... Rose IS_A Flower
• Red roses and yellow roses are both kinds of roses	Semantic connection – red roses and yellow roses are more alike than are daisies & roses Red Rose IS_A Rose, Yellow Rose IS_A Rose
• A petal is a part of both kinds of flowers	Semantic connection – daisies and roses are more closely related than are petals & flowers Flower HAS_A Petal
• Ladybugs eat certain pests such as aphids, which may be infesting certain kinds of flowers	Symbiotic connection – Ladybugs protect flowers from certain pests Semantic Dependency

Are Roses and Candles related? – Both decorate dinner tables

So, if we look into these 5 statements and first what we may do is we may quickly indentify the different classes that we will have. Certainly flower is a class which has all kinds of flowers that may happen. Daisy is a class. It is all daisy flowers that we can think of. Rose is a class. Red rose yellow rose they are classes. Petal is a class. Ladybug is a class. Aphid is a class. So, at least these classes we can clearly identify and we have talked about the nature of classes in the last module.

So, following these principles we can easily say that these are the classes. Now, what is important, what we are trying to focus on now is, each of these statements actually describe some kind of relationship between these classes. So, the first is kind of sharing connection. We can say that daisies and roses, if we just take these 2 statements that daisy and roses are both kinds of flowers. They are kinds of flowers in the sense they share bright color, they have fragrance and all that.

So, we will often express this connection, this relationship by saying that daisy is a flower or saying that rose is a flower. Similarly, when we get into the third statement that red roses and yellow roses are both kinds of roses then the semantic connection is that they are more closely related compared to what a rose is related to with a daisy. Certainly the similarity between a red rose and a yellow rose is lot more than what it is between a red rose and a daisy.

So this will get again expressed, this semantic connection will again get expressed in some kind of a is a relationship I can say that red rose is a rose, yellow rose is a rose and so on. In contrast, when you look into the fourth statement, the petal is a part of both kinds of flowers. Then we have a semantic connection which is kind of describing a whole part relationship that is hardly we will come across flowers which do not have petals and if we see some petals somewhere we know that there was a flower from which the petal has been fallen off.

So, we will model that kind of a relationship with saying that flower has a petal that is a flower that is the whole part which will have petals. Finally, in the fifth statement you have some kind of a relationship such as ladybugs eat certain pests such as aphids. So, if you look into these 2 classes, particularly these 2 classes, ladybugs and aphids or specifically aphids will be pests.

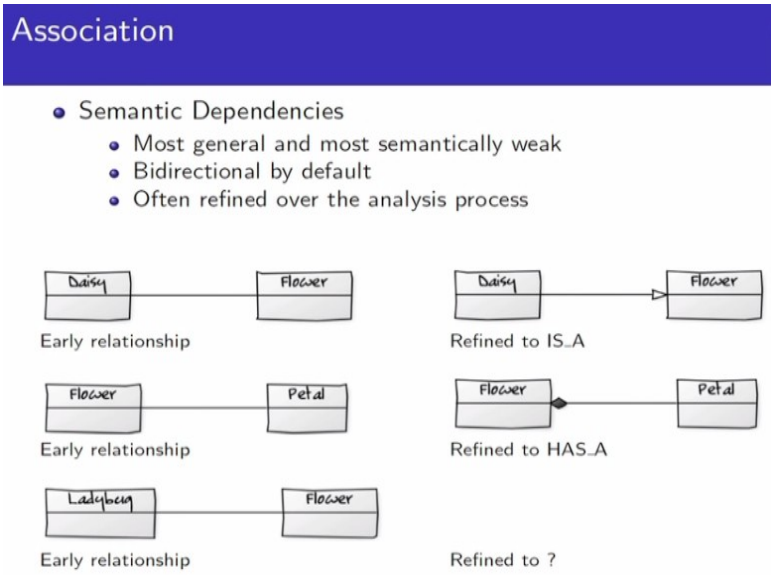
If we look at that then there is no kind of earlier kind of relationship of one among and sharing the properties of the other or being similar to that or one being a part of other and so on but these are also some kind of important relationship because I can say that ladybugs protect the flowers from certain pests. So, in my object module of this reality, we would like to model this kind of semantic dependency as well.

And just before we move on, just to put the question to you, are roses and candles related? Of course even between the similarities between the ladybugs and aphids are more than the similarities between roses and candles because ladybugs and aphids you can say both are dependant for their livelihood and dependant on the plants. But we can also find that in some context rose and candles can be related because both of them happen on the dinner table.

So the relationship, I mean what can be a relationship and what may not be a relationship certainly has a wide range, a wide variety and therefore what we will study is some of the specific types of relationship which are very frequent and they occur very frequently in different designs and I would just like to remind you and refer you to module 12 in this connection, where we discussed about relationships between objects and we will see the different relationships between objects as we have observed.

They will occur in a very similar manner between the respective classes as well.

(Refer Slide Time: 07:38)



Moving on, let us talk about the first kind of relationship that we want to focus on. This is called association. Association does not have a very strict definition. It is one of the most general and kind of most semantically weak, most general and most semantically weak kind of relationship. Semantically weak in the sense that it does not often need a very strict definition. By default, it is bidirectional in nature.

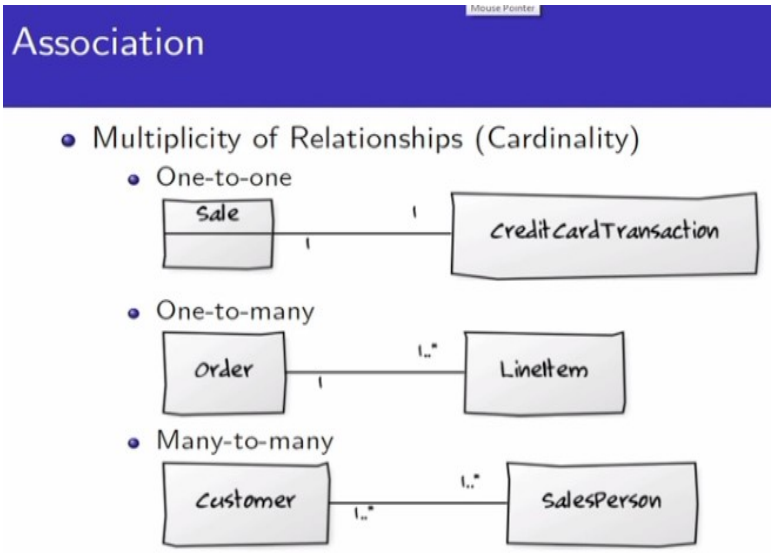
So, we are saying that there is a daisy and there is a flower so there is a relationship between them. In a flower and petal there is a relationship between them. Now, associations often happen in early stages of the design. So, when we have come across a daisy, a flower, a petal, a ladybug and so on then we will just initially try to identify what are the different ways they can, they may

be semantically associated and we just draw this early relationship in terms of the bidirectional links which does not have any arrow heads.

But as we go over the design, into later stages we have more and more information analyzed we may be able to refine those relationships which for association here will later on turn out to be a more refined relationship and we will see that this kind of elevates to an inheritance relationship or whatever was a just an association here between flower and petal will elevate to an aggregation kind of relationship whereas a relationship between ladybug and flower may just continue to remain equally identified.

So, association is a very generic form of a relationship which you should use whenever you find two classes have some way to interact, some way to be related to each other, whenever you find some kind of semantic dependency you should put an association between them and then look forward to further information called the analysis in the stages of object-oriented design, where you may be able to refine some of those and find stronger more structured relationships between them.

(Refer Slide Time: 10:01)



Now associations often will have multiplicity of relationship so and these multiplicities known typically as a cardinality. So what is said that suppose you consider that you are buying an item using your credit card, so what will happen that there is a buaying side on the buying side there is

an order that you place and on the credit card side there is a transaction. So if this order is for Rupees say 1000 then you will need to have a Rupees 1000 transaction on the credit card.

So if we identify the order to be on the sale object which the merchant has sold you and the credit card transaction is a transaction object through which you have actually made the facilitated the payment to the merchant then you will identify that for every order, every sale that you do, every buy that you do that sale object must have a corresponding credit card transaction object.

For every credit card transaction there has to be a sale, for every sale there has to be a credit transaction. So if this is the scenario then you say this is an association which is one - to – one, that is between this class and between this class, one instance of this class will be associated with one unique instance of this class and vice versa, whereas in some cases it could be different. For example, consider an order you are again buying, you are ordering, now often we do not buy one item.

We will buy item 1, then we will buy item 2, item 3 like this. So we will pick up all of those put them together in the cart and then buy that whole thing together. So if I look, if I model by Line item class, if I model each and every item that are placed in the order and by the order class if I model this whole order, then the relationship between them is one to many, that is one order object instance will have one or two or more Line items.

And that is what is depicted in the picture here. Here like this side the number is 1 which say that for one order and this side the number is, if you are not being able to read it properly what is return is basically this. So it means may range that is one to as many as you want. So it could be 1, 2, 3, 4, 5, whatever. So these are typical notations, for example you could have written 1 to 5 something like this, where it will mean that it is 1 to 5 range.

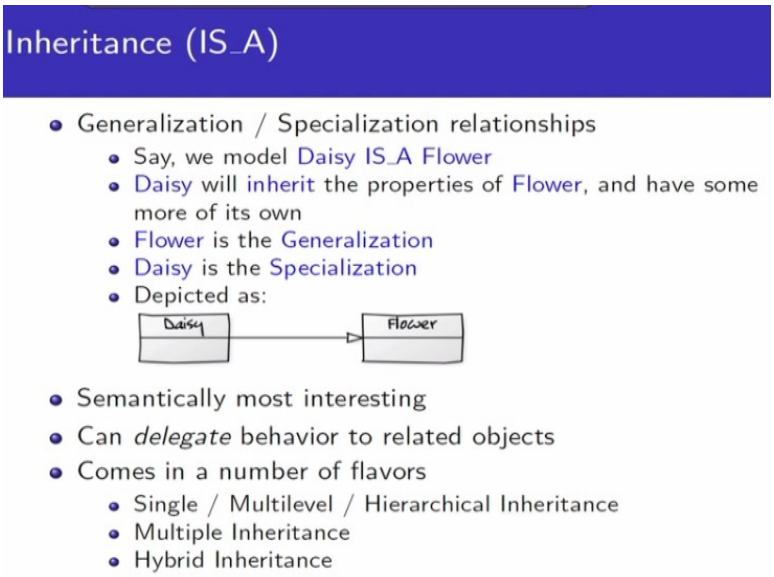
This is a particular one which says it is one to as many as you wish; such associations would be called One - to - many associations. And then finally there could be Many to many associations, think about any sales scenario and having different customers and different sale persons,

naturally every customer maybe dealing with a number of different sales person. Somewhere is buying groceries, somewhere is buying dresses, somewhere is buying electronic goods and so on. Similarly, sales person is also dealing with multiple different customers.

So if I take an object here and if I take an object here then there may not be any unique relationship. One customer object may be related to one or more sales person object and one sales person object in turn would be related to one or more customer objects. That is why this is designated in this manner, these are the many to many association relationships.

So whenever we depict an association if possible we will try to put the cardinality of the relationship on both sides because and of course in this case it is not explicitly mentioned in the slide but I would like to make a note that here we are just talking about simple binary relationships where it is always relationship between two classes so but there could be ternary associations also that could be, that is three different classes could be together related to each other.

(Refer Slide Time: 14:50)



Let us move on the next relationship that we talk about is already known to you. We have talked about this, since almost the, since the beginning of the course. We have talked about inheritance or hierarchy as a general mechanism of managing complexity and then we have talked about this in length while discussing about relationships of objects. So I will be very brief this time,

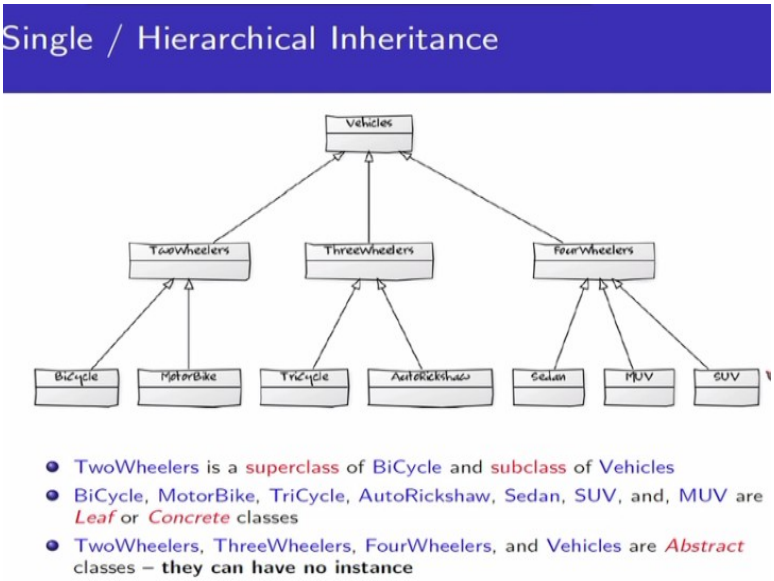
because the concept of inheritance as you learned in terms of objects is very similar in terms of the classes from which the objects actually arise.

So inheritance as you know is a generalization or specialization relationship we have just seen if you model Daisy is a flower then it means that daisy will inherit all the properties of the flower. But Daisy may have some specific properties, some additional properties of its own, so that is a basic principle of inheritance, that you inherit everything from the superclass and you may add some more properties of your own and terminology wise we will say flower is a generalization often also referred to as superclass and Daisy is a specialization referred to as subclass.

If you are coming from experience of some of the OOP programming languages, then you will find other equivalent definitions. For example, Superclass of in C++ is often referred to as base class. Subclass is often referred to in C++ as derived class and so on. And this is the diagram that typically depict, this is a UML diagram and you can see this hollow arrowhead which depicts that daisy is a flower.

Now semantically certainly inheritance is the most interesting relationship to find amongst classes and it can be used to delegate the behavior to related objects and we have seen in while discussing relationship amongst objects that inheritance could be of various different kind. Single, multilevel, hierarchical kind of inheritance or it could be multiple inheritance or it could be hybrid forms of heritance.

(Refer Slide Time: 17:25)



So if we quickly take a look in terms of a single hierarchical inheritance example, we have representing a hierarchy, here were in the top at the root are vehicles. So vehicle is something some mechanism by which people and goods can be transported you can say. Whereas so vehicles can be described as a class with very limited set of properties like, it is a mechanism to move.

It should be able to carry something and the implicit here, which is not separately mentioned but implicit here is a vehicle moves on land. We are not talking about ships or aero planes here and certainly vehicle can be driven. The vehicle will have a mechanism of transportation and so on. So these are the general properties that this class will have. But then we say that there are specialization subclasses like this.

We said that two wheeler is a vehicle, because two wheeler will satisfy all these properties of vehicle that we have just discussed. It can move it can transport people; it can transport goods and so on and so forth. But in addition to all that properties of Vehicle two wheeler has additional property that it has, it moves on wheels first of all. Second it has two wheels, it needs balancing and so on and so forth . So there are certain additional properties that a two wheeler must have.

Then we can say that two wheeler in turn could be specialized into a bicycle or a motorbike. Both of them have lot of things similar two wheels, balancing required and all that but the two wheeler does not specify as to the source of energy that is required to move the vehicle or the two wheeler. Here when you come to this level the source of energy basically distinguishes us to between a bicycle two wheeler and a motorbike two wheeler.

Bicycle two wheeler is manually driven and a motor bike two wheeler is driven by some kind of fuel, some diesel, petrol or something like that. So this is how properties continue to get specialized as we move down a hierarchy and this is exactly what we are designating here and in this some more terminology that you should be comfortable with. All these classes, a set of classes are known as leaf classes or concrete classes, in the sense that these classes do not have any further specialization.

So certainly these classes exist because there are objects that will exists for these classes. But very interesting thing is these other classes, these four classes which are not leaf, the kind of, we will refer to them as kind of abstract classes. And we will go over this concept of a class being abstract more and more but just to introduce the idea, let me say that a class is abstract if it is not possible to instantiate an object for it.

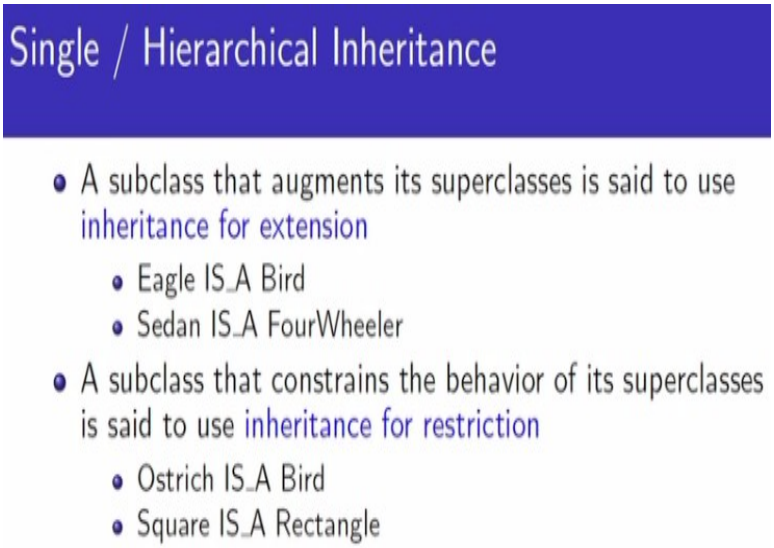
Now do not get shocked in this statement because in the last module itself we had mentioned that classes have instances that are objects but here we are saying that the class, some classes are possible and these classes occur only on a inheritance hierarchy which are called abstract that is they have certain properties but they are cannot be an instances of that class, that I cannot say that specifically I have an object of a three wheeler class.

But I can say that I have an object of a bicycle class because it is a leaf node, leaf class and I can also say that the bicycle, the object of the bicycle class can be thought of as an object of the two wheeler class because the bicycle class is a specialization of two wheeler class. So the bicycle class satisfies all the properties of the two wheeler class. Therefore, a bicycle object, the specific bicycle that I own say will have all the properties of a two wheeler class as well.

But if I do not consider the leaf level if I just want to create an instance of the two wheeler class I will not be able to do that, that is what is a meaning that we are trying to say. We will come to more refinements of this concept and soon I will explain with this example only I will show you why such kind of abstract notion is required for the classes. But one point that I would like to highlight here is, in this example all leaf classes are concrete which will always be the case.

And here all non-leaf classes are abstract. Now this part is incident, in non-leaf class is not necessarily abstract and abstract class will always have to be non-leaf but a non-leaf class may also be a concrete class. That is we will explain later on, that is not a very good style of design but there are many hierarchies where non leaf classes are also concrete classes. But you can never have a leaf class which is not a concrete class, which is an abstract class.

(Refer Slide Time: 23:17)



So in terms of this let me also introduce two more notions that of inheritance for extension and inheritance for restriction. We have said that as we go from generalization to specialization, this is a hierarchy as we go from the generalization to specialization, we inherit all the properties and then add some of the subclasses additional properties. These added properties could be adding own to what we inherited but it barred it could restrict in terms of what we have inherited.

So as an example let us say if we have a superclass bird and I say eagle is a bird then certainly eagle has some properties which generically is not added in the bird class. For example, eagle

can stay port without actually flipping its wings on top of a stream of hot air which generically birds cannot do. So we are adding on that property to the bird class when we come to the eagle, so this is an extension.

We can say sedan is a four wheeler is an extension because sedan has everything that a four wheeler has, in addition to that it has some more properties of having a nice boutique and so on, possibly a nice air conditioner and all those which is not necessary for a four wheeler. On the other side, there are some cases like we say that ostrich is a Bird we all know and we know that ostrich cannot fly but a bird can fly.

So when we say ostrich is a Bird, I am actually restricting some of the properties that bird has. So this is an inheritance for restriction. Similarly, we will say square is a rectangle, but what is a rectangle? Which has all its internal angles are right angle 90 degrees. Square satisfies that, so Square is a rectangle. But when we say I have a square then I have the additional restriction that the sides will have to be equal which is not so for the rectangle.

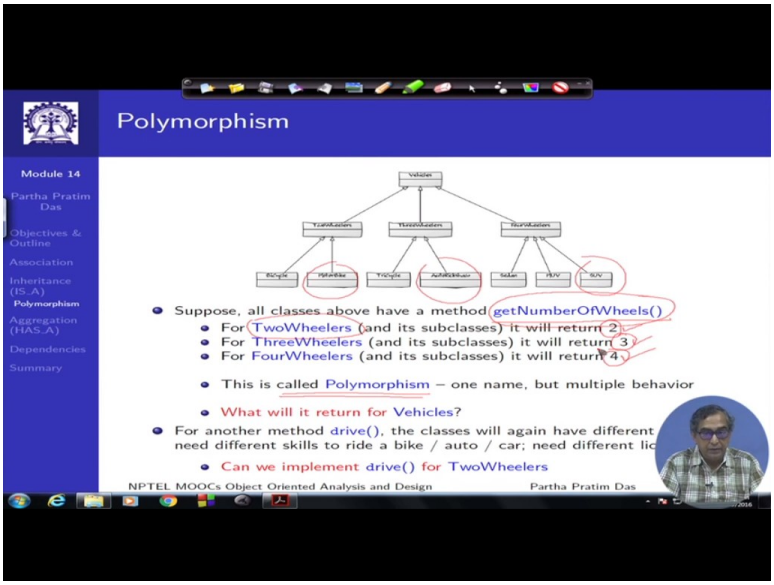
So here when you specialize, we are imposing further restriction on the subclass. So whenever you look at the hierarchy, it is possibly the inherent sense most of us keep on getting is, the inheritance is for extension that I am hiding more and more and more properties but please keep in mind that addition of properties could actually mean extension or it could mean restriction in many cases as well. We will stop here in this lecture and we will discuss about the polymorphism inheritance hierarchy in the next part of this module.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 25
Relationships among objects (Contd.)

Welcome back to module 14 of object oriented analysis and design. We have been discussing about relationships among classes and we have talked about association and we have revisited the concept of inheritance relationship in connection to what was earlier done as inheritance amongst objects.

(Refer Slide Time: 00:46)



Now we continue on the most important concept on inheritance which is polymorphism so let me introduce that with a small example. So we again take the inheritance hierarchy of vehicles that that we have introduced earlier. Now let us say that suppose for since it is a hierarchy of vehicles we would like to know for any class or object of any class as to how many wheels does that vehicle have. So we will have a method called get number of wheels.

Now what is the expectation that for 2 wheeler class or for any of the subclasses of the two-wheeler class, this method should return 2, for 3wheeler it should return 3, for 4 wheeler it should return 4 and also for the respective subclasses. This property is called polymorphism. What is the property? That for all of these, the name of the method is same; the name of the method is get number of wheels. But depending on which class or object of which class whether it is two-wheeler or its motorbike.

Or its auto rickshaw or it is suv depending on which object or object of which class isa invoked the

actual methods are different. The algorithm executed by the method or the implementation of the method is different so we say that polymorphism very simply is one name but multiple behavior, one name is the one name and these are the three different behaviors that we could associate with it. So this is the basic notion of polymorphism and we will see that this will come in extremely handy

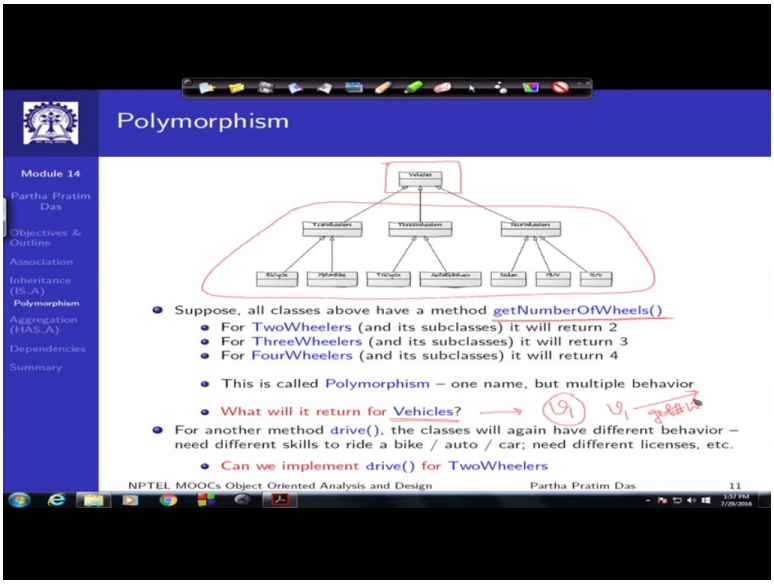
In terms of doing different object based design based on different hierarchies. Okay now let us say that suppose I take this question further and ask that for this particular method if I invoke this on waiting the root class what will it return? What will be the expected output of get number of wheels if asked for some instance of vehicles? You do not know. We cannot say because a vehicle could be a two wheeler when it should be a 2 or it could be a three-wheeler when it should be 3, it could a four-wheeler when it should be four and so on.

In plain and simple terms, this question cannot be answered. Or in plain and simple terms, this particular method cannot be invoked on the waiting. So that gives us a very interesting scenario, I have a hierarchy where a large number of classes are involved, many of these classes are lift classes and they will have several object instances, all of them can respond to the message get number of wheels while the root class from where everybody specializes defines a basic concept of vehicles that cannot answer this question that cannot handle this message.

So this leads to the requirement that when such a situation can exist that over this generalization specialization hierarchy, it is possible that some of these polymorphic methods are not implementable in some of the classes then those classes are considered to be abstract. So that is another way of or more correct way of looking at when a class should be abstract. A class should be abstract when there exists at least one polymorphic method which can be implemented for some of the subclasses but cannot be implemented for this class.

Because if you if vehicle was not abstract then I will I can have an object v1 a concrete object v1 of the vehicle class type.

(Refer Slide Time: 05:35)

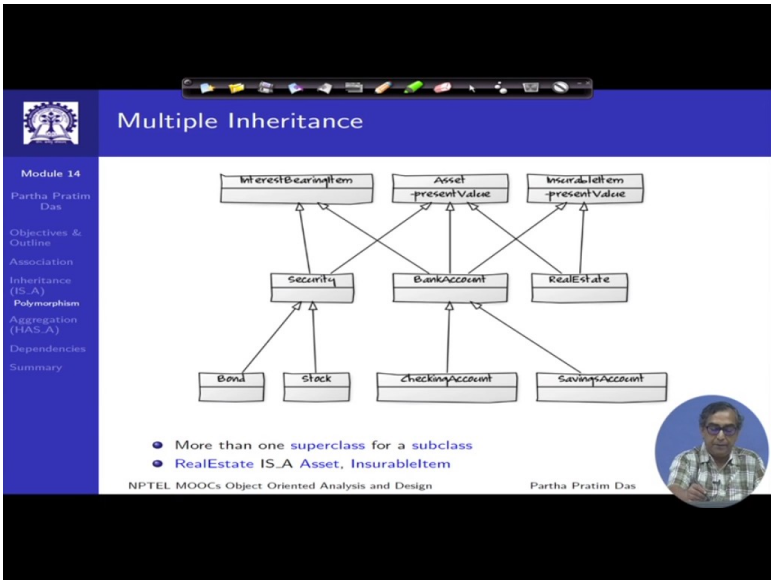


If I have v1 then I can send a message get number of wheels to this vehicle and we know that that message cannot be executed because there is no algorithm for it. So this is the basic fundamental notion of abstract classes and how polymorphism and abstract classes relate to each other you can just think further, you will find that there are you can find reasons of why two wheeler or three wheeler classes will also have to be abstract.

Because say certainly all the vehicles will have some method drive now if you are say that I the class is vehicle you just do not know how to drive because there are different bicycle needs certain kind of skills and auto rickshaw needs a different kind of skills, sedan needs yet different kind of skills, you need different types of licenses, all of different factors are different between driving a vehicle, driving a two-wheeler and driving these specific instance specific vehicles of different number of wheels.

So you will again similarly not be able to implement drive for a two-wheeler because somebody anybody can drive a bicycle if he or she knows balancing but those all of those may not be able to drive a motor bike unless he or she has a license to drive a motor bike. So drive in two-wheeler does not have an implementation whereas it has implementation here. That consequently makes two-wheelers as potentially abstract class.

(Refer Slide Time: 07:10)

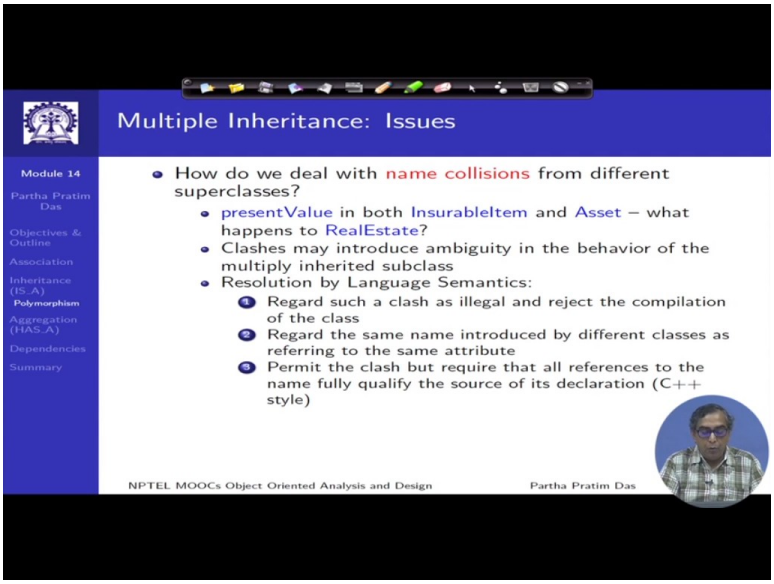


Moving on in terms of inheritance there is a special type of inheritance know as multiple inheritance where more than one super class exist for a subclass. So for example if you look into the bank account here, bank account is an insurable item, is an asset, is an interest-bearing item. Okay you can think through these kinds of examples this example is from Booch's book, so you can read up the whole details of the example from the book.

It clearly says that if I have a bank account I have money in that then naturally I can ensure that money against that. I can use that as an asset to get collaterals and certainly the bank regularly gives me interest on that. So there is a specialization of bank account which has certain properties borrowed from the interest bearing item because it can earn interest certain properties borrowed from the asset class because it can be treated as an asset and some more properties borrowed from the insurable item.

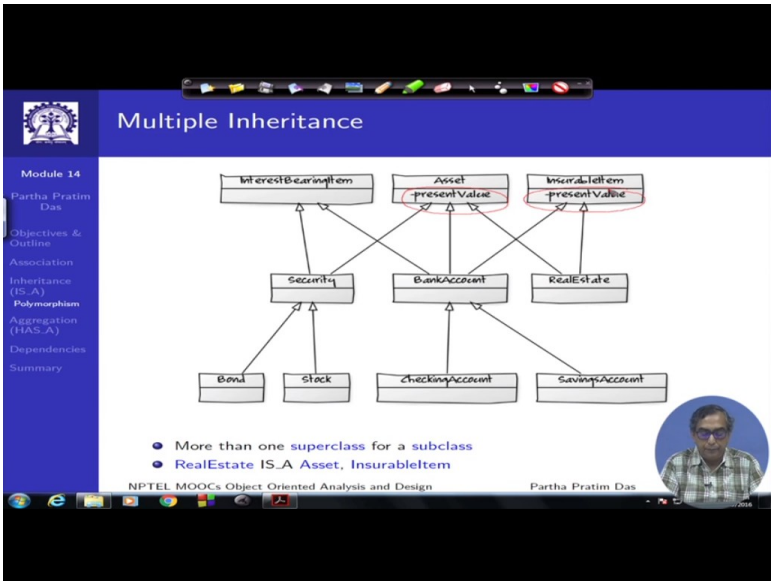
Because it satisfies all the requirements of an item which can be insured. In addition the bank account will have other further other properties if you look at real estate, real estate is an asset at the same time, it is an insurable item as well. So whenever such situations exist whenever one subclass have more than one super class we say that we have a situation for multiple inheritance. Multiple inheritance occur very frequently in real life and but it has some very fundamental difficulties to be handled

(Refer Slide Time: 09:00)



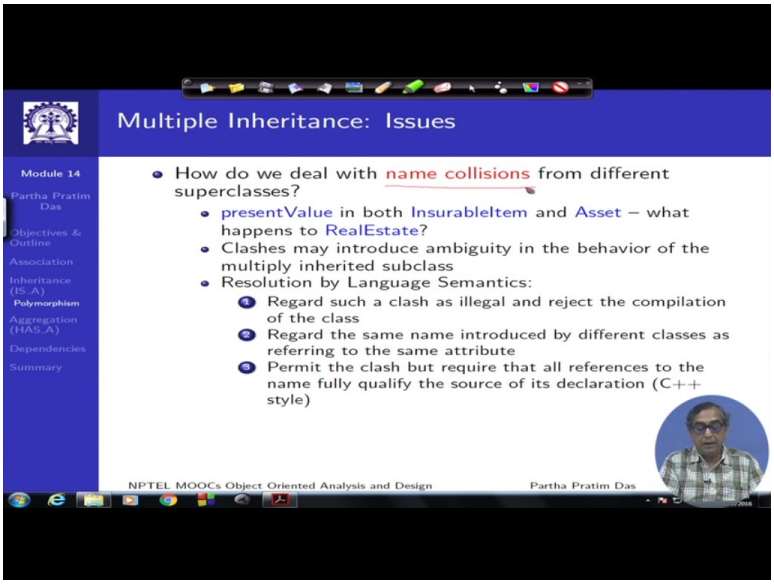
And modeled and therefore we will just mention about those. One is the issue of name collision

(Refer Slide Time: 09:12)



So if you look into this say the asset class and the insurable item class let us say these are independent classes which quite possible that they have a variable a data member which is common between them. So the present value so what happens when I inherit from this I inherit from this in the bank account or in the real estate I actually get 2 present value members. The question naturally is which present value member I should be used.

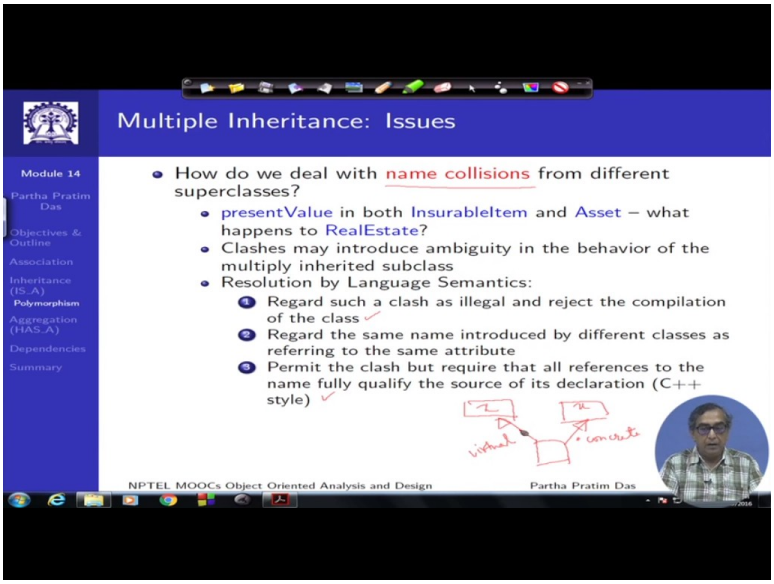
(Refer Slide Time: 09:40)



So this is what is known as a name collision. Present value in both insurable item and asset, what happens to the real estate? So this will certainly create ambiguity or clash in the behavior of the multiply inherited subclass there is no unique answer to how should this be handled, different languages depending on the language you are using, they have different semantics to handle name collisions. Some will not even not allow this to be used at all.

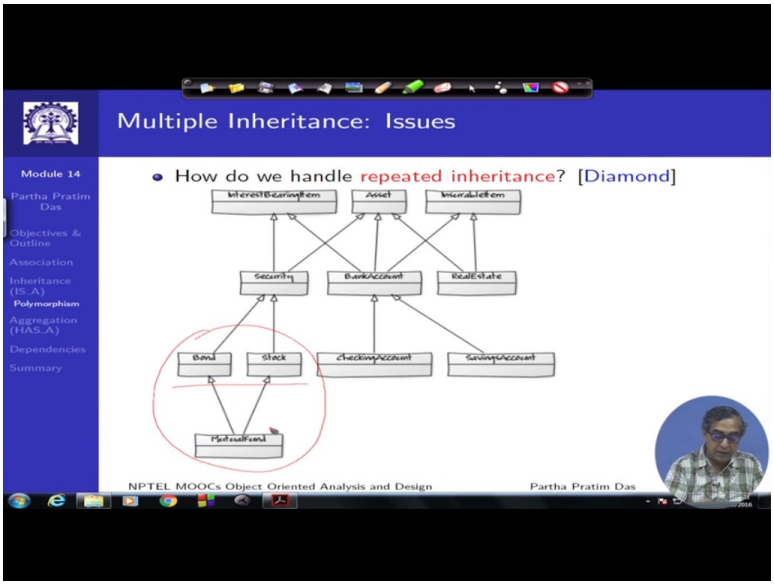
That is 2classes if they are parent of the same class, then they cannot have same data members is one possibility. One possibility is they will treat them as the same data member and then define some semantics based on that or there are some finer semantics for example c define C++ defines a semantics where it says that if I have multiple inheritance like this a situation of this kind and the same variable exists on both, then I will not inherit from both of them. One of them can be concrete.

(Refer Slide Time: 10:43)



Rest of them should be what C++ say should be virtual which means I will only use their methods but I will not use their data members if you are not familiar with C++, do not worry about these comments because the idea is not to go to a specific language, the idea is just to highlight that these are some of the very core issues that can happen if you are using multiple inheritance but at the same time multiple inheritances and extremely useful modeling feature.

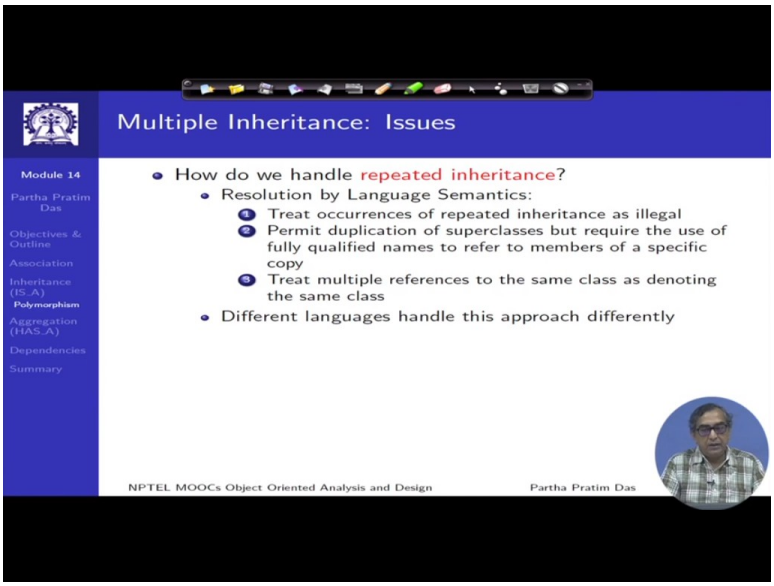
(Refer Slide Time: 11:26)



Moving on the second issue which will often be seen just look at this part this part is what I have added now. So earlier it was bonds and stocks or securities I have added mutual funds as a multiply inherited class from bond as well as stock. So what do we have? You have here, if we just look into this a diamond shaped like a diamond which means that this class this security class is actually inherited by multiple mutual fund in two different parts.

It is inherited as mutual fund is a bond, bond is a security in that part or mutual fund is a stock and stock is a security in that part naturally if you do it both on both these parts, then the kind of properties that get associated with them differ and the behavior will differ and you may have often ambiguous contradictory situations happening.

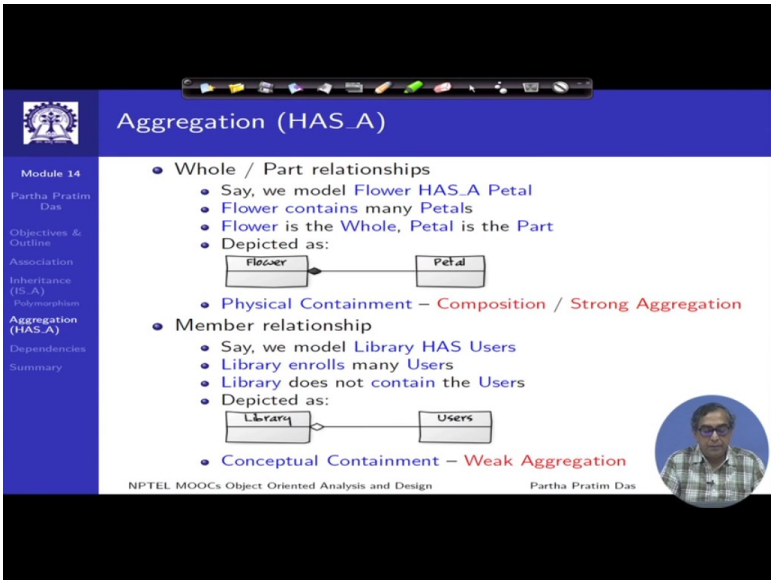
(Refer Slide Time: 12:31)



Again there is no unique solution to this repeated inheritance problem again the different languages when you go to implement such situation will have different restrictions of doing that, some will simply not allow that, some will only allow one parent but you will have to decide on which part the parent will come and so on. So please remain informed that multiple inheritance as such is a very strong modeling relationship amongst classes.

But you will have to be careful about the name class and the repeated inheritance problem when you are doing your design and as far as possible you should try to avoid both of these in your design.

(Refer Slide Time: 13:21)



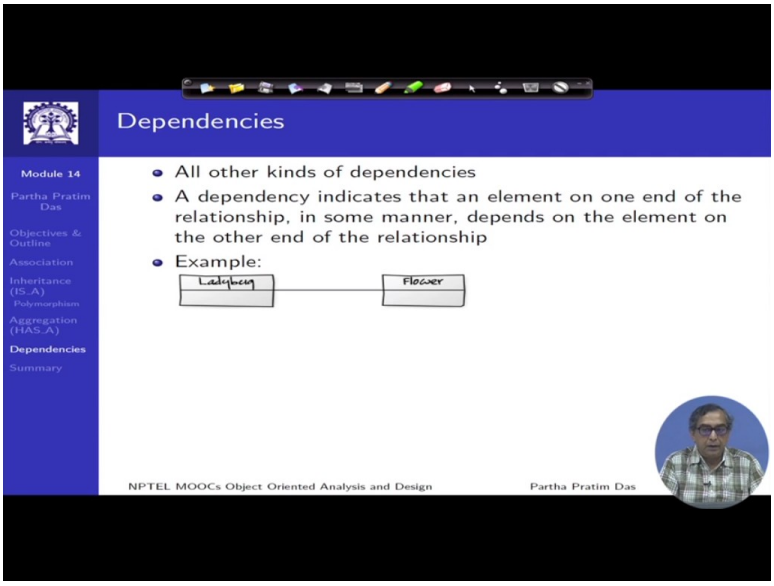
Let us move on to the 3rd relationship which is aggregation which we know is has a kind of relationship and I will be quick on this because we have explained this in depth in terms of relationships amongst objects so it is primarily a whole part relationship. So we say flower has a petal so that means that flower contains many petals and we say the flower is the whole and certainly petal is a part.

And in terms of the depiction in diagrammatic depiction we will show it like this and in this case part-whole relationship, in this case there is a physical containment, the petal is physically contained in the flower. So often it is referred to actually as composition and some authors call it as a strong aggregation also. In addition we have seen that there is a different kind of aggregation which is can be referred to as member relationship where library has its users. Now naturally a library will have many of its users.

And without that users, the library has no purpose but at the same time, the library enrolls the users but library do not contain them, the users are not physically contained within the library, we talked about this earlier in the context of objects also naturally in terms of classes as well. This will have to be kept in mind and this is how we diagrammatically depict that and since this containment is kind of conceptual there is no physical containment. We call this as a week aggregation.

In terms of the diagrammatic depiction of weak aggregation and strong aggregation I would like to just draw your attention to the fact that in case of weak aggregation the diamond that we use at this end is empty one in case of strong aggregation the diamond that you use is a filled up one. So that is how you will be able to depict different aggregations quite easily.

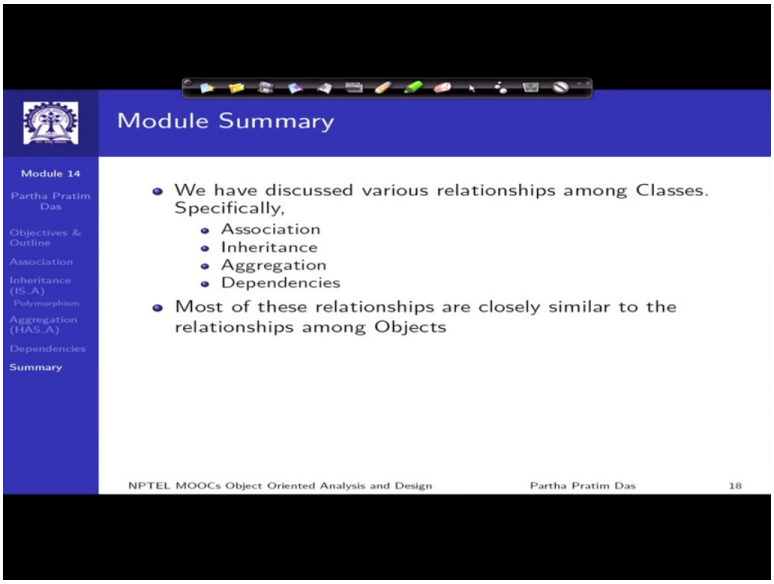
(Refer Slide Time: 15:32)



Finally so beyond association, inheritance and aggregation which are the three main forms of relationship the classes can have. Then there could be all other kinds of dependencies anywhere when you find that some element of a class is influenced by is related to or depends in some way with another class. Then you should mark that there is a dependency so when we talk about ladybug and flower, there is a dependency relationship.

There is something semantically dependent in terms of protecting ladybugs, protecting the flower but these could be there could be lot of variations on these dependencies and we will see as we move further into particularly the UML diagrams and modeling of variety of situations, we will see that a large number of finer relationships can be clubbed are clubbed together in terms of a general dependency relationship but till we go to that depth you should always keep in mind that anything that comes across where you say that things can be somewhat related you put it dependency and put it in the diet.

(Refer Slide Time: 16:49)



So in this module we have discussed various relationships amongst classes, specifically we have talked about association, we have talked about the inheritance which is the most important relationship amongst classes the hierarchy, particularly hierarchical as well as multiple inheritance and in the multiple inheritance, we have highlighted the issues that can commonly happen in terms of the name clashes and in terms of repeated inheritance, we have then discussed about the aggregation composition and weak aggregation relationships amongst classes.

So most of these relationships I will again repeat, reiterate are pretty much like the relationships, we saw amongst objects only difference being that when we talk about objects, we are talking about concrete entities that exist and when you are talking about classes we are basically talking about abstract plans based on which objects will be created at the runtime so like objects have a lifetime which is very dynamic.

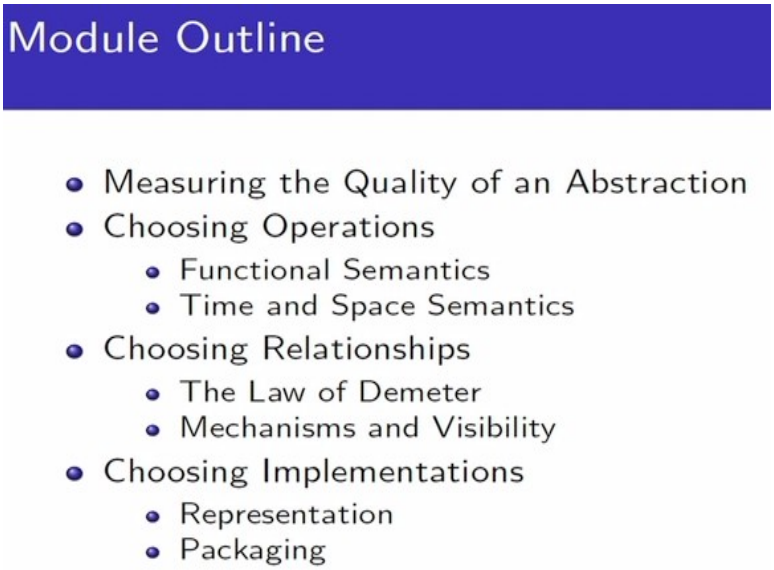
Classes will also have lifetime but they are more static they will exist from one version of the design till the next version of the design and within the lifetime of one class, several objects will be created, operated and destroyed.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture - 26
How to Build Quality Classes and Objects

Welcome to module 15 of object oriented analysis and design. Over the last 4 modules, we have been discussing, introducing about the nature of objects and classes and their relationships. In this module, we will talk about a few basic parameters to decide how to build classes and objects of good quality, that is what defines if my design of classes of objects is good, is moderate, workable or it is pathetic and will cost me lot of pay.

(Refer Slide Time: 01:00)



So in this module, we will briefly introduce some measures of quality and then on the specific aspects of design like if I am doing a class design then I need to decide on the operations, the relationships and certainly find the implementation. I will highlight the four issues that need to be you need to take care of; you need to be careful about when you do these designs.

(Refer Slide Time: 01:28)

Measuring the Quality of an Abstraction

How can one know if a given class or object is well designed? Use measures:

1. **Coupling**
   - Measure of the strength of association established by a connection from one module to another
   - Low inter–class coupling desirable
   - Inheritance is highly coupled!
2. **Cohesion**
   - High intra–class cohesion desirable
3. **Sufficiency**
   - The class or module captures enough characteristics of the abstraction to permit meaningful and efficient interaction
   - Leads to minimalist design
4. **Completeness**
   - The interface of the class or module captures all of the meaningful characteristics of the abstraction
   - No aspects of the abstraction is missed out
5. **Primitiveness**
   - Primitive operations are those that can be efficiently implemented

So in terms of the measure of quality a lot of, after a lot of years of research and experience of various projects, successes and failures, the community has more or less agreed to use five major measures to decide whether an abstraction has been captured well or there is scope for improvement. So the first is coupling, coupling basically says that if I have two classes of modules then how couple they are, how closely interrelated they are.

Now, certainly if two classes are; there are two different classes and there is a lot of coupling that this class uses a lot of information from the other classes, that class uses a lot of information from this class and so on, then certainly it may not have been a good idea to make them as separate classes. They should have been in the same module, should have been in the same class.

So that is what we are talking about that if we have the two classes and the coupling that we have which is basically the density of messages that they need to exchange has to be low. So the inter class coupling should be kept at a minimum. Now of course, this will contradict one of the earlier design principles that we have been preaching is inheritance is a good idea because it gives such a good hierarchy.

But certainly if we are using inheritance then we have a very high coupling because any superclass, rather any subclass will use the whole lot of information from the superclass, so there

is a lot of coupling in them. But there is always a trade - off in terms of how much you couple or how much you use the inheritance of. The second principle or second measure is cohesion.

That is if I have an object, if I have a class then all the data members and methods, messages that exist whether they should be together. Am I putting too much in to one class that is things which are not related, am I putting them together, we will have to decide based on that. So cohesion must be high, which is intra class all different methods and members that we put in a class, they must be closely related.

So, we had discussed this overall in terms of different aspects of separation have concern and all that these two we had always said that between two entities, between two modules, the traffic that goes across modules must be significantly low, whereas the traffic that goes within a module must be significantly high and that is the basic meaning of coupling and Cohesion. So those turn out to be the measures as well.

The second next two measures are also related, one says that the design should be sufficient that is you should do a minimalized design, you should not do a design, which has lot of methods in the interface which may not be necessary. For example, you are designing a stack we always talk about designing stacks, there is a push, there is a pop, I could think about a method which does a double push, that is text two elements and pushes them one after other.

I can do that; I mean this this that way also I can make the semantics of stack correct but there is not something that is a good design because it goes against the minimalist principle of design. It is possible to do with the standard push, so I should not have a double push kind of operations available on my stack. And completeness on the other aspect is, on the other side which says that in terms of the design, how I covered everything that I needed to cover.

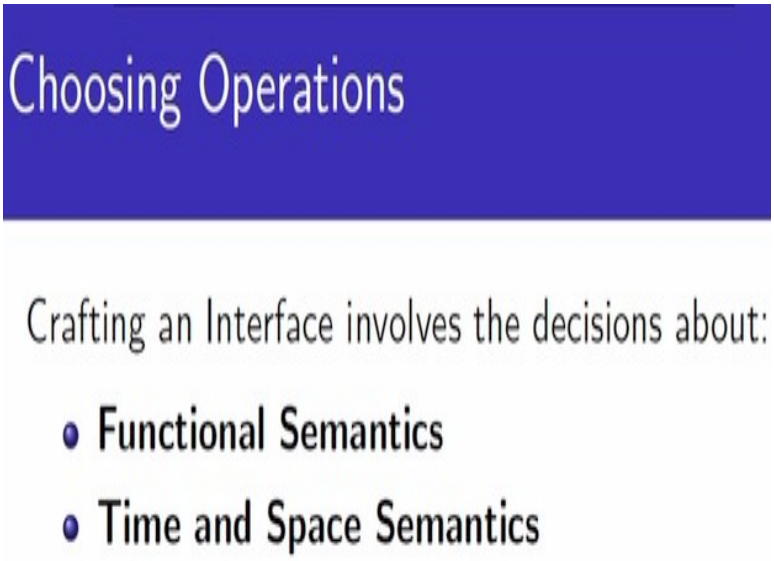
For example, if I design a stack which has a push, pop and top and does not have an empty. By analysis, I must be able to reason that this is not a complete design because there will be situations, where you will not know whether you should actually invoke top, send that of

message or you should actually invoke the Pop message by adhering to the contractual guarantees that we have talked of.

So in this way, the sufficiency and completeness are two measures that we regularly should apply, whenever we are designing the class and finally, it is the primitiveness, you have to decide in terms of what is the primitive that you build with. Now certainly few primitives have to be given by the OOP language that you are using, so they are typically the built-in types, those are the representatives of the classes.

But based on that what you decide to be different classes is based on the primitives that need to be performed. So, this is a kind of a subjective decision, but it will depend significantly on what can we efficiently implement. So these five principles of, measures of coupling, Cohesion, sufficiency, completeness and primitiveness will be necessary for judging any abstractions.

(Refer Slide Time: 07:16)

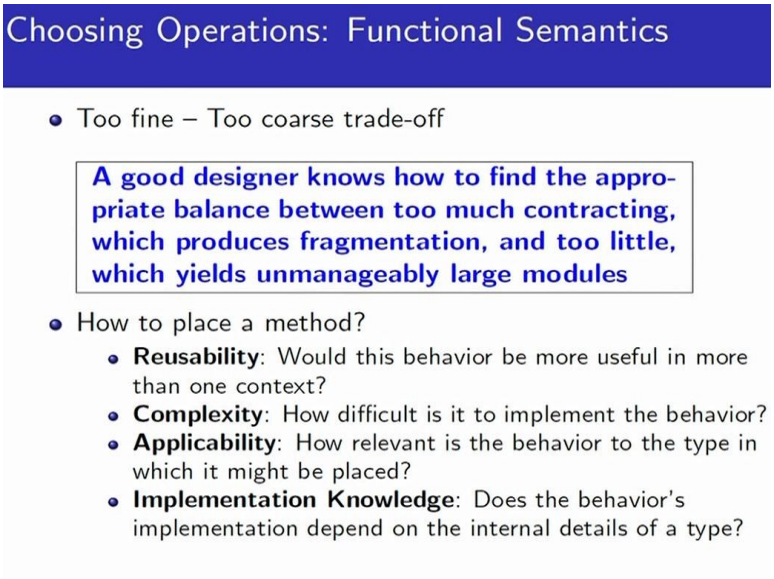


So whenever your design is done, you should try to focus on doing these measures and checking out, if you should improve on the design. Next, we will let us talk a little bit about how should you should choose your operations so that is the first thing you need to do, how do you get started with the class design. Certainly we have seen that the whole scenario is a Client - server based, whether there are clients there are servers and messages passing between them.

So the first thing that certainly we need to decide is what could be the set of messages, what could be the operations that need to be supported for a certain class before we can talk about what is its internals or what is its relationships with others. So when you are crafting a interface is not an easy problem at all.

But when you are crafting that then you need to primarily take decisions about these two aspects, one relates more to the overall behavior of the interface and the other relates more to the non-functional implementation related nature of the interface. The functional semantics and the time space semantics, so let us talk about this one by one.

(Refer Slide Time: 08:32)



In terms of functional semantics that is what should be the different methods that you perform, that you provide to the class, that is a big trade of is, I mean these are standard prescriptions from very successful designers but I will put it in very simple two word terms for you to remember is when you are designing operations, when you are designing methods, you have to always make a trade - off between being too fine and being too coarse which I mean is if you have operations, which every operation which does very small, small things like.

I have an operation which just increments the value, I have another operation which just takes a number and takes out the LSB of that number and so on. Probably, I am getting too fine, at the same time if I am designing for a Bank account management, then I say that I have one interface

through which the username, user ID and password is taken. It is authenticated the account is identified.

It is displayed and within that the same method itself the account balance is shown. If there is any overdraft default that message is thrown out and so on. So all of that I will do in a single message, in a single operation, then I am getting two codes. So that trade - off is very very important. And certainly that is where object oriented analysis and design become somewhat little bit towards arts feel than being a very concrete science.

Because you cannot exactly ever say us to, what is too fine and what is too coarse. But you will always have to ask yourself this question so whatever operation you design, you think, I mean should I have broken this down into two or more other operations, at the same time, ask for the class that given all these methods, is there are a scope combine two or more operations and do something better.

For example, I could talk about stack you will find in many text books, you have an interface for a stack, where the push works in the same way, but pop has a definition that removes that top most element and returns you that element and there is an empty. Now, if you define a pop like that which removes as well as returns you the element then you will have to put a judgment as to, are you making it too coarse, should you split this into two operations, in terms of one which removes the element and one which returns the top element and be strong that you should do the design.

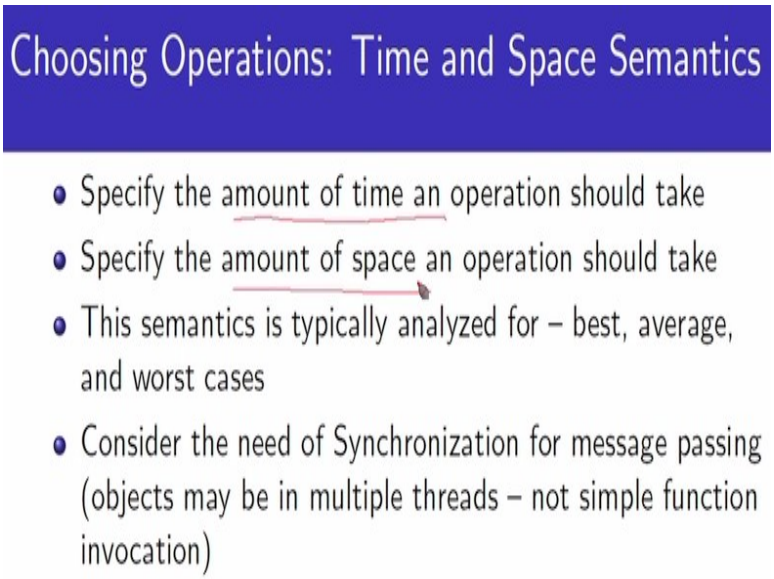
And a quick analysis in this case, will tell you that it is always better to have a separate method to return the top element and a separate method to actually remove the top element, which then you would not return very simply because if you do not have a top, you use pop to do all that then after pop is a only way to check an element, so after checking that if you feel that you did not want to remove this element then you will again have to push this element back.

So that means additional work that means additional time and in some cases, it might mean a very different semantics. So that those kind of choices are what is important and to place a

method that is if you find that ok this is the operation that aI can see then I need to find out an appropriate class for doing that, decide whether there should be a class for this, these are the factors that you should keep in mind that certainly you should put it in a class which can be reused.

You will have to look out what is the complexity of its implementation, how often it can be used and of course, it will depend on the knowledge that you have, the designer has in terms of the behavior's implementation, the platform dependence and so on. But, these are some of the factors that we will need to keep in mind in deciding how we close on the functional semantics of an operation.

(Refer Slide Time: 13:07)



Now once we have the functional semantics then there are factors relating to more details of implementation and some of the questions that need to be answered is, in terms of amount of time and amount of space. Certainly you will need to have some idea as to what you want, for example if you are a very arctic example will be about finding elements from a large pool of existing data elements.

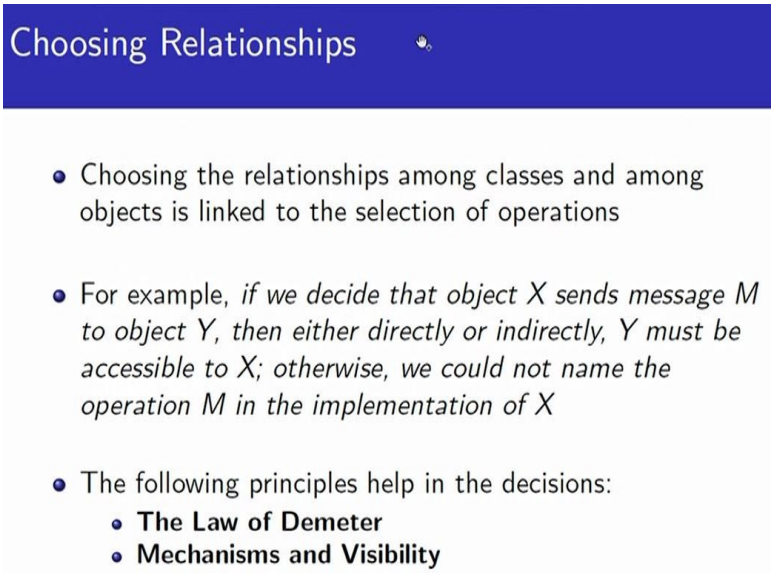
So, if it is important that you need to find the elements very quickly, then you will possibly design the operations in a way, so the time is optimized. But you may be able to afford some space but it could be other ways, if you really do not care about how much time it takes, but you

are trying to implement the functionality of this search on a mobile phone, where space is a bigger premium than time.

So those are the time, space semantics that need to be decided, you could do that based on a best case, average case, worst case and so on. And specifically we call that whenever you have operations, the operations are messages acting on objects, objects are in a concurrent situation. We have explained the different situation of synchronization between them. So, the time space semantics also will have to look into whether there is a need for synchronizing, whether there is a need for mutual exclusion that we need to put to in the design.

So your overall choice of operations should be guided by this requirements of the functional semantics and the time space semantics.

(Refer Slide Time: 14:58)



Moving on once you have chosen the operations, the next would be to choose the relationships amongst classes. What it means that suppose I have say, I have object X and that sends message M to object Y. Now, certainly this message M is what I have decided on, because I have chosen my operation, so I know the message M. Now, once we have that then naturally if this message has to be send either directly or indirectly, the necessity would be that Y must be accessible to X.

That is in terms of different encapsulation visibility restrictions and so on, unless is Y accessible to X this message design, the operation design will not work. So, the relationships have to be chosen keeping in mind the different choice of operations and the way you have started doing the design.

(Refer Slide Time: 16:05)

Choosing Relationships: The Law of Demeter

- The Law of Demeter:

  The methods of a class should not depend in any way on the structure of any class, except the immediate (top-level) structure of their own class. Further, each method should send messages to objects belonging to a very limited set of classes only

- How to analyze the Class Inheritance Hierarchy Structure?
  - **Wide and Shallow**: Usually represent forests of free-standing classes that can be mixed and matched – Forests of classes are more loosely coupled, but they may not exploit all the commonality that exists
  - **Narrow and Deep**: Usually represent trees of classes that are related by a common ancestor – Long chain of inheritance is tightly coupled
  - **Balanced**: Between the above two extremes
- Similar trade-offs among inheritance, aggregation, and dependency relationships to be analyzed
- Should class Car inherit, contain, or use classes – Engine & Wheel?

  Inheritance is appropriate if every instance of B may also be viewed as an instance of A. The client relationship is appropriate when every instance of B simply possesses one or more attributes of A

And their two principles which people follow very frequently. First is the law of Demeter, the law of Demeter says that in terms of choosing the relationship, you have to make a lot of trade - off judgment, which is the first, I am just trying to use one inheritance hierarchy to explain this that I can have a hierarchy which is more like say, like this, 1 base class, one superclass everything else is a leaf child class, is a one kind of Hierarchy or maybe not only that but I have this, lot of here.

So these kind of we will say wide and shallow that is where the depth of the hierarchy is less, but the breadth is very high. So if the breadth is very high then you have a large number of leaf classes, so it will give you a forest. Now, certainly it is much less like a tree, it is more like a forest, so all these leaf classes are independent. So if they are independent they hardly share the properties that could have been shared, that is a basic purpose of inheritance to share properties across classes.

So if the hierarchy is very wide and shallow then you are not exploding the commonality that exists and you are missing out on key design clue that may happen. On the other hand, on the extreme if your hierarchy is very narrow and deep suppose you have this, that is this width is low but this depth is high. Then, also it will not be a good design because you will have long Chains of inheritance where you have just every class is trying to get something from the parent and so on, possibly your find grind is so much that you do not have a good manageable design.

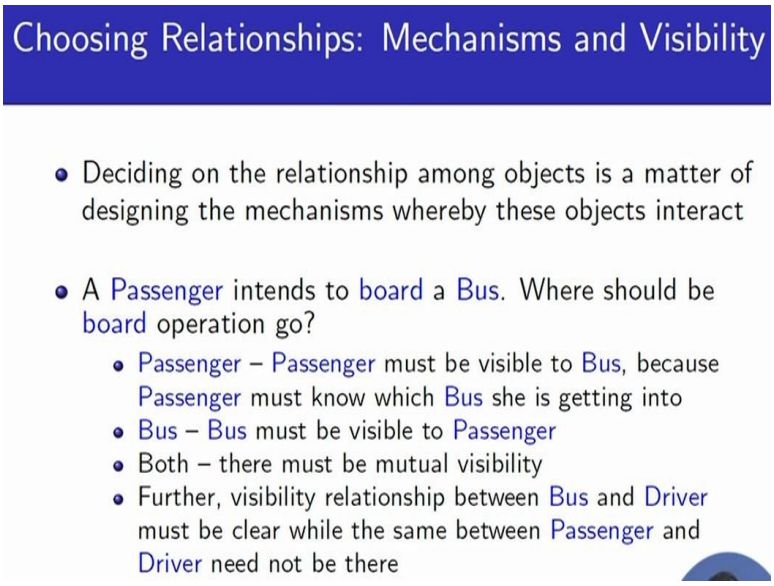
So you recall the kind of vehicle design we discussed about in the earlier module, the module 14 those kind of are known as the balanced design, where you have kind of two extremes that your depth is not very high, at the same time, the width is also not very high so there is a fair amount of commonality that is exploited and at the same time in terms of exploiting that you have not made the whole design very find grind and decide on that.

The similar principles can be used amongst single inheritance in cases of aggregation that is how much to compose, how many different parts should I break it on to do it I manufacture to represent a car, Do I take the car and then break it down in terms of and nuts and bolts and wires and connectors and seat covers and all that or I do a hierarchical aggregation there, what is a dependency relationships and so on.

So on the same principle of trade - off what is very fine and what is very coarse this relationship designs must be done and also there should be design choices made in terms of how do you relate that, for example you have car and engine. Certainly, I just started saying that this is a case of aggregation but who said this has to be a case of aggregation, aggregation is one way of looking at itself. Car contains engine is one view.

Now could I say that car inherits from engine is a specialization of engine, will have to see whether that might sense. But you can certainly say that car uses an engine, I think of car as a separate entity, engine as a separate entity, car uses an engine. So there is certain decisions to be made in all these cases as to, which one is the right way to relate different classes and accordingly you can you should make that choice and that is what the law of Demeter talks about.

(Refer Slide Time: 20:28)



Besides that, you will have to look into the mechanisms, the visibility requirement. Suppose a passenger intends to board a bus, so there is a passenger class, there is a bus class and you need to place the board method and for that board method you need to have the visibility between passenger and bus.

Now there are certainly 3 choices you can have a board method in passenger and let the bus goes as a parameter to that, so that where you send that or the bus can have a board method with a passenger sense it and goes a parameter or both could have board methods which collaborate between themselves. Now depending on which class has a board method, the visibility will have to be accordingly. For example, if bus has a board method then bus must be visible to the passenger.

So the passenger can send the message to it. So these are the different choices in terms of mechanism and visibility that you will need to apply to decide on the relationship.

(Refer Slide Time: 21:30)

Choosing Implementations

- Next consider the inside view
- This perspective involves two different decisions:
  - Representation for a class or object
  - Placement of the class or object in a module

The third aspect in terms of the quality of design will depend on the implementation which is often would mean a lot of skill, lot of understanding of your vehicle of implementation that is the implementation OOP language as well as the platform. I would just like to highlight that two factors that need to be critically looked at regularly is the representation that you are using for the class and the packaging or the placement of the class that is in a large system.

There are large number of classes, objects, modules, objects occurring, now you need to certainly group them into certain sub systems, certain modules. So the representation has to be appropriate as well as the grouping has to be appropriate. So that again the traffic across modules can be minimized and they can be more cohesive in that form.

(Refer Slide Time: 22:25)

Choosing Implementations: Representation

- Representation should be one of the encapsulated secrets of the abstraction
- Possible to change the representation (like, to alter the time and space semantics) without violating any of the functional assumptions that clients may have made

The choice of representation is often a fairly difficult one, and it is not uniquely determined by the facilities available. It must always be taken in light of the operations that are to be performed upon the data

- Time or Space? What has priority?
- Search or Insert / Delete? What has priority?
- Compute or Cache? What has priority?

So representation should be done in a way so that you remember the basic principle of the implementation, that implementation must hide, it must encapsulate all the secret, all the details of the implementation. Couple of modules back while discussing about the nature of classes I talked about this aspect with an example of a stack and I show that how for the same interface we could have multiple people different implementations of a stack, which could change irrespective of, which interface the customer is using and without the knowledge of the customer.

So that is a very basic requirement that any implementation must satisfy that, any change in the implementation usually should not impact the interface contract with the customer and that is what you should try to follow but there are several trade - offs to be made, several choices to be made resemble, for example some of the common ones are should I optimize for space or should I optimize for time. The new dimension being added in many places, should I optimize for power?

Similarly, in terms of any application that needs to do some kind of a search somewhere there is always a choice between do I optimize on search, or do I optimize on insert, delete. We all know that if we use some kind of array structure, we will have very optimal search but very inefficient insert, delete. If you do a list, insert, delete is very quick but the search gets very slow. If for that we want to do a binary search tree, then possibly these two get balanced.

But I need more time rather I need more space, because now the binary search tree has different pointers to be managed and so on. So there are different choices, another very common choice in terms of the class design that we will need to do is compute or cache. That is there are several values which you could compute on the fly or compute once and keep it stored.

I will just give an example, suppose you regularly need the age of an employee now what you could do is, naturally age is dynamic so it does not remain the same so what you do you keep the date of birth, so what you could do whenever the age is required you could compute the age and return that.

Typically, you do not need the age in the granularity of the days, you need it in terms of years. So one we could be that every time you do this that is basically compute your choice, the other way could be that ok, you have an age field and which you compute on 1st January for all employees. Because your logic is that within that year the age will not change or whatever changes will happen that is just for one year which does not impact or on the date of birth of that employee you compute the age and store it.

So what you are doing, you are caching the computation. So there is a tradeoff between what, which one should be used and these different factors will decide what kind of representation, what kind of implementation you will choose for.

(Refer Slide Time: 25:51)

Choosing Implementations: Packaging

- How to put Classes and Objects in Modules?
- Visibility and Information hiding trade off

> **Applying this principle is not always easy. It attempts to minimize the expected cost of software over its period of use and requires that the designer estimate the likelihood of changes. Such estimates are based on past experience and usually require knowledge of the application area as well as an understanding of hardware and software technology**

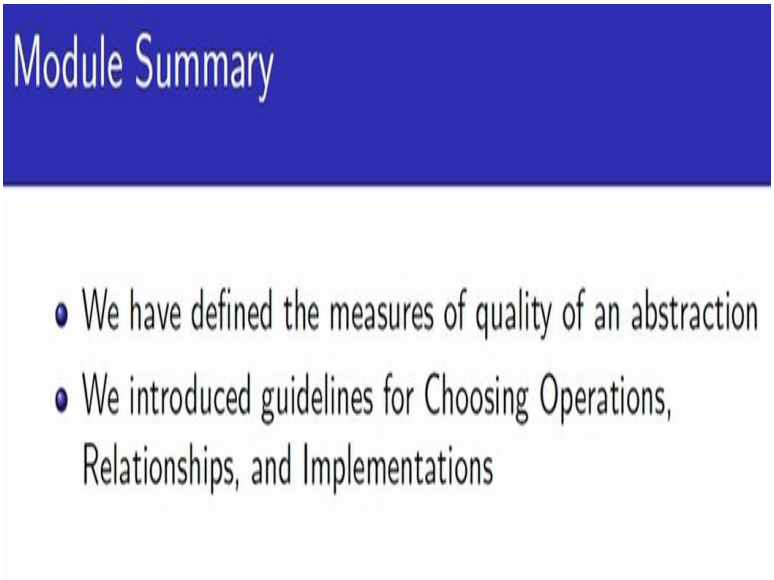
- Many nontechnical factors – matters of reuse, security, and documentation, ...

Finally, in terms of packaging you have to put the classes and objects in modules. If you are using a platform OOP language like Java or C sharp then you will, these are actual physical decisions because you have to decide, which java package some classes going and the other classes will possibly go in some other package and so on. If you are doing in C sharp, you will have to decide on the projects, if you are doing in C++, you will need to decide on your own packaging in terms of different source files, header assemblies and so on.

But this is a very critical decision to make because this will directly have an impact on the visibility and information hiding the trade-off between visibility and information hiding because if you put a certain classes together in to one module then it is much easier to make them mutually visible and hide all of their visibility from someone, who is not in the same module and so on and so forth.

So those are the factors in packaging besides that there could be lot of non-technical factors like how you want to reuse, if you want to reuse something across variety of different modules you would not like to put them hard into one module but may be have a separate module for it. The security could be the matter, the documentations several other factors which could impact the choice of packaging in implementation.

(Refer Slide Time: 27:32)



So to sum up in this module, we try to introduce you to the concept of having certain quality measures for designing the abstraction and for designing the classes and objects in terms of coupling, cohesion, sufficiency, completeness and primitiveness and then for the three major aspects of the design dealing with the choice of operations or methods, the choice of relationships between classes and finally the choice of implementation we have tried to talk about some of the pinpoints that can happen in the whole design, some of the parameters that you need to keep in mind while you go ahead with the design.

We will take this forward in the modules for the next week, where we will expose you with different semi-structured techniques of identifying objects and classes and relationships and as you do that all these principles that we have talked about in this module you should bear in mind and more and more you can put them into practice, you will become a better designer of the object system that you are given to work with.

Object-Oriented Analysis and Design

Ms. Tanwi Mallick
Ms. Srijoni Majumdar
Teaching Assistant
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
Tutorial I
LMS

Good morning everyone. We are here to present the first tutorial of the courses object-oriented analysis and design. What we are going to do in this tutorial is that we are going to introduce a problem, the first level discussion of that problem between a client and a vendor. That is the first initial thing which happens for any object-oriented development. So, the problem is leave management system and this leave management system will be provided the all the requirements of this leave management system will be provided to the vendor by the client in this tutorial.

"Client – Vendor conversation starts"

Client: Hello, I am Tanwi, am acting as a client.

Vendor: Client yeah and I am Srijoni I am acting as a vendor. So, we will start the discussion then.

Client: Yeah so, I am coming from a company which need a leave management system to record the attendance and the leave of the employee. So, can you provide us automated software which can do this thing.

Vendor: Yeah fine I actually I have been sent by my company which has been contacted by your company to build this module.

Client: Hmm.

Vendor: I am here primarily to gather the requirements of you leave management system. So, we will start with this requirement analysis now. Firstly, can you give a brief overview of what you want your leave management system to do.

Client: Yeah actually in previously it was a manual task.

Vendor: Hmm.

Client: Employee used to give their attendance and and apply for the leave manually and the leave approval was also manually. So, we want a system by which we can use all this functionality by clicking a button only.

Vendor: Yeah.

Client: So.

Vendor: You want a automated leave management system.

Client: Yeah, we want a automated leave management system.

Vendor: So. Okay. we have some queries which we are going to ask you now. This is going to enable us to understand the system better because we need to understand all the requirements of the system to build it accurately for you. So, we will ask you the questions now.

Client: Yeah of sure.

Vendor: So, the first very important question about the system is that who will be using the system in your organization.

Client: All our employees are going to use the system.

Vendor: Okay, so are there any categories of your employees like they will be having some different access rights in the system like some may be doing something, some may be doing another functionality.

Client: Yeah.

Vendor: Okay.

Client: We have these categories.

Vendor: Okay.

Client: In the base level, we have executive.

Vendor: Okay.

Client: Executive employee can apply for leave only and they have they will give attendance. That's it.

Vendor: Okay.

Client: And then there is lead.

Vendor: Okay.

Client: Lead can approve the leave.

Vendor: Okay.

Client: Okay and the manager. Manager can credit, debit and adjust leave of the employee.

Vendor: Okay.

Client: And also, can approve the leave.

Vendor: Okay so what I understand that there are mainly three categories of employees in your organization. The first level of is your executive.

Client: Yeah.

Vendor: Who will be only using this system to request for leaves and to give daily attendance.

Client: Yeah.

Vendor: The next level is lead and that the leads will be able to approve the leave of an employee of the

executive who is reporting to that lead and then above another level is a manager who will also be able to approve the leave and may be do some more functionalities.

Client: Exactly, exactly.

Vendor: And so, I think the lead and the manager all will be doing the the other task which the executives are allowed to do, I mean they will be.

Client: Yes absolutely.

Vendor: So, they will raise their leaves through the system and do whatever the employees are doing, the executives are doing but apart from that they have some extra responsibilities.

Client: Yes.

Vendor: And and obviously that is building up like for example for leave you have some extra responsibilities and then the managers will have all the responsibilities of the lead as well as some extra rolls which they want from the system. so kindly can you just know summarize what you actually wanted every category of employees to do.

Client: Yes, Yes, I have a written document let me look into it. Em a employee can can record daily attendance okay.

Vendor: Okay.

Client: Request for leave, cancel and approve leave, not yet avail.

Vendor: Okay.

Client: And avail a leave if it is approved only.

Vendor: Okay.

Client: Check and export own leave status for a period.

Vendor: Okay.

Client: This is the this is the functionality of employee.

Vendor: Okay.

Client: Okay, on top of it, a lead can approve or regret a leave, request from an executive.

Vendor: Okay.

Client: Okay.

Vendor: Okay.

Client: And revoke and approve a leave from of an executive.

Vendor: Okay, okay, okay.

Client: And manager is on top of lead.

Vendor: Okay.

Client: Okay so manager has some other permission also like credit, debit and adjust leave for

employee.

Vendor: Right.

Client: And perform administrative functions.

Vendor: Okay, okay. So, which means that just to summarize and understand this is one major thing and your organization, we need to understand the employee responsibilities.

Client: Hmmm.

Vendor: And the functionalities they want from the system. So, your base level employees that is the executives they want around five functionalities from the system.

Client: Yes.

Vendor: And the lead wants theses 5 functionalities and on top of it, they will be having 2 extra functionality.

Client: Yes, Exactly.

Vendor: To approve and to revoke any approved leave and the manager will have all these functionalities, on top of it they will also require some extra functionalities like crediting, debiting.

Client: Yeah.

Vendor: Leave adjusting and all these things.

Client: Yes.

Vendor: Okay so this is a detailed document, we will take it from you like but we understand the organizational structure like what responsibilities they would be having and what functionalities they would be wanting from this system. Okay, so now we have some questions about the leaves like all leaves are the same?

Client: No no no.

Vendor: Okay okay okay. So, do you have many categories of your leaves.

Client: Yeah.

Vendor: So like can you tell me how many categories.

Client: Yeah, we have 7 categories of leaves.

Vendor: Okay Okay.

Client: So.

Vendor: And can you name them?

Client: Yeah, Casual leave, earn leave, Duty leave, Sick leave, Maternity leave, Parental leave and leave without pay.

Vendor: Okay so these are thee 7 types of leave which you give to your employees at present.

Client: Yes.

Vendor: So like can you just briefly describe any one of the leave may be a casual leave like what are the.

Client: Yeah.

Vendor: Features of those leaves on.

Client: Okay, so emm 10 causal leave are available in a calendar year. Okay, all causal leave is credited to an employee on first January. Okay. For employee joining in the middle of the year, the number of casual leave are propagated. Casual leave cannot be carried over in the next year. More than 2 casual leave cannot avail at a time. Okay and casual leave do not pre do not need pre-approval but must approve within 2 days of availing.

Vendor: Okay, so listening to your description about your leave, I think I have noted down some important basic points who based on which I can I think you can describe a leave. So, for example like if we do this thing for a earn leave now.

Client: Hmm.

Vendor: I will be asking you these basic points.

Client: Hmm.

Vendor: And you can reply me what the earn, for what earn leave falls for.

Client: Hmmm.

Vendor: And then we we maybe we can you know finalize these basic points based on which you will differentiate a leave.

Client: Yeah sure.

Vendor: For example, emm like can you tell me what is the duration of leave for an earn leave.

Client: Yeah, it is 15 here.

Vendor: Okay, Okay. So, this is a distinct feature of any leave that you will be having specific duration.

Client: Hmmm.

Vendor: Okay. This is one feature which we fix it now. Is the leave clubbable like in causal leave I think it was not clubbable? So, earn leave, it is clubbable.

Client: Yeah it is clubbable, casual leave is not clubbable emm rather than casual leave, all other leaves are clubbable.

Vendor: Okay, okay, okay. So apart from casual leaves, all your other leaves are clubbable.

Client: clubbable yeah.

Vendor: Okay like if there is any holiday which is coming in between the leave duration taken, so will it be exempted for.

Client: Yes.

Vendor: For earn leave.

Client: Yes, it is exempted from for earn leave. But it is not exempted for casual leave.

Vendor: Okay, Okay.

Client: For all other leave, it is exam exempted.

Vendor: Okay, okay, so so this is an exemption policy which you qualify your leaves.

Client: Yeah.

Vendor: Okay does it have to be pre-approved? I think casual leave doesn't need to be pre-approved.

Client: Yeah but earn leave.

Vendor: Okay.

Client: earn leave need to be pre-approved.

Vendor: and.

Client: And all other leave except sick leave and parental leave, all other leaves are emm preapproved.

Vendor: Okay okay, all your leaves are pre-approved ok and now like your leaves get carried over or like accumulated like earn leave, does it get carried over?

Client: Yeah earn leave gets carried over ehh and but not not all leaves are carried over.

Vendor: So, what is the encashment policy for an earn leave.

Client: Yeah, so yeah it is also written in my document. Okay, 1 and 1.25 earn leave is credited on the completion of a full month service. Okay.

Vendor: Okay.

Client: So earn leave can be carried over to the next year and accumulate up to 45 days. Okay.

Vendor: Okay.

Client: Once it crossed 45 days, then on the completion of the current quarter 30 days are encashed and paid to the employee.

Vendor: Okay, so I understand. So, this is your encashment policy which and obviously the leaves which will be allowed to be encashed will have a specific encashment policy and obviously I think you do not need any certification for earn leave.

Client: No no no, no certification is required for earn leave. But certification is required only for sick leave. Okay.

Vendor: I mean you need your medical certificates for them.

Client: Yeah.

Vendor: Okay. Fine. So, I, like what I understand is that way I.

Client: Sick leave.

Vendor: Yeah, yeah.

Client: There are some other leave also which needs certification.

Vendor: May be parental leave.

Client: Yeah parental leave and maternity leave.

Vendor: Yeah whichever there is a medical.

Client: Medical issue yeah.

Vendor: Medical issue involved, you will be needing certificates for that. Okay so now what I gather from your requirements is that your leaves, your you have 7 types of leaves in your organization and all these leaves, they have some basic features okay.

Client: Hmm.

Vendor: And like these basic features like which we discussed now, whether it is clubbable or whether it is holiday exemption, whether it can be pre-approved, whether it can be carried over, whether it can be encashed, whether it needs certificates. So, based on these basic points, we can actually describe your leaves.

Client: Yes.

Vendor: And differentiate between them. So, what we will do is that kindly we if you could give us any more document if I have missed out on any of your basic points and if you have any more.

Client: The written document which I have, I will give it to you.

Vendor: Yeah, okay so that would actually help us to understand the nature of your leaves because based on these basic points we can also differentiate between the leaves because this is important for us to build it, build these needs into the system because they are going to behave differently. So, we have to understand the behavior. Okay.

Client: Okay.

Vendor: So, based on that we can I mean give you a proper system. so I understand that you are 2, you have 2 major modules of your leave management system. One is the employee module where actually we have understood that there are 3 categories of your employees. They will be having different responsibilities; some responsibilities are common all throughout and some have extra responsibilities. So, these responsibilities which will be built as a functionality in our leave management system.

So, they will be having access to extra functionalities and the data on which it is going to work is the leaves okay.

Client: Yeah.

Vendor: So, the behavior of the leaves are been, we just now described like how are the leaves going to behave, what are the different types of leaves. Okay, how do we differentiate between the leaves

because they are going to enter the leaves, so we need to understand behavior and give them options to you know fill in the required details which we require. So, I think this is a fair enough discussion, so kindly can you share with us the document.

Client: Yes, I will share the document with you and you gave me the specification whatever you have extracted from our written document.

Vendor: Yeah, yeah sure we will give a.

Client: And we check whether you have missed out anything.

Vendor: Yes yes, once we get a formal document from you, we will do the requirement specification document and provide it to you. Thank you very much. Yeah.

Client: Thank you Srijoni.

Vendor: So, good, Hello everyone. So, what we discussed here was the requirement gathering of leave management system.

So, hi I'm Srijoni, I'm a TA of this course and am Tanwi, I am also a TA of this course and we will be sharing with you this document which we have written down like basically yeah to you know explain the leave management system. We have everything written down, what we discussed. So, we will be sharing it to you for you know which you can refer and we are going to use this example all throughout the course like whenever we understand any aspect of object oriented systems, we are going to relate to this example.

Yeah. Okay. Yeah fine. Thank you.

Thank you.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

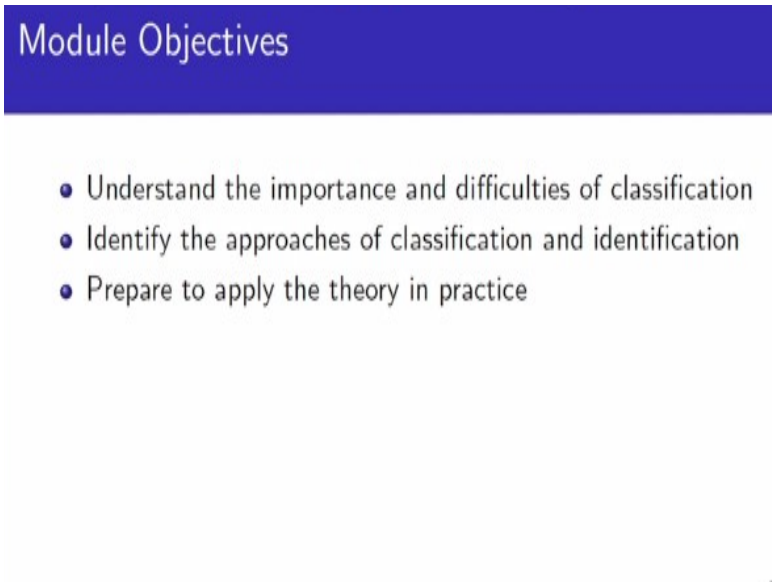
Lecture – 28
How to Identify Classes and Objects

Welcome to module 16 of object-oriented analysis and design. We have so far been taking different analytical looks into complex systems, their attributes and variety of commonalities and approaches that could be adopted to analyse these complex systems from various aspects of objects and classes and their relationships.

In specific, we have looked at different elements of major and minor elements of objects; we have looked at the nature of objects and that of classes we have looked at their relationships in depth. From this point onward, we are slowing taking the course towards practicing all the different principles that we have already discussed.

So from this module onwards where we will talk about some of the major tools and techniques of actually identifying objects and classes from a given specification of written, verbal, interactive or pictorial kind and start taking example and active exercise example with a leave management system and some of the other systems to illustrate how as a practitioner you actually engage in the object-oriented analysis and design processes.

(Refer Slide Time: 02:05)

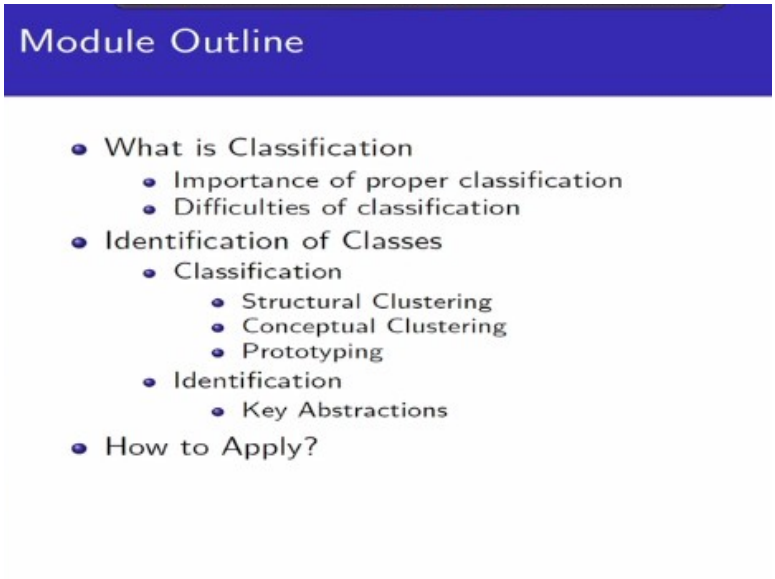


So this module is to understand the importance and difficulties of classification. We have

seen in the earlier discussion theory that whatever where we look at the system, it is very critical to identify objects, identify classes. So given the whole bunch of concepts and dynamics of the system, it is a critical requirement to classify them according to certain commonalities which we can capture in terms of the classes.

And they have been a lot of work on this in terms of classical approaches and then they were followed by several modern approaches. We will not go into that historical development; we would rather focus very specifically into how we can adopt these and apply them in practice.

(Refer Slide Time: 02:54)



This will be the gross outline of the module and we will continue to display this on the left hand side of every screen.

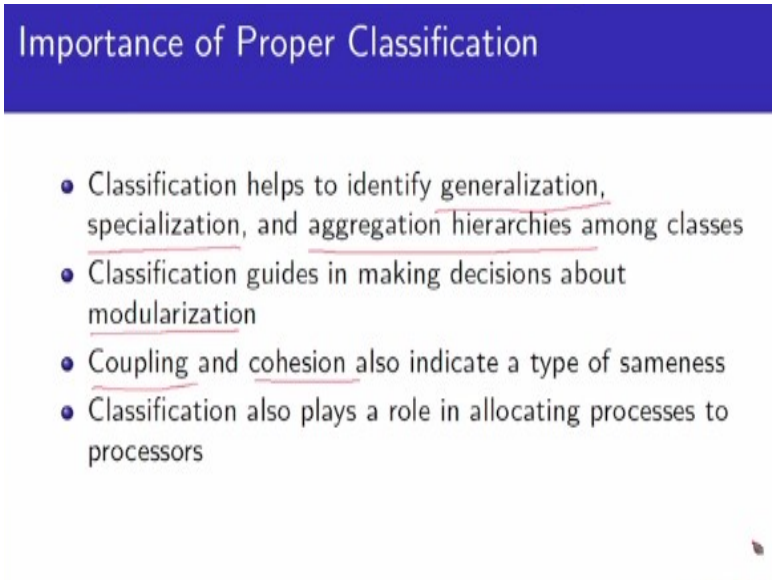
(Refer Slide Time: 03:07)

So, first let me briefly talk about what is classification, of course, most of you would know that classification is a process whereby we put some kind of an order in other knowledge. Some kind of a boundary around things that are similar, tasks that are similar or actions that are similar or they share something.

So here in this illustrative diagram, I am trying to show that we often, when we these days go to dispose garbage, then we get a number of different vats, some written with glass, some written with plastic and so on and depending on what we have whether we have a vegetable waste, a biological waste or we have a plastic cup in depending on that, we need to decide which vat we should put our garbage into.

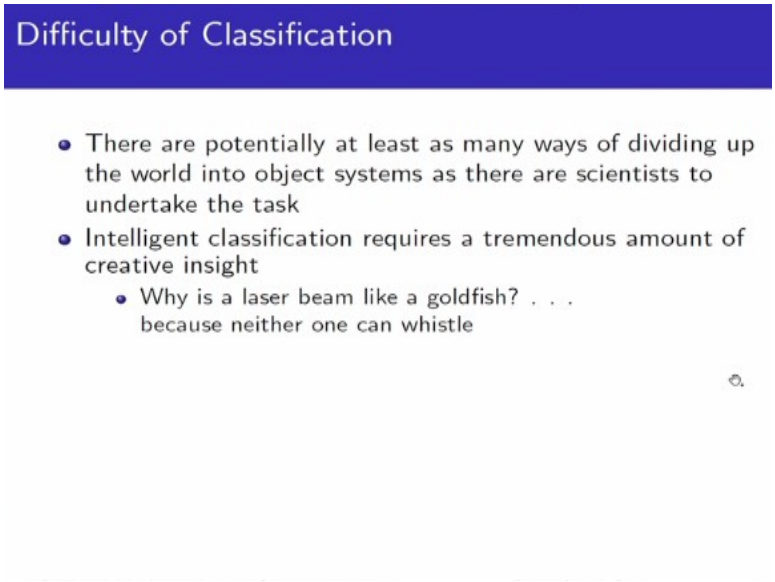
So this process that we are doing mentally is actually a process of classification and in general we will need to do this for given the system that we would like to analyse.

(Refer Slide Time: 04:17)



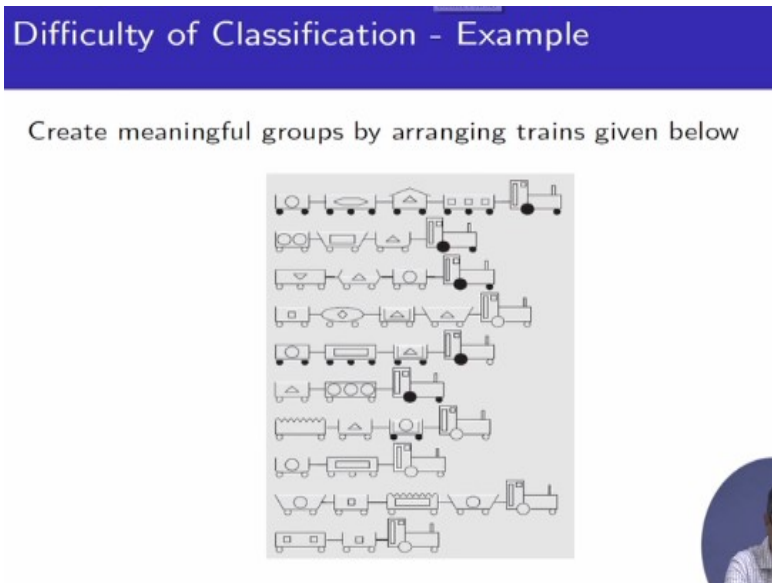
So classification will help to identify generalization, specialization and aggregation amongst the hierarchies amongst the classes. It will also help in making the decisions about to modularization that is what all items, what all classes, what all processes we should put together. In terms of coupling and cohesion, the two main quality matrix that we have studied in the earlier module indicating the sameness is also decided primarily by the classification and it plays a critical role in terms of allocating processes to different processors.

(Refer Slide Time: 05:10)



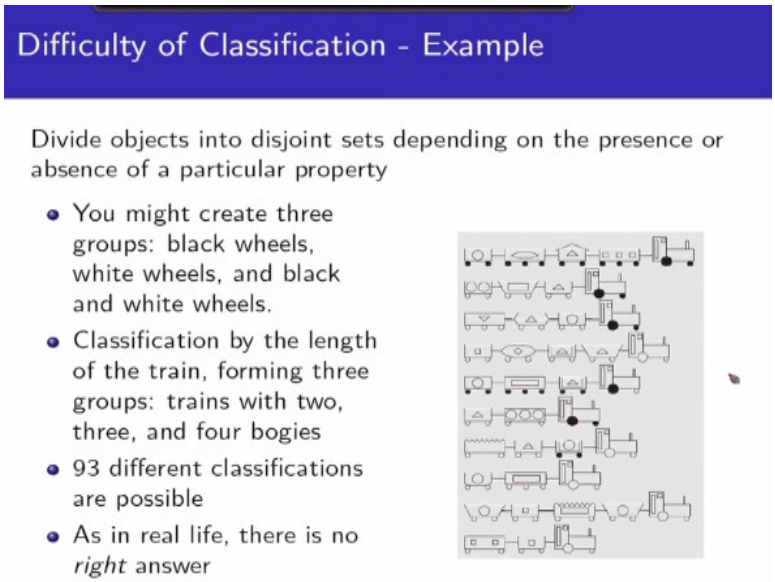
So having said that, let us recognize that the classification is not an easy problem because given the same scenario multiple observers could classify them in multiple different ways because it is not, there cannot be any very straight forward or mathematical or algorithmic ways to classify given a scenario because it pretty much depends on the experience of the observer it depends on the domain, it depends on what you and I identify as the most discriminating parameter in the classification process.

(Refer Slide Time: 05:53)



So it is a, it is quite a difficult process to go with. So just to illustrate with a little bit more concrete example, let us take this example from Booches OOAD book where we are trying to create, we have given pictures of ten different trains right. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 different trains and these trains have an engine and a couple of compartments that it drags and we would like to create meaningful groups, meaningful classification of these trains.

(Refer Slide Time: 06:35)



So, we can divide them, classify them according to, if you just do not have a guideline, if you just look into this picture and you will be able to create different kind of classification depending on, for example, the presence and absence of a particular property, you could decide that okay. I will look for certain common properties. This is pretty much like the you know the IQ test stuff that we keep on doing all that time.

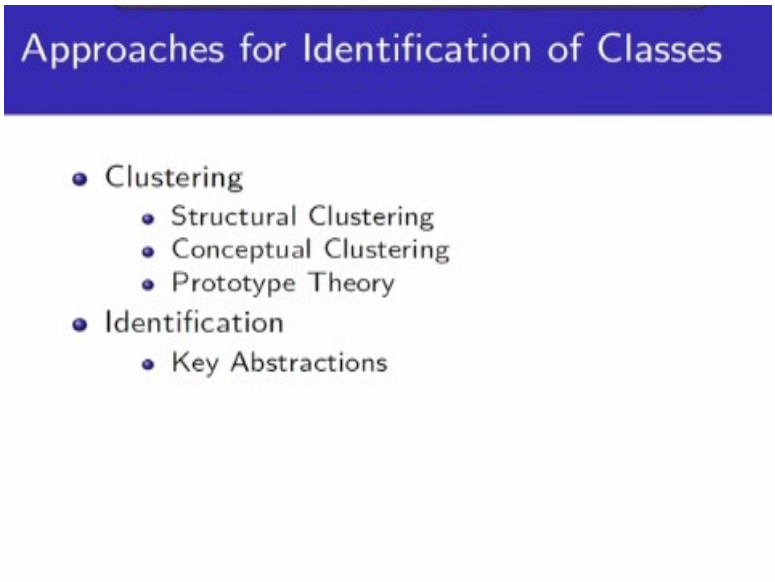
For example, we can observe that different trains have different kind of coloured wheels. Some have black, some have white, some have white and we can decide a classification based on the colour of the wheel. So we will have the trains which have black wheels, white wheels and which have black and white wheels both. So if you look at that, this is completely black wheel but this is completely white wheel, this is completely white wheel whereas this one is black and white wheel. So we can classify the trains according to this.

Alternately, we could say no, it is, that is not the right way to classify, I would really need to understand how long the train is, so I will classify based on the property of the length of the train. So if you look at that then they will fall in three groups where trains with which have two bogies, three bogies and four bogies, like this has two bogies, this has two bogies whereas this train has three bogies, this train has four bogies and so on.

Accordingly, we can classify into different groups that will get three classes for that and in this way if you try to look at all this different patterns that exist in this image then as it turns out that there are 93 different classifications that are possible and in real life unfortunately,

and that is where the major challenge of the practitioner lies is there is no exactly right answer where this is right and this is absolutely true, there is nothing like that. So we will have to work within this level of uncertainty and ambiguity that exists in classification.

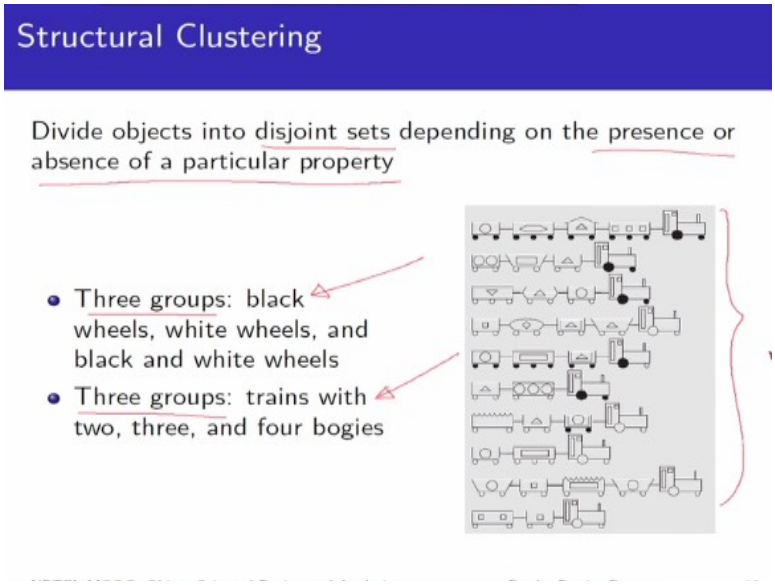
(Refer Slide Time: 08:50)



So based on this, the identification of classes have been tried in various different ways and as of the current practice it primarily relies on one on clustering and then next is on identification. The clustering is the process by which that you say that given all the different concepts that we have in the domain how can we group them together, how can we classify and make groups. Every group that you make by classification we say is a cluster.

And the other is once we have created clusters, then we identify what is the characteristics of that cluster, so that is the two basic process and once we have been able to identify the characteristics, we say that these are the key abstractions and as we had mentioned several times before, key abstractions becomes classes in your system. So in terms of clustering there are three main approaches which we will slowly illustrate in the next couple of slides.

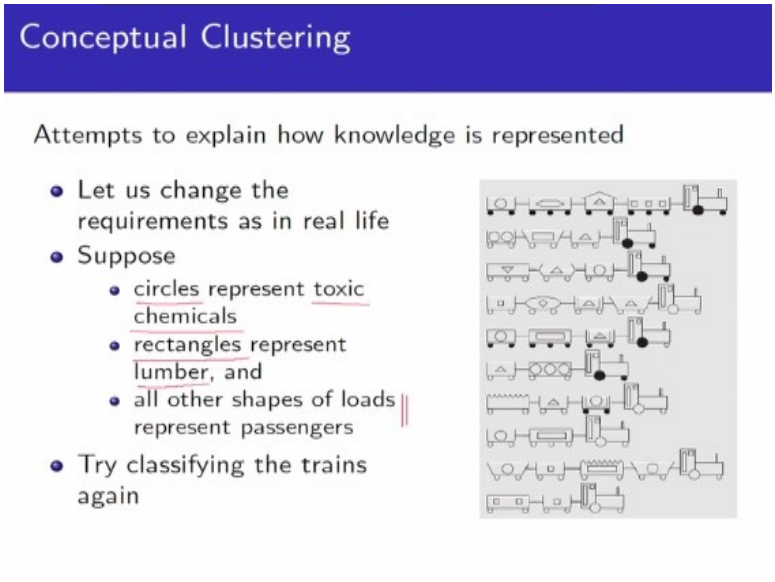
(Refer Slide Time: 09:52)



We start with the structural clustering. In the structural clustering the objects are divided into disjoint sets depending on the presence or absence of a particular property, pretty much the way we were addressing this earlier. So if we look at the colour of the wheels, then we get three groups that could be one structural clustering. We will again get three groups if you look into the number of bogies that every train has and so on.

So we are using different properties like colour of the wheel or the number of bogies and to cluster the given set of possible objects into different cluster groups and this process, since it is based on certain concrete properties is known as structural clustering. So in many situations, a structural clustering, it is relatively easy to do structural clustering because once you can (()) (10:47) on some property, then you can apply it or apply the property with its all possible values and they lead to the different clusters that will emerge from the system.

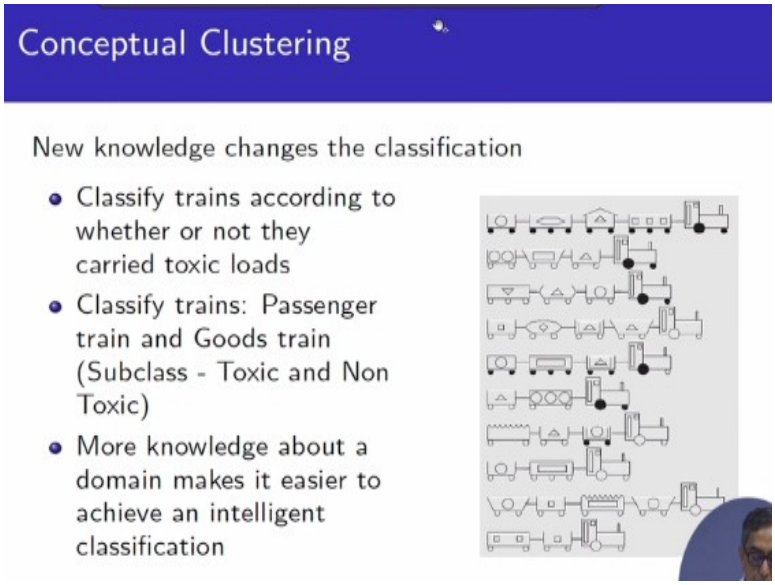
(Refer Slide Time: 10:58)



The next is often okay, I can change the game as is the reality in most cases that it is not only one or two structural parameters that define the way objects will work, the way objects will interact. So let us look at some of the conceptual properties. It is like, let us say, you can see in this diagram in this picture, that the bogies have different kinds of symbol like this as a circle symbol, this has a rectangle symbol, this has a triangle symbol, this has another circle symbol and so on.

So let us, suppose we know that these symbols characterize the type of content that the bogie transports. So let us say, just hypothetically, the circles represent toxic chemicals. Let us say rectangles represent timber logs, the big wooden logs and may be all other shapes represent passengers. So if we say that this is our basis of classification that we will classify based on these concepts then let us see where does it take us.

(Refer Slide Time: 12:18)

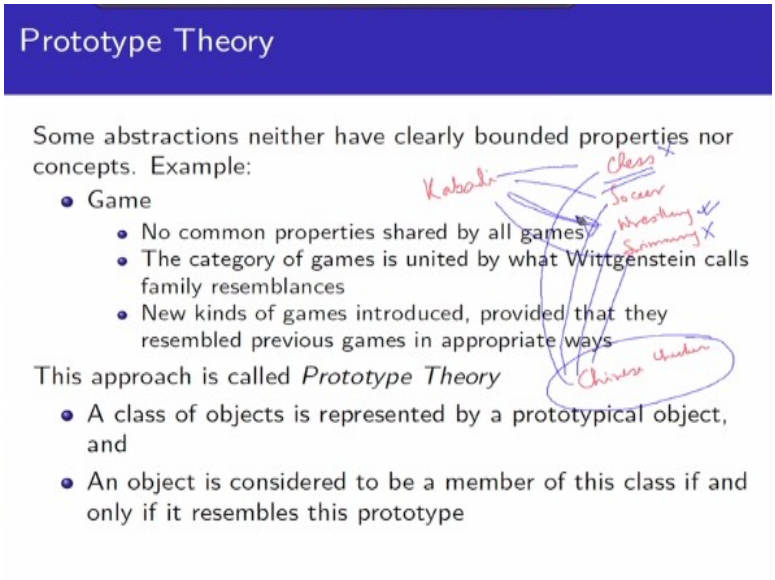


So we can classify the trains according to, for example, whether or not they carry toxic goods. We can classify the trains based on whether they are passenger trains or goods train, that is whether they just carry passenger or they just carry goods. Again if we do based on this, then we can sub classify that, is it a passenger train which carries toxic goods or is it a goods train which carries toxic goods.

So as we get more and more knowledge, now if you just look at the picture, if you just look at the diagram, then the kind of structural clustering that we had been doing is a first attempt that you can make because it is immediately visible. But here what I have done along with that picture I have given you more information from the domain that circles main toxics items, the squares mean lumbers and other means passengers and so on.

So using that you can look at the objects in a more conceptual framework and perform a better classification in as we have been highlighting. So this process as executed is known as the conceptual clustering process and it is subversive to the structural clustering process.

(Refer Slide Time: 13:35)



A third approach which at times is taken is where you will find that abstractions neither have a clear bounded properties nor a very well defined concept. In the earlier two we had either had clear bounded properties like colour of the wheel like number of bogies or concepts like toxicity, like being a passenger train or a goods train but in some domains, in many times, we will see that the abstractions do not offer that.

For example, if you talk about game now, there is no common property which is shared by all the games that we have or that we practice but the interesting thing is that usually if we say that this is, we are talking about Chinese checkers, we are talking about chess, we are talking about soccer, we are talking about wrestling. We all would be able to identify that it is a game.

So the classification in terms of games and in terms of varieties of games is not easy using the bounded properties or concepts. So it needs to be identified by what is commonly known as family of resemblance that what you do is a simple approach, you identify, you suppose that a class of objects is represented by a prototype, is okay, this is the prototype that I have.

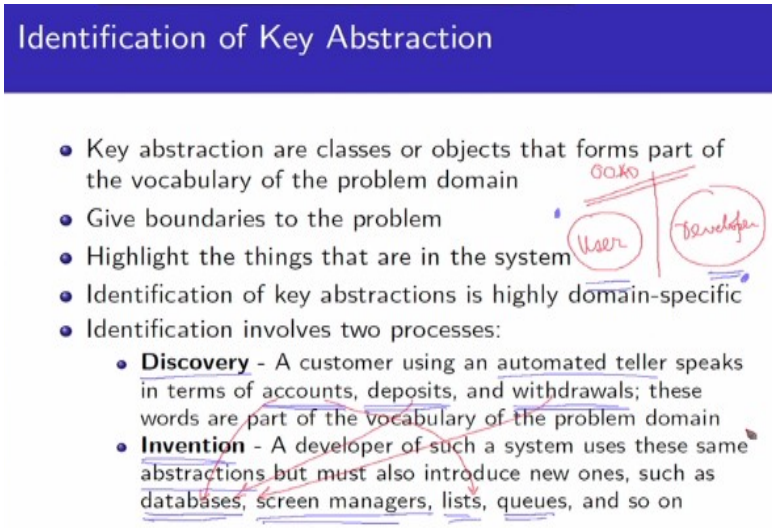
So I have a prototype which is a just let us say, and then we consider, so let us say that all the possible classes that we have, they have prototypes. So given a particular game, we would like to see what is a prototype that resembles the given one most closely. So let us say for, if we have classes like represented by chess, represented by soccer, represented by say wrestling, represented by swimming.

Suppose these are the representative ones then if you are given say Chinese checkers, what you will try to do, you will try to analyse that what it is more similar to of these different prototypes and of course if you analyse that it is expected that you will come to the conclusion that the Chinese checkers is most similar to chess and we will classify that along with chess.

And then you will try to look for some common property that may have been shared between them for example, you could come up then with an identification that okay, you classified it this way because both of them are board games but suppose if you are given a different game, say if you are given say, let us take kabaddi, So if you given kabaddi and you try to again similarly try to do this measures then certainly you will not find it similar to chess, you will not find it similar to swimming.

Possibly, you will find it most similar to wrestling and you will classify kabaddi to be in the same cluster as of wrestling. So going by this kind of a prototype based approach is known as the prototype theory which is very common in identification of abstractions in classification of abstractions particularly when the structure and the conceptual clustering approaches do not produce any meaningful results.

(Refer Slide Time: 17:36)



Now once you have been able to perform the cluster then you need to, you will get a whole lot of so in the system, now you have various different clusters that have been formed, may be too many of them. Now, your next process is some kind of a filtering that you need to do, is if you say that everything that I could cluster into segregable, identifiable items, if I say

that everyone of them has a class and it should have its own relationships and identification and all that, it might become, you might have too much of incoherent set of system design.

So then, you try to be more going to more refinement and primarily try to decide on key abstractions. That is out of these clusters, out of these classes that we have created, what are the key ones that are important and that is a very subjective decision. Primarily, the approach that OOAD takes is key abstractions try to follow the vocabulary of the problem domain that is what is the predominant vocabulary the problem domain is using and what is the boundary of this problem.

It also has to highlight the things that exist in the system and in this process the whole identification of key abstraction is extremely domain-specific. So again, it is a difficult process but if you focus on some of the typical characteristics of the system that you are discussing what are the abstractions that is classifications as identified by you, what are the abstractions that are more often talked about.

What are the abstractions that are more often active or take part in critical activities. What are the ones which fall within a certain boundary of the system and so on, if you focus on those, it will become usually easy to identify the key abstractions. Now at this point you should also note that identification actually involves two processes, one we say is discovery, other we say is a invention.

Now the basic difference between the discovery and the invention is, if you look in to the whole requirement of OOAD, then we have two sides of this whole, one is the user and one is the developer right. That is the basic thing that we are trying to do right. The user has a, is with a system, user needs certain functionalities to be built on the problem domain, that the user uses and you are the developer, we are the developer who is trying to do this.

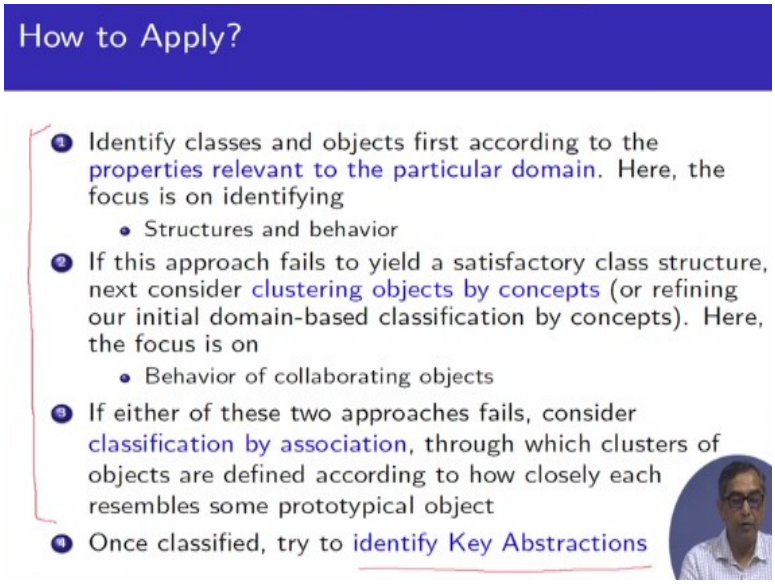
So the user has certain view of the system. So if you ask the user is to what the key abstractions are, user certainly talks about the vocabulary of the problem domain in pure form. So if the user is using an automated teller, then the user will talk about accounts, automated teller means the typical ATM machines or the bank counters where you can withdraw cash.

So that kind of user will talk about accounts, deposits, withdrawals, so these are the words which will form the vocabulary of the user and therefore will become key abstractions in your system and analysing to find these key abstractions is the process of discovery but once you come to the developer side who will implement the system, the developer will need to often deal with in addition to the key abstractions identified.

The developer will also need to deal with various different abstractions like database, like screen managers, like lists, queues and so on so forth because the implementation will happen in the domain of the developer which has a different set of vocabulary and identifying these key abstractions is a process of invention.

And naturally what the actual design process involved is how do you map these key abstractions into these different key abstractions that the developer will need to deal with, that is basically mapping the problems into the implementation and that is the role of the process of identification of key abstraction and as ever it is a highly domain dependent and quite a complex process and we will need to go through that more.

(Refer Slide Time: 22:46)



So to be able to apply these techniques and please be reminded, what I said at the beginning of this module this now we are trying to talk about techniques which more than theory will have to be practiced by us from the next module onwards, we will start applying this on an exercise and try to see how things go. So what is most important is as you recapitulate on the approaches that we have talked of please realise that the, if you are given the problem of identification of objects and classes that the first thing to do.

Then first what you should try is to, the first you should try to go with the properties relevant to a particular problem domain which means that the first approach that we are taking, so you try to learn what are the properties that exist in that particular domain, what is the typical vocabulary and all that and try to do a clustering based on those properties.

So you are trying to follow a structural clustering and identifying the class structures and behaviours that exist in the system. With this you will be able to make some progress, some of the concepts abstractions will get identified but it is likely that this will not solve the whole of the problem. So it will fail at a certain point.

It will not give you a satisfactory class structure possibly and if it does not succeed, then you go to the clustering objects by concept. Then you try to identify that from amongst the vocabulary again that what are not primarily properties but more like concepts from the domain and let us try to apply those concepts and that will give rise to the behaviour of collaborating objects.

Because concepts often are not just like, I mean if they are very concrete and with respect to only one object or only type of class then, we will be more like properties but concepts are more across different objects, across different classes. So they will show up more as collaborating objects, behaviour of collaborating objects and that is what you should try to discover from the conceptual clustering which is your second step.

A few things are still unresolved if both these approaches have not been able to completely solve your problem, then you try to approach classification by association which is the basic approach of we are going by prototype, that you try to see okay. This is, I know this is a chess and I have a Chinese checkers and let me see how that associates with a chess, a soccer, a wrestling or a swimming and based on that association.

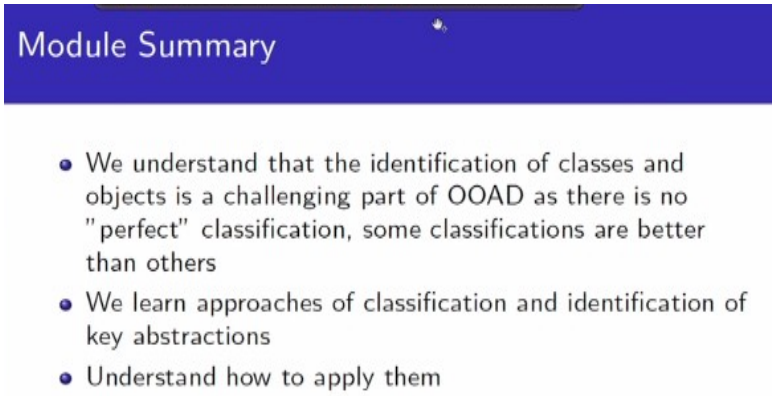
I will classify the concept and I will get a, I am using that prototypical object, I will get a classification of the concept and by this hopefully, you have been able to classify by this three. We have been able to complete your classification process and you have identified all the classes that you could do at this stage. So once, this is done, then you should engage in identifying the key abstraction that is you go deeper in the problem domain again.

You look into the vocabulary more closely and amongst all the different possible, candidate classes that you have got. You try to identify which are the key abstractions which are the most important one, the most engaging ones and try to pick them out and then look for more of their relationship to go forward. So this is the broad starting approach and again please recall that what I have been saying very repeatedly is that the whole process of object oriented analysis and design is a iterative refinement process.

So you start somewhere, apply some of the methods, make a hypothesis, make a design, based on that design you try to check how are you doing in terms of the quality of that design and then you try to refine the design. You try to use some more knowledge. So to extract some more knowledge, if required you go back to the customer and talk to the customer again.

Try to get more information possibly go the domain documentation, domain knowledge expert, get more information about the domain and again refine the design that you have done, again may be go over the whole of structural conceptual clustering and prototype been and re-identification or refinement of key abstractions discovering more of associations and then you come back and do the measure. So this repetitive process is very critical in terms of doing this design.

(Refer Slide Time: 27:56)



So in this module, we have understood and tried to understand the identification of classes and objects and we have observed that there is no right answer to this question. There is no

absolutely proper way to identify and organize classes and in many a cases one between competing designs, one may be better in certain aspects and one may be better in certain other aspects. There may not be a clear winner.

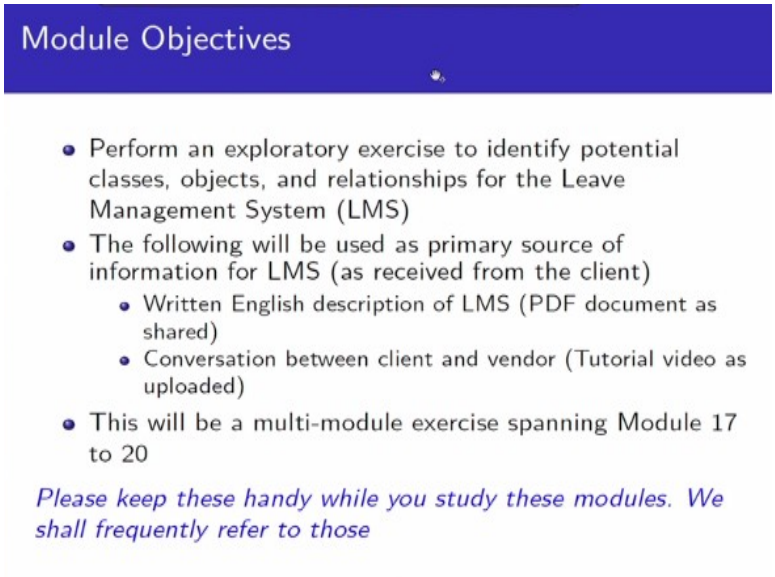
But certainly there are various aspects, various measures and representations with which we will be able to say that this is a more proper design than another one. So based on this, our next task as we go into the module 17 onwards, we would now try to with this tools at our hand will try to take up the leave management system exercise and we will try to apply them and see actually on the ground how we can do this clustering, how we can identify the key abstraction, relationships, associations and slowly start building up the design.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture – 29
Identification of Classes, Objects and Relationship in LMS

Welcome to module 17 of object-oriented analysis and design. In the last module, we have discussed about techniques of classification and identification of key abstraction for actually bringing a system specification into a concrete design. We have talked about some examples and outlined an approach. From this module actually we will continue this current exercise over a number of modules, we will start then this module and continue over the next couple of modules.

(Refer Slide Time: 01:09)



Here our objective is to take the leave management system exercise and actually try to apply all the different techniques, principles, measures that we have talked about so far. Try to apply them may be in a small scale and see how they help us, get to an object oriented design of the leave management system.

So from this point onwards, it will more an exploratory exercise, so you will also have to as you listen to this lecture, it will be good to have the leave management system exercise specification handy, we had uploaded that earlier and there was a video where two of our TAs Tanwi and Srijoni interacted as perceived customer and vendor to talk about the leave management system.

So before you go through this modules, please revise on that specification and the video and keep those handy with you and also keep some worksheet alongside so that you could at times pause the lecture and try to work out things by yourself. What since even though leave management system is a small problem, you will find that it is quite big to be covered in full in terms of this lecture. So I will just try to work out the representative parts and leave the rest of the details for you to go through.

(Refer Slide Time: 02:51)

Module Outline

- Input and Output Specification
- Informal English Description of LMS
- Identification of classes
  - Extraction of Nouns
    - Quality Check
  - Identification of Attributes
  - Structural Clustering
    - Quality Check
  - Behavioral Clustering
    - Quality Check
  - Identified Classes

So with this we will start off with the module, the outline for this particular one would be to start with the English description of LMS and try to identify classes. That is the first thing that needs to be done. So this module, we will focus on identification of classes.

(Refer Slide Time: 03:10)

Input and Output Specification

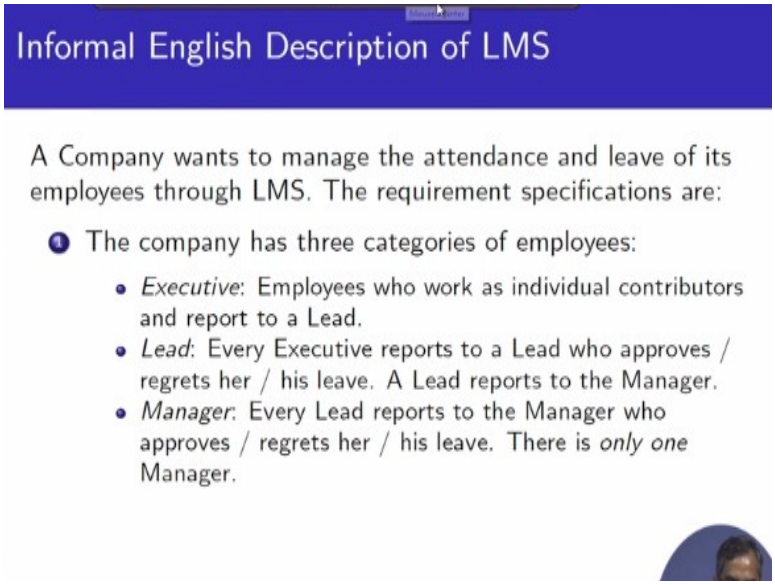
Input | **Output**

Informal English Description ⇒
- Classes
- Attributes
- Responsibilities

Class Name	
Attributes	Responsibilities
•	•
•	•
•	•

Naturally, broadly I specify the input output that we are trying to achieve. The input therefore of this exercise is informal English description, the leave management system description and the output would be classes associated with that if in the process we can identify some attributes, some responsibilities, we will try to do that.

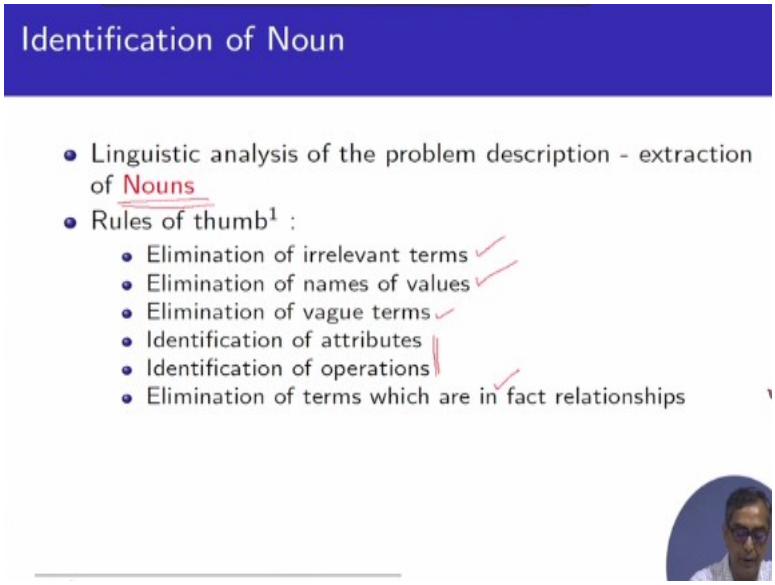
(Refer Slide Time: 03:31)



So this is, what is our input, now this is if you refer, please refer to the PDF LMS document at this stage and you will find this is directly lifted from that document. So it talks about the basic system, there is a company which wants to manage attendants, leave of its employees and it starts saying that the company has three categories of employees and so on.

So certainly, in this space I cannot copy, I do not need to also, copy the whole specification as given in those two pages, you can always refer to that, that is what we are talking about as an informal description to get started with.

(Refer Slide Time: 04:09)

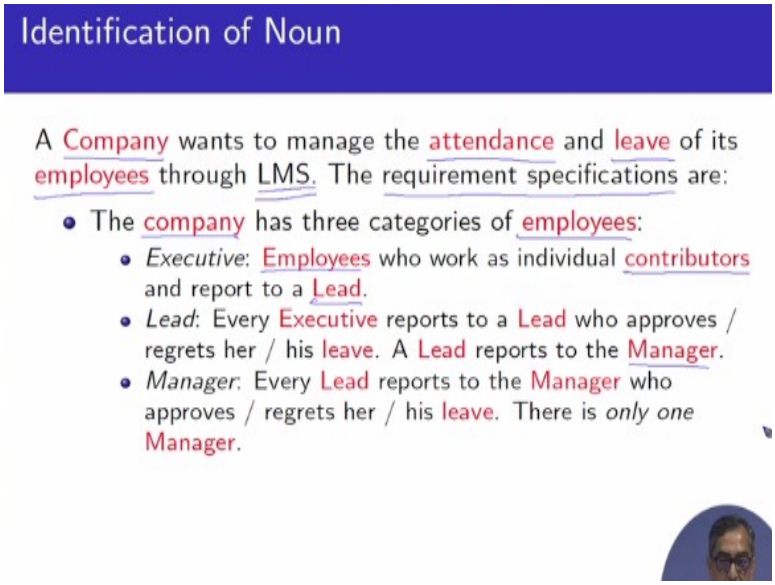


And please always supplement that with the discussions that Tanwi and Srijoni had been doing for this. So the first task that we will do first attempt that will take is taking this English description, we will try to identify, we will try to extract nouns. So this is an approach specifically used for textual description or interactive notes is called a linguistic analysis.

That is you basically try to parts the description in terms of parts of speech and then based on the parts of speech you try to identify concepts and then you start relating them. So if you are talking about the noun being the major part of speech, then our approach would be to identify nouns which is a easiest to identify and then in the process, we would like to eliminate irrelevant terms and you will find lot of nouns which basically does not have a sense in this whole thing.

We will try to identify, eliminate the names of values, they may not give as classes. We will eliminate vague terms, but will certainly try to focus on identifying which nouns are representing attributes operations, so that they can be grouped into in terms of classes and so on and we may also eliminate the certain nouns which represent relationships and take them up when we are actually talking about relationships.

(Refer Slide Time: 05:43)

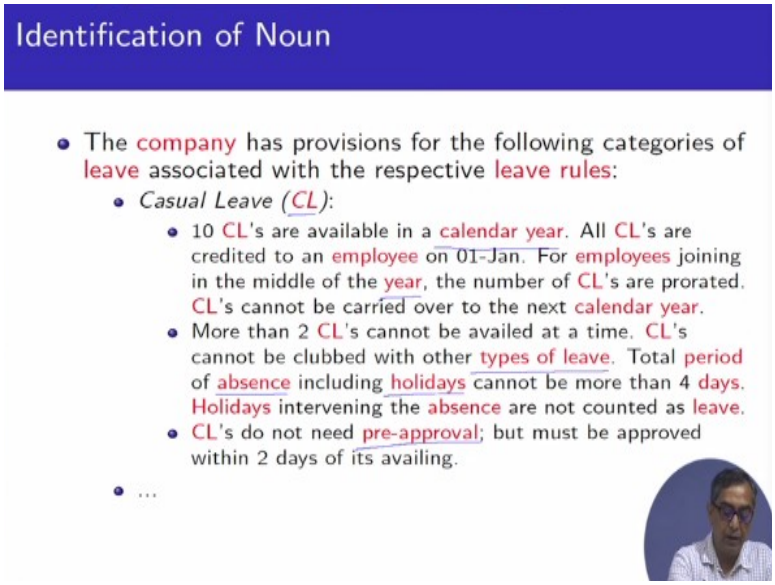


So let us get ahead with this exercise, I am again reproducing the beginning of the document and now you can see that in this I have highlighted certain words in red. Now these words as you can see the company, let me make the pen tip as blue now, so the company is certainly a noun. If you keep reading once to manage the attendance is a noun and leave is a noun, of its employees is a noun.

Through LMS, LMS also is a noun but we have not highlighted here, I have not highlighted here because it is obviously an irrelevant terms so far as the nouns or classes of the domain are concerned. Requirement specification is a noun but again we have not picking it up because that is a irrelevant term. But company we have already identified, employee we have identified employee.

Then we find a next noun in terms of individual contributor, we found a noun in lead, we found a noun in manager and so on. So that is a relatively straight forward process that you can do but of course you will have to keep on applying your intelligence, common sense and the domain sensitivity to see what could be nouns and what may not be nouns.

(Refer Slide Time: 07:11)



So if we move forward, this is as it continues the company has provisions for the following categories of leave. These have already been identified as noun. So we already know that. But then it comes with a noun which is leave rules. So we still do not know whether leave rules are relevant term or less relevant term or irrelevant term but we need to mark that. The causal leave which is short named as CL, so I am just using the CL as a noun.

Then you find several nouns coming here, calendar year, employee had come earlier, year, then we get absence, we get holidays, we get types of leaves, we get pre approval. These are according to part of these are all nouns. So the basic exercise is to actually go through this and keep on identifying the nouns. So I would not proceed further. I would suggest that at this point, you pause the lecture and take your time of to go back to the document and identify all the nouns in the document.

(Refer Slide Time: 08:22)

List of all Identified Nouns

Company	Attendance	Leave	Employees
Contributors	Lead	Executive	Manager
Leave Rules	Days	Year	Name
Type of Leave	Period	Absence	Holiday
PL	CL	EL	DL
SL	ML	LWP	UL
Pre-approval	Month	Service	Quarter
Medical Certificate	Parenthood Certificate	Disciplinary Action	Administration Function
Daily Attendance	Personal Details	Calender Year	Batch Task
Account	Balance	Designation	SysAdmin
Parent	Salary	Week	List
Privilege	Right	Login ID	Leave Status
Employee Code			

Presuming that you have already taken the break, you have identified the nouns then more or less you should get this set of nouns. I am saying more or less because there is always some possibilities of interpretations, some possibilities of missing out items but these are the typical nouns you will get.

And you can see that well this exercise just by itself may not be helping us a lot because you can just count how many we have, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13. We have 13 rows, 12 full rows, so four in each, 48 and last one. We have in this exercise itself we have identified 49 nouns. So we will have to see that how far have we actually understood about the system by doing this exercise.

(Refer Slide Time: 09:20)

Quality Check

- The Metrics at this Stage
  1. **Coupling**: Very low
  2. **Cohesion**: Very low
  3. **Sufficiency**: Unclear
  4. **Completeness**: Unclear
  5. **Primitiveness**: Unclear
- Actions required
  - Identify Attributes
  - Perform Structural Clustering

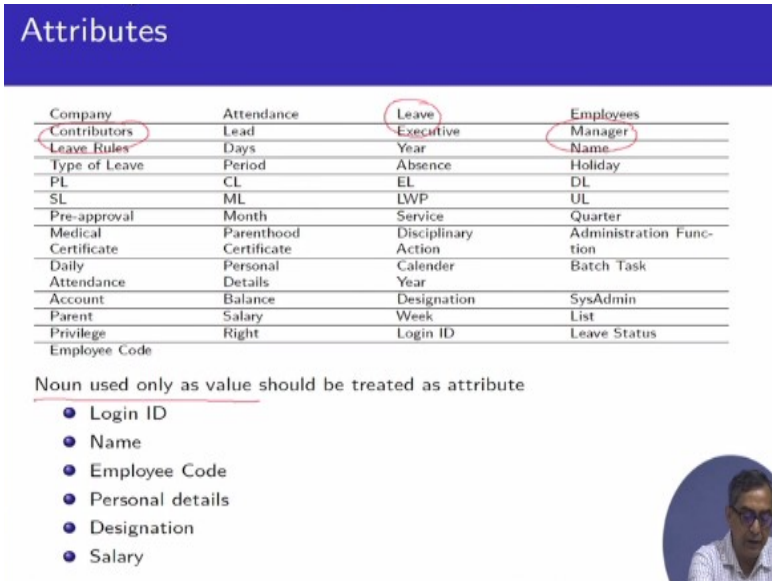
So what all continue to do is I introduced you to the different quality check parameter. So of

and on we will go back to that and just do a quick check. Basic metrics, coupling, cohesion, sufficiency, completeness and primitiveness, what do you know. In terms of coupling and cohesion, we have very low coupling, cohesion.

We just do not really know which is related to what and whether these nouns are coherent ones, these are, how they are really coupled and so on. So we have very low information about that and regarding sufficiency, completeness and primitiveness, we are absolutely unclear.

So the next step in the design process, we pick up in terms of the what we learnt in the earlier module, that we will try to identify the attributes out of this nouns because an attribute is expected to be a noun and we will perform some structural clustering to see if we can group these concepts better. So let me take you do that.

(Refer Slide Time: 10:20)



So here on top again, you have the set of nouns, as we have extracted and let us look at what could be an attribute. So what is the characteristic of an attribute, the characteristic of an attribute is, it should typically be a noun which is used only as a value. That is an attribute does not have a behaviour. Attribute does not have its other attributes or operations and so on. So but it is something concrete.

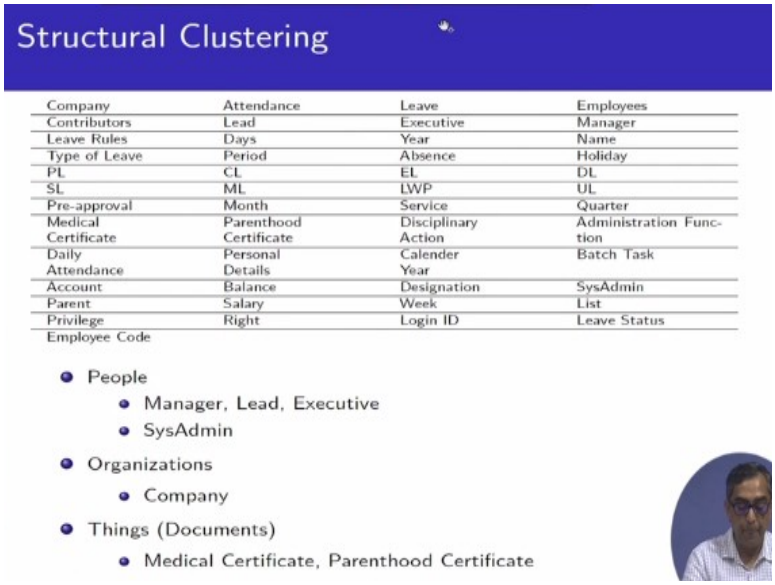
So you will expect it to appear as a noun but not have anything other than a value. So if we look at some say let us say, if we look at, say leave or if we look say contributor, if we look at a manager, we will typically know these cannot be attributes because we have seen, please go

back to the LMS document very frequently.

If you see through that document, you will find that manager does this, contributor works, leave is approved, leave is regretted. So all these kinds of, so there are several actions associated with this. But if we look at say the salary, or if we look at designation, if we look at personal details, login ID, so on so forth.

If we look into these and there are some more like employee code. If we look into this, these are nouns which adjust use as values. So we can go through that and say that these will be the attributes. So I just trying to do a basic process of classification okay, using this property. So this is also a kind of structural clustering only that we are doing, using the property that we expect nouns to be, nouns treated as values to be attributes, we are tried to get the attributes out. So we can say that these are the attributes that we can get to see right now.

(Refer Slide Time: 12:50)



Then you can do a more organised structured clustering by trying to kind of go in, kind of a discovery mode and say that okay, do we see people in this, like since this is a moderately generic domain, this is not a very technical domain like you know aircraft model or a very specific domain model. So we will try to throw in some concepts and see that do we have clusters here which shared the properties of that concept.

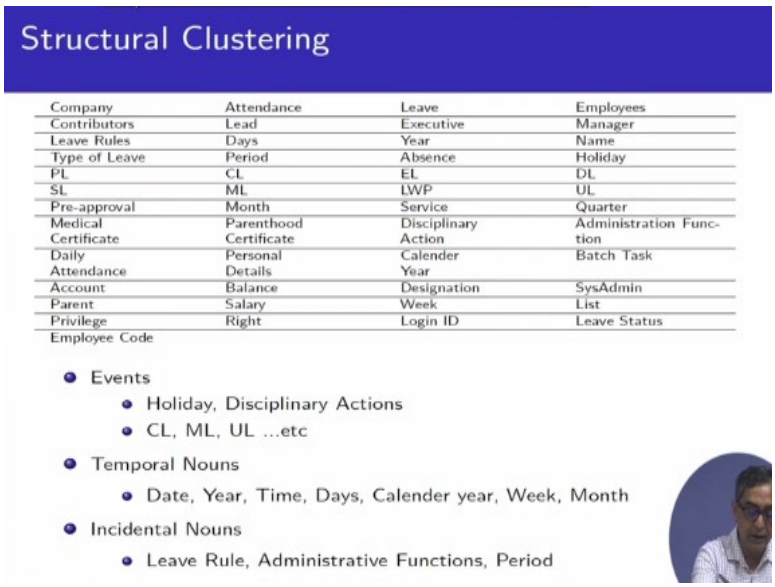
So people, let us look at what could be people and certainly we can find out managers, employees, executive, lead, these are all people because they will have the ability to act, take decisions and so on and we also get something like a SysAdmin, here I have kept these two in

little bit two different buckets because their specification says that SysAdmin is a batch.

So we are really not sure that SysAdmin in one sense can be a people that is there could be a physical person and therefore the person can act but it could also be an automated process which could act. So kind of we will put them in people but just try to remember that okay somewhere it said that SysAdmin is a match. If you look at organisation, we get company that is pretty straight forward, then one certainly main thing to cluster structurally are things.

So we get a whole lot of them, the different kinds of documents like parenthood certificate, medical certificate, these are certainly different things or documents that we can get so in the whole system.

(Refer Slide Time: 14:54)



If we proceed in doing structural clustering, one kind of clusters that often happen in the system are known as event where something is going to take place. Usually events have a temporal sense, events have a beginning and end, events have a sense of activation and certain side effects and so on. So that is typically an event. So let us look for other events, the certain things happening.

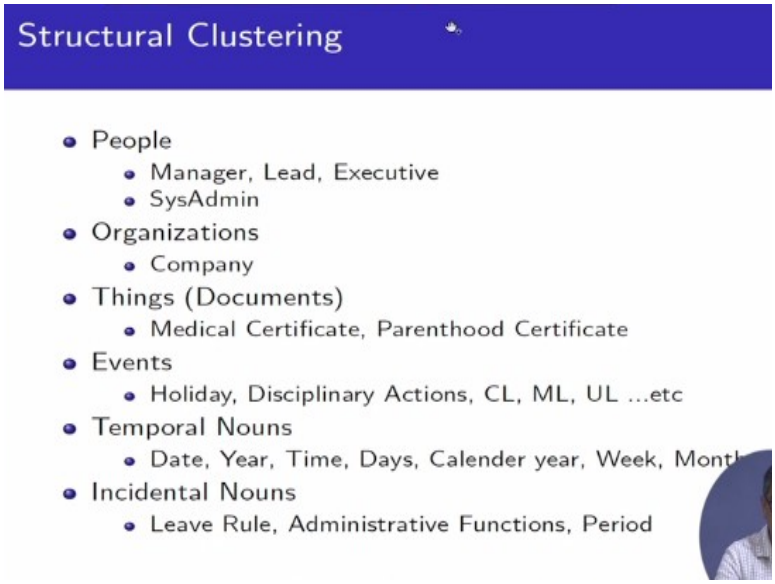
And we find that plenty of them are here. For example, leave is an event, is an event where an employee is going, will be absent from attendance in the company as it starts at a certain point, ends at a certain point and so on. Similarly, all of these different types of leave as discussed, the all these are kind of events. So is a holiday, holidays are not event because that is a designated day on which the company does not work as a whole.

So several these kinds of events including disciplinary action is an event because certain things specifically is done. So we can say that events tell us that these are the possible clusters that may be present here. Then certain things emerge from what you get to read if you go through, go back to the annotated document that you have created by marking the nouns, you will find that the word, the nouns like week, year, particularly days, month.

They occur not only the exist, but they occur very frequently. So there is a dominant presence for them and we can easily identify that they are structurally similar because all of them talk about time. So we can say that these are the temporal. So here I create a group which is kind of a temporal nouns.

Then there are lot of incidental nouns like leave rules, administrative functions, period of time so. You really have to go back to the document again and proceed further, work on the refinements to understand whether they care concepts nouns for concepts or they are just incidentally came in providing the description.

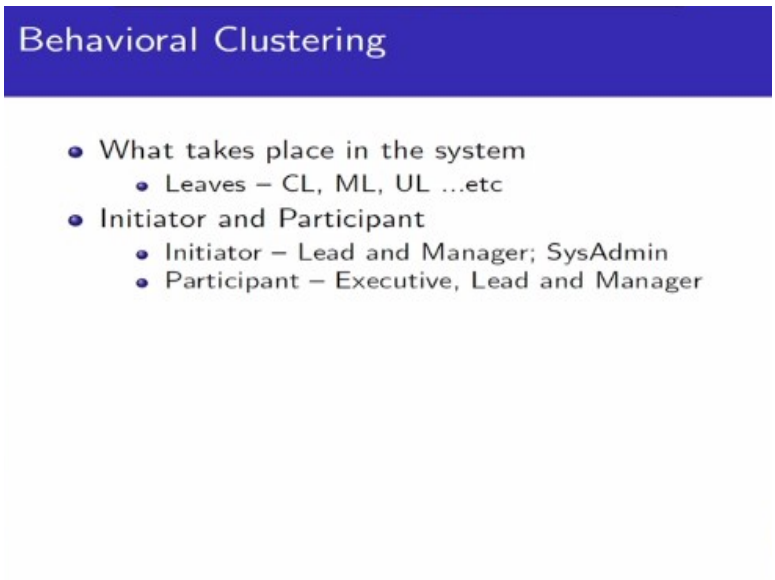
(Refer Slide Time: 17:20)



So given this, these are the concepts that abstractions that we have been able to identify through structural clustering and let us quickly do a quality check. Obviously, we have improved somewhat in terms of cohesion and coupling because, cohesion because we have been able to, we knew nothing almost about all these concepts that came up as nouns and now we have been able to group them into some seven eight broad classes.

So those classes I can say are coherent and in terms of, so if they are coherent they are similar by property. So there are certain amount of coupling that is coming up but certainly we have not made any progress in terms of the other metric parameters and we will go ahead and do the next which is the behavioural clustering.

(Refer Slide Time: 18:09)



In terms of the behavioural clustering, the main approach is to look for dominant behaviours and one way to look for dominant behaviour is to look for what happens in the system, what takes place in the system. So we will see in subsequent modules that they will be more of discovery of behaviour through identification of verbs and adverbs and so on but having read this specification by now for three four times I presume, because if you have sincerely done the noun extraction and have trying to do the structural clustering.

Then you would have got a sense of certain thing taking place in the system which is leave, various kinds of leaves. This is a leave management system, they are certainly that is a leave behaviour is kind of a dominant concept that will exist and that is coming out and proliferating through a variety of different leave types.

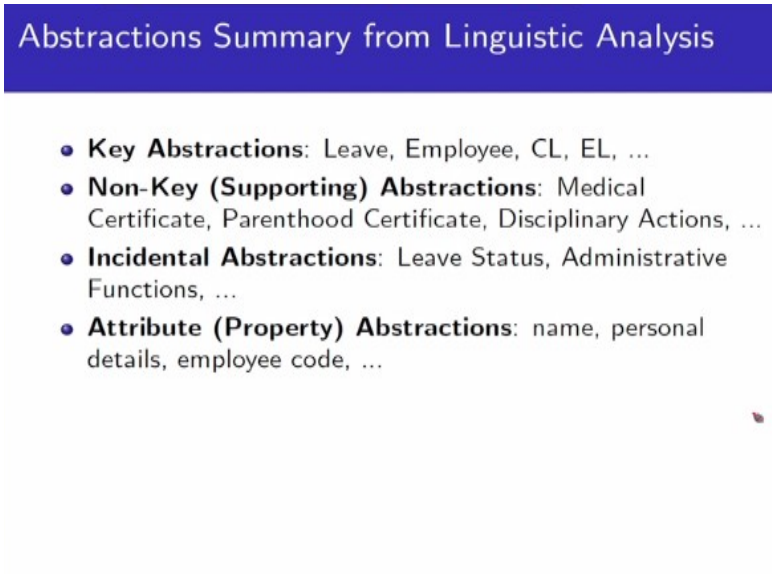
The other that you look for typically in a behavioural clustering is to find initiators and participants that is in terms of a behaviour, we say this is an initiative which in terms of the final model will turn out to be clients and participants are kind of serves or just passive participants.

Do we find that those kind of behaviour and if you look and go through the specification

again through these expectations through this classes, we will find very well defined initiators like Lead, Manager, SysAdmin who are frequently talking about doing something in this system. Initiator is would does something in the system, who is responsible for starting some activity action in the system.

And you also find lot of participants like executive, lead and manager again because if the lead is approving or regretting the leave application of an executive, then the executive is a participant in the whole thing. Of course, I mean, these are not exhaustive, I would suggest strongly that you go through the document and try to identify more of the actions that are taking place in the system and more of the initiators participants in the systems and make the list more big.

(Refer Slide Time: 20:34)



Now having done this, then we can summarize the abstractions that we have found from the simple linguistic analysis which is been based only on nouns, will do more of that later and we can say that we have the key abstractions now, we can identify the key abstractions as something like leave, employee, casual leave, earned leave and so on.

And they do satisfy the requirements of the key abstraction we had specified earlier that is they come directly from the vocabulary of the domain, they are dominant concepts in the fact that they are repeatedly talked of in the specification. They are involved in a major part of the activities that are proposed in the system and so on. So looking into this symptoms, I propose that we accept leave, employee these as key abstractions.

And there are few others which are supporting abstractions non-key kind of medical certificate, you know if there is a lady employee needs to go for maternity leave, she needs to produce a medical certificate. So that is an abstraction but that is kind of a non-key abstraction because we do not find many references of medical certificate. We do find another reference in terms of sick leave but we do not find too many of them.

So we probably would not like to put it as a key abstractions and that is a judgement choice. I mean, may be some of you would like to make it key abstractions but probably I will not make it at this stage. Certain abstractions are also found in terms of structural and behavioural clustering is incidental abstractions kind of administrative function, leave status so on.

So we are kind of putting them on the side line because we really have to go through the document more, we really have to go through the video more to understand that if they are critical for the LMS system or they have just occurred in the specification because when you talk about the activities in certain context, the related subsidiary tertiary activities also get talked of.

For example, one that we will talk of in terms of the next module is the fact that does employees work because we know if their employees there, we will should work in work related work related abstractions are they in any way important here, we will talk about that. So these are kind of incidental abstractions which are of that similar type and of course we have identified a set of, this is not exhaustive, it is just indicating we have started identifying a set of property abstractions or attribute abstractions in the whole thing.

And mind you, this is this whole thing is kind of little bit structured than completely go, I mean, going wild by saying that you pick up the vocabulary of the problem domain and start making them, clustering them. We have taken at a little bit more organized approached in terms of doing a linguistic analysis which also is a very powerful tool.

(Refer Slide Time: 23:51)

Quality Check

- The Metrics at this Stage
  1. **Coupling**: Low (was Low)
  2. **Cohesion**: **Moderate** (was Low)
  3. **Sufficiency**: Unclear (was Unclear)
  4. **Completeness**: **Very low** (was Unclear)
  5. **Primitiveness**: **Very low** (was Unclear)
- Actions required
  - Decide on Classes (first level)

And just with a noun analysis, this is what we have been able to achieve and if you re-evaluate ourselves as to where we stand right now we find that coupling is still low but cohesion of course has been improving because we not only have been able to group in terms of structural and conceptual similarity but we have also applied more considerations to identify what are the key abstractions, what are the supporting abstractions, what are incidental and so on and so forth.

The sufficiency is absolutely unclear as yet. Completeness of course, I have no sense of completeness, but sounds very low sense of completeness is coming to forth, because now you have at least some confidence to say that well of the different concepts that you have identified through these nouns may not be 100 percent complete but they must follow from about 90 percent of the basic concept that exist.

You do not still know whether those concepts should be used in that form itself, whether the abstractions need refinement, how they will be organised on hierarchies and so on but you have some sense of the fact that it is not these 49 nouns have, may be, may be you have some more may be around 50 odd nouns is kind of the domain within which we will have to define our system.

And as we have done that then we have started seeing that certain things are keys, certain a less key, so you have also some early sense of primitiveness coming in as to what are the concepts that are completely on their own and what are the concepts which are defined in terms of the others. So given this, we can take the last step given this.

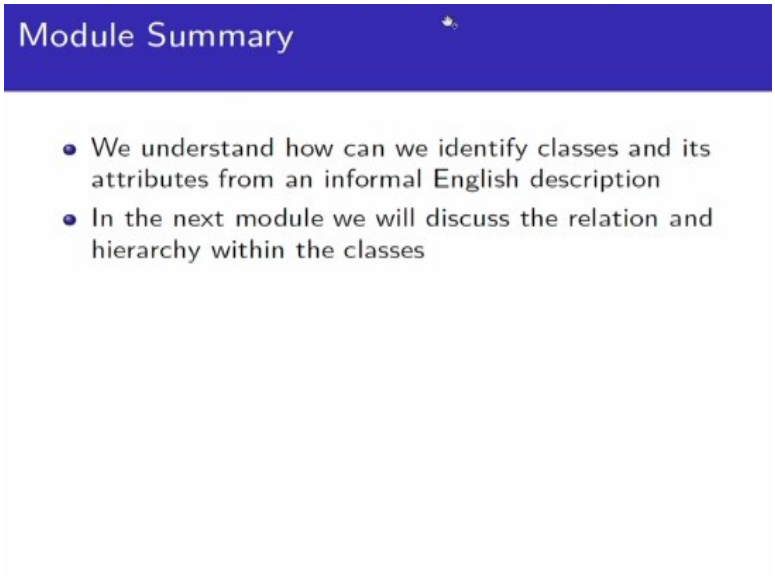
So you can see that, I have shown at three stages of acting, starting with the extractions of nouns as we go and apply some more considerations, revisit and refine, the quality, the whole purpose of this refinement is to keep on improving the quality metrics and which we are being able to do and at the objective of this particular module, we want to end with deciding on the classes of this key abstractions and the supporting ones.

(Refer Slide Time: 26:05)

Let us at least initially agree that all the key abstractions initially would be formed as classes, so if you put the key abstractions, so this I say is the resultant identified classes, so we have the all, we have the manager, lead, executive, employee, these kind of ones, we have leave and all different types of leave, we also have SysAdmin and we have the central company, the organisation on which.

So this is what from the particularly from behavioural clustering and from identification of key abstractions where we are tried to make an assessment on the density and importance of activities that different abstractions do, based on these we proposed that this could be at least our initial choice of classes that we make for the LMS system.

(Refer Slide Time: 27:01)



So this brings us to the end of this current module but as I had specified at the beginning, this will be an ongoing process of doing this exercise of finding more classes, their relationships, their hierarchy and the objects and different operations and we will continue in the next module in terms of that.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture - 30
Identification of Classes, Objects and Relationship in LMS (Contd)

Welcome to module 18 of object oriented analysis and design. Starting from the last module we have taken up the task of doing an exploratory exercise to design our leave management system. So we will continue on that task over this module as well. So as I had advised in the last module 02 please keep the document of the LMS specification the PDF handy keep the video handy.

If you need clarifications, please watch the video again and please also keep pen and paper with you so that you can pause in between and complete parts of design that I am discussing in this module. So here we will primarily again quickly recap what we did how much part of the design has been done primarily the identification of classes and their attributes have been done.

And based on that this particular module we will focus on the identification of responsibilities and as we had been doing in the lat module we will continue to do periodic quality checks to make sure that we are actually refining it for the better. So just to remind that our specific input is the English specification, informal English description and we are trying to make the class attribute and responsibilities.

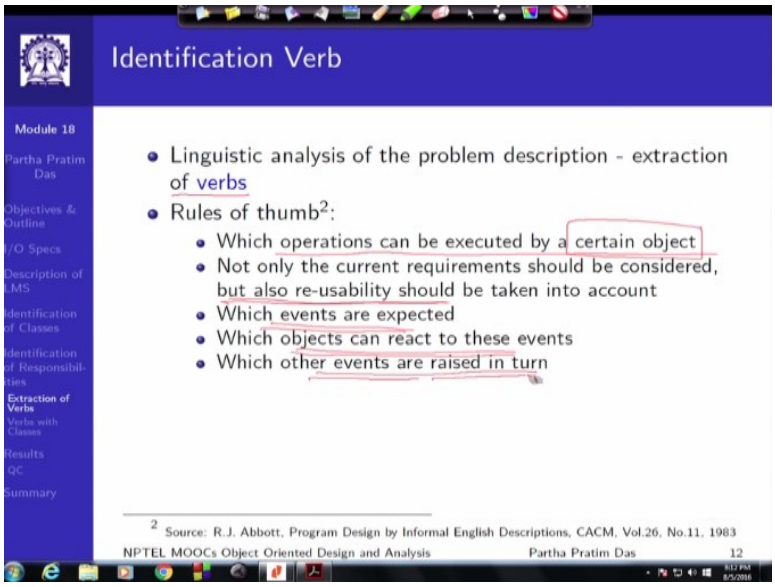
So this just for a recap again that this is a kind of a specification that we are following you already know this. Now in the earlier module we have followed a primarily a linguistic approach for analyzing the English description based on that identification of nouns and we have done that analysis. We have identified a whole lot of nouns of the order of 50 different nouns.

This is possibly the complete list there maybe few more which you may be able to identify and then analyzing as to which nouns just occur as values we identified some of the attributes of course there are lot more attributes which need to be completed by you and with all this after the stages we did a quality check to see that the coupling in the system is still low

because it all disintegrated kind of some kind of cohesion has started coming up in terms of similar items which we are putting down in terms of classes.

And sufficiency is not clear and completeness and primitiveness is also very low and with that we identified about 15 potential classes there would be more, but these are the potential classes, these are the key abstractions which stood out from amongst the different nouns that we had analyzed.

(Refer Slide Time: 03:40)



Now having said that now we move on to the identification of responsibility and we will continue with the linguistic analysis approach and this time we will focus on the verbs to do our task so it will be like this that we will use the linguistic analysis and try to extract verbs and naturally verb as we can understand verb inherently gives a sense of action, it is a sense of something being done.

So verbs like nouns have more strong correlation with existential realities and therefore are more indicative of objects and classes. Verbs would be more indicative of actions, but there would be exceptions in some cases verb may actually stand for a connective may actually mean a property and so on, but primarily from the verbs which we will try to see is what operations can be executed by a certain object.

We already have made a good understanding of the possible objects by the identification of classes and we will try to take into account from the analysis of verbs as to whether there is reusability that is if the same action or very similar verbs are repeated amongst various

different agents then those are potential places for reusability which we would like to identify.

We will try to see that naturally actions mean events we talked about events for structural clustering also you will remember while we were analyzing for the classes. So the event is some happening so we will expect that with the identification of verb we will be more clearly able to talk about what events are expected and when those events will happen and so on. We will the verb naturally in many cases will have a subject and an object in the linguistic sense in the grammatical sense.

So from the object of a verb, we would be able to understand, we will be able to extract what is the expected object of the OOAD sense on which the verb is the action is actually being taken. So we will get idea about responses the events raised in turn and so on. So the analysis of verbs should give us quite a bit of good hold over what can happen in the system. So like before we just colorize the text this time I am using the blue colour to identify the verbs.

So the company wants to manage the attendance and leave of its employees through SMS. This is a verb, but we would simply ignore that because we understand this is just a verb for expression not for the system, but here is a verb which has specific subject of employee the verb work then report again report then there is approval approve, there is regret again report approve and so on.

So in this way we can see like in the case of noun we can identify several verbs that will be off use in terms of the identification of responsibility. Continuing on the document further if we go towards the part where leave different kinds of leave are discussed we can say that we see a verb credited. So we will know that leaves will have something to do with the credit system.

It has something to do with the joining another verb allocation is prorated so there is another verb that defines actions carried over or carry forward is another verb and we can see that there is some qualification in terms of that. So when we analyze verbs we can actually see that what is a primary verb and what are the auxiliary one. The primary is a carry over and the auxiliary is cannot be which basically is a negative action that is being said.

So we can make a note of this all of this are of that kind and if you go through we will be able to identify many of them in the whole document. So I hope you have understood again as to what we are trying to do in terms of analyzing the parts of speech. So I would strongly suggest that you pause at this point and go back to the PDF specification and go through it very carefully this time analyzing the verbs and connecting them in your notebook.

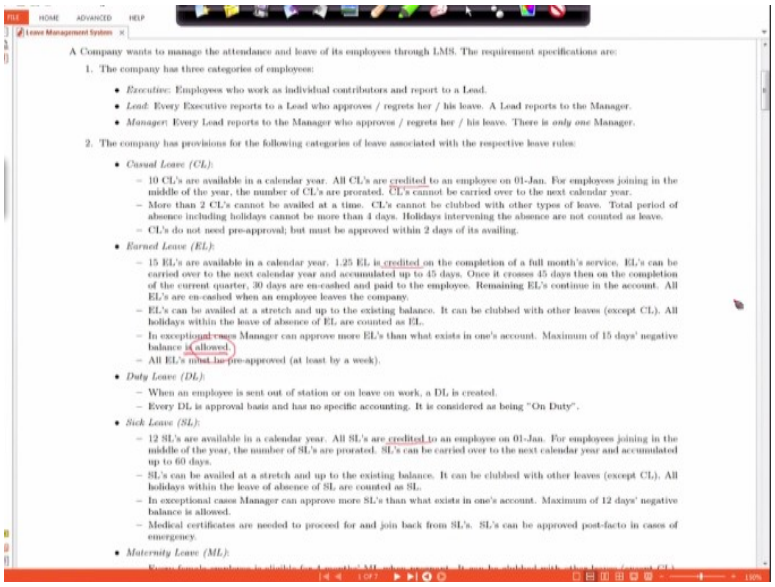
(Refer Slide Time: 08:54)

So I hope you have done this exercise of collecting the verbs this is the list of the verbs identified from the LMS specification again I would never claim this to be an exhaustive list you may be able to find some more verbs that I have missed out, but this is more or less the kind of verb that you get. So there is interesting verb like approve, like regret like adjust and there are lot of as I said is auxiliary combined verbs the negative actions.

And certainly verbs which are kind of more connective like become and provide these kind of so all kinds of verbs actually come in as you do the analysis and you will also find that there are certain verb forms which are not in the pure form like these ones. So the verbs pure original verb is the club then it is coming with an auxiliary extension. So what we will do just to clean up the system and reduce the number of verbs that we have to deal with we will try to just take the pure form and take the list of verbs which are in the unique in terms of their stem.

So if we do that clean up then we are left with these kinds of verbs on which again we can try to look into couple of different things like what are the verbs that we observe in very core part of the action so maybe I could just take a look at the specification here.

(Refer Slide Time: 10:37)



A Company wants to manage the attendance and leave of its employees through LMS. The requirement specifications are:

1. The company has three categories of employees:
   - *Executive*: Employees who work as individual contributors and report to a Lead.
   - *Lead*: Every Executive reports to a Lead who approves / regrets her / his leave. A Lead reports to the Manager.
   - *Manager*: Every Lead reports to the Manager who approves / regrets her / his leave. There is *only one* Manager.
2. The company has provisions for the following categories of leave associated with the respective leave rules:
   - *Casual Leave (CL)*:
     - 10 CL's are available in a calendar year. All CL's are credited to an employee on 01-Jan. For employees joining in the middle of the year, the number of CL's are prorated. CL's cannot be carried over to the next calendar year.
     - More than 2 CL's cannot be availed at a time. CL's cannot be clubbed with other types of leave. Total period of absence including holidays cannot be more than 4 days. Holidays intervening the absence are not counted as leave.
     - CL's do not need pre-approval; but must be approved within 2 days of its availing.
   - *Earned Leave (EL)*:
     - 15 EL's are available in a calendar year. 1.25 EL is credited on the completion of a full month's service. EL's can be carried over to the next calendar year and accumulated up to 45 days. Once it crosses 45 days then on the completion of the current quarter, 30 days are en-cashed and paid to the employee. Remaining EL's continue in the account. All EL's are en-cashed when an employee leaves the company.
     - EL's can be availed at a stretch and up to the existing balance. It can be clubbed with other leaves (except CL). All holidays within the leave of absence of EL are counted as EL.
     - In exceptional cases Manager can approve more EL's than what exists in one's account. Maximum of 15 days' negative balance is allowed.
     - All EL's must be pre-approved (at least by a week).
   - *Duty Leave (DL)*:
     - When an employee is sent out of station or on leave on work, a DL is created.
     - Every DL is approval basis and has no specific accounting. It is considered as being "On Duty".
   - *Sick Leave (SL)*:
     - 12 SL's are available in a calendar year. All SL's are credited to an employee on 01-Jan. For employees joining in the middle of the year, the number of SL's are prorated. SL's can be carried over to the next calendar year and accumulated up to 60 days.
     - SL's can be availed at a stretch and up to the existing balance. It can be clubbed with other leaves (except CL). All holidays within the leave of absence of SL are counted as SL.
     - In exceptional cases Manager can approve more SL's than what exists in one's account. Maximum of 12 days' negative balance is allowed.
     - Medical certificates are needed to proceed for and join back from SL's. SL's can be approved post-facto in cases of emergency.
   - *Maternity Leave (ML)*:

So if we look at this then suppose if I am looking into these leave parts and we can see that the verbs like the leave is credited, EL we are saying it is credited. So we get SL'S is credited. So you can see that it is credited as a verb or the credit verb has something strong to do in terms of the different leave which is understandable because you need to have some entitlement for the leave.

But some of them maybe relatively less important verb like here we are saying that maximum 15 days of negative balance is allowed. Now this is not this is though this is a verb, but you will have to be careful that in the usage here this is more a constraint than an action. Since we have to express everything here in English you are writing it in terms of naturally subject predicate object and verb and all that, but this is not actual an action of an agent.

This is you would not perceive an action here rather this 15 days negative balance is allowed is more like a constraint that we will need to get into the system.

(Refer Slide Time: 12:11)

Whereas we could look into some of the other verbs like clubbed we can see that here in CL, we can see that here in EL, we can see that here in SL and so on. So all that have meaning is in fact if I just scroll down further in maternity also again I am sorry your underlining will change because here the underline is actually on the display panel, but you can see that clubbed also exist in maternity. So we can certainly conclude from here that club is a verb which has a more of importance more of relation to the core responsibility and dynamics of the system.

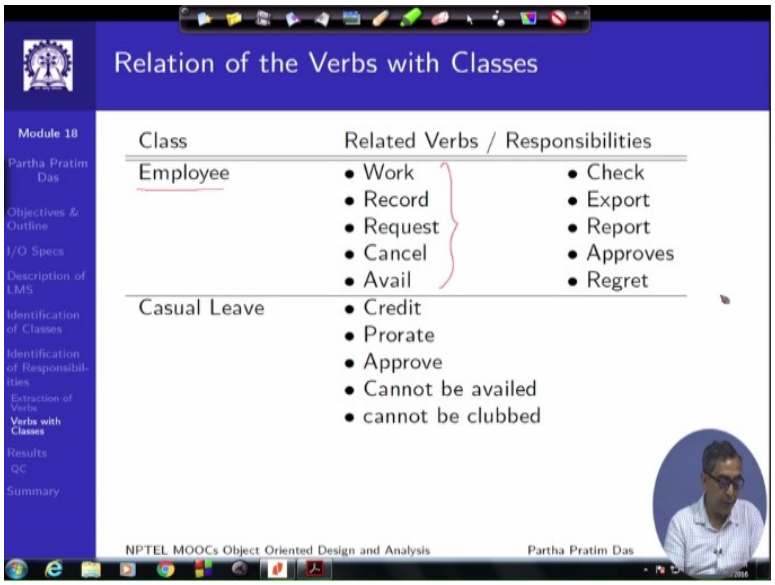
(Refer Slide Time: 13:00)

So that is what I was trying to point out here that is what I was trying to point out in terms of the list of verbs that we have here so from this list we will need to try to primarily identify in a manner that we were doing as to what are the more important, more relevant verbs for the situation that we have. And we will have to obviously keep this in mind that our core

responsibility here is to prepare a leave management system.

So that by the basic definition of the problem most of the actions that are either initiated by leave kind of objects or where leave kind of objects are a participant certainly has more importance and we will have to be looked into in greater detail.

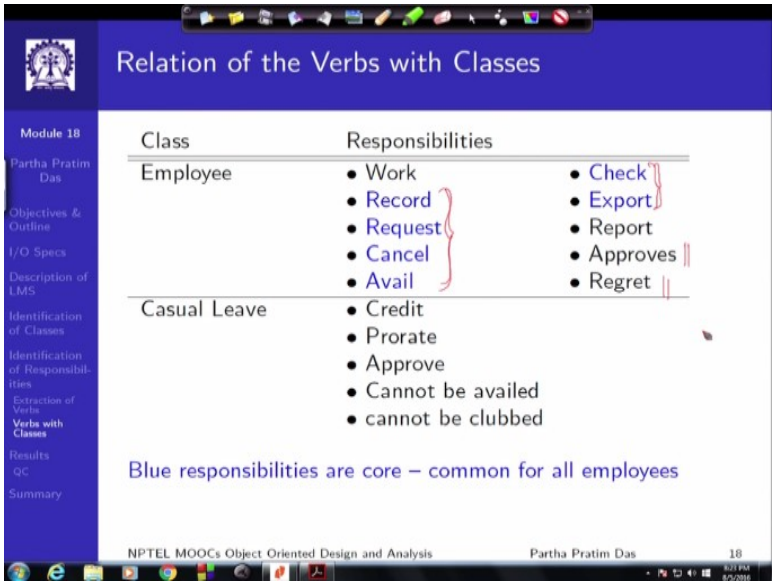
(Refer Slide Time: 13:48)



So with this let us move on let us now try to put these verbs with the classes that we already have the classes identified at least we have a reasonable number of classes identified. So what I am trying to do is actually analyze the document and trying to associate the class with the verbs that can get related to it. So this will mean that if my statement says that say if my statement says, for example, here that it says that the employee who work so that relates these two together.

We say executive reports so that relates these two together. You say lead who approves so that relates these two together. So this is the kind of relationships that we are trying to set forth we already know the nouns corresponding to the classes that we have identified that is how we got them and we are identifying the verbs and from the verbs we will now try to annotate on this relationship to get a better idea of what the responsibilities could be.

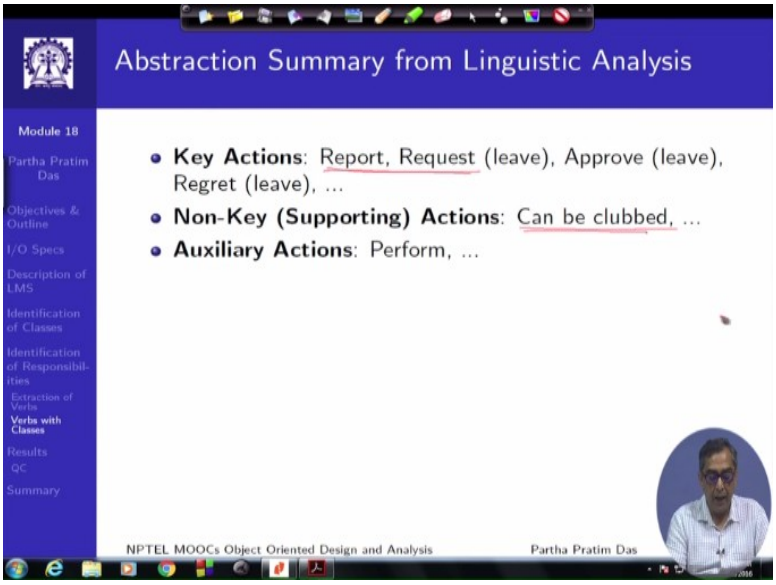
(Refer Slide Time: 15:26)



So we do this identification of work, record, request, cancel, check, report and all these are the verbs associated with employee whereas the verb associated with leave will be credit, that is crediting leave, prorating leave getting a leave approved and so on and so forth. So if we just define that then these verbs we can now think of them as responsibilities and you will see that some of these are marking in blue especially these ones because we know that there are multiple categories of employees.

And if you analyze on the document you will find that these are the responsibilities or these are the relations that have been mentioned for all kinds of employees whereas maybe this is approve or regret is relatively restricted maybe it just applies to lead or manager, but does not apply to executive. So in terms of generalization I will say that these are the more important responsibilities, more common responsibilities that we identify and we will try to put them together with the classes that we have.

(Refer Slide Time: 16:56)



So this now brings us to the abstraction summary of our linguistic analysis. So we will say that we have key actions report, request, approve these are the key actions. So these are the verbs or responsibilities that we have identified and then we are trying to say that what is core key and what possibly is just a supportive one let me just again go back to this and then try to show you maybe I can pick up some parts from the second page because I am always discussing the first page.

So we say that here the specification is saying these are the typical use cases which mean that use case is a term which tell us that how typically a user will actually use this system, what are the different scenario actually through which he or she will use the system. So if we look in here we will be able to see more of the association between the objects and the responsibility.

So if we look at, for example, here say every employee of every kind can do the following from his or her account so that clearly relates to these are the different responsibilities that you have. Now we are saying that of all these we are trying to mark that these responsibilities of say record, request, cancel, avail these we are saying are more key abstractions via these are key responsibilities in contrast to some of the others.

So let us understand why are we saying so let me again scroll up suppose here let us read the leave we are saying that CL cannot be carried forward, CL's cannot be clubbed. Then say EL's can be carried over then it says it can be clubbed. This says SL can be clubbed. So if we look into these club as a responsibility now naturally it will go to a responsibility of the leave

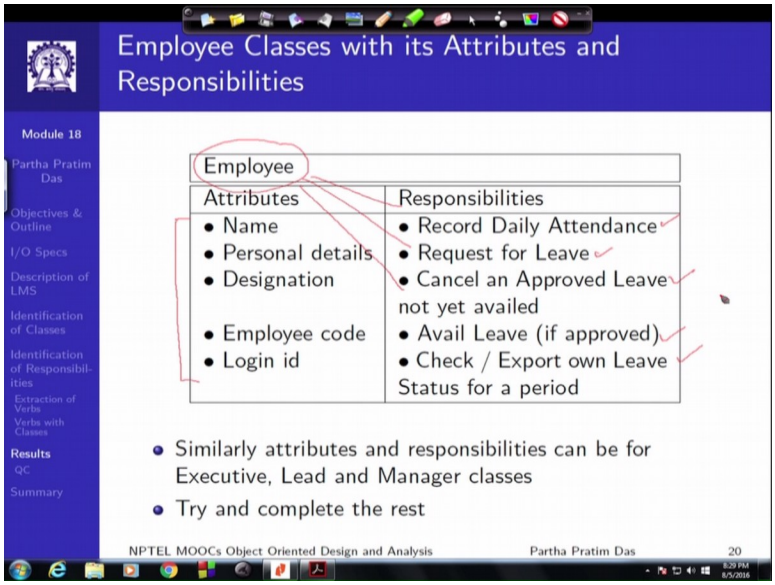
class or the leave objects.

But this is we would say it appears like a more an auxiliary responsibility because this is a responsibility which will again not have a primary action in the system which as it turning out very loud and clear he is that employees apply for leave the leave maybe approved they maybe regretted and so on and then if approved avail of it they can cancel and all that in comparison to that these are also responsibilities and they are responsibilities of certainly the leave class and these responsibilities will possibly lead to refinement of the leave class.

This will possibily lead to different refinements in the class structure that we have it will possibly impose more constraints and that is a reason we will like to make this distinction and try to differentiate between what are the key actions and what are the non key actions so that is I just discuss this to explain that why we are putting this in a non key one and why these are the key actions that you have.

And besides that you may have a lot of auxiliary actions like is something has to be performed and so on which are basically not part of the core system, but is coming across because you need to finally express the whole thing in a English like language.

(Refer Slide Time: 21:26)



So with that if we take back then some more at a very early level or design could look something like this we know we already knew that there are several classes and employee is a class and if we look into the exercise these are different attributes which are specified somewhere in the document for an employee name, personal details, designation, employee

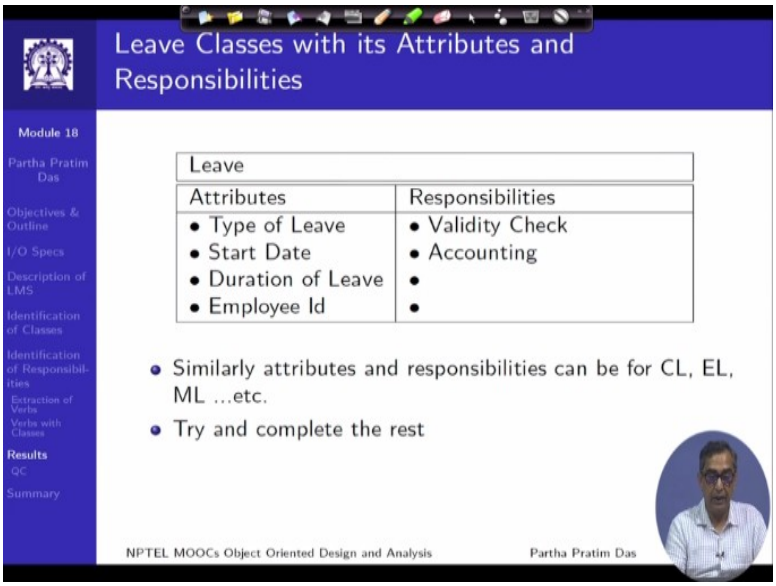
code, login ID these came up through the analysis of extraction of noun.

And then it went through the process of categorization, classification as attributes so we will have these attribute and these are the different responsibilities that we are being able to identify in terms of the analysis of verbs and then associating them as I showed with the actual employee class. So we have a collection of verbs, we have a employee classes and we can associate them based on what has occurred in the document where.

So this is a kind of the employee class with attributes and responsibilities similarly we have executive, we have lead, we have manager all of these you need similar things to be done. So I will leave this to you and I will again advice that it may be a good option to just do a pause and try to work this out for the other kinds of categories of employees as to what their attribute should be and what their responsibilities, how their responsibilities turned out to be.

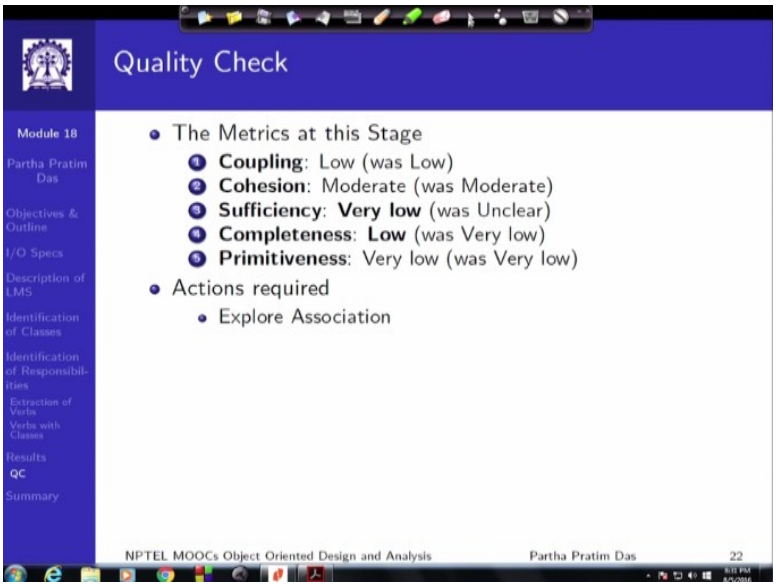
Going forward in later modules we will actually use that information and do different refinements of the design.

(Refer Slide Time: 23:13)



This is just another example of the leave class with the attributes of the leave like type of leave, start date, duration of leave and so on and other responsibilities like validity check, accounting, capabilities all this will come up and again I will request you to actually work out on the leave objects and try to for all the set of classes please try to complete the attributes and the responsibility so that you have a good hold in terms of the basic defining entities of the whole system.

(Refer Slide Time: 00:23:50)

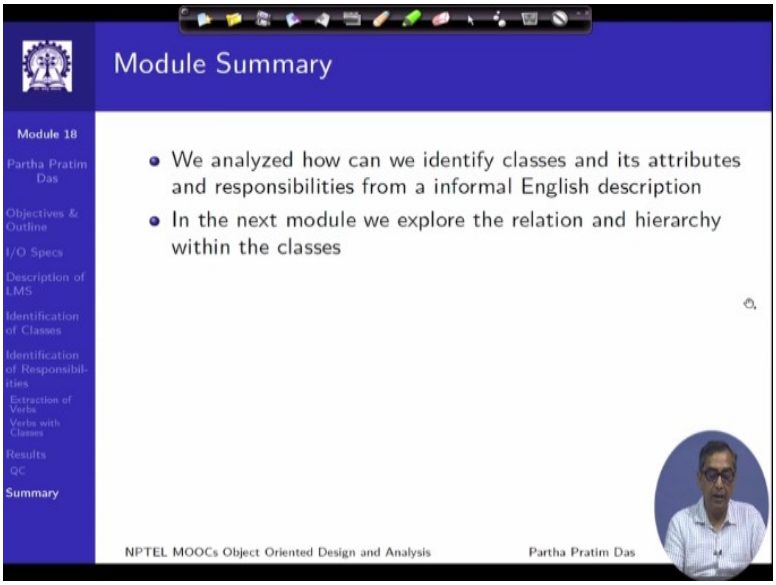


Quick time for a quality check again so if we look into what we have primarily added the responsibilities here. So we have not yet analyzed the interpretation of the classes in terms of the responsibilities which we will what we would like to do in the next couple of modules. So since we have not yet done that so but there is not much that we expect to change in terms of coupling and cohesion metrics they remained to be at the same place.

But my feeling was that we have slowly inching towards completeness we are still low, but at least we have completed one aspect of the system that for all classes we have identified the attributes and the responsibility. So at a certain level we have a basic notion about the system dynamics though we have not yet looked at relationships we have not yet analyzed how those relationships will impact the overall organization of the system.

We have some idea emerging for the sufficiency as well. So our next target would certainly be to explore different association, different relationships amongst the classes within the classes so that we can infer more information about the design.

(Refer Slide Time: 25:13)



So in summary in this module we have analyzed how we can identify classes and its attributes and responsibility from an informal English description. We have primarily in this module and in the earlier one we have primarily used a linguistic analysis approach which is one way of doing it certainly there are several other ways that this analysis can be done, but what has been important is we have shown that at every stage it is critical to go back to the specification.

And also measure as to how well you are doing in terms of the metrics in terms of checking out how the different quality parameters are doing and then come back and refine that. In continuation we will continue this exploratory exercise and in the next module we will explore the relation and hierarchy within the classes.

Object-Oriented, Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture - 31
Identification of Classes, Objects and Relationship in LMS (Contd)

Welcome to module 19 of object oriented analysis and design. For the last two modules, we have been trying to apply different techniques and assessment to explore the design of leave management system. So this module primarily continues on that and I would again repeat that please keep the PDF of your LMS system handy please refer to that regularly as you listen to this module.

And also frequently refer to the tutorial video where we have provided a client vendor interaction on the LMS system and keep paper and pen handy so that you can frequently pause this discussion and actually work out the details of what we are outlining here. In this module, we will focus on primarily the extraction of collaboration, modules and hierarchy. So as you have studied earlier in the earlier modules we have discussed the core fundamental items of object oriented analysis and design of the object and classes.

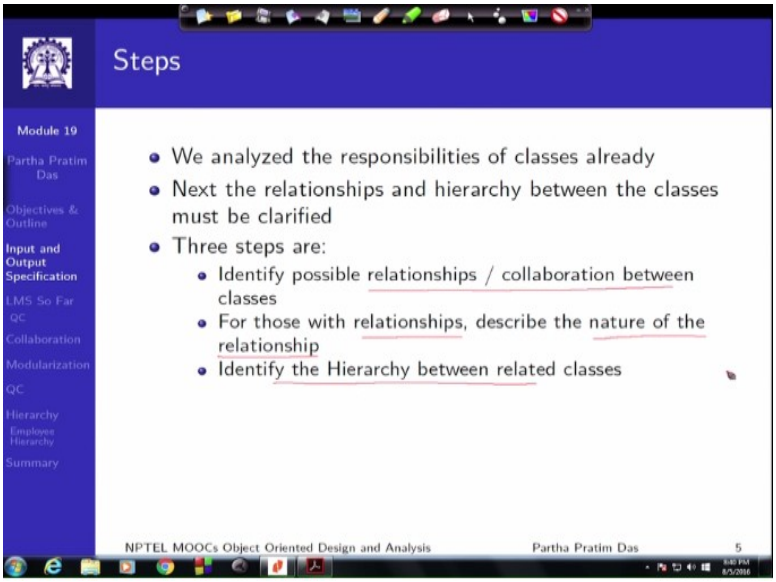
And next comes their relationship. The relationship defines how the system can be organized, what are the different structural and behavioral properties in the system that can be observed. So in the last two modules we have been focussing primarily to extract the information of the classes and object along with that attributes and responsibilities and now we will try to focus more on the possible relationships that different objects, different classes may have in this system and we will use two major source.

One is certainly going back to the specification that we have and also we will try to now refer to what we have already designed and try to look for refinements in terms of the relationship that we are trying to find out. So in contrast to the earlier two modules where we were using the specification and trying to find out the classes attributes and responsibilities. In this module, we take these classes attributes and responsibilities as primary input.

But that does not mean that we forget about the specification document because there is still a lot of information, domain information there which need to be found out, but primarily from

the classes their attribute and responsibilities in this module we are trying to extract the collaboration the modularization in the hierarchy. So these are the main output that we would like to generate from here.

(Refer Slide Time: 03:53)

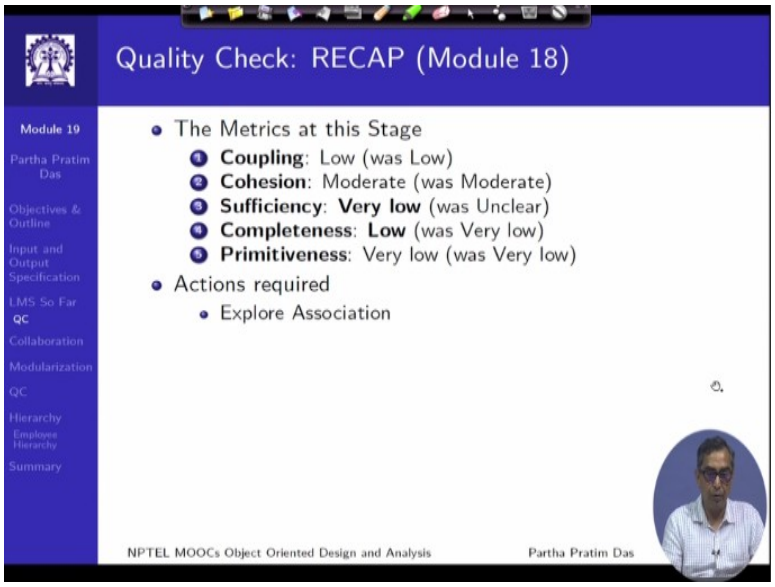


Now just a quick recap of the broad steps as we have already analyzed the responsibilities of the classes like we did in the last module. The next will be the relationship and hierarchy between the classes that we need to establish. So the main three steps are the possible identifying the possible relationships and collaborations between classes and for those relationships we need to describe the nature of the relationship.

And we will do some bits and pieces of that here, but more and more as we will go through the OOAD design process we will see that identifying relationship and describing their nature or characterizing them are key components of the design. And finally the third step would be to identify the hierarchy between related classes and particularly in the next module we will try to show how this hierarchy can be really developed well using the study of different relationships that may exist.

So before we start discussing on this module design part let us recap what we have in our design by now. We have about 15 classes which we did in module 17. We have identified the attributes for this classes partly you have completed them already I hope and in module 18 we have completed that task of associating the classes with attributes and responsibilities and worked out two samples for employee and leave and left it as an exercise to complete for you all.

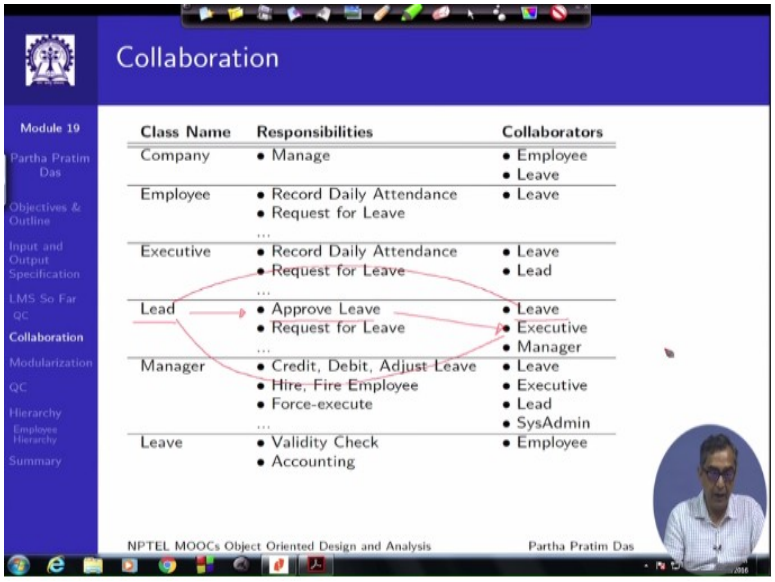
(Refer Slide Time: 05:45)



And at the end of that that process our the metrics at this stage I just to remind you that we are regularly measuring the metrics in understanding how well we are doing. So in the metrics at this stage stand at this level the coupling is low, the cohesion is moderate, the sufficiency is very low. We just have no idea about the whole design.

So we certainly cannot talk about what is the minimalist design the completeness is low because we have just completed a basic round of classes and responsibilities and the primitiveness is also very low because we have not done much in terms of identifying the primitives.

(Refer Slide Time: 06:36)



Now let us first talk about the collaboration. So what the collaboration does. Collaboration is

kind of a I mean you can say it is a kind of re refactoring problem that is for example let me for once again go to the specification document. So let us look at some of these say company wants to manage the attendance or leave of its employees this is what so if we look into this manage aspect of this this is defining a relationship.

This is defining a responsibility for the company then certainly the company cannot disburse this responsibility of management if the company has to disburse this responsibility of management and certainly the company need the employees. So we say that these are there is a collaboration between the company and the employee in terms of the management that happen through them.

We can say that for example if we look into some others then we can say that every executive reports to lead certainly there is a collaboration here which is through the reporting relationship the responsibility. More importantly executive or let say the lead approves leave for some executive. So approve is a responsibility of lead and what does the lead approve? The lead has to approve leave.

So there is a collaboration between the lead and the leave. So this is kind of a factoring problem or you can say refactoring problem where you look into the different responsibilities and try to identify that if there is some agent on the other side of that responsibility. If such an agent exist then we will say that there is a collaboration between these objects or these classes.

So if we come back we can say that a lead this is what I was referring to a lead can approve leave and what does it approve his leave. So the collaboration is here. Now there is a different kind of collaboration in the approve leave is the lead approves leave and for whom does he approve the leave is a executive so there is a collaboration between the lead and the executive.

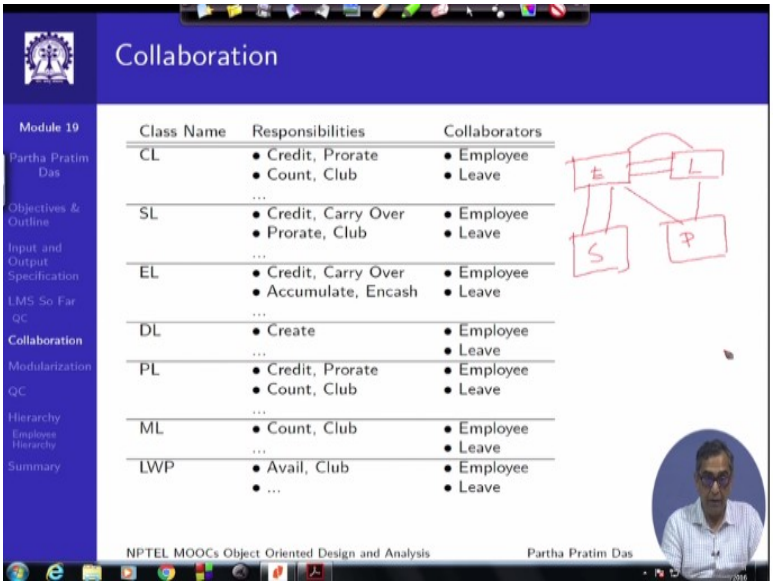
And identifying collaborations is basically again minding through deeper in terms of the specification to say that what are the objects, what are the corresponding classes that would be related through there mutual responsibilities. Managers collaborates with executive through the process of hiring, manager hire executive. Manager would collaborate with the lead in terms of the usual approve leave kind of which is not listed here, but that will also be

here so approve leave kind of responsibility that the manager has.

So in this way we try to identify different classes and their collaborators and again please try to follow similar kind of technique that we were doing in the linguistic analysis also they try to look into every class all responsibilities for every responsibility you try to find out what are the recipients, what are the other actors that exist and put them as collaborators and once you have done that you will find that again there will be lot of repetitions.

And you will be able to then refine it to actually find the minimal set of collaboration that need to be in place.

(Refer Slide Time: 11:28)

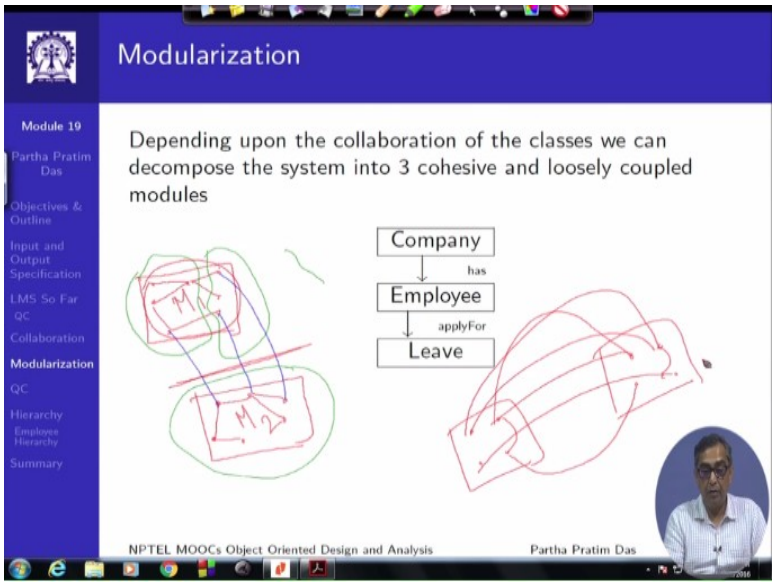


So this is just some more extension if you actually look into the earlier here primarily the first few, primarily where the different kinds of employees whereas in the next these are primarily different types of leaves, these are the different categories of leave and their responsibilities and how they collaborate with other classes and you can certainly see that all of these categories will have lot of commonality in terms of the collaboration and this commonality is what we will see, we will try to leverage further when we refine that design more and more.

So collaboration if you ask collaboration by itself may not result into a specific aspect of the design, but if we understand the collaboration then we understand that if I have employees here, if I have leave here, if I have system admin here, if I have printer here and so on we have not listed the printer, but then we can see what are the different types of collaborations that may exist between these different classes.

And based on that we can find out how coherent they are, what kind of coupling between them exist and that give a good clue in terms of modularization and all those as well. So this is the basic approach and I would again request you to pause and try to identify as exhaustively as you can all the different collaborations that you see in the LMS specification.

(Refer Slide Time: 13:16)



Modularization is another exercise where you start using the collaboration information more strongly is where you try to see that what would you like to put in a model if we put it differently how do we like to modularize if you remember then we have said that if I have two modules that we have talked about this sometime back then we look into the interaction between different objects in the modules these are the different interactions between the objects in the module.

And we will look into the interactions across the module. So we will say that our task our objective is to define the boundary based on these interactions based on all these what else could be a third colour that I might use maybe this one. So based on all this our intention is to decide should we modularize like this, like this and like this or should I do it differently so that the intra module that is the intra module the traffic that is happening within this.

The collaboration that happens within the module and the once in the blue which are basically inter module reach two extremes that is intra module you would expect lot more of traffic, high density of traffic whereas inter module across the modules we will expect very low density of module or traffic. So if we can achieve that then certainly we will say that we

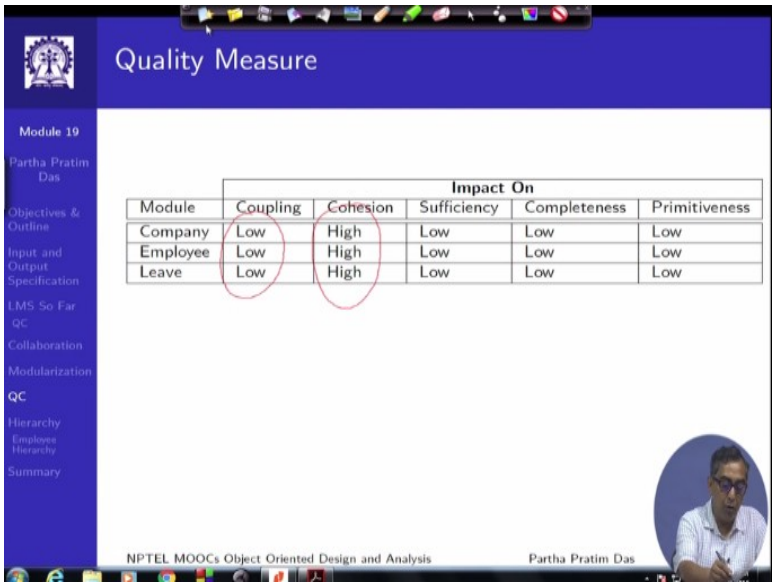
have been able to modularize value.

If we do the other extreme that we did have a module here, we have a module here. We have couple of objects place like this and say this is the interaction, this is the collaboration that exist between them and there is if this is a scenario that we see then we will not say that this is a good modularization possibly we should have put these two in the same module and not across the module.

So this is just to understand that collaboration this information of collaboration would give us a very good inside in terms of how should we modularize. LMS is a relatively simple system so we see primarily the first level we already see three modules emerging one is the company and that module will interact with the employees in terms of companies has employees and then employee module in interact with leave in multiple ways so which can be abstracted in the sense that these are apply for or you know avail of leave and so on.

Of course when you try to complete the exercise you will find that these are not the only modules in the system there would be other modules like your basic accounting maintenance has to be a login module which deals with the login and authentication related activity. There were possibly a backup module to keep backup of the annual leave record and so on, but your basic analysis of the collaboration information which you got from the analysis of responsibilities will give you a initially a good clue in terms of what could be the modules in the system.

(Refer Slide Time: 17:12)

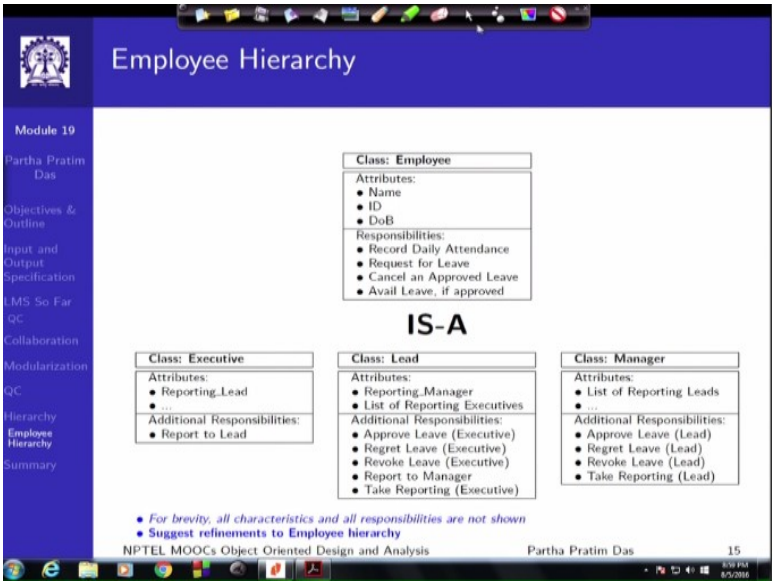


	Impact On				
Module	Coupling	Cohesion	Sufficiency	Completeness	Primitiveness
Company	Low	High	Low	Low	Low
Employee	Low	High	Low	Low	Low
Leave	Low	High	Low	Low	Low

So at this stage we might take a quick another look into the quality measure to see how we are doing. Naturally the cohesion is improving, cohesion is becoming high we have been able to kind of put the employees together in a module the leave together in a module, the analysis of the collaboration has allowed us in terms of doing that. Company of course is a pathological example because that just one class so it is coherent by itself.

Coupling we still do not know much, we have not seen the coupling aspect though we have the collaboration information, but we have not seen in terms of what the hierarchy could be so this is still low and the other parameters are low because we are still to complete the whole aspects of the design as yet.

(Refer Slide Time: 18:06)



So next let me just start off with the analysis of hierarchy and this is a possibly you saw this earlier also in couple of different other context, but we will say that the collaboration and the responsibilities give us a lot of clue in terms of the hierarchy that we at least should have at the first level. So if you look at the collaboration pattern will tell us that these four classes are strongly related that has also been collaborated by the fact that we found them suitable to be put in the same module.

Now the question is in that module, in that employees module at this classes are they going to remain independent standalone or are they going to be related through some specialization, generalization relationships. So what necessarily help is again going back and looking at the responsibilities, analyzing the responsibilities and if we do that we find that some responsibilities like recording daily attendance, requesting for leave, cancelling an approve

leave for self, availing a leave all these kind of are found in the employee and actually found in all of these specializations, potential specialization I should say that exist.

So once we see that commonality we start sensing that there is ceratin way to put them in a hierarchy, but of course just be identifying this we do not know if the relationship should be like this or the relationship should be like this that is we cannot say whether executive is an employee or an employee is an executive all that you can say that they share a lot of things that might make sense to test the waters to see if one can be a specialization of the other.

So in one part the identification of hierarchy means that you identify responsibilities, you identify attributes which are common between multiple different classes which possibly by now you have put in the same module. So another would be to looking at the attributes and if you look at the attributes you will find that all of these 4 different classes actually have these attributes of ID which is employee code, name, date of birth and so on.

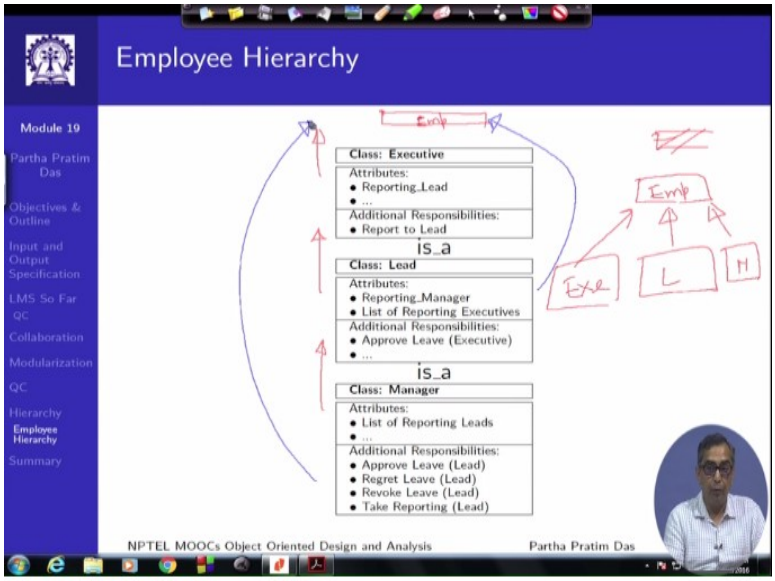
Next to decide on the direction of possible specialization you will need to look at where do they differ so this was a common part, common responsibilities and common attributes. Now you have to look at where do they differ why does an executive differ from in general from an employee is there an executive reports to the lead. It is not explicitly said, but this is where you will do some extensional logic that if the executive reports to the lead then there has to be some attribute for an executive which is a reporting lead.

Remind you if you go back to the document look at a list of nouns and so on you will not find this explicit, but this is where we are making certain deductions from the relationships, making certain deductions from the possible collaboration that if executive reports to lead then executive will have to keep the information of that collaboration object which is a reporting lead.

Similarly if we look at the lead then actually we start finding that though there are lot of responsibilities which are common between the employee and the lead, but there is a lot like approving leave, regretting leave, revoking leave and so on which are additional for the lead. So in this way when we analyze all these four classes we find that this class the employee class has kind of the common minimum staff which all other classes have.

And therefore we agree or we decide on an initial specialization where you say that executive is an employee, lead is an employee and manager is an employee. Naturally we do not stop here we would once we have identified this then our effort would be to see can the quality of this specialization we improve this is just one level and certainly therefore it is wide in space.

(Refer Slide Time: 23:29)



Now if we analyze further and now earlier we were trying to see that employee was the common minimum so we are trying to see between employee and each of the other three classes. Now we try to look at once we know that we have a basic structure which is employee, executive, lead, and manager once we have that then we try to see are there commonalities that exist between these specialize classes.

And when we start doing that we do find a lot of commonality between the lead and the manager both of them can for that matter approve leave, both of them can regret leave and so on. Similarly there is commonality between the executive and lead as well both of them can apply for leave. Actually all three of them can apply for leave and so on. So if you do a similar analysis of the containments of responsibilities and the collaborations you will soon come to the conclusion.

Instead of just every each one of the employee types being a specialization of the employee possibly a more refined model should have a multi layered structure where you say that the manager is a lead because we find that most of the things if not everything exactly that a lead can do a manager can also perform. When we say lead is an executive because anything that the executive can do the lead can always do.

And finally we preserve the original hierarchy that we are created that an executive is an employee. So earlier we had let say the employee was a specialization for executive for lead and for manager and now that has just become multi layered. So keep this in mind that while we do this refinement it will be quite bit if you actually violate the original hierarchy that you had.

So this hierarchy said that lead is an employee so here by transitivity you will also have that lead is an employee. So yo do not explicitly show that because you say that lead is an executive and executive is an employee and therefore by transitivity this holds. Similarly manager is an employee is through the transitivity of all this relations. So when you do this kind of a refinement by exploring more information from the specification, from the problem domain.

You should be careful that your original design aspects are do not broken, do not get violated if they do then you will have to review the design very carefully to see whether it is getting violated because what you had assumed earlier possibly based on partial information what had some wrong notions, some misconceptions so you are correcting it now or whether in order to refine you are actually making new mistakes in that design.

So this is the broad way of designing on the employee hierarchy so if we show it in the final form this is how it looks and along with that there are different other you know relations that they perform we will talk about this more when we talk about the relationships in general. So to summarize this has been the third part of our exploratory exercise on the design of the LMS system.

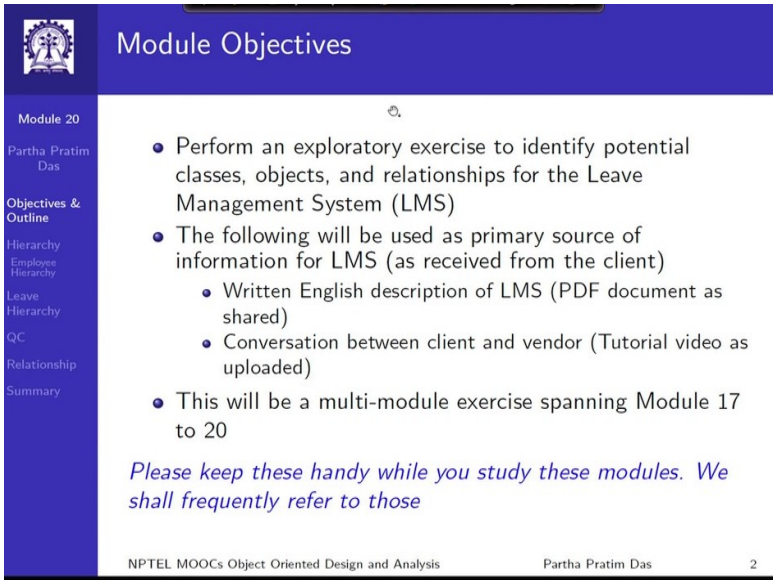
And in this based on the classes attributes and their responsibilities we have first tried to identify the collaborating classes the collaborators and then we use the collaboration information to create a basic level of modularization in the system and finally using the modules as well as the collaboration and responsibility information we have tried to generate a basic hierarchy of employees and a little refinement of that and as we move into the next module we will show further depths of refinement of hierarchy.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology - Kharagpur

Lecture - 32
Identification of Classes, Objects and Relationship in LMS (Contd.)

Welcome to module 20 of Object Oriented Analysis and Design, we have been pursuing the exercise of doing a basic design of the Leave Management System.

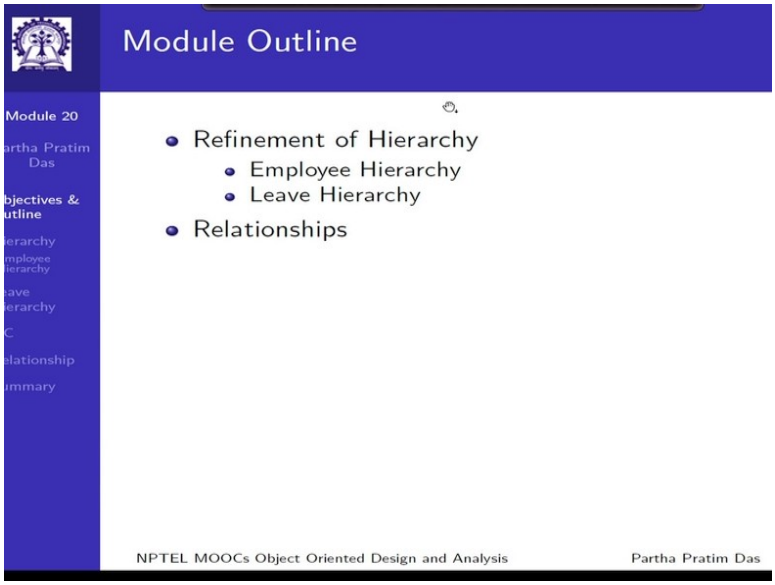
(Refer Slide Time: 00:43)



And this will be the last module on that exercise, so I will just reiterate that please keep the PDF description of the leave management system handy with you, you will need to refer to it very frequently if you have not already very carefully listen to the interactive video of the client-vendor interaction for LMS, please do listen to that carefully.

And while you go through this module please considered to stop and work out things for yourself when we have outlined any new action on the design aspect that way it'll be able to learn the design process much better.

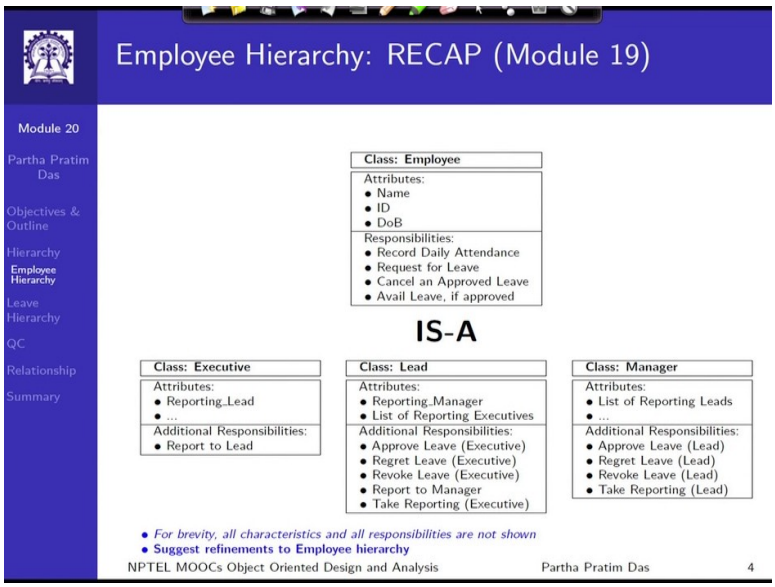
(Refer Slide Time: 01:30)



In this specific module, we- we will particularly focus at refinement of hierarchy and will try to show that in addition to the identification of responsibilities which we did earlier and the identification of collaborators which also we did earlier while we recognize where we extract a hierarchy it is quite possible that a lot still can be done on top of it by going back to the specification and trying to look for a lot more of possibly intangible information that are hidden all over the document to improve on the hierarchy and get to a better design.

And we will just also outline in this a little bit of relationship between classes.

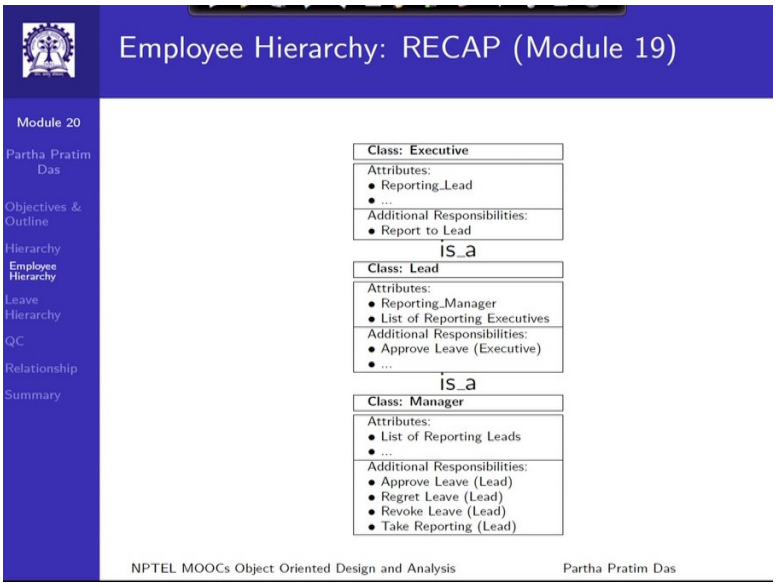
(Refer Slide Time: 02:26)



A quick recap of the recent stuff that we have done and employee hierarchy design that we did in the last module we did first did a flat design of one level which is just by kind of factoring of responsibilities finding the common responsibilities between the employee and

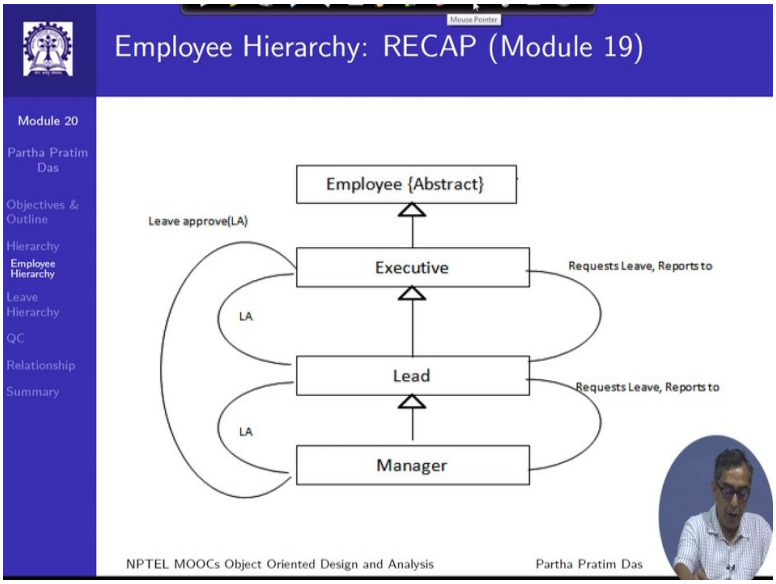
the other classes we made Executive, Lead and Manager all specializations of the employee class.

(Refer Slide Time: 02:59)



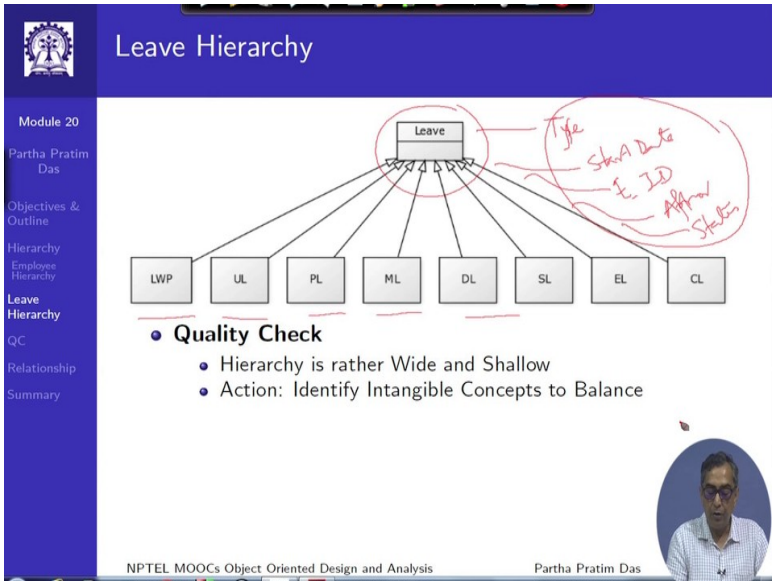
And then will look into refined that further by seeing that it is not only a common subset of responsibilities between employee and others there is a hierarchy of subset of responsibilities between these four classes and therefore they form a linear chain naturally they form a linear chain of hierarchy and which is designated as per here.

(Refer Slide Time: 03:20)



Now we would like to just a take this further an apply that to leave.

(Refer Slide Time: 03:30)



So you have already read the specification document several times so I am sure you will not have to go back to its to conclude that all kinds of leave that we are talking about are fundamentally leave so if we - if we try to say this in terms of attributes sets then we can say that this will need to have a type which we have already received it needs to have a start date.

Certainly it needs an employee ID as to who is taking the leave certainly it needs to have a approver it needs a status and so on. So these are things that a leave will have their maybe lot more, and each one of this special kinds of leave all of them will also have these properties these attributes but depending on a specific type of leave - leave there will be more specific attributes more that will exist.

For example, a medical leave will have certificate a longer duration and so on am - on leave will have the associated encashment benefits and so on so forth, so doing this hierarchy does not take much effort does not take much time so if you just put it back to our quality check I would refer you to the quality of design discussion we had particularly in terms of choosing the relationships and so on.

Then we will see that this is what is considered to be a certainly it is a very wide and shallow hierarchy is very wide everything is and very shallow, so which means they are potentially could be lot more of in a communality of a behaviour lot more of possible natural coupling that we're missing out here, so we want to discover that and so the action would be to identify intangible concepts to make this a more balanced hierarchy, okay.

(Refer Slide Time: 06:05)

Requirement Specification

A Company wants to manage the attendance and leave of its employees through LMS. The requirement specifications are:

1. The company has three categories of employees:
   - *Executive*: Employees who work as individual contributors and report to a Lead.
   - *Lead*: Every Executive reports to a Lead who approves / regrets her / his leave. A Lead reports to the Manager.
   - *Manager*: Every Lead reports to the Manager who approves / regrets her / his leave. There is *only one* Manager.
2. The company has provisions for the following categories of leave associated with the respective leave rules:
   - *Casual Leave (CL)*:
     - 10 CL's are available in a calendar year. All CL's are credited to an employee on 01-Jan. For employees joining in the middle of the year, the number of CL's are prorated. CL's cannot be carried over to the next calendar year.
     - More than 2 CL's cannot be availed at a time. CL's cannot be clubbed with other types of leave. Total period of absence including holidays cannot be more than 4 days. Holidays intervening the absence are not counted as leave.
     - CL's do not need pre-approval; but must be approved within 2 days of its availing.
   - *Earned Leave (EL)*:
     - 15 EL's are available in a calendar year. 1.25 EL is credited on the completion of a full month's service. EL's can be carried over to the next calendar year and accumulated up to 45 days. Once it crosses 45 days then on the completion of the current quarter, 30 days are en-cashed and paid to the employee. Remaining EL's continue in the account. All EL's are en-cashed when an employee leaves the company.
     - EL's can be availed at a stretch and up to the existing balance. It can be clubbed with other leaves (except CL). All holidays within the leave of absence of EL are counted as EL.
     - In exceptional cases Manager can approve more EL's than what exists in one's account. Maximum of 15 days' negative balance is allowed.
     - All EL's must be pre-approved (at least by a week).
   - *Duty Leave (DL)*:
     - When an employee is sent out of station or on leave on work, a DL is created.
     - Every DL is approval basis and has no specific accounting. It is considered as being "On Duty".
   - *Sick Leave (SL)*:
     - 12 SL's are available in a calendar year. All SL's are credited to an employee on 01-Jan. For employees joining in the middle of the year, the number of SL's are prorated. SL's can be carried over to the next calendar year and accumulated up to 60 days.
     - SL's can be availed at a stretch and up to the existing balance. It can be clubbed with other leaves (except CL). All holidays within the leave of absence of SL are counted as SL.
     - In exceptional cases Manager can approve more SL's than what exists in one's account. Maximum of 12 days' negative balance is allowed.
     - Medical certificates are needed to proceed for and join back from SL's. SL's can be approved post-facto in cases of emergency.
   - *Maternity Leave (ML)*:
     - Every female employee is eligible for 4 months' ML when pregnant. It can be clubbed with other leaves (except CL). All holidays within the leave of absence of ML are counted as ML.
     - Medical certificates are needed to proceed for and join back from ML's.

So if we want to do this then let us see what are the things that we can look at what can we get those intangible ones, so this needs little bit more careful reading of the specification so please bear with me for one to leave I am going to read out for your benefit the what is stated for the leave.

So it is a ten CL's are available in a calendar year first sentence, all CL's are credited to the employee on such and such date, employee joining in the time - middle of the year the number of CL's are prorated therefore the sentence being said. CL's cannot be carried over to the next calendar year more than two CL's cannot be availed at a time, they cannot be clubbed the total period of options including holidays cannot be more than four days.

But the holidays intervening are not counted right so, these are the different sentences that are coming up so our - our mental process is as you go to study about the next type of leaves Earned Leave we read it and we tried to see, okay if 15 EL's are available in a calendar year we mentally make a - a kind of map that available is something which mere relate the different kinds of leave so available.

Then we can say that we see that we already seen the verbs the credited so we know that available is some factor credit is certain factor we say the carried over is talks about carried over now I do not know EL's can be carried over EL's can also be carried over so carry over is another aspect.

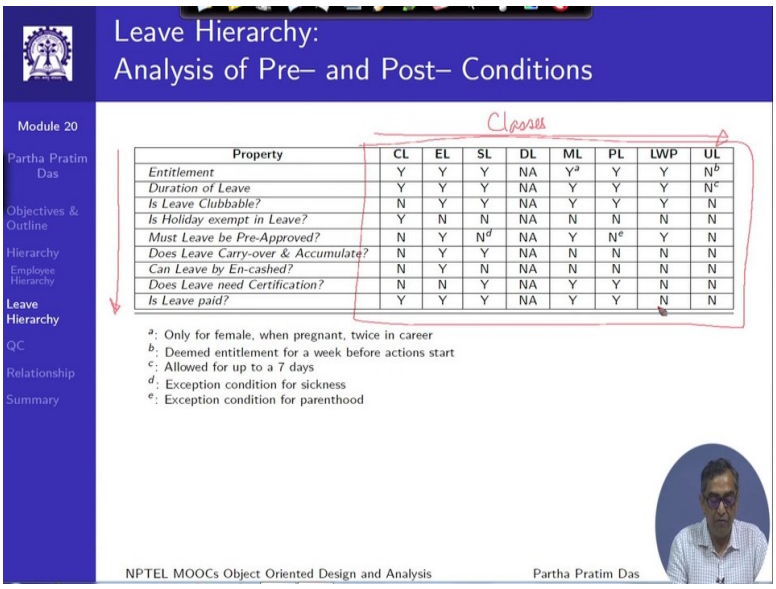
Then there is some encashment encash is another aspect right, these are the different things that we are seeing here, then it says it must be pre-approved and then we say CL's does not need to be pre-approved so we can say pre-approved. So all that I am doing I am trying to go through this different kinds of leave.

And for example I could - I could scroll up of course this set of markings will go here on that but we can - I can use a different colour, I can use a different colour and try to do some more marking like this here is again talks of available, talks of credit it does for SL it does talk of what else carry - carried over this is the basic concepts that we have been we have been seeing here. So what - for example it talks about something new it talks about a Medical Certificate.

So what I am trying to point to is if you try to go through each one of these leave categories and try to see what are other than the factor that it is a leave what are the different kind of conditions constraints properties that this system is talking about and then kind of try to make a representation for that you could represented in whatever way you please so what I will present you here is certain representation that I tried to extract from this description so that we can use it in terms of the design.

So I have represented in terms of a table so if you look into the left hand side.

(Refer Slide Time: 10:24)



Property	CL	EL	SL	DL	ML	PL	LWP	UL
Entitlement	Y	Y	Y	NA	Y[a]	Y	Y	N[b]
Duration of Leave	Y	Y	Y	NA	Y	Y	Y	N[c]
Is Leave Clubbable?	N	Y	Y	NA	Y	Y	Y	N
Is Holiday exempt in Leave?	Y	N	N	NA	N	N	N	N
Must Leave be Pre-Approved?	N	Y	N[d]	NA	Y	N[e]	Y	N
Does Leave Carry-over & Accumulate?	N	Y	Y	NA	N	N	N	N
Can Leave by En-cashed?	N	Y	N	NA	N	N	N	N
Does Leave need Certification?	N	N	Y	NA	Y	Y	N	N
Is Leave paid?	Y	Y	Y	NA	Y	Y	N	N

For example, we - we did see the question of whether let me use red that is better whether a leave needs to be pre-approved we saw that CL said that it does not be pre approval where is

EL said that it needs pre approval, we saw the issue of carry over, we saw some leaves can be clubbed we saw ML leave need certification, so every time you come across cert - certain tangible or intangible properties and constraint you try to put them as the system property and that is why I got this list.

And there can we make a chart of these are the properties and these are the different classes I already know that classes and I know why do I just do this for these classes why do not I take employees why do not I take an admin and why do not I take for that matter calendar year into this because I am trying to refine a hierarchy so my containment is within that hierarchy I have already done those analysis my basic modularization is there within that I have actually created a entity hierarchy so if I am just focusing on that.

So well on those to access of property and the classes so this - these are the properties so this basically is not written its obvious these are the classes, I tried to fill up this is the compliance to what is the compliance of different class to these properties so this kind of gives me a very compact representation for what is intangibly described over two pages or one and half pages of different leave conditions.

if you want to read it through say - you say is leave clubbable so we have read that it says that CL cannot be clubbed but naturally if its unauthorized leave then it cannot be clubbed anyway but and if it is the duty leave there is no question of clubbing it because you are on duty anyway, but all of this now you can go back and actually reads to the document you will find that all of these are actually clubbable.

Similarly, say can the leave be encashed while doing the reading about EL we studied that leave EL can be encashed so you put yes here and rest of them are basically not applicable are known, and of course this this will not give you very uniform design always

For example, in case of entitlement, entitlement is basically what - what I write here is entitlement is what we noted as credit do you remember that we say that every year in 1st January 10 CL's are credited then for every month 1.25 EL's credited and so on, so that cred is what I am calling that entitlement as to what are you entitled to get if you look into the entitlement these are all different types of leave.

But certainly you have a special qualification here when you're talking about maternity leave the qualification is and that is what this suffix represent at this leave is available only for females when if female employee is pregnant and also at most twice in the career so this kind of you also try to keep track of some of the special conditions which might exist in terms of this general broad illustration of the intangible properties that you'll get to see.

For example, you could look at some of the others for example here which is d or does it mean must leave me pre-approved and we are talking about so this must pre-approved for sick leave that is d so which says that exception condition for sickness that is the if we- if we go back and see what we are encoding if we look at that then let us - let us specifically focus on the sick leave part it is it is tried to see it says that SL's can be pre-op - can be pre- post-facto in case of emergency.

Because certainly you cannot predict always that you are going to fall sick so if you fall sick and you cannot attend office if you apply for sick leave in the post- facto manner that is after you have taken the leave so we have basically utilizing that information and encoding it here saying that in a may be a little bit different manner that must leave be pre-approved'

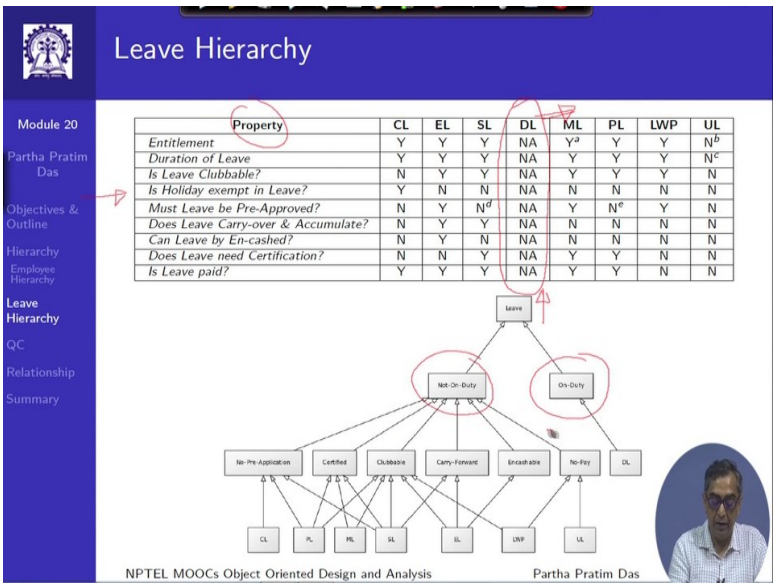
If you are being able to take it post-facto take the approval post facto, then is not pre-approved for a SL and but that is a more exception case it is not that it for every SL at least that sentence does not mean kind of suggests that it is always says that it can be approved post facto in the case of emergency so if you have you know pre-planned medical checkup or you are taking up some sir surgery something then certainly it will be expected that it will still do a pre-approval all that.

So these are the different things and you can - you can just go through the details later on and see the encoding so all the time trying to show you it is not always that you just read and you are not try to analyse and do things in your mind but you can try to use variety of tools of tabularization and compaction to come to some better conclusion.

So what does this table lead us to? This table say that the leaves are different kinds of precondition and post condition and we can say that can we not then try to reclassify reorganizing the leave based on these properties that is based on some of these properties

For example, if certain leaves are need certification certain say no, this say no, this says no but some say yes if some need certification and some do not - does not it impose a specialization on the set of leaves that you have and that is the basic idea that we will use here.

(Refer Slide Time: 17:24)



Property	CL	EL	SL	DL	ML	PL	LWP	UL
Entitlement	Y	Y	Y	NA	Y^a	Y	Y	N^b
Duration of Leave	Y	Y	Y	NA	Y	Y	Y	N^c
Is Leave Clubbable?	N	Y	Y	NA	Y	Y	Y	N
Is Holiday exempt in Leave?	Y	N	N	NA	N	N	N	N
Must Leave be Pre-Approved?	N	Y	N^d	NA	Y	N^e	Y	N
Does Leave Carry-over & Accumulate?	N	Y	Y	NA	N	N	N	N
Can Leave by En-cashed?	N	Y	N	NA	N	N	N	N
Does Leave need Certification?	N	N	Y	NA	Y	Y	N	N
Is Leave paid?	Y	Y	Y	NA	Y	Y	N	N

And here I again show you the same leave hierarchy and I should have actually use different colour but in terms of the original hierarchy this was the root and I am sorry this was the root and the participating classes are this, so the original hierarchy was just kind of a flat where we just said that each one of the leave is DL, CL, SL, ML all are the kind of leave.

Now we are saying if you look at if we say is clubbable so we have defined a - we defined that let us have an abstract kind of a leave class as clubbable in class which will have all the properties of leave so that will be a specialization of leave, but it will have some additional properties that are required for checking out the clubbability condition or imposing that condition and so on.

And then we make all that if this is clubbable, then this is clubbable, this is clubbable, this is clubbable, this one is, this one is, so you say EL, SL, ML, PL Parenthood Leave and the Leave Without Pay these five are clubbable so we show that this five are specialization of clubbable leave.

Now this is- this is an interesting exercise for example if you see that in this - in this whole exercise if you look into this column DL you have everything has any nothing applies so that

clearly tells us that it is not only always that the horizontal factorization will have to be used you could use a vertical factorization also that you say that this particular leave category is out of the general.

So let us try to identify some property which is not identified here so that I can take this column out and that is precisely what I do here. I just see introduced a concept that is a On Duty leave and is a On Duty and a Not On Duty, so naturally DL becomes only specialization that and all the other leave actually come under the others part of the category.

So certainly you can - you can realize you can expect that this intangible classifications we will give you certain bit of benefits positively because if I know that this is the On Duty one and these are all Not On Duty so which means that for any of the other leaves you will not at all look at or you will not at all ask the employee to fill up is work responsibility during the period of absence whereas only for this you will need to fill it up so that is the different specialization that can be worked out.

For example, if we look at one more example let say the does leave need certification so SL need certification, ML need certification and PL need certification so certainly if it needs certification then your final effective processing behaviour is going to be different and if you - if you note in this exercise we have made some progress in terms of just the first round identification of responsibilities and analyzing that we got the basic hierarchy.

But now we are looking deeper into and kind of if you - if you remember kind of we are moving somewhat from what was Discovery you would remember we had talked about Discovery and Invention in terms of identification of key abstraction Discovery deals with that user who is domain vocabulary and so on and those are the leave and earned leave casual leave and all that and Invention talk about the developer the implementer.

So that is what the user will not look at the leave as whether it is clubbable or whether it is - it is a certifiable on so on so forth but does not implemented it does help to look at leaves which are certified and note that there are three leaves are that kind and make them specialization and clubbed all certification related functionality all certification related responsibilities in this class.

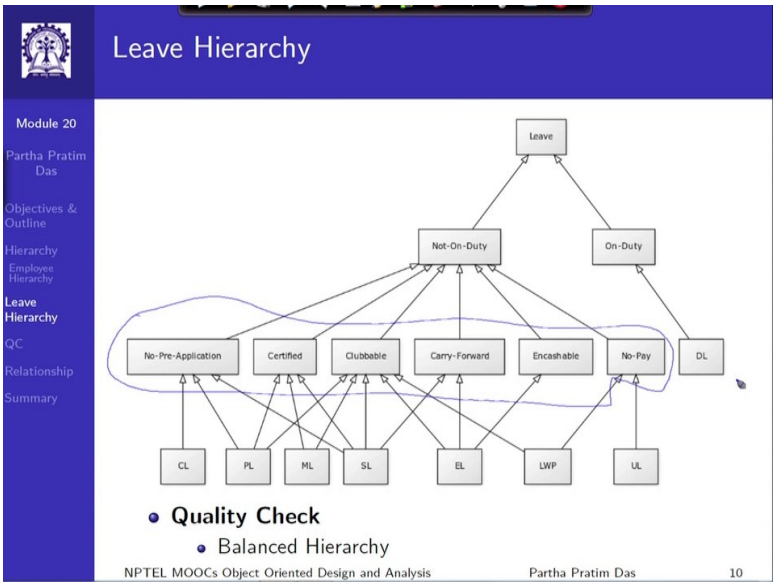
And certainly you can -you can easily figure out that finally a leave is of one of these kinds so anything else that we are doing any of the other that we are doing the rest of this whole hierarchy that we are doing are abstract so we will never have a leaf which is just certified, you will have a specialized in stance of that in terms of either PL or an ML or an SL, so that is a basic.

So this exercise I wanted to really so you mining information from the leave document to show that you will whenever you were given to work on a hierarchy which is a very very critical we have repeatedly said that hierarchy of abstraction is a very very critical part of the OOAD design because if you get - if you do not get the hierarchy right then the whole implementation of the system whole conception of the system becomes is not on solid foundation.

So you really should do a very extensive exercise in terms of trying out different and an again, again, please do not think that this I am saying that this is an absolute hierarchy for the LMS system you could think of different other parameters and that you could organize them differently but as long as you have reasonably covered all that is mentioned in the specification.

And as a developer you have all your requirements all your questions answered you have a fairly good design and certainly if we now take this as a - as a final that we have in terms of this leave part of the LMS.

(Refer Slide Time: 24:11)



Then we can certainly say that we have a much better balanced hierarchy now which goes into one, two, three, four different layers and you would see that many of these classes have very limited number of specialization showing that they are very focused abstract classes which kind of focus on the small set of aspects of the design.

And all of this intermediate classes that we show here are represented you of that for each class is basically going to implement on top of the basic leave functionality some of the very specific functionality that need to be imposed on the certain types of leaves. So that is - that is a basic of the hierarchy design, I would emphasize that you please go over this design more carefully and try to refine that further.

(Refer Slide Time: 25:14)

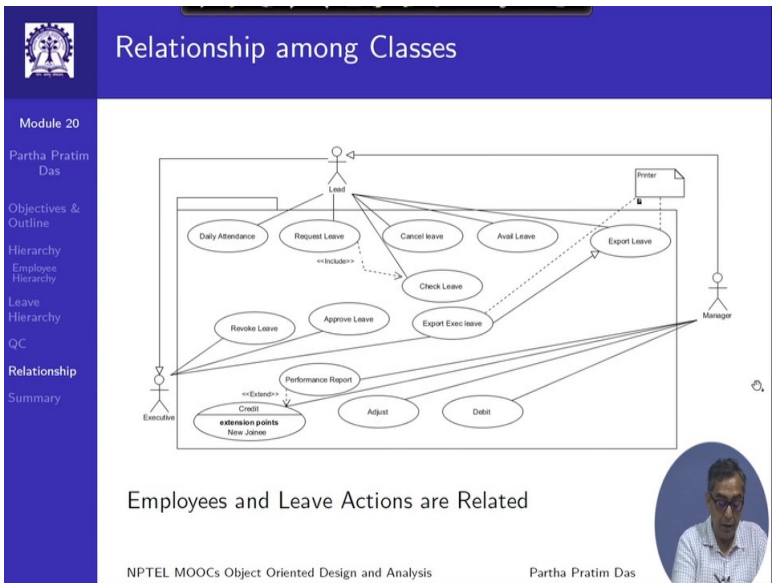
	Impact On				
Module	Coupling	Cohesion	Sufficiency	Completeness	Primitiveness
Company	Low	High	Low	Low	Low
Employee	Moderate	High	Low	Moderate	Moderate
Leave	Moderate	High	Low	Moderate	Moderate

Coming back to the quality measures certainly we have made some more progress and the very significant one would be in terms of Coupling because now we have the hierarchy in place and as you know I work in process a lot of coupling so in terms of an employee and leave we are at a moderate level of coupling if not at high level of coupling the Cohesion does not change.

But certainly we have made a lot of progress in terms of completeness because particularly for employee and leave we have made write a detailed analysis of how they are structured how they are collaborate how they can be putting modules what are the responsibilities and so on.so in terms of days we have made some progress and going ahead in terms of the quality measure in our design.

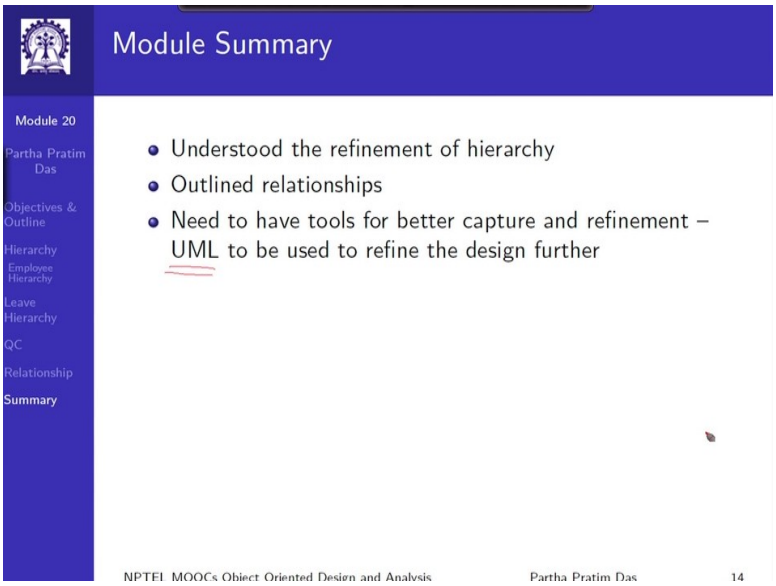
Besides that, there could be besides the hierarchy there could be several other relationship that we like to capture we would not do it any detail exercise of that in this current module for example between executives there could be relationships of leave approval and there could be request reports and so on, they can be shown in multiple different ways.

(Refer Slide Time: 26:39)



Similarly, if we looking in at a bigger perspective then between the employees these different employees the Manager, is a Lead, is an Executive and different these are different leave actions actually we can see lot of relationship association. And in this manner I would not go on in terms of just elaborating on the relationship because relationships are so many varied kinds associations could be of so many varied kinds that it is about time that we formalize ourselves and start taking help of a more structured tool the UML tool.

(Refer Slide Time: 27:17)



And learn how to first represent whatever we have designed for the LML systems so far the basic information of classes atributes responsibilities and their basic relationships in terms of modules and the hierarchy that we have, and then we will go deeper into learning the Unified Modelling Language taking different aspect of these are called diagrams and as you pick up different diagrams will see which details of the system are captured by the diagram.

And these four modules where we started exploring the and doing kind of a hands on for the LMS system design for just the introductory part we will continue doing this over the next couple of weeks as such when we go through the UML a so that by the end of the course at least for the LMS. We will be able to have a set of completed UML designs that you can refer to and in terms of your assignment and practice.

We have - we will be putting up other systems like we have already mentioned about an Assignment Management System and you should do similar exercise of analyzing and designing for that system.

Object-Oriented Analysis and Design

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology - Kharagpur

Lecture - 33
Overview of UML

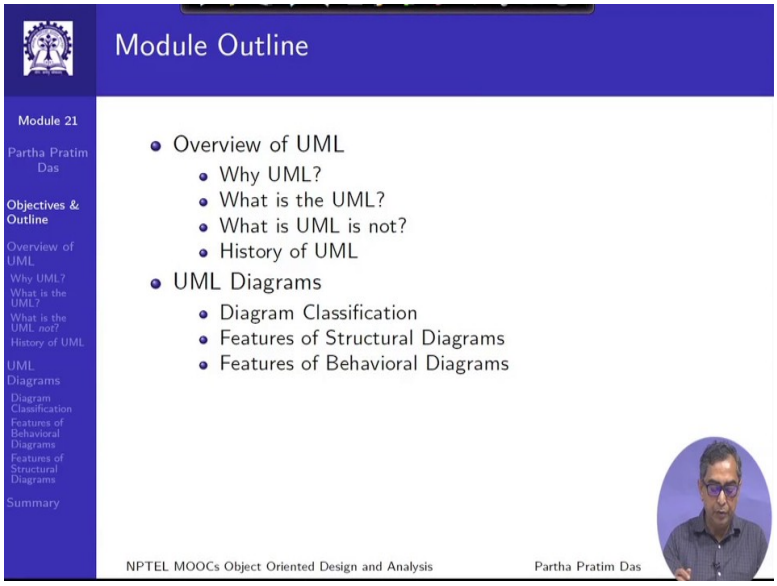
Welcome to module 21 of Object Oriented Analysis and Design in the last couple of modules, we have discussed and actually worked out an exercise with a Leave Management System to illustrate how different linguistic and important analysis techniques can be engaged to extract the classes the key abstractions their relationships and so on.

In this context we can go further and that is our endeavor to do so but before we proceed further on extracting more and more system information from the specification and start modelling them for proper design we need a vehicle to express the information that we are extracting, naturally we are our starting point has been certain English documents interaction videos and it could be other supporting documents images charts and so on.

But as we extract the information and start creating the analytical design of the system we need a language which can be used very compact way and very concretely to express all the information that we are extracting. UML or Unified Modelling Language is such a vehicle so in the module today and in several modules that are to follow this.

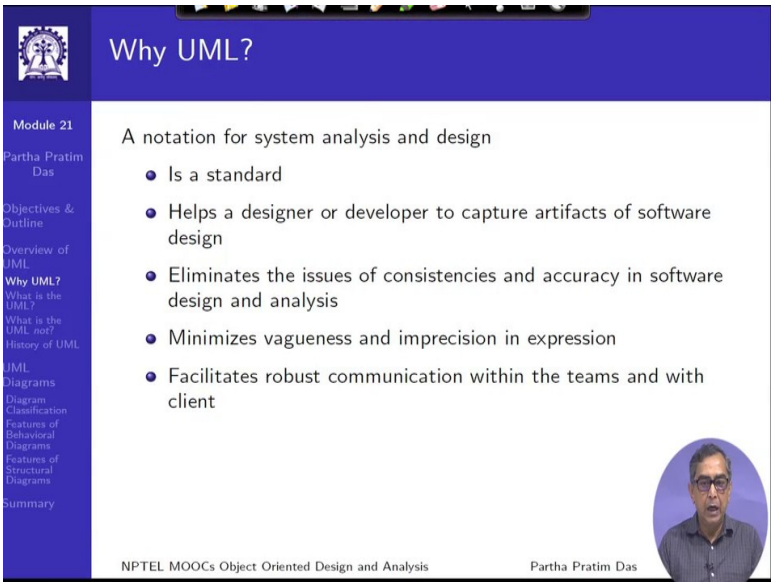
And in the next couple of weeks we will take a deep look into what is UML, how UML is represented and as we work through different representations of system concepts designs parameters into the UML we will illustrate as an ongoing process as to how we can use that Leave Management System and create its Unified Modelling Language models so with that a broad objective we will start this module by discussing it basic overview of UML.

(Refer Slide Time: 03:17)



This is what we intend to discuss and as usual this outline would be available to the left of every slide as we go through the presentation.

(Refer Slide Time: 03:27)



So the first question certainly is why do you need something like UML, as we have seen that so far there are two kinds of descriptors that we have for a complex system on one side on one extreme is a descriptor which specifies the requirements of the system specifies different behaviour and entities of the system and typically as we have observed it is written in a natural language.

And as we have repeatedly explained there could be supporting information discussions, notes, images and all that the key difficulty in this form of representation of a system is natural language is often ambiguous vague imprecise from the same text what I understand what you understand and what someone else will understand would vary widely the diagrams help and yet they could be drawn with different conventions with different practices.

So this way of representing a system is not a uni-clear method by which we can unambiguously precisely understand what the system is all about, the extreme of representing a system is its implementation when the system is gets implemented in terms of possibly a Object Oriented programing Language so if we intend to use C++ or Java or some such, then we will have the whole system again described represented in terms of a set of classes methods exception conditions dependencies modules of sub systems messages threads and on so forth.

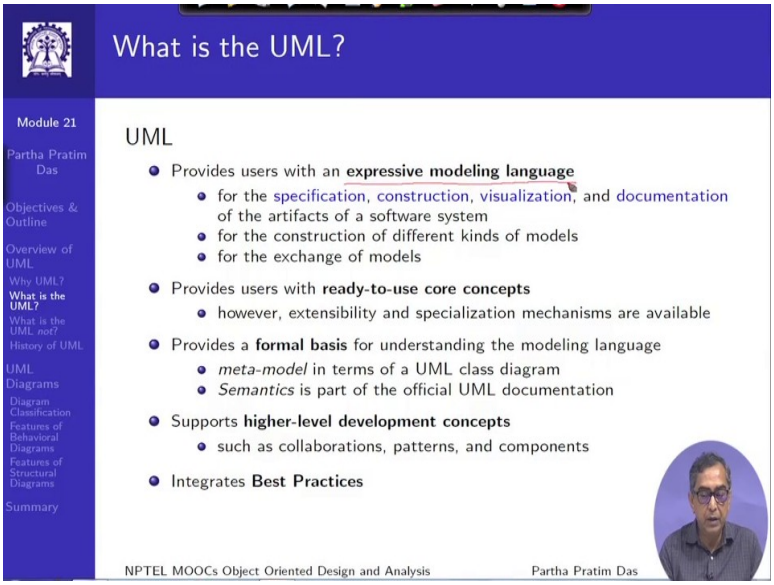
Again another description which is very specific and would vary across different platforms different languages different paradigms and so on, our intention our requirement in the whole process of OOAD is to start with the requirement specification of a system and reach the implementation software implementation of the system both of them we find are difficult complex to handle they are not very clear precise in the expression.

So that brings in the context of using something like a UML or Unified Modelling Language which is basically a notation for system analysis and design the first importance of why should you use UML it is a standard it is an International standard maintained by the object modelling group or OMG.

So it helps the designer developer even the user to capture various artifacts of software design various inventions details of software design can be uniquely captured in UML without any scope for imprecision vagueness ambiguity or the viewer or reader dependent understanding so it eliminates the issues of consistencies and accuracy that typically will exist in a software analysis and design problem it minimizes vagueness and imprecision.

As I mentioned and coupled with all these it facilitates a robust communication with within teams robust communication between the vendor and the client and so on, so the basic process would be that the initial stages of our design and analysis we will try to interpret the specification from different multiple natural language and multimedia sources and create this UML diagrams which will be the unique representation and kind of the golden on which the whole system will then be built up.

(Refer Slide Time: 08:01)



So what is UML, UML has can we look at in various dimensions first of all its an expressive modelling language it is a language to model its set to be an expressive language because it is allows you to model almost anything that you need to model starting from specification, to construction, to visualization, to documentation everything that you need in the whole life cycle of a software development system implementation development can be captured in UML.

For the need of clarity expressiveness and precision UML is almost totally designed as a graphical language so it is not which depends either on the vague semantics of natural language nor does it deal with the complex logic and analysis of the programming languages it is a graphical pictorial kind of language and it allows us to use a whole lot of ready-to-use core concepts.

You recall from the earlier discussion that we had talked about reuse we're talked about common patterns that occur frequently in complex systems that occur across domains and so on, so what UML has done UML has tried to create semantic expressions for those concepts and met them available as a core in the language itself so that very often once you come across such a concept in the system you are going to represent you would simply be able to find a primitive find a feature in the UML which directly represents that.

Besides that, UML provides a formal basis may be for some of you it may be difficult to understand what a formal basis means what it means is though it is a graphical language it should not be just taken as a pictorial representation it has certain strong mathematical foundation which can be proved to always provide a consistent correct description of the system alone, and in that process it uses certain meta-models it uses a semantics which is the part of the official UML standard documentation.

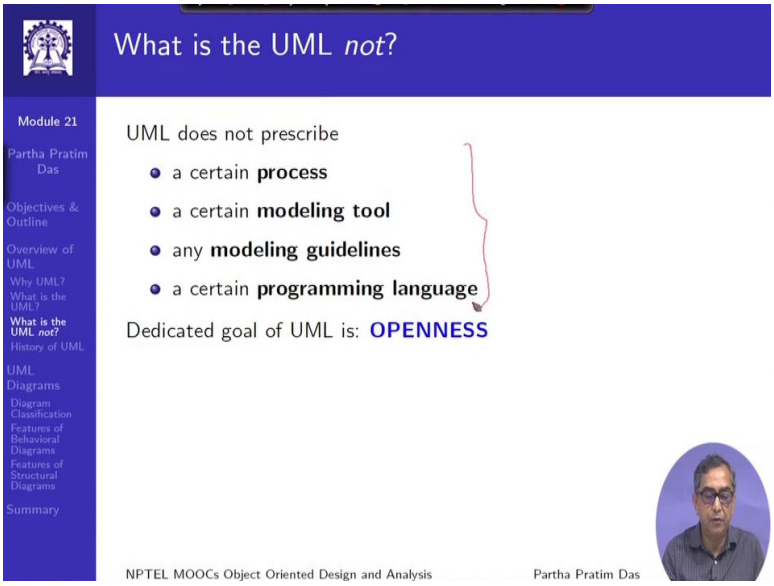
In addition in the supportive side it supports high level development concepts will often find UML talking about terms and items which can be directly represented and immediately implemented in an standard OOP language like C++ some of those UML concepts a supported concepts will actually have given standard library implementation in some of the common OOP languages so it is this kind of collaborations patterns these are the some of the terms that I have mentioned here patterns and components would be very convenient to represent in UML.

Last but not the least is UML inculcate a set of best practices if you refer back to the very first module where we started discussing as to what at the challenges of constructing a software we address that there are different issues and maturity issues and so on, for which we still set the fact software is developed but all other kinds of engineering development is actually called construction whether there is a guarantee of success.

We had mentioned that all these other fields of engineering succeed so well are so reliable these days because of the years and years of practice we have been able to develop a set of norms set of industry standards and best practices which every civil engineering design or a mechanical machine design has to follow.

Similar practices still do not exist that much in software so UML can be used as a vehicle to encode this best practices because the UML is giving as a platform in which I can express a concept in a platform independent language, language independent even at times domain independent manner, so this is the basic essence of UML.

(Refer Slide Time: 12:45)

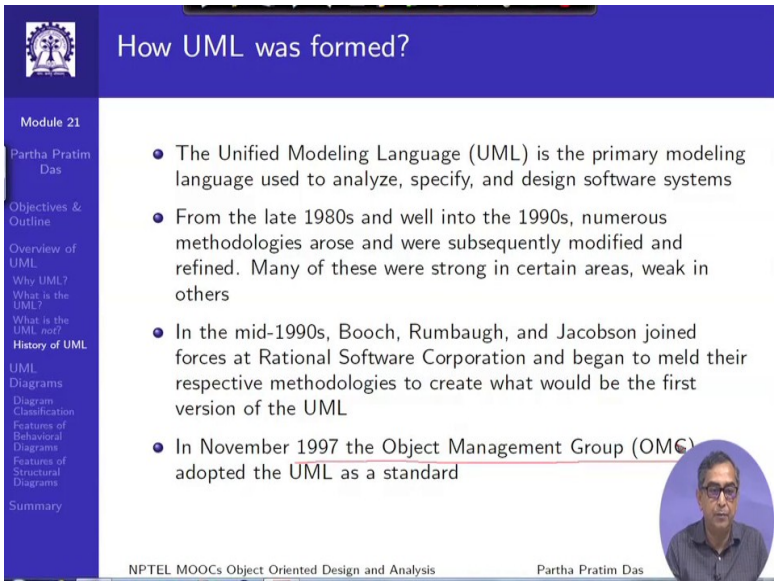


And while we discuss this it is also important to understand and highlight that UML is a - is characterized by Openness, what does that mean that UML does not say that you have to use a specific modelling tool or you have to use a specific language it does no - it would not say that you must use C++ or you must use Java or you must use python.

It will not tell you that this is you should use waterfall process of development or agile process of development or you should use industrial development it will not make any prescription it is not meant for prescribing any of this or any of the other actual detail practices that are involved.

It is open so that irrespective of what process you follow, what modelling language, what programming language, what platform, which hardware you are planning to use irrespective of all these parameters UML would be equally useful for you to model the system and to take it through the life cycle of development.

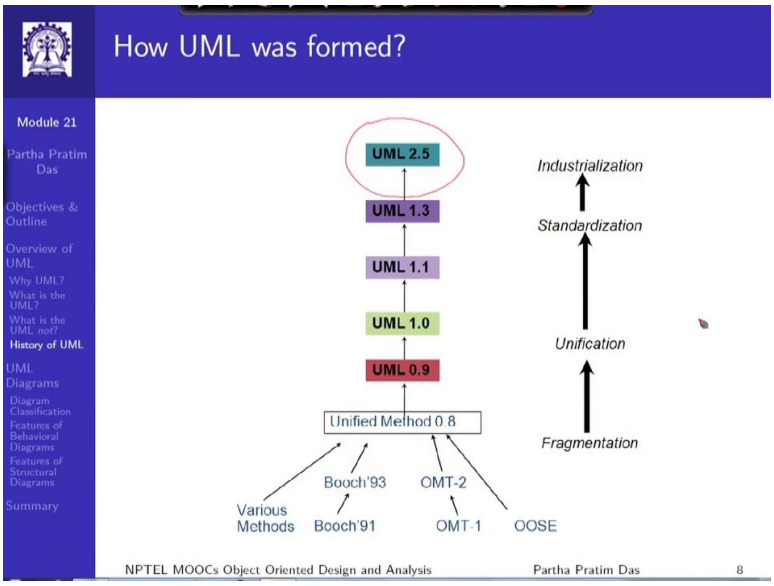
(Refer Slide Time: 13:55)



So a quick bit in terms of how UML got formed I really do not have the time to go over the details history but that history is indeed very interesting, the concept of being able to model system in a formal way started quite long back and in 80s we saw a lot of activities and which went into at least the first part of the 90s and it saw a proliferation of several different ways several different languages and protocols to model systems, till majority of the proponents of various models Booch, Rumbaugh, Jacobson and so on.

You - you are already familiar with Booch's name we are following his book very thoroughly they all collaborative join hands and UML was formally that way created with the formation of the Object Management Group there is a website on OMG I would suggest everybody that you go to OMG site and you will find a whole lot of very useful information not only on UML but in general on Object Oriented Analysis and Design, and especially you will find information which is oriented towards a current practices.

(Refer Slide Time: 15:21)

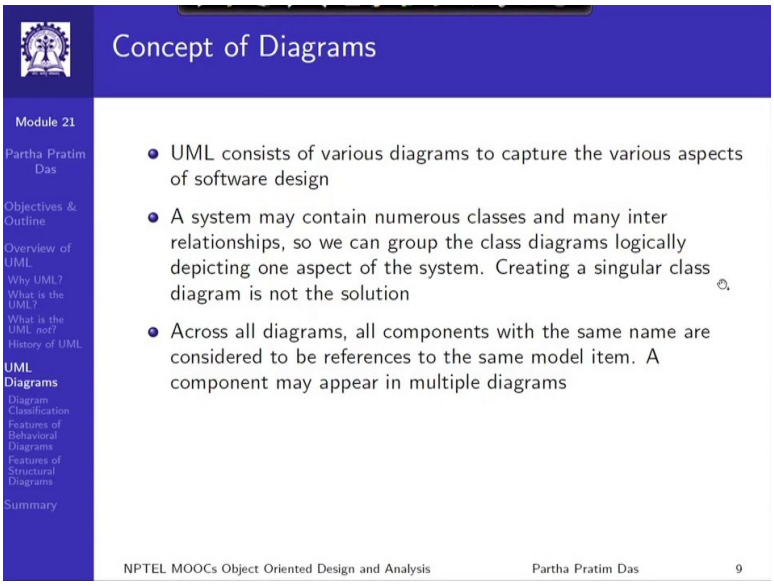


So in this background this kind of late 90s that UML started existing in its current flavor so this is - this is kind of the era of fragmented development then people has joined hands and stay put all this together and the U in UML that is in Unified in UML is actually the unification of all these methods you know some people tend to thank that it is called unified because it can be used over multiple phases of SDLC are used to express variety of concepts.

Those are also true but in primarily got its name by the unification of methodologies that it represented in the late 90s. And then over this whole around 25 odd years nearly 30 years I am sorry its nearly 20 years it has developed and OMG managers and the versions UML from one to the other and this UML 2.5 is a current version that is existing in the standard. So while we discuss UML, we will over discuss some of the starting from the early concept.

So not we will often restrict ourselves to the 1.X kind of standard because 2.5 has lot of advanced concepts which we may not have time for but we will take glimpses of 2.5 from time to time as it needs.

(Refer Slide Time: 16:54)



As I said the UML is graphical so what it has the core of UML is collection of diagrams so it consists of various diagrams that captured various aspects of software design, so the idea is simple that a software system or a system in general we have seen there several different complexities and there are several different ways that we can look at the system we have discussed these things at length in the earlier modules.

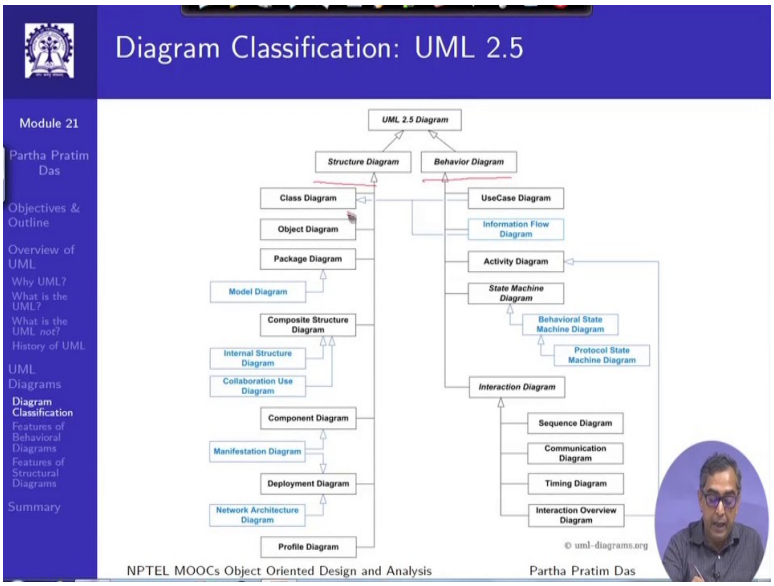
So the UML takes a very simple approach to address this, that it has identified what are the major aspects of that you need to know that you need to decide on for a complex system and for each one of them there is a category of diagrams and that diagram has specific primitives which help you actually capture as well as analyze the aspects of that.

So for example you will have a bunch of class diagrams necessarily for - we have already seen that is not easy to capture all classes in to one set so a system will have multiple classes, multiple kinds of relationships so the class diagram can be built up over this collection into a smaller to bigger set to give you a very unique UML description.

And it has UML supports a strongly support a name unification which means that across different diagrams and across different modules of the design the same name can be used and therefore we will see that a name that is used in use case to represent a use case will be used in the class diagram to represent a method the corresponding method, a an actor in a use case will

be become a class in the class diagram representing a particular group of individuals who is taking that action so that's the - that's the broad basic point.

(Refer Slide Time: 19:09)



And based on that this is just to give you an overview glimpse of the whole thing so you can see that, the route is the UML 2.5 diagrams the collection of all diagrams and then you have two types of diagrams the Structure diagram and the Behavior diagram.

And as you must have noticed by now is whenever we draw this kind of an arrow let me draw it clearly again that's an open triangle with this then which means a is a or generalization specialization relationship so to describe the UML diagrams we are taking care of the same notation, this notation is a part of the UML class diagram and some of the other UML diagrams so we say structure diagram is UML 2.5 diagram this arrow.

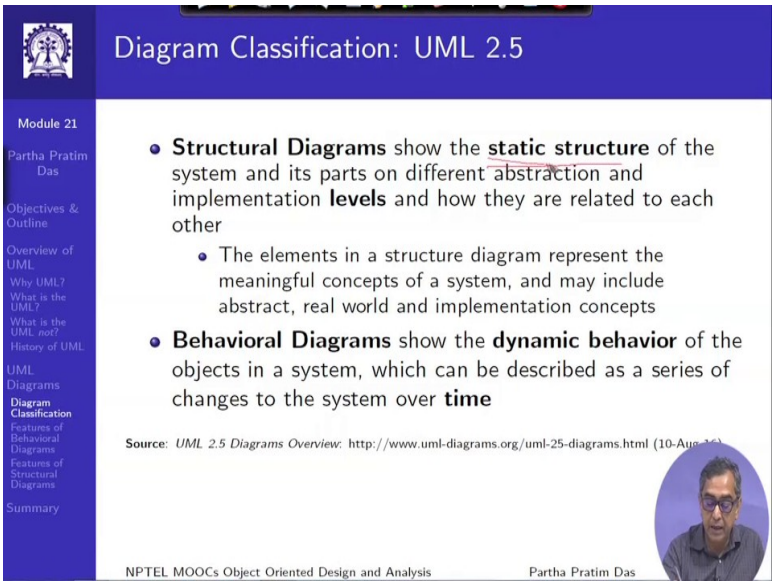
The behavior diagram is a UML 2.5 diagram this arrow and so on, so basically here in this slide not only see a collection of these diagrams but you see kind of there specialization generalization hierarchy so we know that these are the two basic types of diagrams and then we have a variety of different class structure diagrams like this, their variety of different behavioral diagrams like this and so on.

In reading this chart you have to remember a basic color code the diagrams that are written in black are part of the standard so Class diagram or Use case diagram are part of the UML 2.5 standard, but the diagrams that are written in blue are the level of proposals they are being discussed and they have not yet made to the standard some of them may become a part of the standard UML in future some of them may get dropped also and there are several inter-relationships between them.

For example, if you are talking about a composite structure diagram then you are saying that there are two specific sub types of it in terms of the internal structure and the collaboration use diagram so these are at the level of proposal so what we will do this I am presenting you the whole picture to just give you the current truth of what the UML 2.5 looks like.

But we will concentrate primarily on the diagrams in the standard and there too we will just take up a subset because it may be otherwise quite a lot of time taking to complete all of the diagrams we will just take a measure ones and discuss this.

(Refer Slide Time: 22:03)



So first let me say a few words about the basic tool sub classes of the UML diagrams, one is called Structural diagrams and the structural diagram show the static structure of the system so we have had we have been seeing this notion of static and dynamic description all through I mean - I would refer you back to our discussions on objects and their nature we observed that

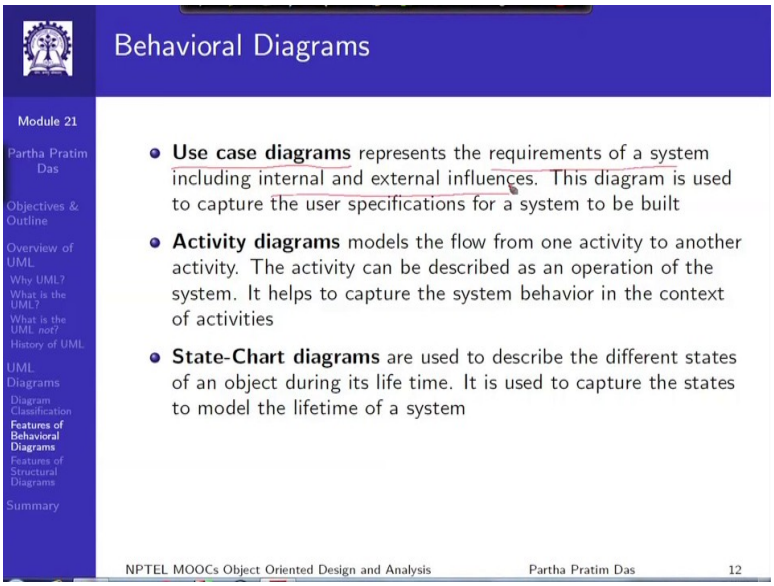
objects have an identity but besides that they have attributes and they have properties, attributes are the static structure and properties are the dynamic behaviour.

So similarly the diagrams here are also in a way static structural diagrams give us the static structure, that is how different parts of the abstraction at different implementation level are related to each other that, but this is static which means that when the system is actually built up deployed and works these are not going to change over time these are - this remind kind of fixed.

The other side behavioral diagram which talks about the dynamic behavior that is how does the different objects the behavior of different objects change with time what happens if I want to get into a system and I want to perform a login I have to perform a sign in kind of activity then what are the different actions that into happen in what sequence, in what temporal order, in with what dependencies to allow or dis allow me to sign into the system is a part of the behavioral or the dynamic behavior.

So if we cannot that some of the subset of the diagrams we will talk about structural properties and subset we'll talk about behavioral properties.

(Refer Slide Time: 24:11)



So let us start just take a very quick look in terms of this so the - the way of organized is for all the diagrams that exist in the standard we are first talked about the behavioral diagrams and then

we talk about the structural diagrams. Here I am just trying to make a very brief overview to you about this each and every diagrams we will take up many of them at length and we will have separate modules discussing regard at - at times multiple modules discussing them how to use that and wherever applicable.

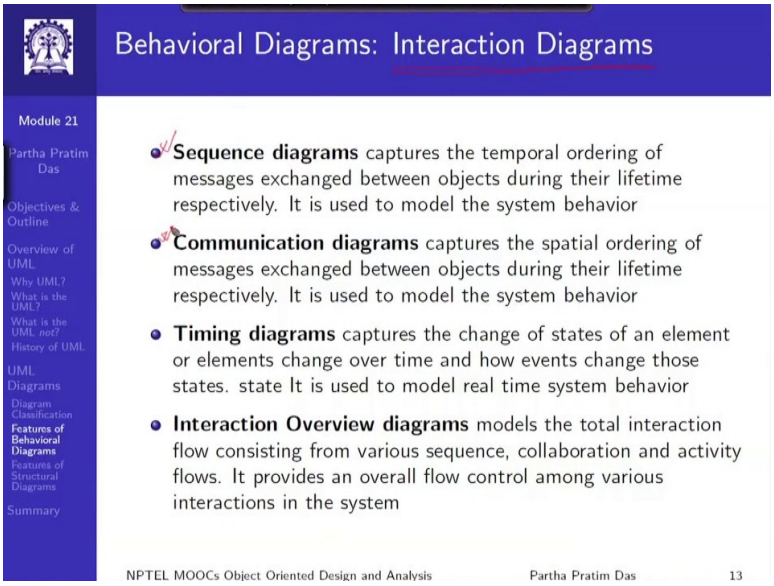
We will take the case of the Leave Management System and illustrate how the corresponding diagrams can be met up. So the most important possibly to start with and the first fundamental diagram that UML behavioral has to start is Use case diagram where you represent the requirements of a system including the internal and external influences what is happening around the system there many gross ways.

Then Activity diagram talks about what are the different activities that internally happens in the system what are the operations of the system. State-chart diagram basically borrows its mean from the state transition machine or finite-state automaton are finite state machine so it means that I can think of that in the system there are several situations where there are a finite set of possibilities and the system keeps on moving from one possibility to the other and that is what we say is the finite state machine.

State-chart basically is an enhancement of the finite state machine which along with telling me what are the different states that a system can be in for example in the sign on system I could be I may have signed up and I may not have signed up if I have signed up then I'm signed in or I am not signed in if I have signed in then I may have a state of coming in next time without having to sign in again and so on.

So these are the different states and state-chart that allows me to play around with these different state information.

(Refer Slide Time: 26:17)

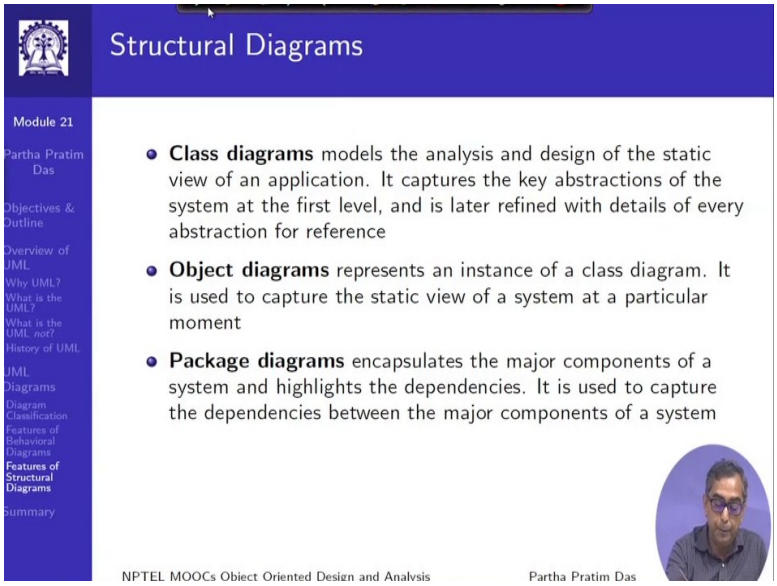


The next set of behavioral diagrams are all clubbed in to one sub category known as interaction diagram because the define different kinds of interaction behaviour other the Sequence diagram is most frequently used which will tell you what is the order of events that happened. The Communication diagram will tell us that what is the order in which objects exchange messages I would just again remind you that our vehicle finally is a client server model.

So when you do all these modelling finally we would like to take it to some messages being sent from one object to the other so the sequence diagram tells us the temporal order in which the messages are sent, and the communication diagram tell us the spatial order or the object to object order of how messages are sent. Timing diagram as the name suggest is closely related to the timing information and Interaction Overview diagram basically tries to model that how interaction flow be from various sequences collaborations and activity flows.

So it is kind of sets little bit at a higher level compare to the sequence, communication and timing diagrams which are particularly object based, interaction overview diagrams is often at a level of modules.

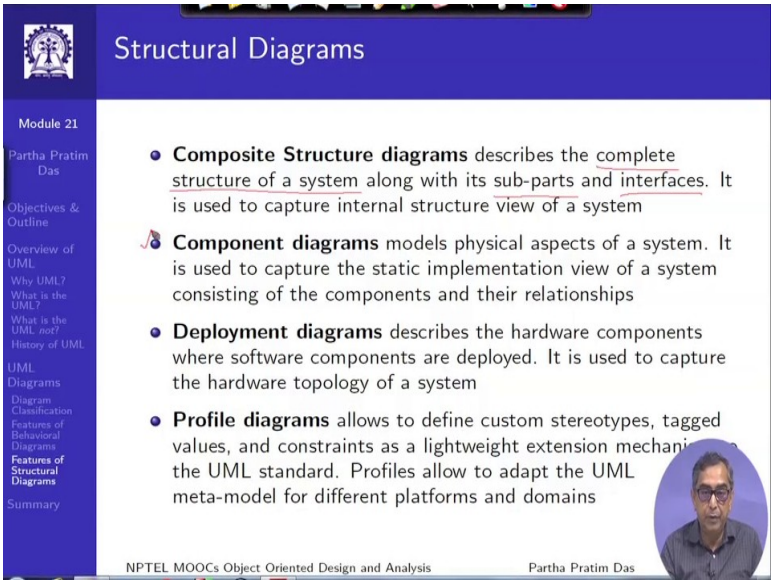
(Refer Slide Time: 27:41)



Coming to the structural diagrams the main structural diagram of course are the Class diagrams and Object diagrams, class diagrams and as we go into detailing them after a couple of modules we will find that you have already come across much of the notation or the diagram convention of the class diagrams and the objects in terms of our process of identifying classes and our defining objects.

Certainly the other structural diagrams is a Package which basically encapsulates the different components and puts them as a part of a bigger module.

(Refer Slide Time: 28:19)



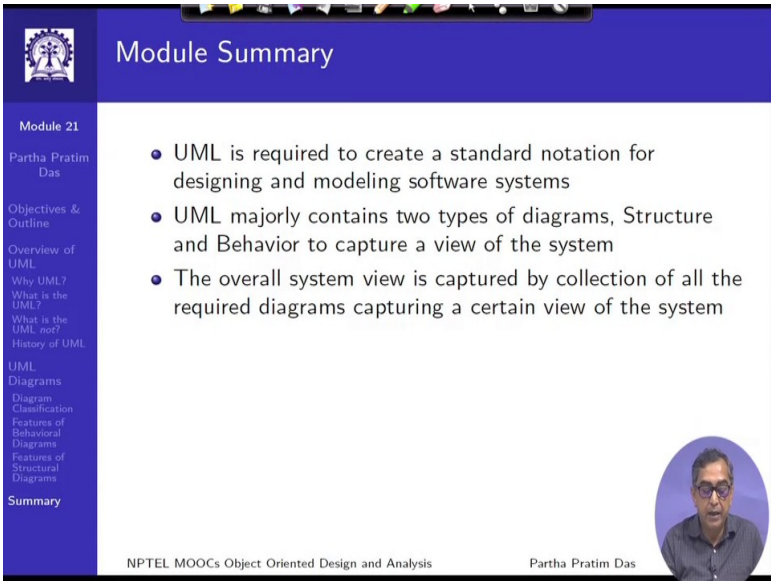
Further structural diagrams include Composite structure diagram which gives a complete structural view of the system with all its sub-parts and interfaces and so on and the complementary to that is the Component diagram which primarily show you what are the physical aspects of the system so the composite structure if it talks about various interfacing relationship between components the component diagram specifically talked about what components and what static implementation view of this components that you need to describe.

And though we are talking about analysis and design we will find that several of this diagrams actually go beyond that boundary and Deployment diagram is a direct example of that where you not only just talk about the early phase design but you talk about the late phase implementation where you decide, okay.

We finally this system will be deployed over five servers and it has 17 modules you actually decide which module will run in what server whether some module will run in multiple servers whether one module - one server would accommodate multiple modules and so on, that whole description of hardware component and software component mapping is given in terms of the deployment diagram.

Finally, Profile diagram is an auxiliary one which talks about what kind of profile, for example if you are implementing on Java what kind of would you like to use J2EE or you will be using J2ME, if you are using J2ME would you use an Android profile or you will use a IOS profile, so these kind of information's are often shared in terms of the profile diagrams.

(Refer Slide Time: 30:13)



So to summarize we have understood that UML is required to create a standard notation for designing and modelling software systems to liberate ourselves from the vagueness and imprecision of natural language and also not to get trapped in the intricate analytical and expressive power of the programming languages. UML majorly contains two types of diagrams structural and behavioral diagrams.

And we have seen variety of structural and behavioral diagrams overview and the overall system is will be captured in terms of all these different diagrams, so with this overview you will in the next module we will take a brief look into the basic design process design and implementation process of the software to realize how we can start using these diagrams and then we will move into the discussion on specific diagrams.

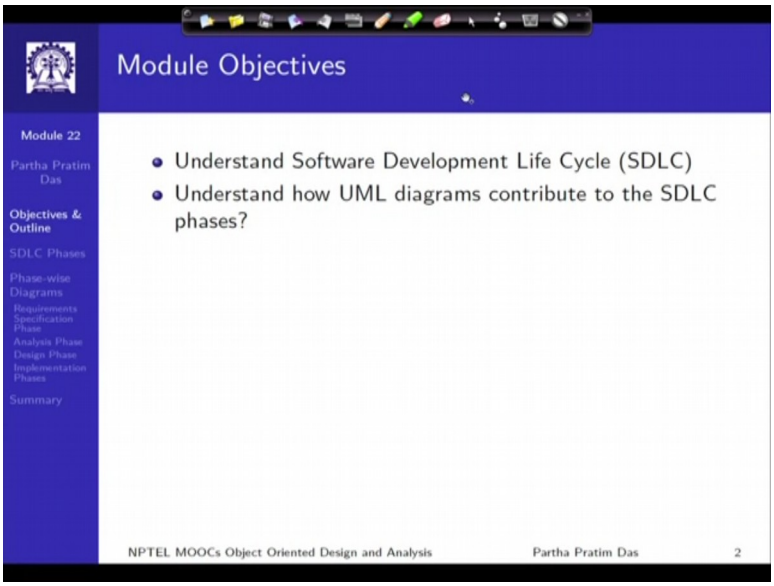
That Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 34
SDLC Phases and UML Diagrams

Welcome to module 22 of object-oriented analysis and design. In the last module, we discussed and took a brief overview of uml, unified modeling language. We saw that it comprised a collection of diagrams which can be sub classified as structural or the static behavior, static properties or the behavioral diagrams which are dynamic properties and in each of structural and behavioral diagrams there are variety of specific diagrams that are involved.

Now the question certainly is that we want to use uml because we want to bring in better clarity and better expression in the design process. But with so many diagrams around, certainly it is a lot of confusion, fresh confusion as to which diagram should be used when and in what context and so on. So, in this module we will try to answer that question.

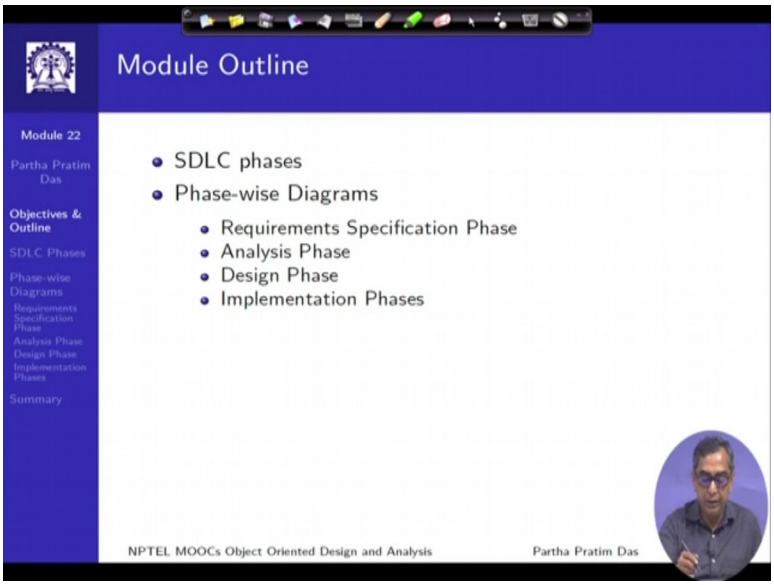
(Refer Slide Time: 01:39)



And to answer that question, we take a look brief look into the software development lifecycle. The whole process of system, analysis and design is a part of n is targeted for the finally the system being implemented as a working software. So, there is a moderately agreed process for software development and it is called a life cycle because starting from the initial development till the end when you finish we see that actually we are at the beginning of the lifecycle again.

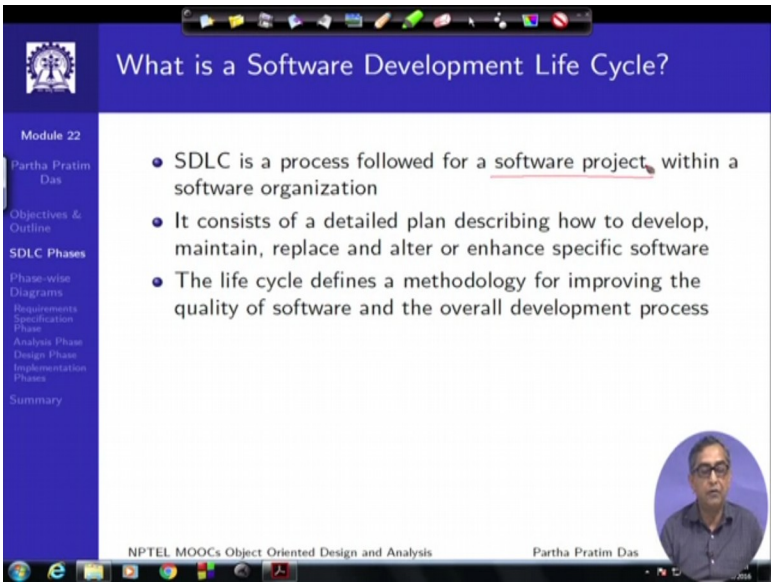
So, we would like to take a brief look into this, some of you, many of you may have had experience with SDLC if you had studied software engineering in some context and once we have taken a look into SDLC, we would try to associate different of the uml diagrams at different phases of the SDLC. So that is the basic objective.

(Refer Slide Time: 02:52)



So, these are this is the outline which will be available to the left of every slide

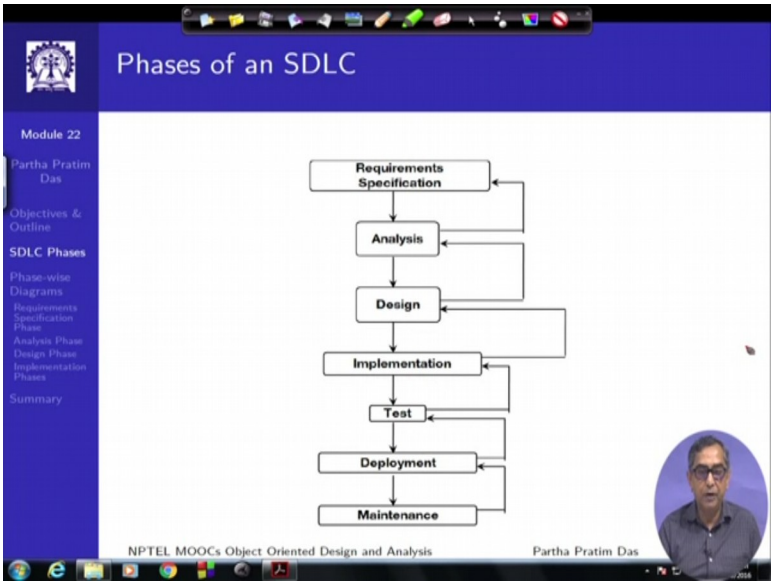
(Refer Slide Time: 02:57)



Now what is software development lifecycle? It is a process followed by a software project or a software organization even for in a software product and it consists of detailed plan describing how to develop, maintain, replace and alter, enhance specific software. So, it's everything that goes on in the time that is software solution is conceived to the time it is deployed to the time it is matured to the time

it is different version created and tot the time that is finally goes out of use is the in under the purview of the software development lifecycle.

(Refer Slide Time: 03:41)



So, it is typically this is not a standard prescription but it is typically software development Lifecyle is divided into a couple of phases. So quickly on these phases, this phase is a requirement specification where actually I should have had a arrow here, where the request or requirement for developing a software has come from the customer from the user. So, who are providing the domain information and we have been discussing a lot about that.

And here the requirement is that you analyze what the customer is trying to say or trying to specify in terms of the system through going through documents, interactions, videos, making notes, images and so on. And generate a very early specification of the system which can be further analyzed for looking into the depth of the what the system involves. So, when you start analyzing it, I would just re relate you I will just refer you to an activity that we did couple of modules back possibly it was module 20 where we.

We are trying to identify the classes relating to leave in the leave management system and we initially had all leaves being a specialization of a general abstract leave concept and then we did an exercise of finding out what are the different properties, the different leaves satisfy and accordingly we tried started grouping leaves as clubbable, certifiable and so on so forth. And this exercise that we did is necessarily the analysis stage of an SDLC behavior.

So, where you go through all these, refine your captured specification and get into better and better details. Now in this process so this is the this is a forward arrow and in this process, you may require, you may find out that the analysis tells you that well some information is missing, some information some more information is required from the domain, from the customer and so on. If you require such then you need to go back to the earlier phase earlier stage.

And that is the reason you have this reverse arrow and you go back and recapture some additional information in the specification. So, the it it now onwards it goes like that so once I have the analysis output then I am working on the design where I want to more specifically now look at how these concepts can be converted or directly mapped into different software primitives different software details and that too I can do in multiple phases till I am able to generate an output int of the design which will be very detailed design of every system, subsystem module.

If I talk in terms of say my vehicle being C++ has an implementation then out at a output here I am talking about C++ class I am talking about methods, I am talking about specific libraries and so on. So, the design must specify all of this so that the coding of that can start. Now here again you will see that there is a iterative return arrow which is taking me back to analysis for it because the input of design which is the output of analysis could have something which is not easy to design or which is not consistent.

And then I will need to go back and do the analysis again or do some more analysis and it is quite possible that I in some cases at least I need to go back to the requirement specification itself. So, you will see that this whole whole diagram is is goes something like this, it keeps on doing something and coming back. So, this confirms to the basic principle of iterative refinement that we have had talked about. Now the output of design goes into the implementation phase, this is basically the what we say is coding

and programming where you are using your chosen object-oriented programming language to actually write the programs which realize the design that you have done, again it might mean that you will need to come back to the design at some stage or it might need to go back up totally up to the requirement specification phase. Once the implementation is over, you put it to test which is a very big issue in the in software implementation any of you who have done little bit of study on software engineering will understand this is a very critical stage

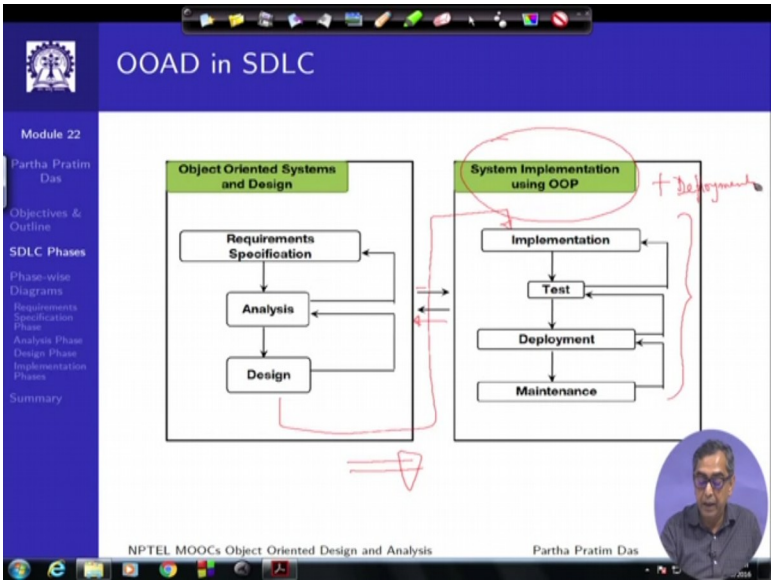
where you are trying to see that whether 2 main things, whether you have through this process been able to build the correct system that is whether what you have built is what you are required to build given the specification of the customer and the second you try to answer whether you have built the system correctly that is what you have understood in the requirement specification and brought it through all these stages of implementation in the process, you have not made any mistake but you have been able to build it correctly.

So, test primarily is a very extensive process engages about 70 percent of the total effort of software development and tries to answer whether you have built the correct system and whether you have built the system correctly. Again, test anything that fails in the test we say is a bug, you need to go back and make changes in the implementation. Once a system has been tested then the resultant system goes to deployment that is you take it to the customer and put it to customer place and so on.

Let the customer use that and try it out and if it fails then it comes back to test, reimplementation, redesign and so on so forth. And finally, if the deployment succeeds then the software goes into the maintenance phase where you are using it but you may still get bugs at times or it is possible that at a late I mean after using it for a certain period of time, 1 year, 2 years, whatever you need a whole set of new functionalities, you need the software on a completely new platform, you need a different kind of performance and so on.

So, you request to upgrade the software and again from this we will, you go back to deployment or you may directly go back to the requirement specification and say well I have a working software, I have been using it for 2 years now. But now I want such and such and such features included and I start all over again. So, this whole set of phases basically talks about what happens to the if if a software can be considered to be like a human being then whole set of phases will tell you what happens to the life of a software and therefore it is called the software development lifecycle.

(Refer Slide Time: 10:52)

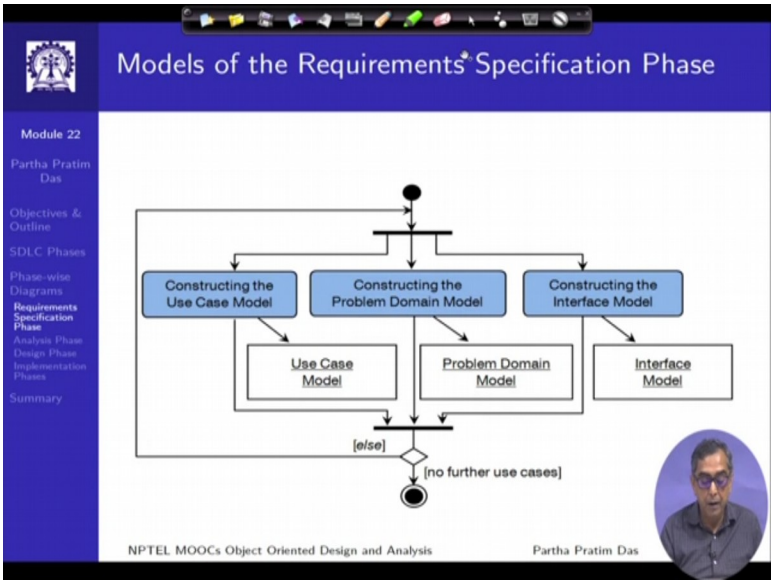


Now if you look into these stages then we can group them into 2 broad parts, the front part which is from requirements to design which I am showing here where the OOAD, what we are studying has a very critical role to play. So, we will see that a whole lot of uml diagrams will be related in this stage. Whereas there is a after the design has been done, then basically after the design has been done, then you go to the implementation as we have seen and then this phase is primarily around the object-oriented programming language.

So, it needs more of in addition to OOAD, it needs a knowledge and understanding of the programming systems, the software different software components, the hardware systems and so on. And if there are any issues in the implementation then you will come back to the design again. So, in this part 2, uml all the uml diagrams that we do in the in these 3 parts will go as an input to the system implementation part and in addition I may have a lot of or at least a few of diagram, uml diagrams which will take part in the details of this system.

So that's the basic role of OOAD in the SDLC and we will keep that in mind to identify the different diagrams.

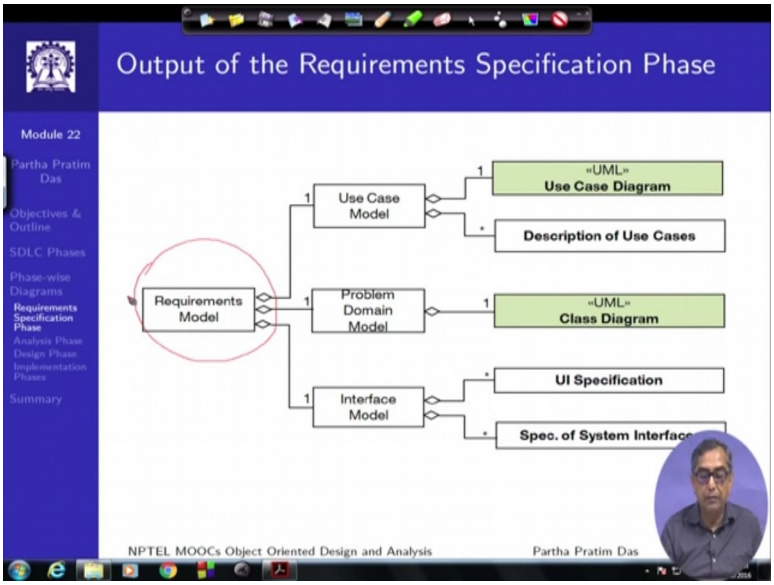
(Refer Slide Time: 12:19)



So, starting from the first phase which is the requirements phase that it typically uses 3 different kinds of models or diagrams. So first you need to construct the use case model. So, you use the use case diagram, you need to construct the problem domain model which is actually trying to capture what are the basic properties, key abstractions, relationships of the problem domain are. So, use problem domain models and you need to have an interface.

So, this interface I mean the moment we say interface it it kind of a suggests as a graphical user interface GUI but it may not necessarily be that. GUI is also included but it could be all other kinds of interfaces including keyboard based interface different kinds of voice interface, auditory interface and even program interfaces which come in the interface model.

(Refer Slide Time: 13:22)

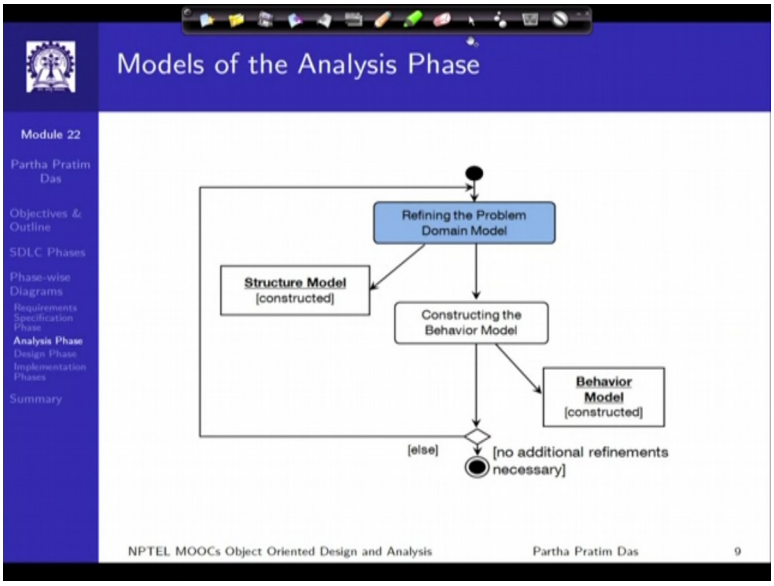


So, when we do the requirements phase, requirement specification phase, then we expect that we will come out with a set of models which as a whole we will refer to as requirements model and this notation we have used it to show aggregation at at a certain stage earlier this show that basically this is a union of or comprise the use case model, the problem domain model and the interface model. So that same uml notation is used to describe uml here.

So, the requirement models have these 3 types of models as we have seen and in terms of the uml diagram, this is what the use case model will do. So, when we say we need to identify the use case model, how the system can be used by whom to do what, we capture that in the use case diagram. So that will have to be the output of the requirement specification. Similarly, class diagram will identify the basic objects and classes and is a part of the problem domain diagram that problem domain model that we need to create.

In terms of interface model yet uml does not have very strong diagrams, so UI specification and system interfaces and so on are still often done in a little bit unstructured way, little bit with the use of you know natural language and hand drawn drawings and so on. But then in these 2, the use case and class will be the core components of what you expect to generate after the requirement specification

(Refer Slide Time: 15:08)

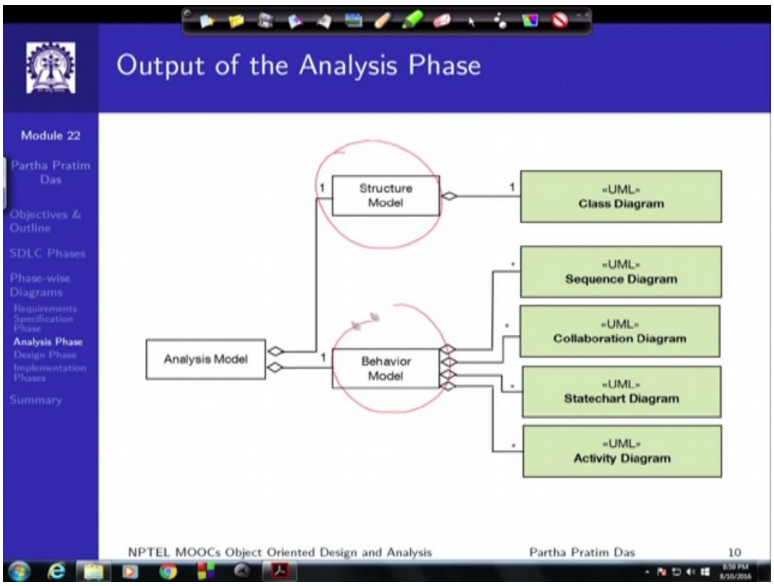


So naturally these will now go into the analysis phase and in the analysis phase the main structure is analysis and to define the problem domain model again remind yourself of our exercise of refining the way we had modeled the hierarchy of leaves. So that is the activity that you are doing here. Naturally what you expect to generate our structure models which are constructed and you start constructing the

behavioral model.

In terms of the requirement phase output possibly you can still you can still have certain behavioral description in the class but is primarily a structural model but when you are completing analysis, you are expecting more and more of the beh behavior to be extracted and behavioral models to be explicitly constructed. So that is your dealing with the uml diagrams in the analysis phase and as a output of that, this is what you expect to generate, a set of structure models and a set of behavioral models.

(Refer Slide Time: 16:13)



The structure model is again class. So, you can note you saw this in the requirements phase output also. So, you again say this it does not mean that the fact that you are is using the same diagram does not mean that it is necessarily exactly the same diagram. It is a class diagram that you are using but the set of diagrams you did at the end of requirement specification through analysis that has got refined and you may have a differently organized class diagram, you may have a lot more of details in the class diagram.

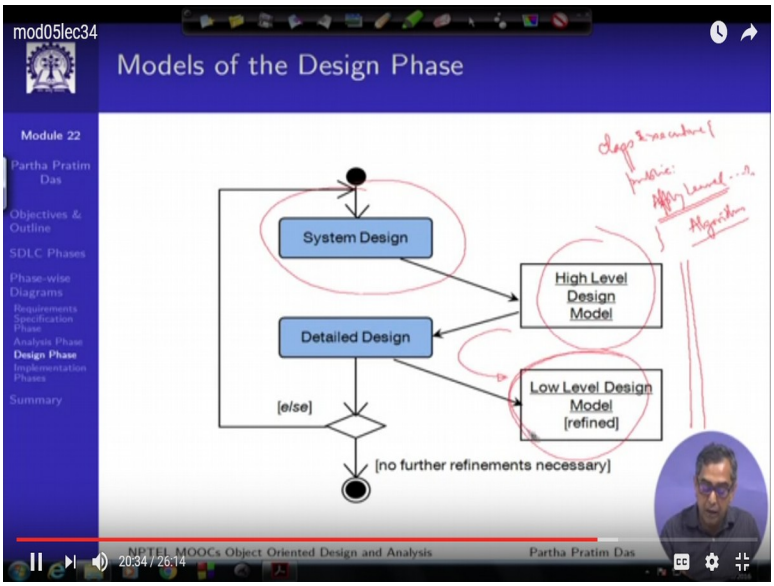
Now again the example is at the output of requirements phase you just have one leave class and all those leave are considered to be specialization of that and at the end of the analysis phase, you have the same class diagram, now refine in terms of a complex hybrid hierarchy. The behavioral model includes these 4 which I briefly touched about in terms of sequence diagram. The communication or collaboration diagram, collaboration diagram I have,

The term is mentioned here because I I mentioned that will switch will keep on switching between the

stand the current standard and the earlier one-point x standard. So, one-point x standard used to whatever one-point x standard used to call as a collaboration diagram from the 2-point x, this is now called the communication diagram. So, you can this is same as the communication diagram, not a different diagram. the state chart diagram to show the different states and the activities.

So, these are the diagrams which have to get created through the analysis process and each one of them will capture a different aspect of the system.

(Refer Slide Time: 18:02)



Naturally the output of the analysis phase go into the design where the main activity is the system design that is now it is expected that you have been able to understand the specifications, understand the problem domain, the requirements very thoroughly and you have been able to represent them in terms of this whole lot of different uml diagrams and now keeping all these in the in your mind, you are now trying to design how actually we will go about organizing the code,

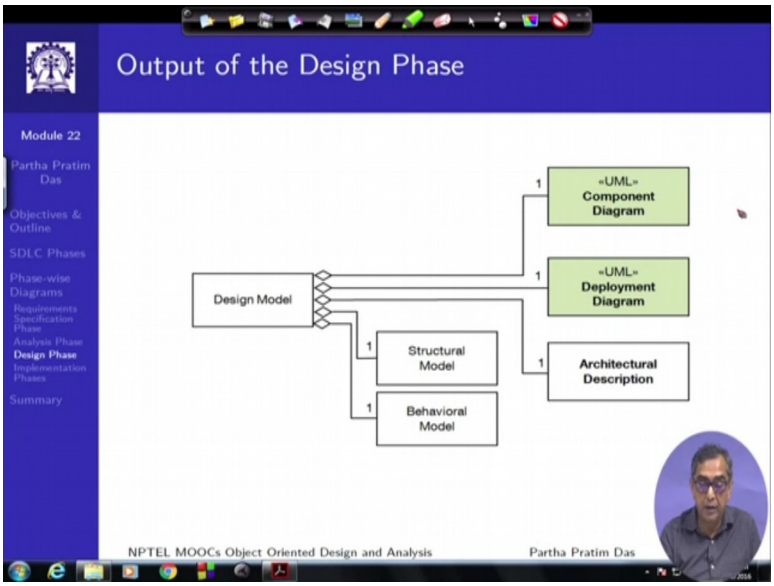
what should be the different code component, what should be the actual classes that you write in the oop and so on that is what you mentioned about system design and as you can see that its typically done in 2 stages. The first stage is you do a high level design model. For example, if if in the leave management system, we decide that we will have we have after knowledge we know that we have class like employee, executive, lead and manager.

Then at this stage of the system design stage, for if C++ is my vehicle since I have a class executive and so I I write it down and I say that this will have different methods like apply leave kind of and so

on. So that is I I work out a basic high-level design. Then I want to again take it through another design phase which is called the detailed design where I will refine on this high-level design. So here possibly I was happy saying that I have apply leave, here I would like to know the algorithm that will be used in applying for leave.

I will glad to know that do I have some idea about at least what are the data members that I should have. These data members are nothing but attributes of the class but if I say that the attribute of the start date has a for the leave, it has a or rather we are talking about executive so it has a joining date of the executive, it has the age of the executive and so on. We really need to detail out as to what kind of type the every data member should be, what should be the size and so on. And those are the tasks at the low level design phase.

(Refer Slide Time: 20:58)



So in in short you will find these terms very widely used. this is called HLD high level design and this is called LLD and this whole process of design will split out again a set of different uml diagrams. So now please keep this in mind that is go over from one phase to the other, all outputs of the earlier phases still exist and are still valid unless they have been rejected and removed. So, we already have all those use case, class diagram, different behavioral diagrams of state chart sequence

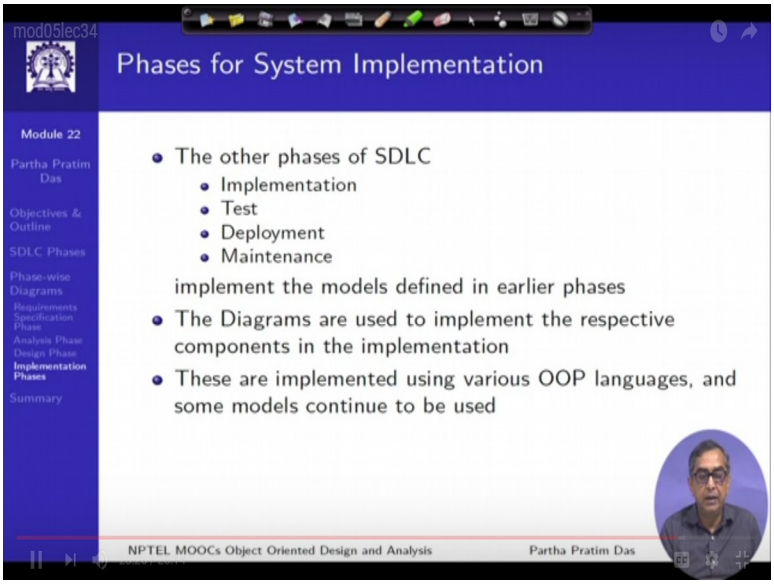
and all that but now in addition we have been able to create these kinds of diagrams like component diagram. So component diagram will tell me saying leave management system it will tell me that I have an employee component which implements variety of employee related classes, employee executive, lead, I have a leave component which certainly talks about different kinds of leave classes, I have a

joining component which talks about how does not employee start existing in the system and so on.

So, I talk about all those and I also outline the basic diagrams in terms of what is going to be my target system. What kind of hardware do I have and how do I lay out these components in the target. So, what is my deployment plan that we will come around with. Now along with that I may have other structural models, behavioral models that I have already got from the earlier phases and in addition there could be some more details, now as you can see that here I am calling this as architectural description.

If you could refer back to the last module, module 21 and look at the hierarchy of uml 2.5 diagrams you will find that in terms of deployment, people are often now talking about say network architecture diagram. So, in in in may be 5 years back we would have been just happy to talk about a general you know English like description or diagrams defining what is architecture of the system, how are different servers and front ends and network cables are connected. But now we may need to slowly work towards formalizing that in terms of better diagrams as well.

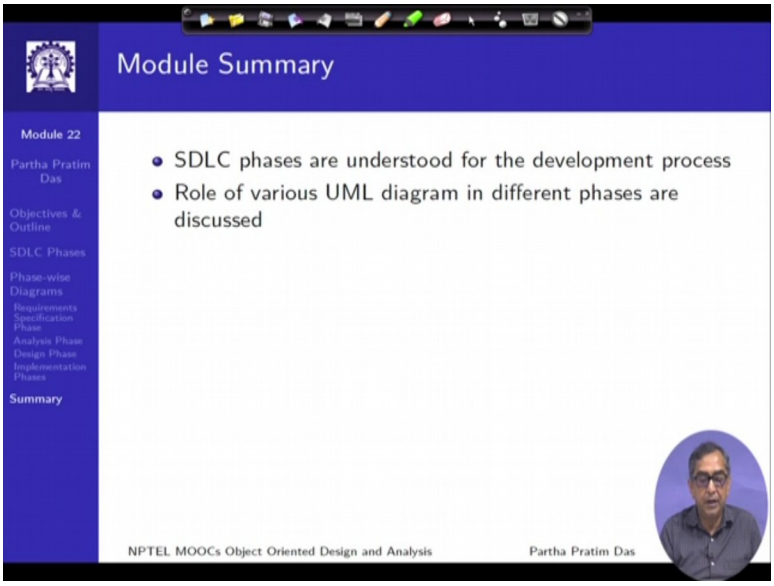
(Refer Slide Time: 23:18)



So, this is the kind of diagrams in the design phase and that is the those are the 3 stages, phases of SDLC requirements, analysis and design where uml's have a big role in terms of getting formulated. Then in the later phases of implementation, test, deployment, maintenance, their main role is that these diagrams are then used and the implementation is designed. So, in this other phases, it is typically the use of these models but certainly in that process also we might generate some more details in some of these models.

We may need to go back to the design or analysis phase or even requirement specification phase to refine the models and work through that.

(Refer Slide Time: 24:11)



So, to summarize in this module, we have tried to take a brief journey around the commonly accepted software development lifecycle process. I have talked about the major phases of this life cycle and how the uml diagrams will be used in different phases or will get generated in different phases. So, from the next module onwards when we specifically talk about the details of every diagram, we start learning about what their primitives are, how they should be formed and so on.

Then we will be able to relate to which particular stage of SDLC software lifecycle we exist. before I close, I would also like to put a disclaimer in terms of the SDLC because the kind of approach that I have shown or discussed in the SDLC here is the most classical and the flat approach, there are several different SDLC variations that exist today starting from different agile methods as print method, test driven methods and so on.

So even if you are actually using a different SDLC flow you will find that maybe the checks and balances of what phase you do, when we complete a phase and how do you repeat a phase or how do you refine a phase, those details will vary but you will find that whatever methodology you may be following, these stages of requirement specification analysis design all these kinds of activities would be there in the SDLC that you are following.

So, once you quickly identify that, you will be able to also relate to which particular uml diagram you

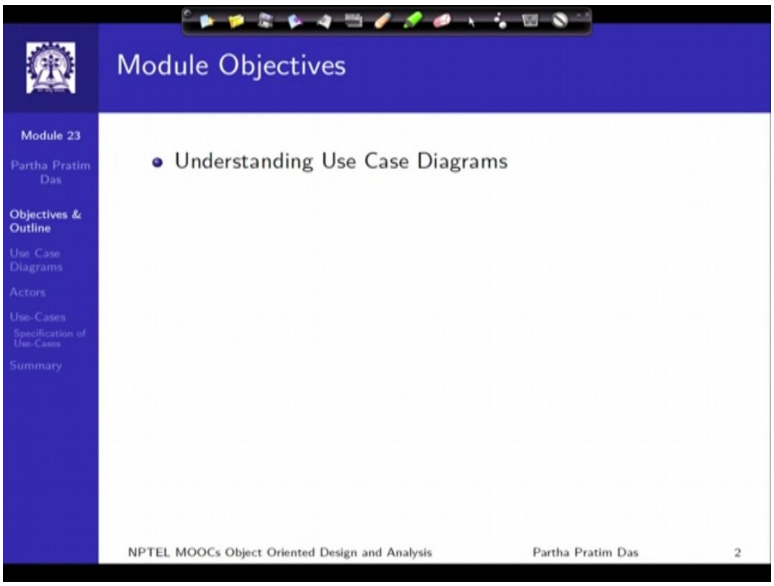
should be using for doing what kind of task at different stages.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 35
Use-Case Diagrams Part I

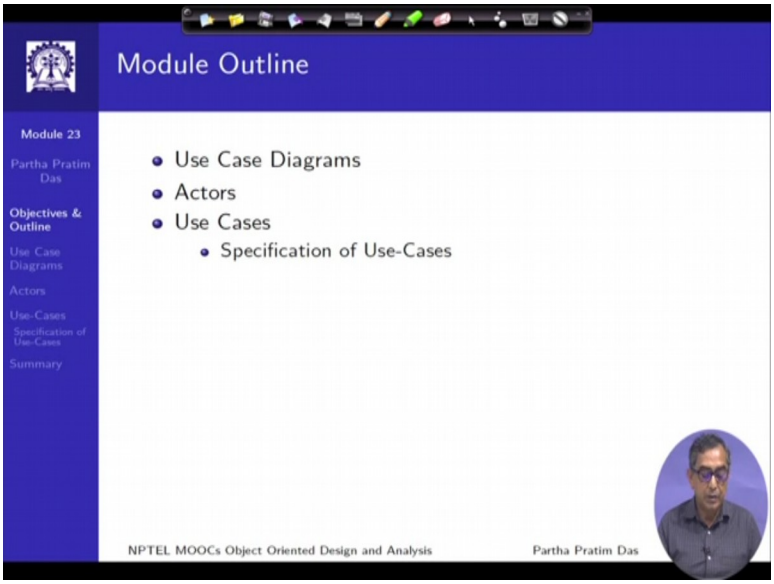
Welcome to module 23 of object oriented analysis and design. We have already in the last 2 modules, taken a look at the basic premises of uml, the unified modeling language. We have discussed very briefly though the stages of software development lifecycle and related both of them together.

(Refer Slide Time: 00:54)



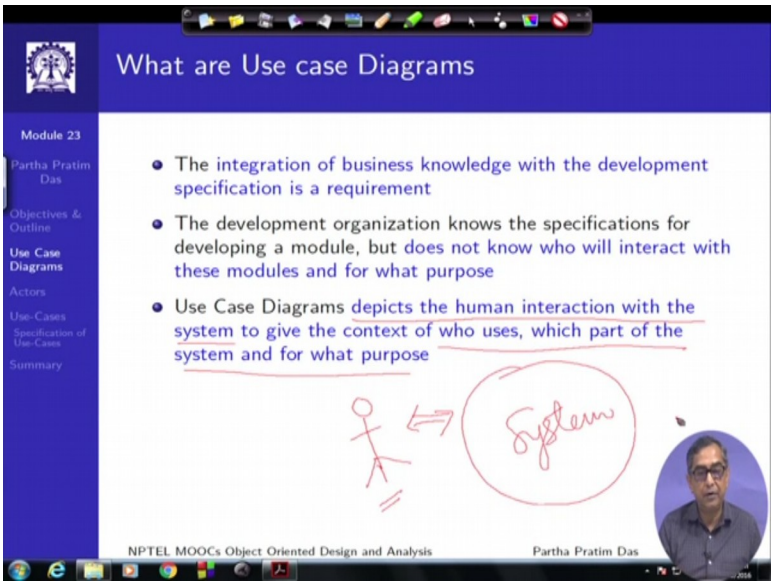
Now from this module onwards and this journey will continue for several modules to come. We will start discussing and explaining about different specific uml diagrams. So, in this module we start the discussion on use case diagram and we will continue that in the next module as well.

(Refer Slide Time: 01:20)



So, in terms of our discussion we will discuss use case diagrams and specifically actors and use cases, the outline will be available on your left always.

(Refer Slide Time: 01:29)



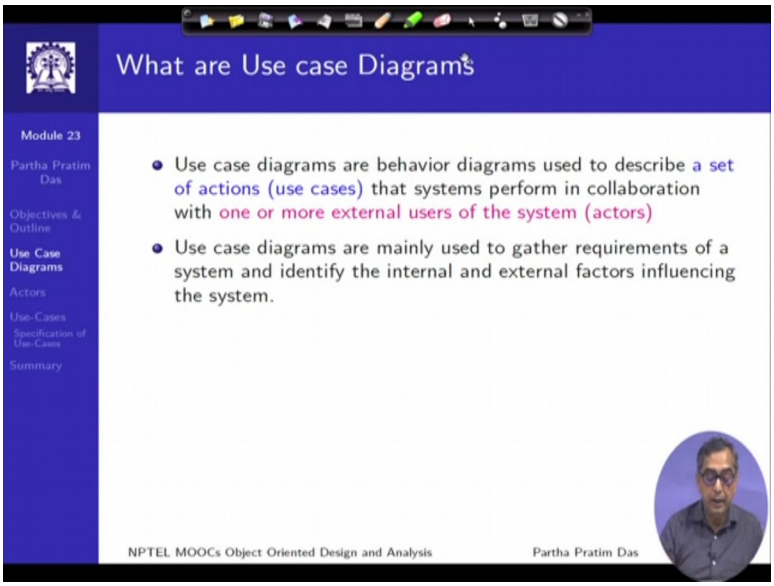
So just to remind what are the use cases? Is a integration of business knowledge with the development specification is a requirement? That is that is the basic starting point. So that is how we got the 2 pages on the leave management system where the organization decided to have a system of lms and they put it down, integrated the business knowledge which is how they how the employees report, how the employees apply for leave, what different kinds of leave they have and so on and created this requirement.

Now we still do not know who will interact with these modules and for what purpose. What user will

do with this is as still not clear, certainly these are not so well specified. In fact, the lms specification is actually not that bad because if you go back to the specification you will find the use of the word use cases. But usually you will not get that. So, the use case diagram depicts a human interaction with the system that is this is my system that am trying to build.

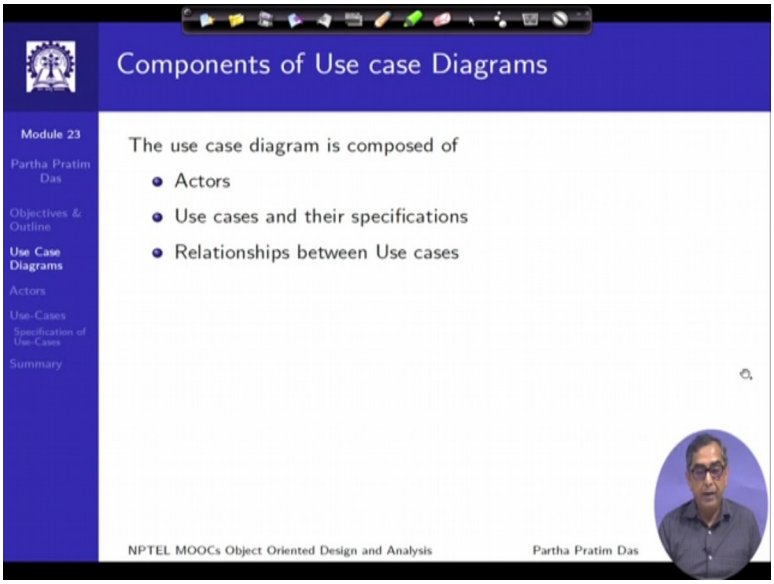
This is my system and finally why am i building the system because there is some human who wants to interact with the system, give input, take output and so on. And this is this human is not a not a single individual, this human could be a various different groups of users who would use and different parts of the system for different purposes. Certainly, in any reasonable system, no 2 users would be using the system exactly in the same way or may be at least there would be variety of groups which will use the system in multiple different ways and the different ways in which the system can be used is the main premise of the use case diagram.

(Refer Slide Time: 03:40)



So, it is a behavioral diagram as we have already said and it is used to define a set of actions, use case would mean it is a particular case of use or case of action that we are interested to look at and who takes that action, if an action who will take that action, an action will be taken by the actor. So, the use case diagrams are mainly used to gather requirements of the system and identify internal and external factors. Now before wasting more time let's get into the actual details.

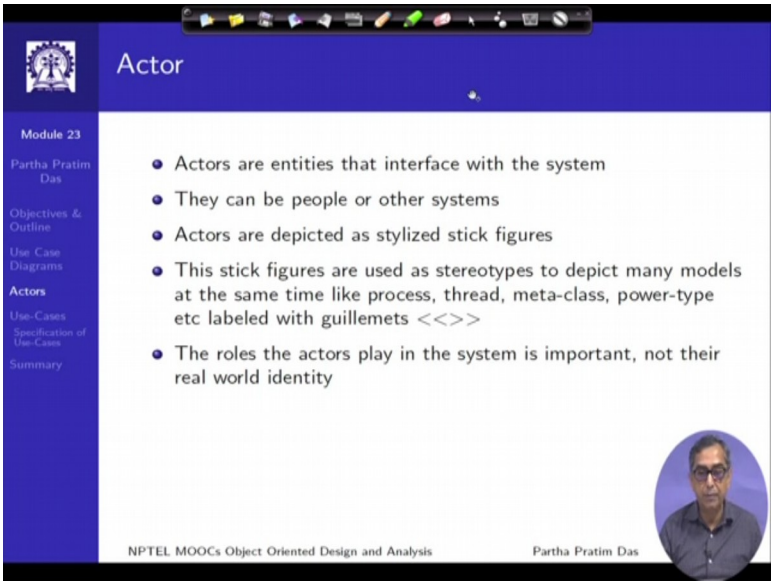
(Refer Slide Time: 04:21)



A use case diagram is composed of 3 major components, the actors, the use cases and the relationships. In simple terms, the actor is a user, in a simplistic term the actor is a user who wants to do something with the systems, use case is an action is an activity, or is a way that the user would like to use the system and finally relationship is no action that the user would want to execute would be all realizable alone by itself, different activities have different dependents.

If I want to take leave then it depends on what is my availability of leave, what is the validity constraints on the leave, what is the opinion of my reporting manager and so on. So, the relationship is most important to understand what how the actions will interact.

(Refer Slide Time: 05:25)

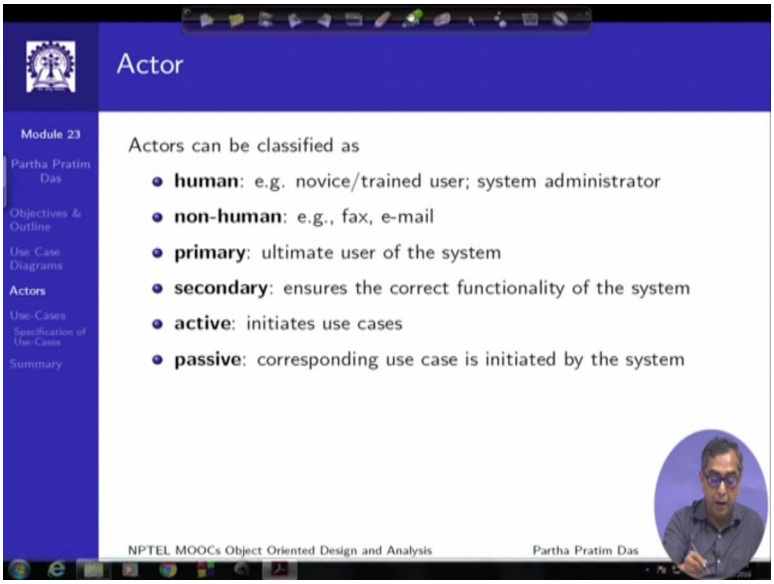


So, the uml diagrams gives various different notations and semantics for defining these components.

First the actor. As already explained the actors interface with the system. Actor though the name sounds that as if actor is a human being but uml considers actors who are people or human beings or actors who are other systems that is I can get a request from a human being sitting on the terminal working through a gui or I can get a input, I can get a request for service from some other system also. Actors could be non-human.

These are typically shown in terms of typically shown in terms of stick diagrams figures like this. Or they could be level by guillemets also I mean instead of the figure I could just put guillemet is this pair of symbols. We put a name within that and that means an actor.

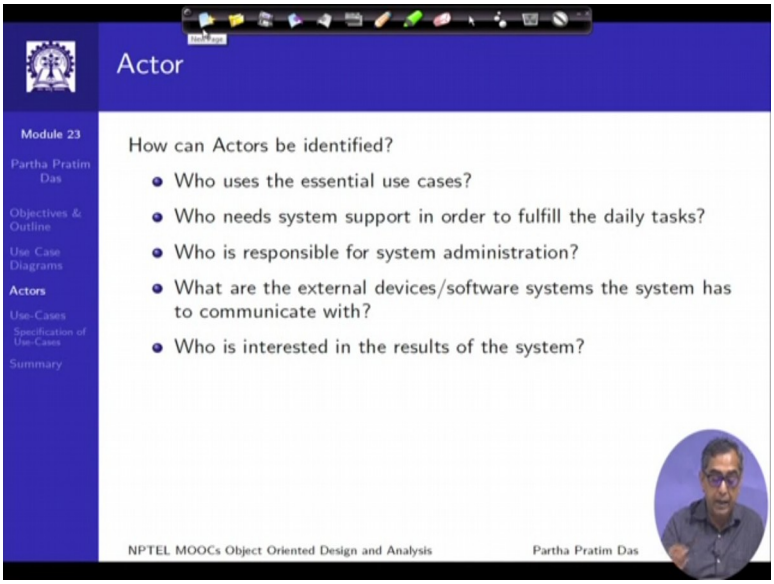
(Refer Slide Time: 06:39)



There could be variety of different actors or or rather the actors can be classified in multiple different ways. For example, the first-class verification I have already talked of, an actor could be human or non-human. An actor could be primary or secondary, the primary user is kind of who is a targeted end user of the system so in the leave management system, an employee or executive is a primary user. And a secondary user is someone who is required for the correct functionality of the system.

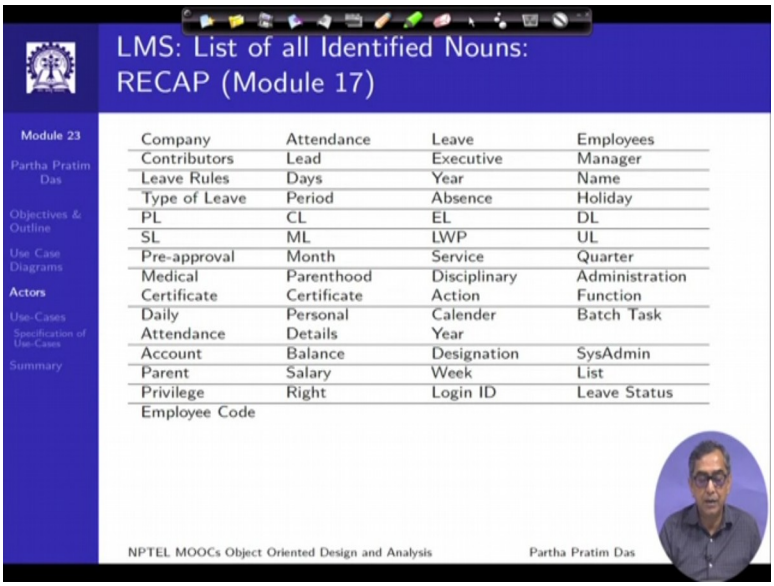
So, system administrator who has to make sure that the proper leaves are credited the leave, records are reconciled and all that is considered to be a secondary user. Then users could be active or passive. Active users are those who initiate a use case and passive users are those who are recipients of the use cases. So, an active user starts applying for a leave, a passive user is like a ehm actor is like a printer who based on a request to print a report would actually print the report.

(Refer Slide Time: 07:50)



So, the question again is how can we identify the actors. So, we use a essential use cases who needs the system support who is responsible for system administration, what are the external devices, software systems to communicate with, who are interested in the results of the system and so on. These are the questions you can keep on asking and answers to those will give you the actors.

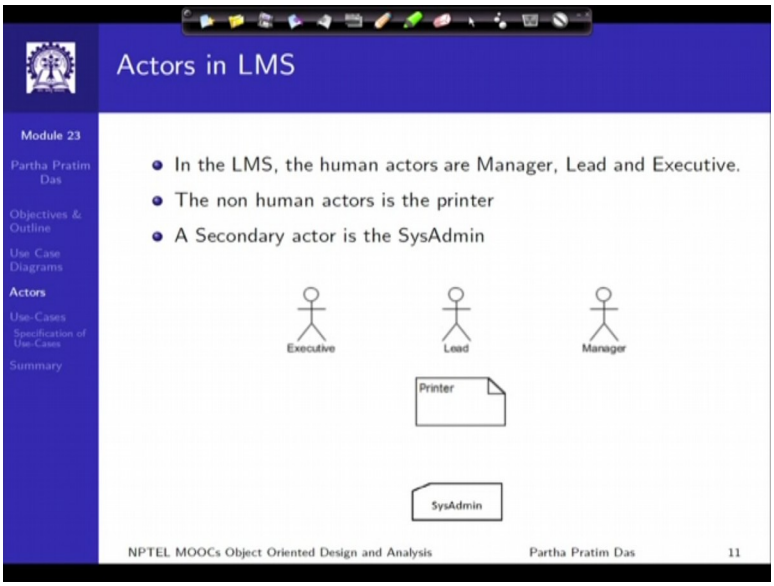
(Refer Slide Time: 08:24)



And extending on our earlier approach of analysis to identify objects and classes, I could easily conclude that certainly an actor will be one who is an identifiable class, identifiable object. So, if we refer back to our linguistic analysis of nouns that could give us some good clue in terms of what the actors could be, so we look in here all around and try to find out you know nouns like employee, manager, lead, sysadmin and so on which could initiate some action in the system.

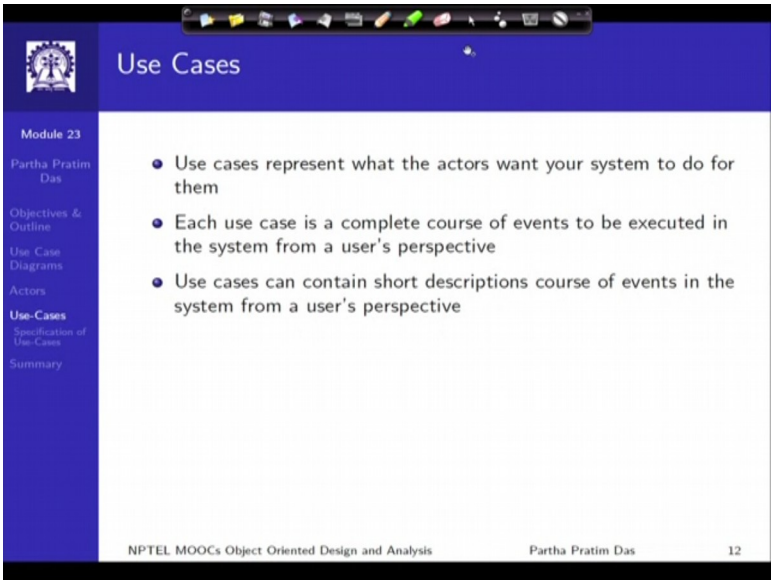
The employee initiates the request, lead initiates a approval process or a revocation process, sysadmin initiates the leave crediting process and so on. So, you can very well understand that there is there is no very standard mechanism for identifying actors but again the linguistic analysis of noun will give you some good clue and then clubbed with that if you do some you know action analysis based on the works you will be able to get a better confidence as to what, who are the people who would need to act on the system and must be actors in the uml diagram.

(Refer Slide Time: 09:49)



So, this is the lms example which I have already discussed. So, these are the 2 different ways so typi typically you use a this kind of a icon when you know that the actor is a human and you use this kind of an icon when you know that the actor is a non-human. So, these are the actors in the LMS system that you can easily see.

(Refer Slide Time: 10:16)



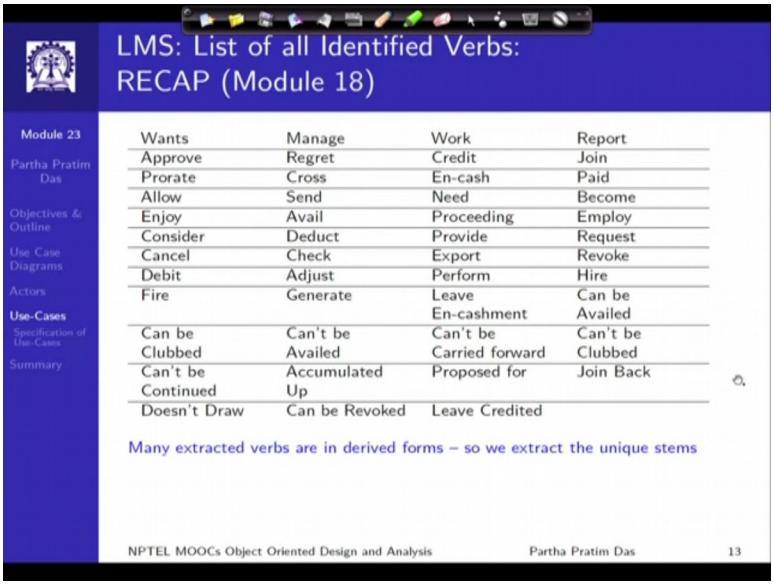
Now the next component of use cases diagram and certainly the most important component is the use case itself, that is a use case represents what the actor want the system to do, the required action. So, if an executive is an actor, executive wants to apply for leave. The lead needs to review and accept, approve a leave. So, these are what the actors want the system to do. So, we just do not end by identifying this action is required but you say that a use cases has to give us a complete picture, complete course of events.

For example, if if an executive is applying for a leave, then it's just not applying for the leave, it's the whole course of actions have to happen, you must have a reason for the leave, you must have a enough account balance for that leave, it must block those dates, it must calculate how many days it turns out to be, it may have salary implications and so on. So that complete course of actions from the use case for a particular action. In addition, use cases that their graphical diagrams as you say, but they could have some attached nodes, short nodes to describe what the use case is talking about and what the conditions is and so on.

You would I would like to remind you at this stage that use case is a diagram to be used at the very beginning of the whole activity. So, you are using use case at the requirements phase so when you are working on the use case you do not have any of the other uml diagrams possibly at hand. So, all that you are relying on is your classical natural language descriptions and discussions and so on. So, in one step you may not be able to put everything very concretely in terms of the required graphical notation of the uml.

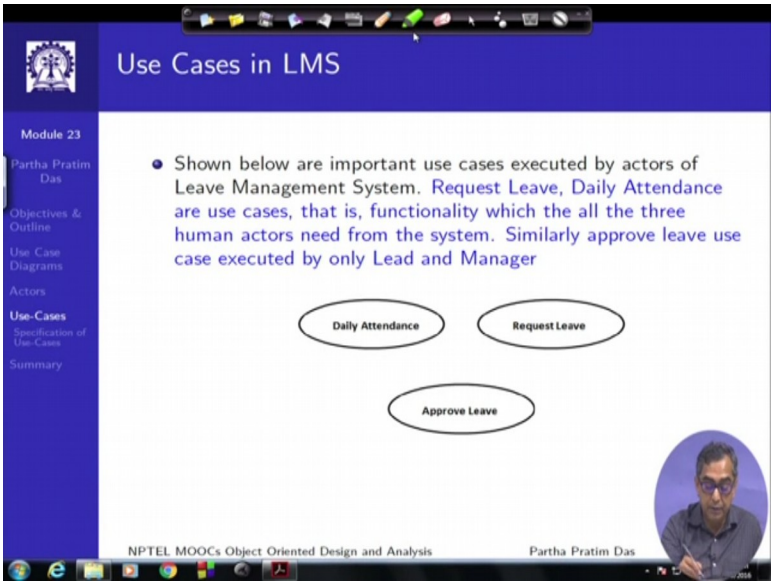
So, you may still need to have some further details, even some some details may not be representable in use case diagram. So those are the things you keep putting down in terms of the notes and attach them with different parts of the use case diagram.

(Refer Slide Time: 12:40)



To identify use cases again I would refer you back to the linguistic analysis of verbs and we can see that an employee needs to approve o I mean a lead or manager needs to approve a leave, regret a leave or revoke a leave, an employee can cancel a leave and so on, can avail a leave. So, these are different actions that are expected in the system and therefore they will very legitimately qualify as candidates to be considered for different use cases.

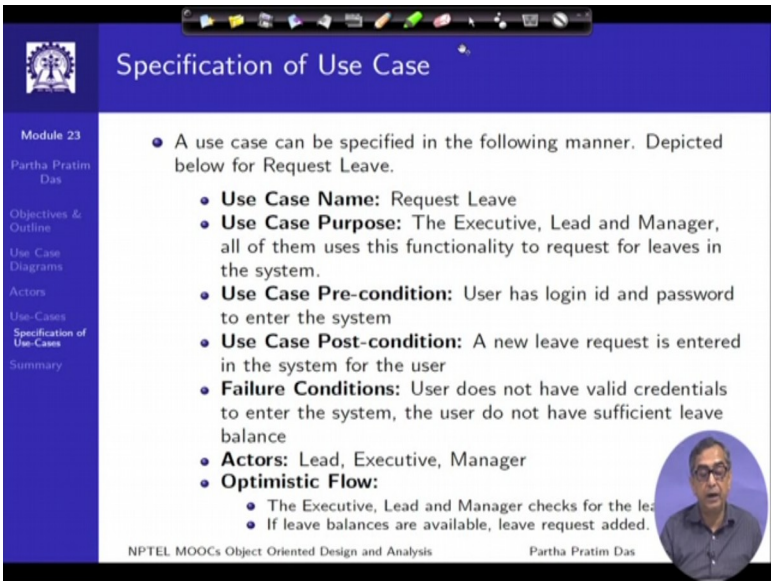
(Refer Slide Time: 13:23)



So very explicitly these are some of the request leave, daily attendance are some of the use cases, a use

case is typically drawn in a kind of a oval shape and with the use case name written here. So, this is the shape and this is the name of the use case. So, this is the minimum the use case must have. It may have other annotations, other ornamentations along with it but it must at least basic have a closed oval elliptic shape and a name to being identified as a use case.

(Refer Slide Time: 14:07)



You remember I have started saying that uml is grossly a graphical language, so everything every concept, every primitive we talk of there is a iconic representation for that. So along with certain textual annotations its primarily this iconic primitives which will tell us what's exactly being represented and going on in the system. Now a use case may be specified in the in the following manner that is this is kind of a possible way that you can specify a use case.

So again, we are at a very early stage of design so we do not have much of the details codified in the uml diagrams. So, in terms of a use case, at least the first thing naturally is obviously required then name of the use case, you must give it a name. it is good that in the notes if you put what is the purpose of this use case so that will clarify as to why this use case is important or how this use case could later stage be found to relate to other use cases and so on.

The next one is very important is it says that many of the use cases have a precondition that you can do something provided certain conditions have met in the system. Of course, in terms of the system this is a very trivial pre-condition that the login and password signing in is required to enter the system. But you can get a whole lot of pre-conditions in the LMS system that for example to be able to avail of a sick leave. for the basic approval for the or for the application I do not need a certificate but when it is

to be finally reconciled I need a certificate.

So, this reconciliation need as a pre-condition of a certificate. The post condition is what happens when a request has been entered am sorry when a use case has actually executed has performed and after that what has to happen. For example, you have applied for leave, the leave is approved, once the leave is approved it has to get temporarily locked in your system to show that you have less number of leaves now. Now when you avail on the termination of avail of the leave it must get actually deducted from your leave.

So, the post condition of having availed is that your leave permanently go down by the number of days that you have taken leave. The post condition of approval is it is provisionally locked to a lower number by the number of leave days that you have been that you have got approved. Because at this stage at the approve stage if the leave is revoked by your approving manager or you decide to cancel the leave, then it will again the post condition would be that it should get unlocked and go back, should get credited back to your leave records.

Often, it's a good idea to also document in the specification that the failure conditions, what happens if the pre-condition is not satisfied or if the use case fails to execute successfully. So, what should be the what should happen in those cases must also be documented. For example, the user has an employee has tried to apply for a leave, the balance in the account of the employee satisfy but in terms of the clubabililty condition, the leave was not a valid one. 2 leaves were being asked for together, just oppose together which are not which cannot be taken together.

So, in that failure condition we need to specify as to what happens to the apply leave use case. Trivial to say that actors need to be identified as to who all possibly can work with this use case and often people would also like to put a optimistic flow. An optimistic flow is not a fixed flow I mean is not that, this is what will happen but it's kind of we are designing the systems we are designing the system with a purpose so the optimistic flow say that if everything else go right then this is what we would want the use case to do.

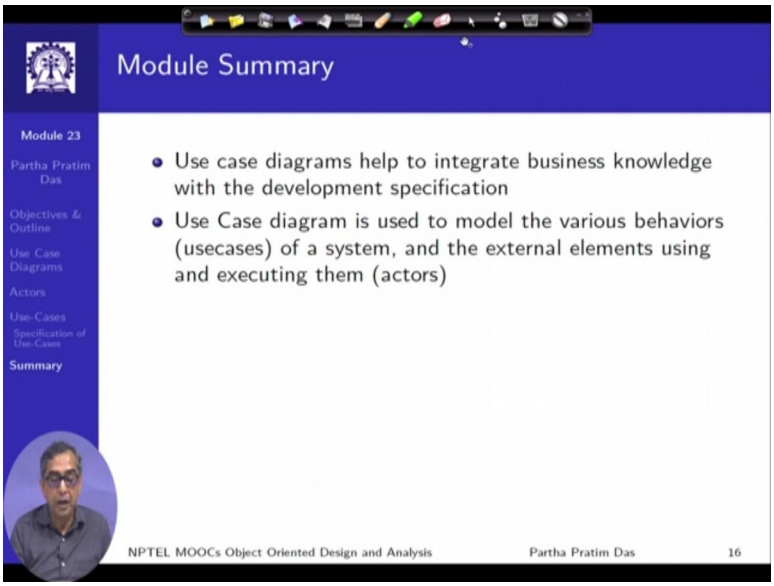
So, these are some of the possible optimistic flow for the request leave use case we are looking at. So, it's often good to document the optimistic flow because when it later on this use case will go to the analysis phase, the design phase and finally and finally the you know the low-level design will get

detailed out, it will help the designer to understand that this is typically what the system will be used for.

So, the designer at this stage could make recommendations to the test system later on that put a lot of emphasis on these cases because these are the cases in which the user would normally use and test the system. Whereas failure conditions are important to also design tests to decide as to if the system will work in the corner situations. So these are the typical these are typical structure of a specification of the use case.

It is not necessary that all of these of course the name is necessary but many of these may be missing in a specification but certainly as much of that can be put in the specification of the use case would certainly help.

(Refer Slide Time: 20:18)



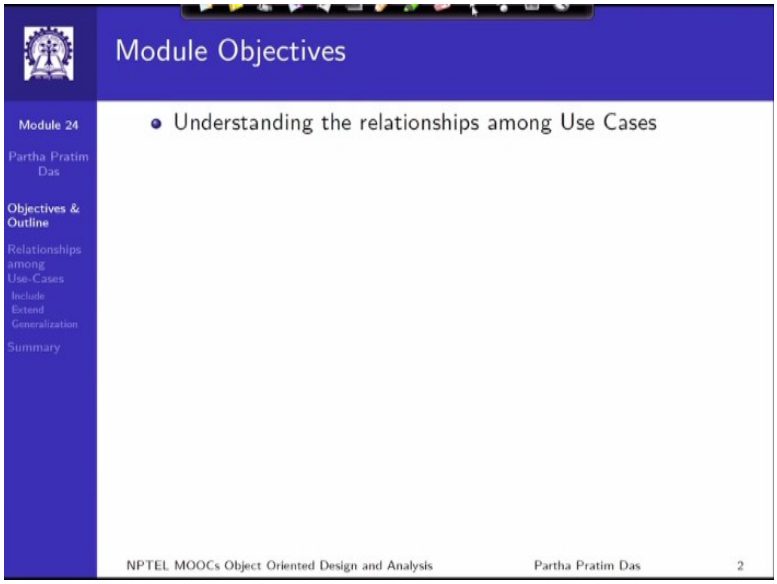
So, to summarize on this module, we have started discussing the first uml diagram, the use case diagram which we know is to be used for the requirement specification stage of the SDLC process and in the use case diagram we have talked about the 2 primary components of actor and use cases. In the following module, we will continue our discussion on the use case diagram and talk about the relationships that exist between different use cases.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture – 36
Use-Case Diagrams: Part II

Welcome to module 24 of Object Oriented Analysis and Design. From the last module we have started discussing about different UML diagrams; specifically, we have been talking about Use-Case diagram and we will continue on that.

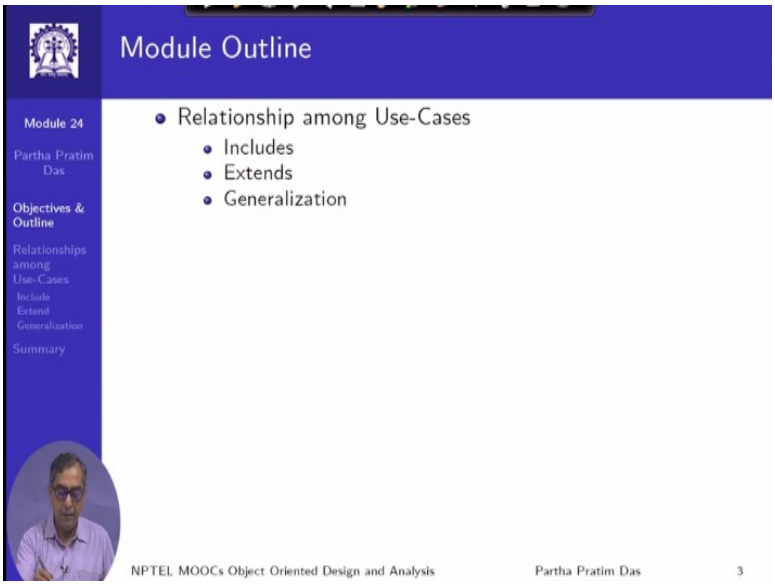
(Refer Slide Time: 00:44)



In the Use-Case diagram we have seen that there are few components that define the diagram the actor, actors in a Use-Case diagram are responsible for starting Use-Case actions or can be responding to some Use-Cases as passive actors. We have seen different classification of actors. The second component are Use-Cases which are well-defined perceived functionalities of the system that we want to put in place.

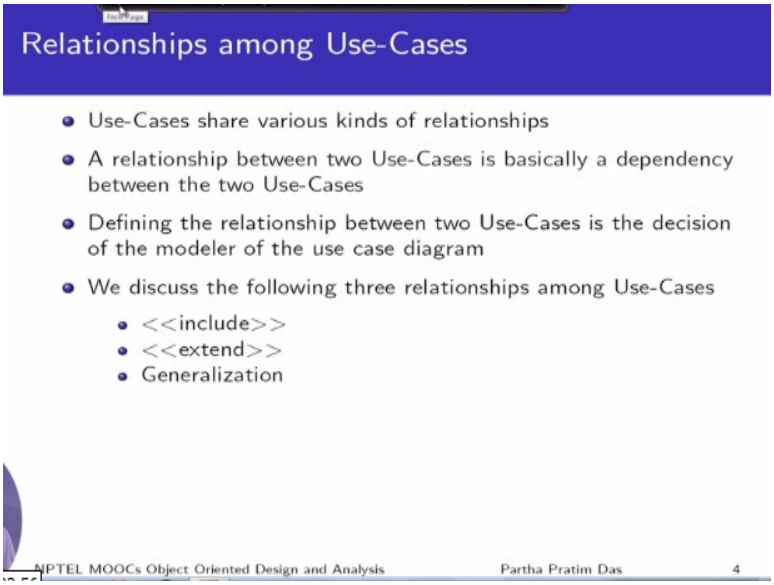
So these are the actions that the actors initiate every Use-Case is described by a name and we have seen that in the specification of the Use-Case we can state its purpose pre-condition, post condition, failure condition, optimal or most expected flow and so on. In this module, we will discuss about relationships among Use-Cases.

(Refer Slide Time: 02:04)



There are typically-- there are three major relationships that exist between Use-Cases; this is given in this outline which will also be available on the left hand side of every slide.

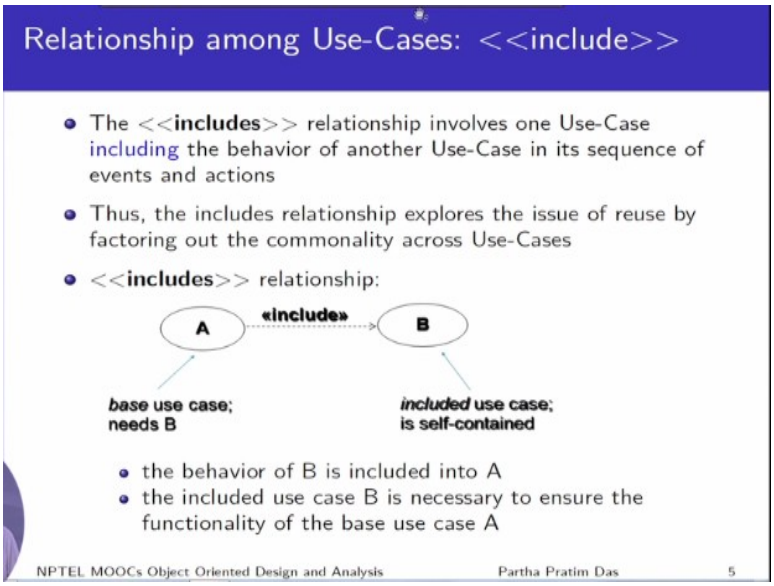
(Refer Slide Time: 02:13)



So the relationship among Use-Cases there could be various kinds. When you talk about relationship it is basically a dependency between two Use-Cases. So a relationship is I have a Use-Case here have another Use-Case here and I am putting some kind of a dependency between these Use-Cases and that dependency is known as relationship. Defining the relationship between the two Use-Cases is usually a decision that the person modeling the system would take.

So analyzing the specification looking at different conditions that may exists between these actions, we will identify different relationships. Here we will talk about three relationships the include relationship; the extend relationship; and the generalization relationship.

(Refer Slide Time: 03:18)



So we will start with the <<include>> relationship. The <<include>> relationship involves one Use-Case that involves another Use-Case that includes another Use-Case. So the <<include>> relationship basically explores the issue of reuse. So if you just draw a parallel with the common programming paradigm then <<include>> relationship is kind of one function calling another.

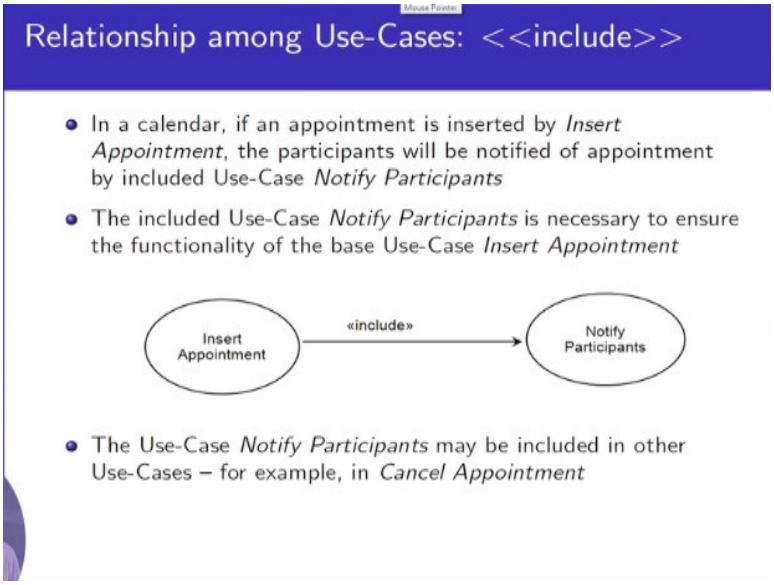
So when we want to perform some action say main function wants to perform some action and that includes printing certain values then main would not be directly printing it rather main will call a function printf, to take care of the actual printing process. So main if main is one Use-Case, then Printf would be other Use-Case so that is a typical programmatic view but in general we will say whenever we have a Use-Case A and to complete its own functionality.

It might need another Use-Case B then we will write it like this A <<include>> B. So you should note the direction of this relationship A is the base Use-Case which needs B, B is the included Use-Case which we will assume a self-contained. Of course in turn B may include other Use-Cases that is quite possible. So the arrow direction of the arrow would be from A to B and the style of the arrow would be dashed.

And to make it explicit that it is an <<include>> relationship we will use a gulimet and say that is an <<include>>, so this is the way we write it. So that would mean that the behavior of B is included in the behavior of A because while the Use-Case A is executing at certain point it will invoke B and complete that and the included Use-Case B is necessary to ensure the functionality of the base case.

So without B, I cannot complete the functionality of A and this is the basic notion of <<include>> relationship.

(Refer Slide Time: 05:35)



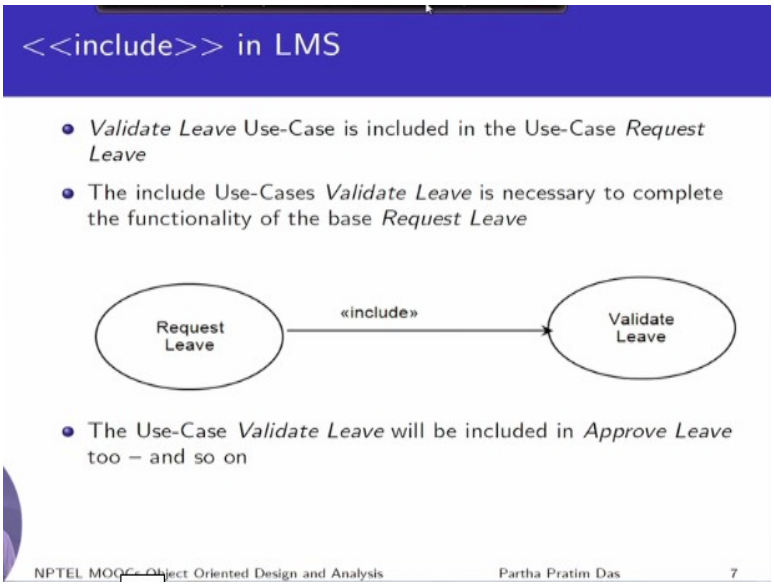
So for example if we have a calendar kind of an application were there are different appointments that I can insert, create. Now certainly we will expect that once an appointment has been created then there will be notification given to the participants. So one Use-Case-- I am sorry, one Use-Case is the Insert Appointment, the other Use-Case is Notify Participant and there is a <<include>> relationship from the Insert.

So Insert Appointment is the base case and Notify Participant is the included Use-Case in. Now we may note that like in functions which can be called from multiple functions the Notify Participant this Use-Case Notify Participant maybe included in other Use-Cases. For example, if

I later on somewhere, I have another Use-Case Cancel Appointment then certainly when an appointment is canceled the participant needs to be told.

So Notify Participant would also be included in the Cancel Appointment Use-Case.

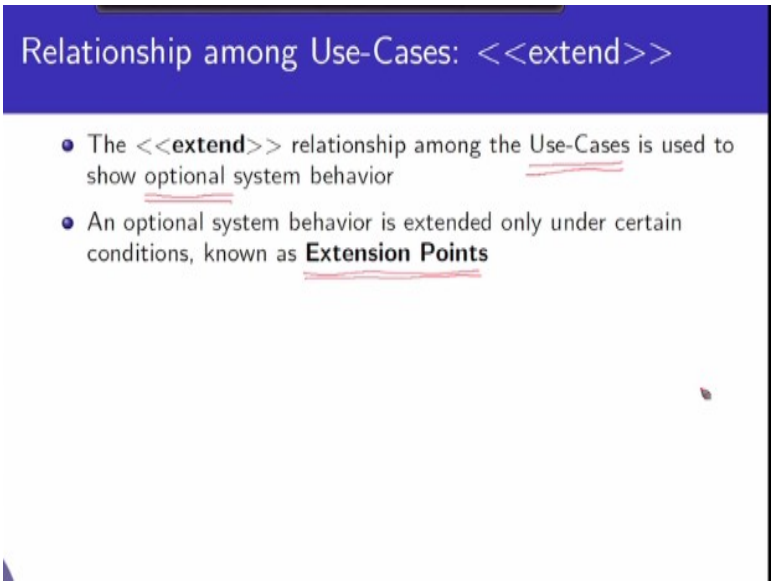
(Refer Slide Time: 06:51)



Talking about are ongoing LMS system we have-- for a leave we need to run a validation to check whether a particular leave is a valid one. And if we say that that Use-Case is validate leave which has all the actions performed to validate a leave. Then the <<include>> use—this will be can be included whenever we are doing a say Request Leave. So when an employee is requesting for a leave certainly before the request a leave can be created.

We need to validate that the tentative leave record is a valid one, so request leave is a base case and validate leave is a included case that will get included. Certainly validate leave can be included also in other Use-Cases like approve leave. When the lead goes onto approve a leave naturally the -- it will need to be checked that the leave being approved is a valid one under the conditions that the lead may have imposed on the leave.

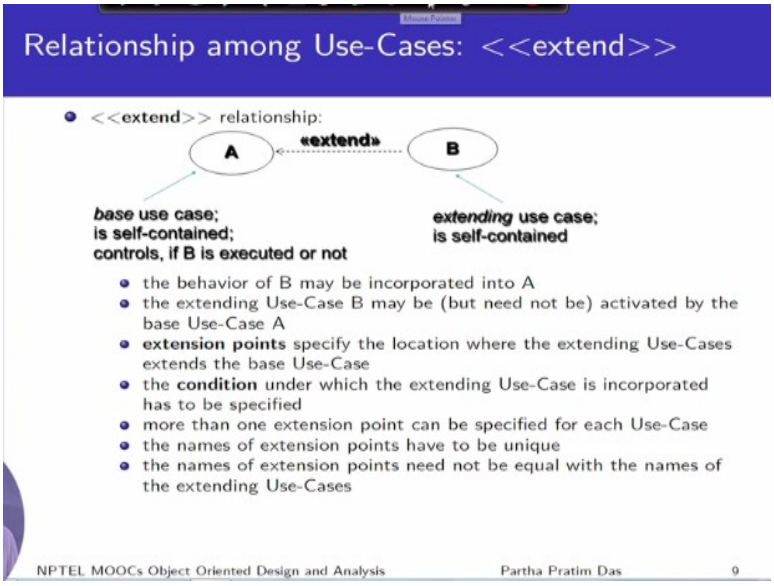
(Refer Slide Time: 07:57)



The second relationship between Use-Cases is known as a <<extend>> relationship. <<Extend>> relationship is a kind of an option adding an optional behavior among the Use-Cases. So it can be used to add some optional behavior to what has already been defined for a Use-Case. Now when you want to add this optional behavior; now this optional behavior means that it could actually depend on certain conditions.

So you can also say it is a conditional Use-Case. So the conditions are represented in terms of Extension Points. So you see if you reach the specific extension point then you invoke the extended Use-Case.

(Refer Slide Time: 08:51)

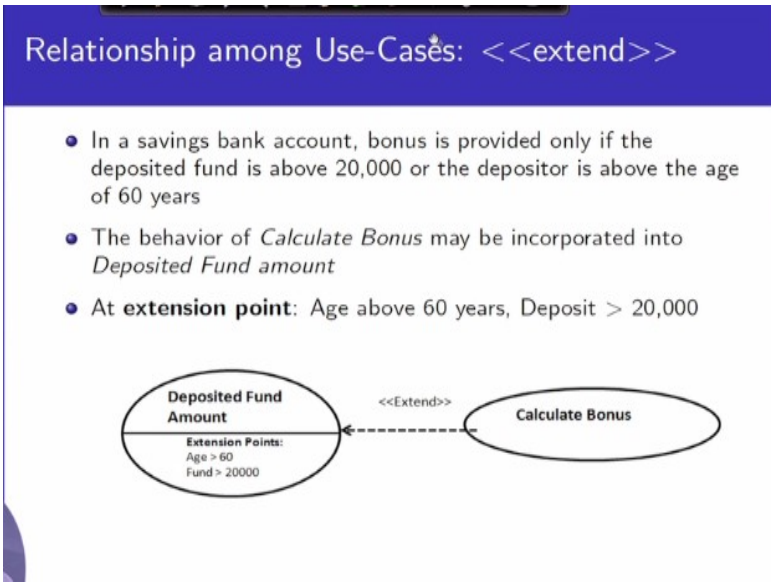


So if we say, we have a base use case A as an here; we have an extending use case B as an here and in this case both A and B are self-contained unlike in <<include>> where B was self-contained but A the base case was partial because it need B always—here that is not the case because this is the case of optional or conditional invocation. But A controls if B should be executed or not, that is a conditional one that is the optional one.

So these are the couple of points we can note the behavior of B may be included maybe incorporated into A provided it is invoked so extending the Use-Case B is optional that is it maybe but it need not always be activated by A. And there are Extension Points that specify the location in Use-Case A where the extending Use-Case will be used. So there is a-- the condition under which this is incorporated the condition is actually given by the Extension Point.

Now some more details will include that more than one extension point can be specified for each Use-Case. There could be multiple different conditions under which the same Use-Cases execute. The names of extension points must be of unique of course otherwise we will not be able to identify them. And certainly since one Use-Case maybe-- may extend multiple extension points it is quite possible that the number and names of extension points and the number of extended Use-Cases may not be one to one, there could be multiplicity of relationship.

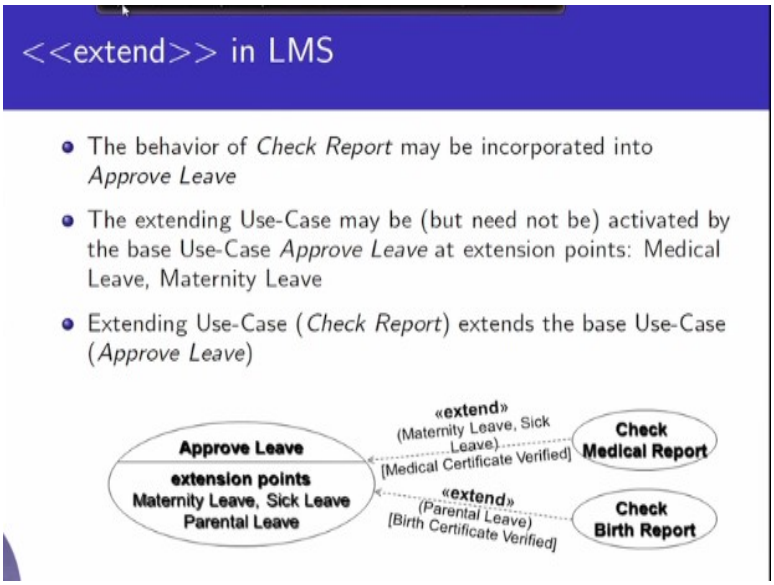
(Refer Slide Time: 10:51)



So as an example say savings bank account provide bonus if the deposited fund is above 20,000 or the depositor's age is above 60 years either that depositor is a senior citizen or their fund values is more than 20,000 then a bonus is provided. And if the Use-Case for providing the bonus is calculated bonus and the Use-Case of actually computing the fund is deposited fund amount then we can have an extension point which incorporates these two conditions.

The senior citizenship condition as in here and the fund level condition as in here and you find those in the diagram. Let me just clean up and show you again. This is the extension point and the deposited fund amount is a base case and this is the extended case that will get invoked at this extension point; that is the basic idea of extended relationship.

(Refer Slide Time: 11:58)



So in LMS, we know that there are certain types of leave; we have seen certain different types of leave which can be approved provided there is a certain condition that will satisfy, for example for a maternity leave certainly the certification is required that the employee in question already has the medical condition of pregnancy or for a sick leave we need the certification that the person was sick and doctor has checked and so on.

Similarly, for a parental leave we will need birth certificate of the baby that has been born to the required-- the associated employee. So let us try to put that behavior of as a Check Report which we may want to incorporate into the Approve Leave which means if the lead has to approve a

particular leave which is of either these are-- you remember the leave types the ML, SL, ML is Maternity, SL is Sick Leave, and PL is a Parental leave, if any of these leave have to be approved then for those leave we will need to check for this.

And certainly for the maternity and sick leave it is kind of medical report, so I can have-- we can model the extended Use-Case which is Check Medical Report which has to do all the necessary tasks for actually checking out the validity and authenticity of the medical report and so on. Whereas for Parental leave we need to check some corporation documents, municipality document regarding the birth of the baby having taken place.

So that is another Use-Case check birth report. So we can we can model this by having an extension point or having actually multiple extension points I should have marked them here in the Approve leave Use-Case. One extension point is Maternity Leave, Sick Leave. So in terms of Approve leave if the condition is that you have a situation of Maternity or Sickness then the corresponding check medical report Use-Case has to be invoked.

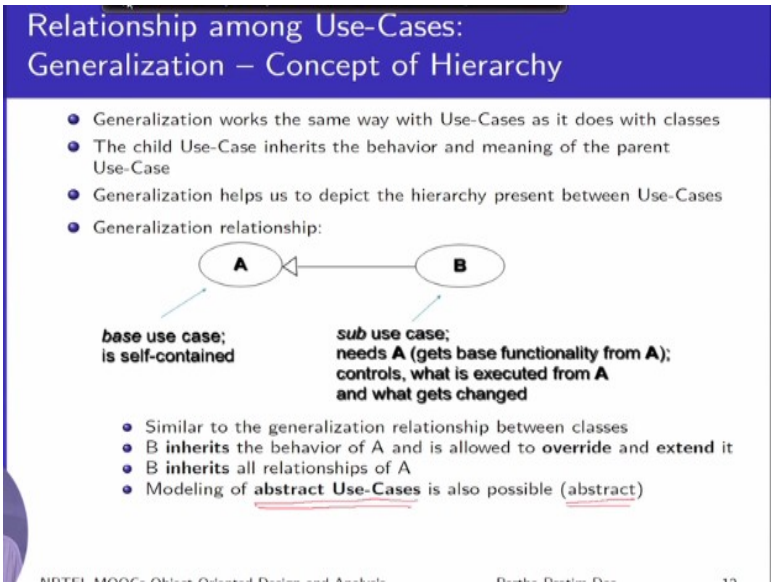
So that is designated by this arrow so here also note that the direction of the arrow is from the extend Use-Case to the base use case. And on that again we put the guliment <<extend>> to mark that it is an <<extend>> relationship. On top we put the condition of extension which is extension point name, so the name of this extension point is Maternity Leave, Sick Leave so that name has to be match here Maternity Leave, Sick Leave.

And then below that we try to mark what is the action that this extension will take which in this case is verification of medical certificate. So for example for the other extension point this are the other extension point where we have Parental leave so the extend relationship will show what is the Parental leave condition on which this checkbox report to be done and here we specify is to what is action that this will take.

So extension often is a very convenient way to relate different conditional Use-Cases to one or more base Use-Case, it is possible that same extended Use-Case may be associated with multiple different extension points in different base Use-Cases as well. For example, checking medical

report here is happening in terms of approval it may need to be also invoked in case of say revoking a leave, you may really need to check on the report to actually decide whether the revocation can at all be permitted in the system or not.

(Refer Slide Time: 16:31)



So this was the second kind of relationship. The third kind of relationship that may exist between Use-Cases is the Generalization relationship. The concept of generalization and you must have experienced by now that this basic concept of generalization specialization or hierarchy is repeatedly occurring in the OOAD scenarios. So it works in the same way as does with classes. So there is a base case A; there is a sub use case. Here is use the terminology sub use case B.

So base case A is self-contained, that it has a within that it needs for the invocation. But the sub case B needs A because what it does as in class associated generalization specialization or inheritance the sub use case inherits-- this inherits all the functionalities of the base case A. And after inheritance-- so this is similar to this so it inherits the sub use case inherits the behavior of A, the base case but it is allowed to override and extend it.

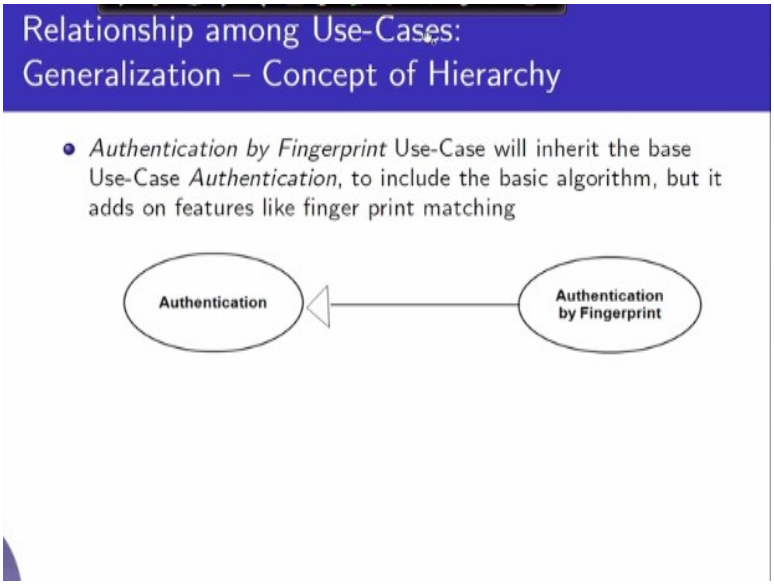
Override means that once a Use-Case has been inherited it has different behavior it has different possible operations that the Use-Case may invoke. The sub case B make use these operations as it is from A or it may want to redefine the behavior of the operation. If it redefines the behavior of the operation this basically means redefine then we say it has overridden the particular

operation from the base use case. Or it may want to extend it which means that it will first execute the operation as in the base use case and then it will add something more.

So the possibilities are as it is that is just inherit, override that is use the same name of the operation but have a different functionality or extend that is first use what you have inherited and then you do something more, these are all the possibilities. So in overall the B inherits all the relations of A, the whole functionality of A. And in this context it is possible that we can have some Use-Case which is abstract.

So if a Use-Case is abstract will normally write within the Use-Case that this is an abstract Use-Case. What is the meaning of the abstract Use-Case is that this is a scenario of action which will never take place in the real-world but this is being put in place more for the modeling purpose to unify a number of different possible Use-Cases that can happen. And like in abstract classes this is also a regular functionality of Use-Case.

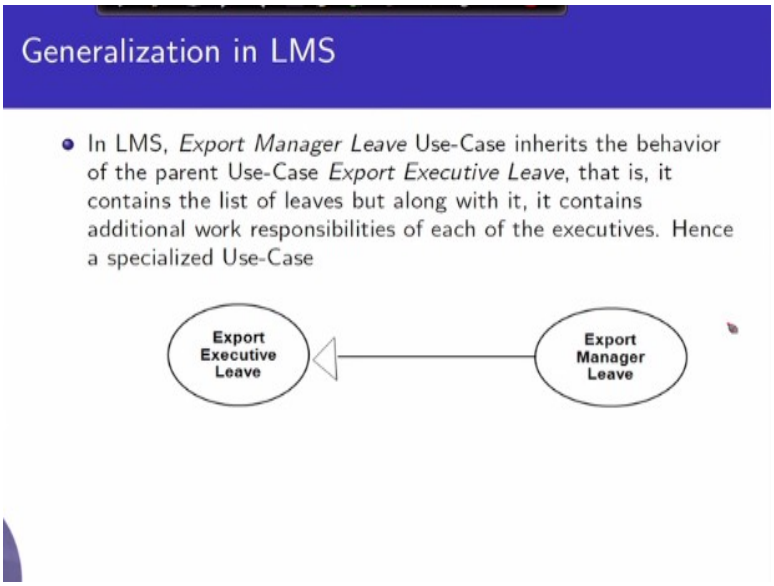
(Refer Slide Time: 20:16)



So as an example let us say I have an authentication system. So I have Use-Cases for authentication which does variety of authentication like password base authentication maybe SMS, OTP based authentication and so on. And I want to you define another Use-Case authentication by fingerprint which specializes from the authentication use case. So what is does, it inherits all the different algorithms and operations that the authentication Use-Case has.

But, in addition, it will include some fingerprint matching strategies in the authentication in this Use-Case and that is how the authentication Use-Case will get a specialized in the authentication by fingerprint.

(Refer Slide Time: 21:09)



Talking about LMS example if we think of a Use-Case to export executive leave and another Use-Case as exporting manger leave then we will have again a generalization, specialization relationship this being the base. If I want to export the manager leave certainly I will do everything that is required for exporting the leave of an executive which is the base case. But in addition, I will need to possibly do some more export of the data in terms of the different employees who report to this manager and their availability during the leave of the manager.

So this will typically have then when I extend I inherit all the functionalities of the export executive leave Use-Case and then execute them and do something more on top of that. In LMS we could also think of abstract Use-Cases for example we can say that we have different Use-Cases like 'Request' is a Use-Case then 'Approve' is a Use-Case then 'Avail' is a Use-Case and so on.

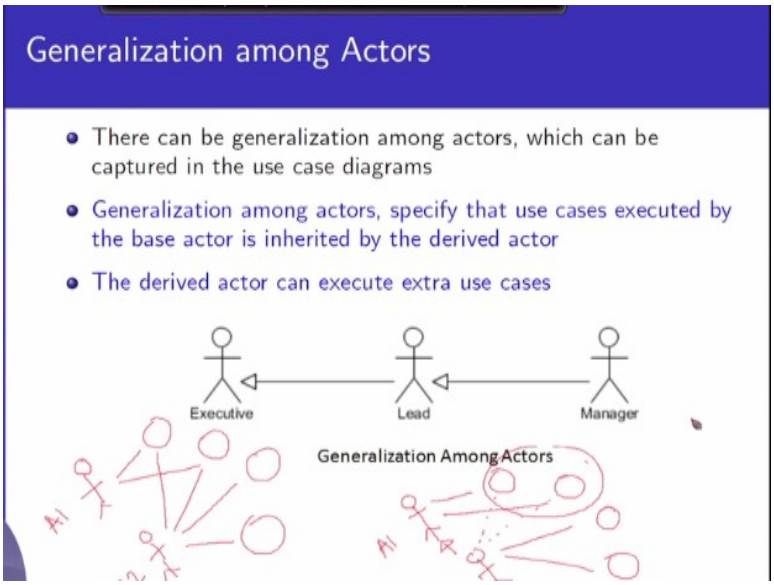
Again say that Use-Case 'Act on Leave' which is an abstract. And we can say these are these basically are all specialization of this particular 'Act on Leave' Use-Case. So what does the 'Act

on Leave' Use-Case specifically, it is some action on the leave so this will have the basic functionality of checking out on the leave record maintaining in the leave store and so on which is something which all of these leave Use-Cases will need for its own invocation for its own implementation.

So instead of that all those leave store management related stuff being implemented as a part of all of these specific Use-Cases we can club them together and conceive of an abstract Use-Case like 'Acting on Leave. Now why is this abstract? Because by itself just the functionality of the leave being maintained on the leave database on the leave records that being accounted and so on just by itself is not a stated functionality or a perceive functionality if we look into the leave management system.

So- but this will come out through the discussion and analysis of the LMS specification that such abstractions could actually help the system get organized in a better manner.

(Refer Slide Time: 24:12)



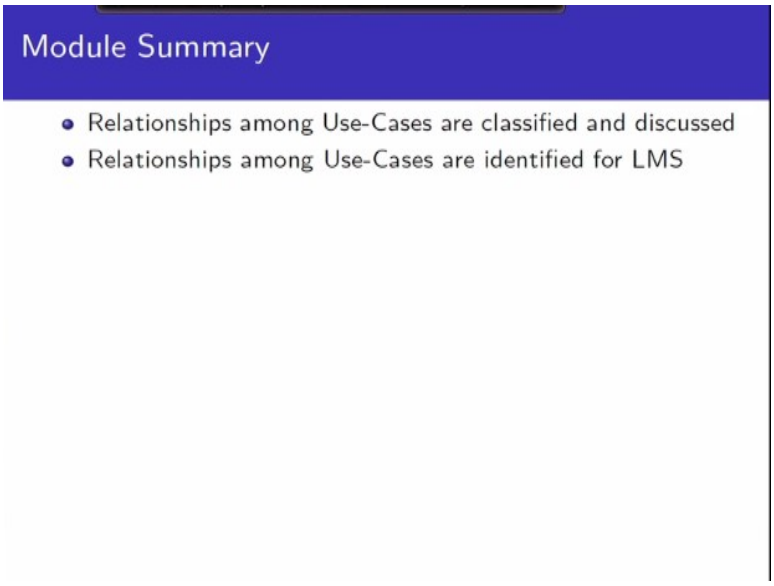
Besides the relationship amongst the Use-Case there could be some more relationship like there could be a generalization relationship between actors. That is we can think that actors like an executive the example is a best way to look at executive is an actor lead is an actor and we have already in terms of identifying classes we observe this that lead as a employee can do anything that executive as an employee can do but lead can do something more.

So again have a base so this is a base actor and this is the sub-actor or the specialized actor who inherits all the functions of what the executive actor and do but lead actor can do something more and in turn that could be generalized-- specialized fact in terms of the manager actor. So what happens is often this is this is very useful as we will see that if we have two actors one is here and one is here and suppose we have four different Use-Cases, and this actor can use this two and this actor can use say this four instead.

So basically it means that say the actor A1 and actor A2, A2 can do anything that A1 can do then it maybe more explicit to model this separately that I say this is A1 this is A2 and there is a specialization of A2 from A1 and of the different Use-Cases I will associate these two with A1 and only these two with A2.

I actually do not draw this association because since A2 is a A1 it is implicit that all these Use-Cases will also be associated with A2 that often simplifies the clatters and simplifies a model much better and give a clarity in terms of the Use-Case diagram.

(Refer Slide Time: 26:36)



So to summarize this module here we have for continued to our discussion on the Use-Case diagram and in addition to the actors and Use-Cases we have now seen that there could be relationships between Use-Cases and in specific we have talked about <<include>>, <<extend>>

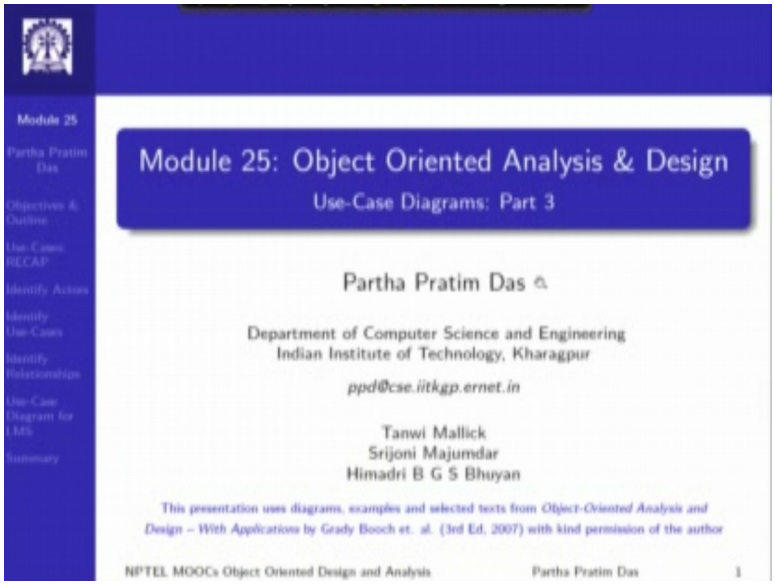
and Generalization relationship between Use-Cases. And we have also seen examples so of the LMS system.

We will continue this discussion in the next module as well were we will try to explore and consolidate the Use-Case diagram for the LMS system.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

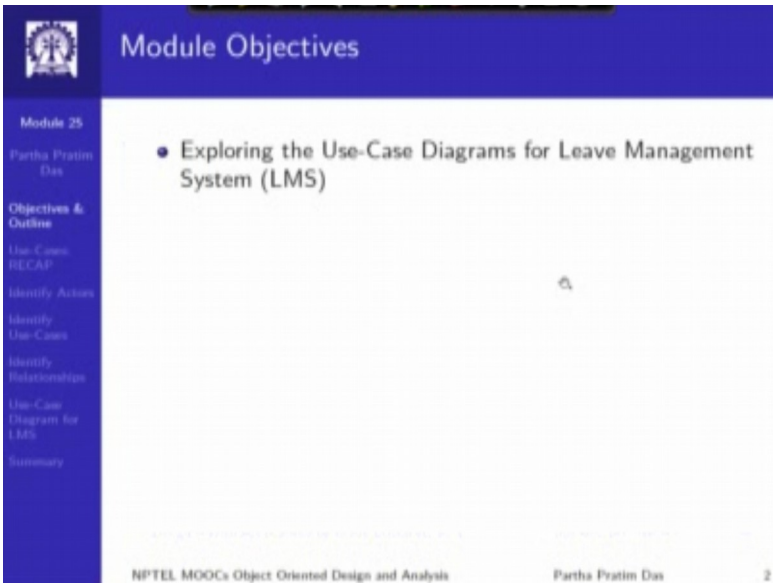
Lecture - 37
Use-Case Diagrams: Part III

(Refer Slide Time: 00:25)



Welcome to module 25 of object oriented analysis and design. We have been discussing about use-case diagrams.

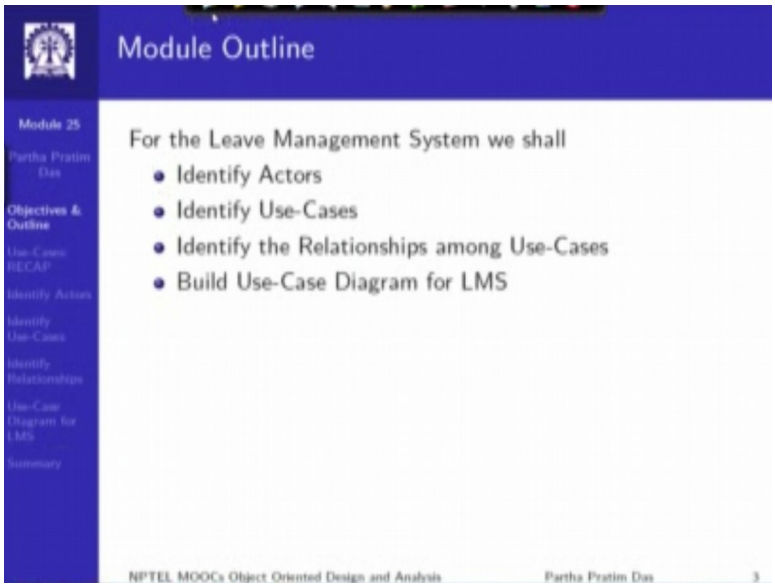
(Refer Slide Time: 00:31)



And we have seen all the major components that constitute use-case diagram. So in this module we will focus on trying to analyze the leave management system specification further

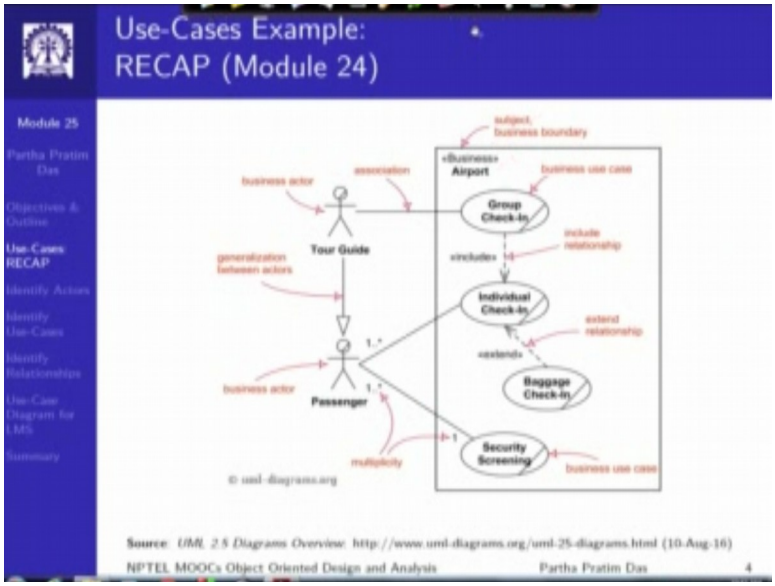
and try to show you how these use-cases can be identified and put in together and we will try to conclude with a first level, very tentative use-case diagram of the LMS system.

(Refer Slide Time: 01:09)



But before I get into that let me just also recapitulate some of the major features of the use-case diagrams that we have already discussed. This will be the outline for this module. So certainly as you can see the main actions would be in terms of identification of actors, use-cases and the relationships.

(Refer Slide Time: 01:39)



So first on to the recapitulation. So this is just based on what we did in the earlier module. So this is an use case example, not one which we have discussed earlier. So this is kind of checking in at airlines counter and the the possibilities are that you could do a group checking that is if you are going on a tour together or you can do an individual checking, actually it

involves checking in of the baggage and before you can get into the flight you need to have a security screening.

So these are the different use cases that exist. So for each one of them we specify that this is the use-case name. This part is where you can put more additional notes and so on. We identify two actors, this is a passenger who is a passenger who is a business actor we say and this another who is a tour guide. So it comes in place when certainly some a group of people need to check in for a particular tour.

So the kind of things that we can expect is, one, is an include relationship between the group check in and the individual check in. Which means that if 5 people have to check in then necessarily it means that each one of them have to check in individually. So this is an include relationship which say that to be able to complete the group checking it is necessary to complete the individual checkings.

Similarly, we can see that between the individual checking and the baggage checking there is a exchange relationship. Why is it an exchange relationship because if someone is checking in for a flight, then whether or not the person will carry some baggage to be checked in is optional. It is not every passenger, some passenger could just fly with one bag and does not need to do a check in. So this optionality is demonstrated by the extend relationship.

So individual check in at an extension point which is not shown here possibly saying that has baggage kind of will invoke the baggage check in use case. Then there is another here. Now some more that we can absorb here is between the actor tour guide and the passenger there is a specialization relationship because any tour guide can also be a passenger might travel with the same group, but a passenger is not a tour guide necessarily.

So a tour guide can do all that the passenger can do like, tour guide can individually check in, will go through the security screening but this is not possible, that is a passenger cannot do a good checking. So that is exemplified by this generalization, specialization relationship between the actors what we are showing here. Further you can observe that certainly actors are associated with use-cases, so the tour guide is associated with the group checking which is marked by this association link.

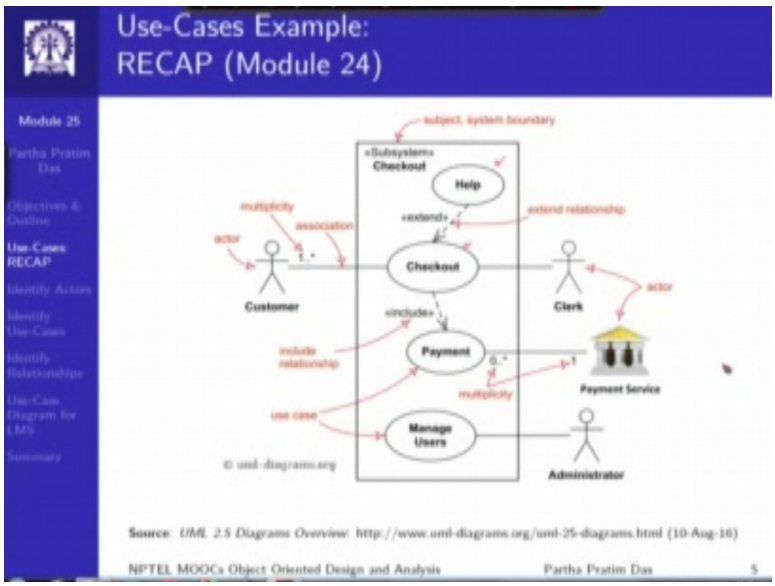
The passenger is associated with the individual checking which is marked by this association link and we can also see that there could be multiplicities at the end, both ends of the association. The multiplicity means that how many different actors can associate with how many different use cases. So here we say it is 1 dot dot star which means that one to any number any number more than one.

And here it is one. So it says that any number of passengers, at least one passenger has to be there, but any number of passengers can go through security screening. Similarly, in this case of association between passenger actor and the individual checking use case, the 1 dot dot star again means that any number of passengers can check in individually. Here the one is not mentioned, if it is not mentioned then it is meant to be one always.

So this is called the multiplicity. What else do you observe? We observe that there is a bounding box that puts all this use cases together. This bounding box is the perceived boundary of the world that we are trying to model and it is often called the business boundary and or the subject of the use case diagram which bounds the whole system and typically the subject will have a name to identify this particular business boundary.

And we will find that always the actors will be outside this business boundary because they are not inside the system dynamics, they come and interact with the system to get the kind of services that they would want.

(Refer Slide Time: 06:50)



This is just looking at another example. This is just an example of a typical internet based purchased process. So what we will have, we have one actor who is the customer. There is one actor in case you are doing a counter purchase kind of, so there is another actor who is the clerk. There is an actor who is administrator and there is some actor like payment service and from the model itself you can figure out that while customer, clerk and administrator are human actors, this is a payment service is a non human actor who is involved in this system.

The use-cases, the typical use-case is a check out which is after you have identified the items and put them in your card, whether it is a virtual card or it is a card at the shop. You go to the payment counter and you try to place the order saying that I want to buy these things and I am placing the order and I will make the required payment.

Naturally if you do the check out, the other actor associated with the check out may be the clerk, this may be nonexistent if you are just doing an internet kind of transaction in which case the clerk may again become a non human actor, but at the counter the clerk is a human actor. So the clerk has to take part in the check out because she accepts the goods that you want to buy and prepares the order for you.

Certainly the check out will include payment because you are not allowed to go away with the goods until you have made the payment. So to complete the functionality of the check out use-case you need to perform the functionality of the payment use-case. Now payment assuming that you are making the payment through your credit card or debit card, he will need the payment gateway, the bank and the payment gateway like the Visa, Master this kind of MasterCard these kind of to take part and make that payment happen.

So you can see an association between the payment service and the payment use case through this association. In addition, there could be certain use-case to manage the users to manage the total floor and administrator is the actor who will be associated with managing this. Administrator will not directly take part in the other activities.

So if we look at the associations, customer is associated with check out and the multiplicity is well understood that any number of customers could do a check out. Then payment service, one payment service is associated with multiple payments and here you can note that the

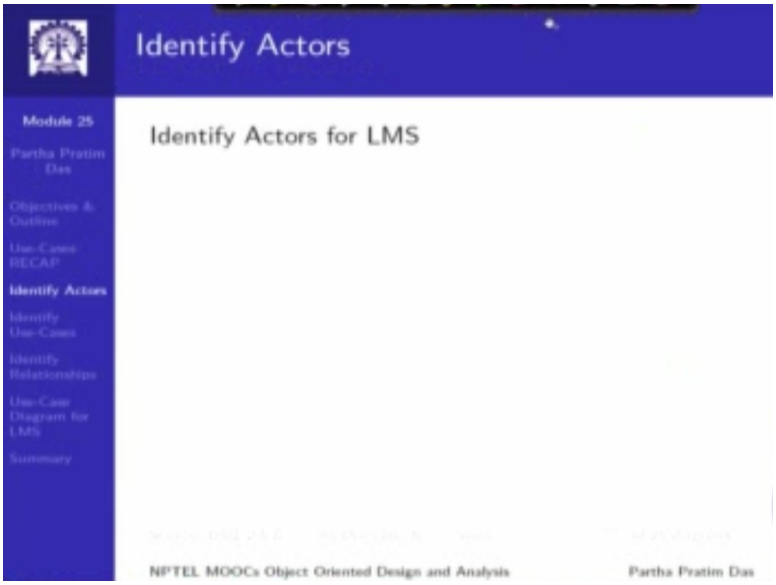
multiplicity includes zero, which means that a payment use-case may not actually use a payment service.

For example, if you are paying through cash then it is just the cash counter. So that is the kind of meaning that the multiplicity can give you. And there is an extend relationship on this site where while doing the check out you might need some help, for example some item may not have a proper bar code and some executive need to be called for the item code and so on.

So you might need different kinds of help, but you do not need them as a regular process, you need them optionally under certain conditions and therefore help is an extended use-case for check out. The whole system falls within this system boundary, this rectangle and is given a subject name of check out. So this is just another example of a different system showing how use-cases can be done.

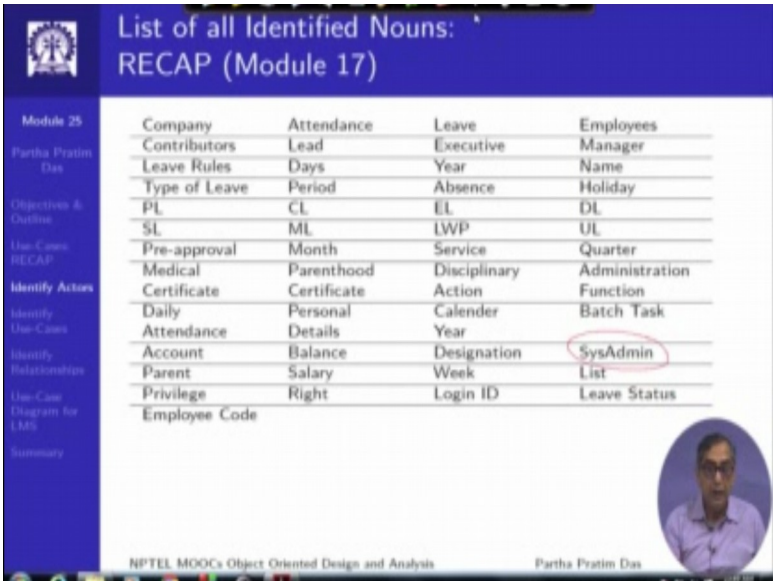
So I just wanted to consolidate all that we have learned in terms of the use-case diagram, in terms of two scenarios airlines check in and in terms of a purchase check out to show you how typically you should be building the use-case diagram for the system that you would want to represent.

(Refer Slide Time: 11:25)



Now let us quickly look over the actions that will need for the LMS certainly to build the use case diagram you need to identify the actors.

(Refer Slide Time: 11:39)

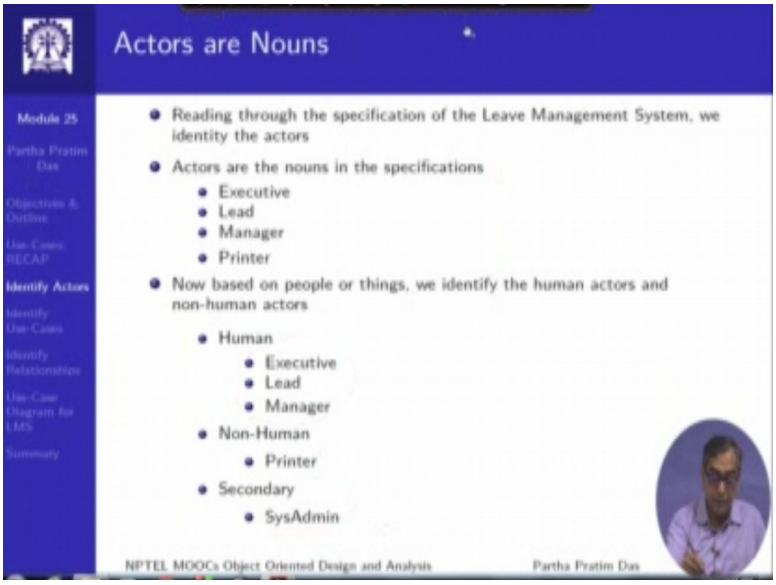


We have already started this activity with the linguistic analysis of nouns. The same thing will be required here, so all that we have analyzed we can look over them and try to identify different actors like manager, like lead, like executive or even like employee and so on. But you will have to keep in mind that not all nouns that you identify could be actors for example leave.

This cannot be an actor because the leave is acted upon, you deal with the leave, so it becomes a subject of what you do. But leave is not something that initiates an action which provides you a functionality. So it is not only that you look for the nouns, for example the another could be sys admin who could be an actor, but not only you look for those nouns, but also on that from that analysis you will have to do a further processing to reason as to who of these could actually act on certain use-cases.

Naturally what we will see subsequently is the identification of actor and the identification of use-case will have to be closely related.

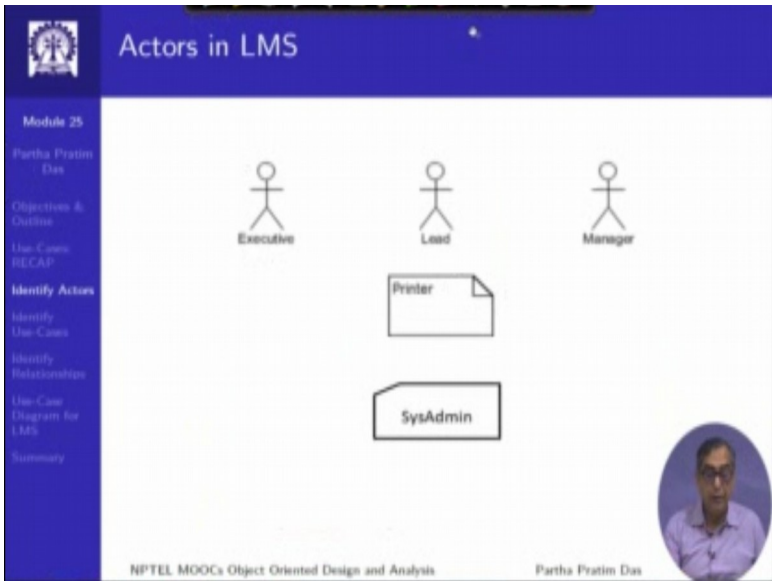
(Refer Slide Time: 13:01)



So if we do this then these are the different nouns as we can identify from this. Now based on these we can have different classification of actors here. The human actors are executive lead and manager, the non human actors could be printer, sys admin could belong here or could also belong here, it depends on what kind of sys admin you have, whether it is just a batch administrator process or it is a human being.

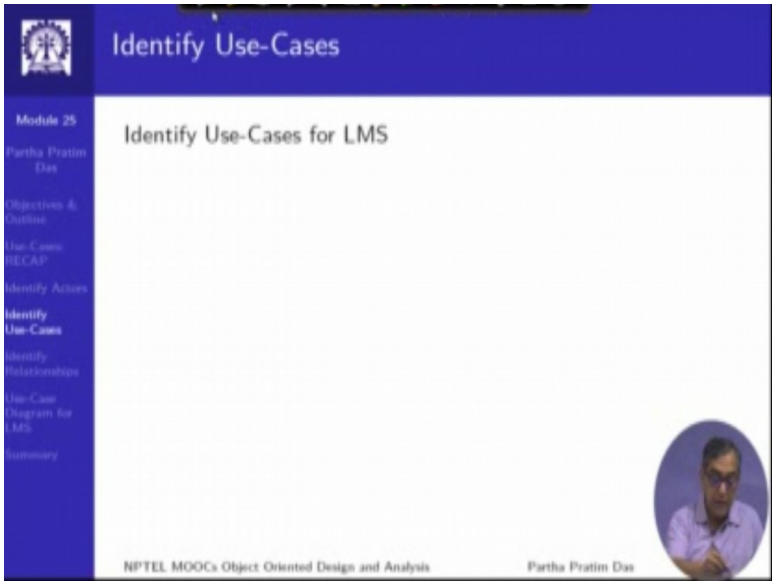
And certainly sys admin kind of actors are secondary because they do not provide the primary functionality of leave management in the system. Whereas these human actors, the executive lead manager are also in terms of our classification are also primary actors. So we could go through this analysis and then based again based on the specification and our understanding we could classify all these different designated actors in the system.

(Refer Slide Time: 14:11)



And accordingly we could choose icons for human and non human actors.

(Refer Slide Time: 14:20)



Second is identifying use cases.

(Refer Slide Time: 14:27)

So we will try to identify the use-cases for LMS and use-case is basically as I said is an action, so we will start by trying to identify. If it is an action, then the specification it must associate with some kind of a verb. So we will try to start with a list of verbs that we had extracted from the specification and I am just referring to the list that has just the stem, not all different qualified and derived forms of the verbs.

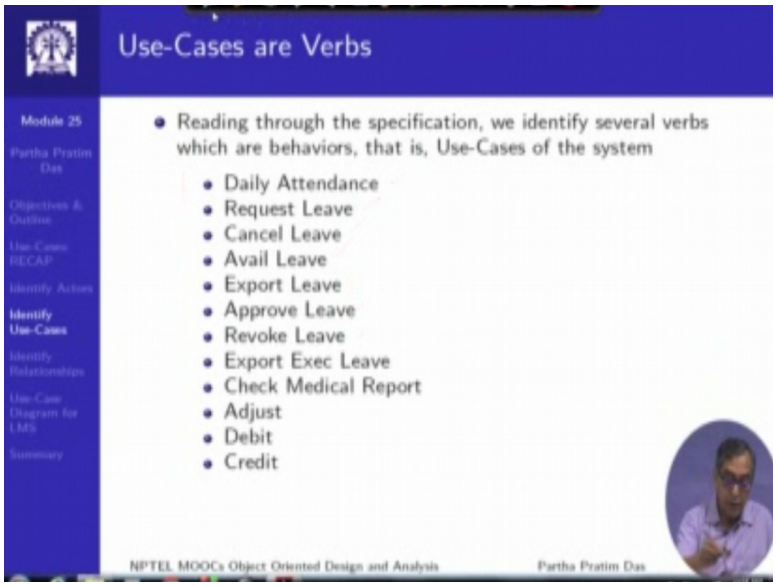
So on this we can see that these are different verbs which approve, regret, request, revoke become use-cases. In fact, you will see that almost all of this verbs in some way or other

represent use-cases, now certainly it is a matter of judgment of the designer of the modeler as to what you want to explicitly say is a use-case and what you just want to ignore. As an example I would just draw your attention to the verb work.

So at the beginning of the specification we have a very small reference that employees work in the organization and of course which we understand also, why are employees there in an organization unless they work. But the question is in terms of modeling the LMS system do we expect to see, a use-case called work, I would say possibly not. Even though work maybe the most important use-case for the organization to survive or that is where most of the actors, particularly the human actors would be involved most of the time.

But from the perspective of the leave management system, working is not the matter of importance, working or idling or how some employee, how productive an employee is, are factors which may be important for the organization, but in terms of the lift system, the verb work may not associate with the use case. So there is a whole lot of judgment and re-understanding revision of the specification that is involved in actually carrying this out carefully.

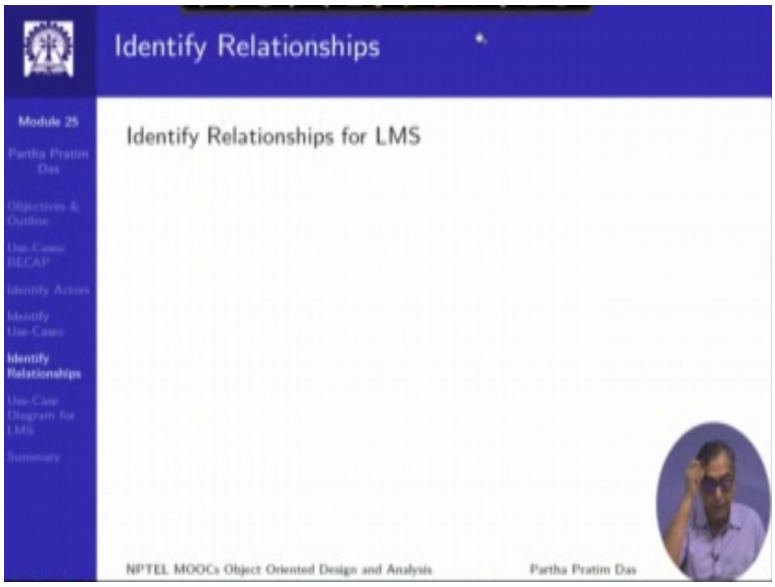
(Refer Slide Time: 16:46)



So if you read through then even after that elimination you will come up with possibly these set of verbs which should be considered as a functionality of LMS system for use-cases like the Daily Attendance process, requesting leave approving leave, cancelling leave, revoking leave, availing leave and checking the medical report, checking other reports and various other supporting use-cases.

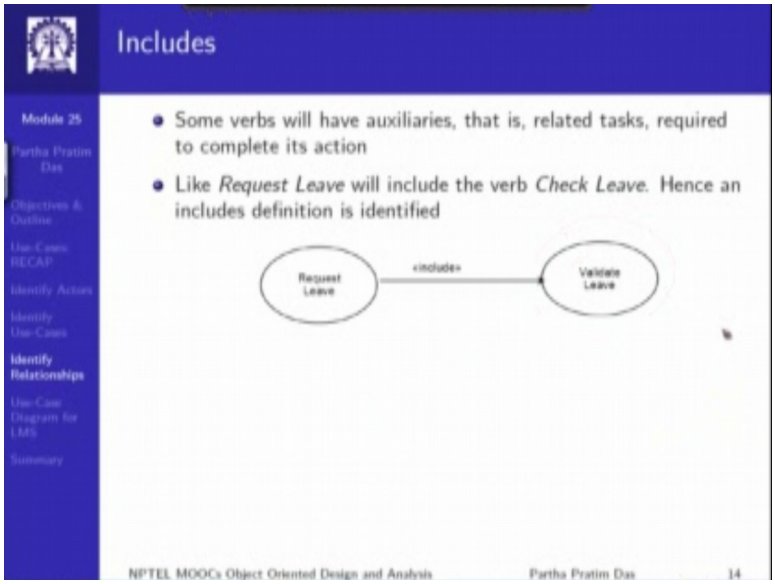
This may not again we can think of that, some are important use-cases like request, approve and so on and some are supporting ones like debit credit which also must happen but it is not primarily visible use-cases of the system. So we will come up with this kind of a list of verbs which could be termed into use-cases.

(Refer Slide Time: 17:52)



Next we need to identify relationships.

(Refer Slide Time: 18:07)



The process of identification of actors and use-cases by analysis of noun and verb and the understanding of the specification, will by itself give rise to a good insight into how the relationship should be looked at. So relationships are primarily of 3 kinds, include extend and

generalization. So for include, we will need to look at that for each and every use case that we intend to put, whether that use-case is completely self contained.

If it is not, we need to conceive whether there could be other use cases which is required to complete the functionality of this use-case and this analysis of whether they functionality needs to, be completed with another use-case will give rise to the include relationship and mind you, while you do this exercise, it is also possible that for example if we just just let me for for once go back to the verb used cases, so these are the used cases that we have seen, now in this we do not have say a use-case, called say validate leave. I think we do not have this.

We have all this different use cases and if you actually read the specification why we did not have this, because in the specification, this has not been explicitly stated and so when we look for the verbs we did not see a verb like validate or check occurring and therefore we missed that out. But once we tried to look at the functionality and we tried to look at the functionality of request approve, then we find that how can I request for a leave unless that request is a valid one.

For example, if if I am depending on the type of leave, I am applying for example, if am applying for casual leave and if it spans over 4 consecutive days, then we know from the specification that it is something that cannot be approved. If I am applying for a medical leave, sick leave and I am not supported by, valid reports, then it will not be sustained. So there are different kinds of conditions and sub tasks that will need to be done.

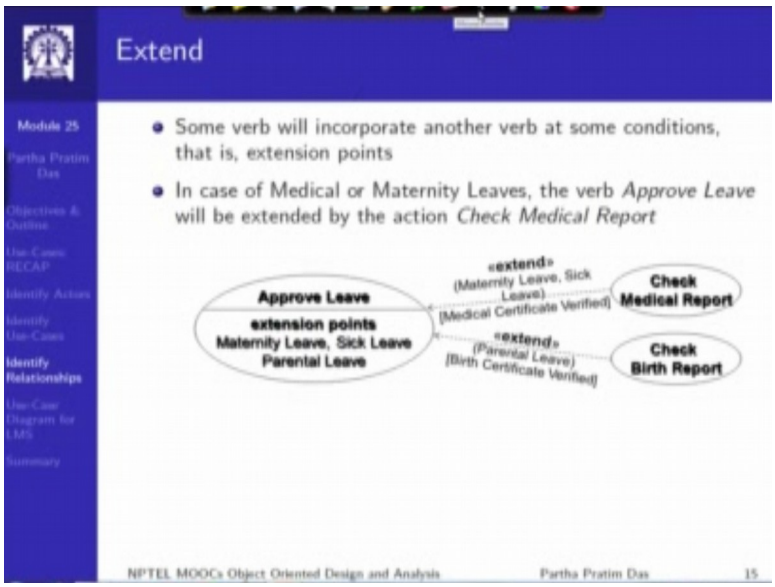
So if you request for a leave or if you approve, want to approve a leave, then you will need to check if that leave under consideration is valid or not. So this is a use-case which we discovered in the process of analysis and it is not explicitly stated in those terms in the specifications. And we try to put those as include here, so I would just like to remind you that the include relationship necessarily is the best handle to provide re-use.

That is if you do not, if you identify less number of include relationships because you may not find them explicitly in the specification document, the consequence could be that request leave has to put all that cord, all that action required for validating a leave and the approved

leave will have to duplicate all of those again as its own functionality whereas actually they are the same functionality of validating the leaves of different types.

So the reuse will be enhanced, if we can identify such use-cases which can be included in one or more multiple cases and can improve the re-use and bring a better modularity to the system.

(Refer Slide Time: 22:05)



Second is the situation of extend, which is deciding that in terms of a use-case whether we have conditional behavior. So what we are trying to do, we are trying to say in this case trying to look at approve. So I want to make approve a use-case which is a very valid justifying one that is a basic functionality, so that will be a use-case.

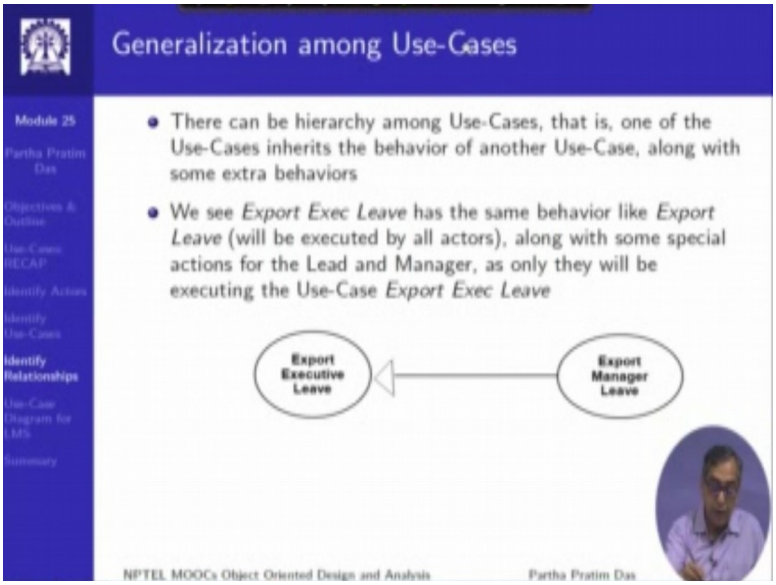
Now I think that if somebody has to approve, I go back to the document, if somebody has to approve, can the approval be done blindly. Now mind you this approve leave has already included a validate leave, so the system has already validated that the leave according to the basic structure of leave duration, leave type, availability of leave to that particular employee and so on has already been changed.

So is it that for all leave, if it is a valid one, then the lead can simply blindly go and approve it provided the lead is okay with the absence of the employee during that period or there needs to be further use-cases to be used. And we find that some of the leave have other conditions attached and that is the situation of medical leave, the sick leave as we mention there, the maternity leave, parental leave and so on.

So if I want to talk about approve leave it will have to for the case of medical leave or sick leave, for the case of parental leave it will have to involve further conditional processing which is not applicable to approve leave of other kinds and that is the kind of situation which gives rise to conditional optional use cases and that is exactly the extend behavior.

So in terms of identifying the extend behavior you are specifically looking at, what are the what are the situations where a general processing or a general use case action, may require very specific sub use-cases to be identified and used as extension on a conditional basis and that is the example. We had discussed this particular example earlier also, but that is the process that you do. The identification of conditions, the process that you do to extract the extend relationship in the model.

(Refer Slide Time: 24:41)

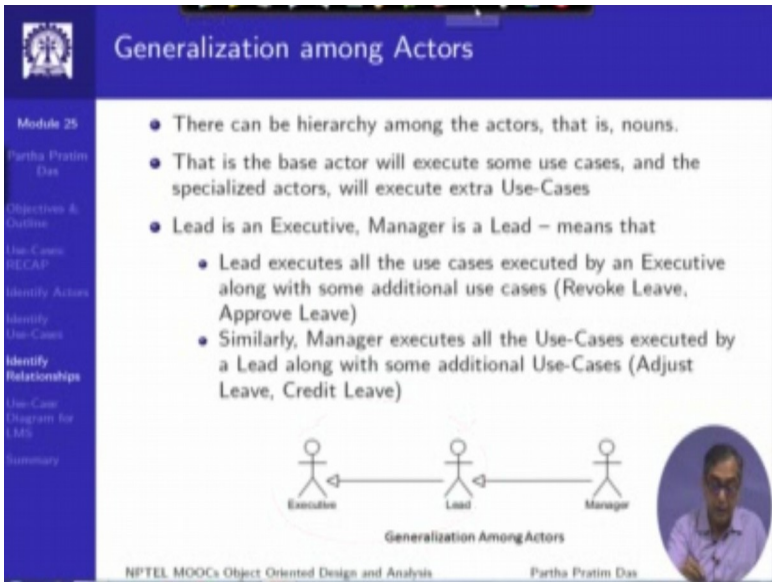


Generalization is certainly everywhere. Generalization of use-cases will look for, that once you are have identified first should be that you try to identify as many use cases as you can and then you try to see that between different use cases if there is total subsumption or total commonality of behavior as you may have in terms of exporting the leave for an executive and the exporting the leave for a manager.

You see that this behavior of this use-case is totally subsumed in the behavior of the export manager leave or in other words export manager leave needs to possibly do few things extra but otherwise it is like the export executive leave and like in the way we try to establish the generalization specialization between classes in a very similar way we try to establish the

generalization specialization between use-cases and we can build those up in terms of the system.

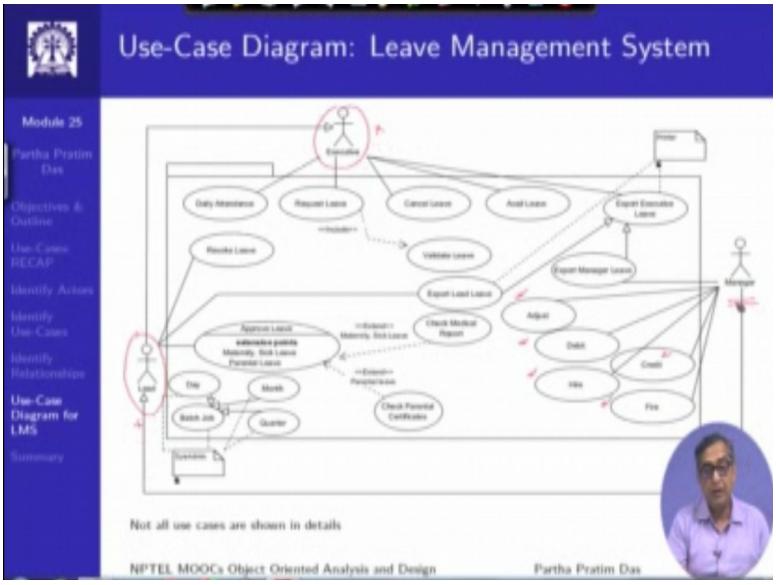
(Refer Slide Time: 25:48)



Generalization among actors are very similar. You will need to identify what actions the, what use cases the actors are associated with and if we find that the use-cases or actions for an actor, say executive is completely covered by the use-cases that lead employee, the lead actor has to perform, then he can say that lead is an executive, or lead is a specialization of the executive.

Mind you, whenever we put this generalization specialization relationship that anything that this actor will be associated within the use case, this actor would be able to perform that. So you must, if there are exceptions then it will, this generalization will not work. So you will have to be careful about doing this.

(Refer Slide Time: 26:48)



So using all this, combining all these techniques of identifying actors, use cases and relationship we finally consolidate for the leave management system. So here I am trying to show, the use case diagram with the actors the executive lead and manager and we can see that there is a generalization relationship amongst them and there are other non human actors as well.

And with that now all the major use cases have been put in place, for example an executive will go for daily attendance, can request for a leave. If the leave is requested, then request use-case will include the validate leave, so that the leave can be validated. The executive can cancel and avail leave and so on. And certainly the export the list of leave.

Now what happens is as we have seen that lead is a specialization of executive which mean that the lead will also be able to provide daily attendance, will also be able to provide request leave, will also be able to cancel leave and so on, but we do not actually need to depict them on the diagram, because since a lead is an executive, the lead will, through the executive the lead will be associated with all the use-cases to which the executive is associated.

So what we do, we just here try to associate the use cases that the lead can perform, but like revoke leave or approve leave, certainly the executive in turn cannot be associated with this use cases. So we have the approve leave, and as we had earlier discussed the approve leave had couple of extension points, so it extends and it needs to check for medical reports in case of SLML it exchanges and check for parental leave condition and the parental certificate for this.

So all these are being put together and then if we look at the manager, we find that the manager also is a lead. So which means that the manager can actually perform all the, can be associated with all the use-cases like approve leave, like revoke leave, as what the lead can do, and since lead in turn is associated with request leave with cancel leave and so on, the manager also will be able to.

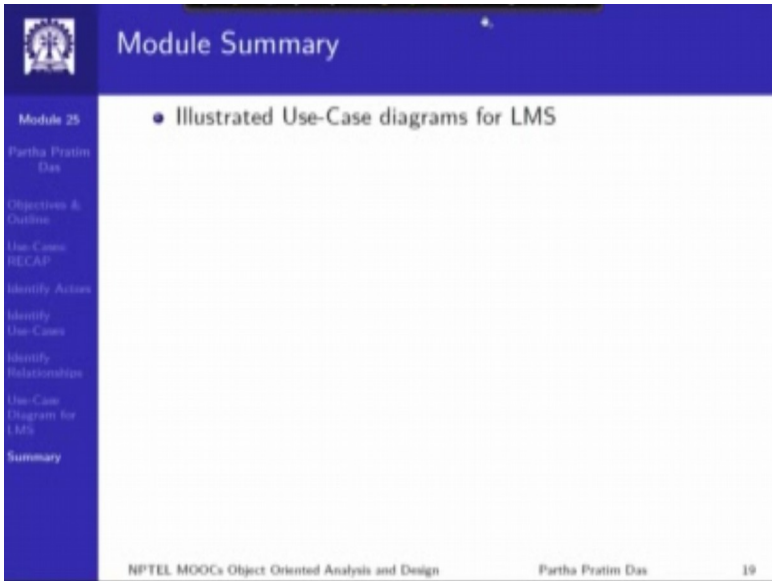
So manager basically kind of get associated with these use cases to which directly in the diagram the executive is shown associated in two levels, since lead is an executive, lead is gets associated with request leave and since manager is a lead he in turn will get associated with the request leave. But specifically manager can also be associated with number of use-cases like adjust, debit, leave, credit leave, hiring firing of employees and so on, which neither the lead not the executive will be associated with.

So these are the exclusive associations of use-cases with the manager. Naturally manager has separate use-case to exports managers leave. Lead has separate use case for that. And both of them are specializations of export, executive leave because executive lead does not need to take care of the reporting employees where as the lead and manager cases need to do those.

So this kind of puts the different pieces that we are doing together, But I have mentioned here, please note that all cases in all details are not given in this diagram because that would have been very cluttered in one slide. Later on we would like to publish the more complete solution of use-case at other UML diagrams on the LMS system.

So you will be able to see more detailed design at that point of time. Mean while I will urge that you follow the LMS specification and work out., the use case diagram in full depth. So that you can really appreciate how the modeling is happening and when we finally, later on when we published the solution we will be able to compare and check whether you have been on track or you have been different.

(Refer Slide Time: 31:56)

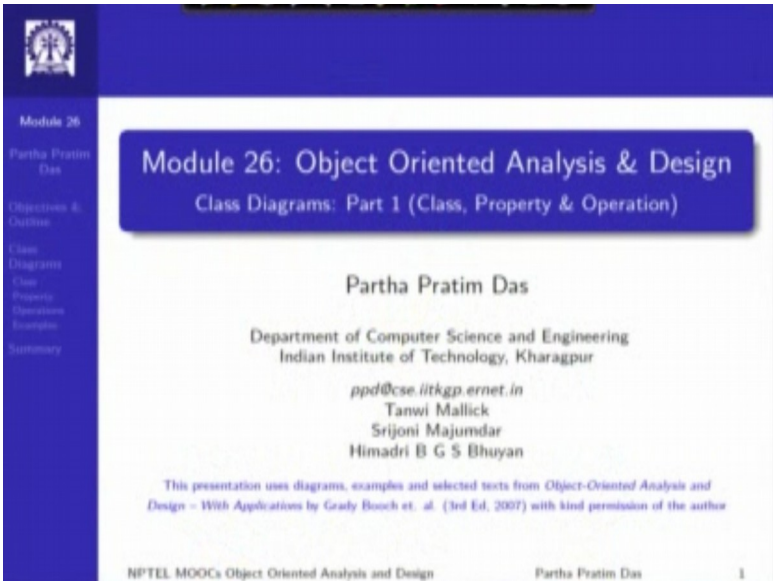


So to summarize, in this module we have looked tried to take all the different features of the use-case, diagrams and suggested outline a methodology, to extract information appropriately from the specification and populate the diagrams for the case of leave management system which you should use for your practice. And we have also published the assignment management system, on which assignments have also been given. So you should can also use that and start practicing on the UML diagrams.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

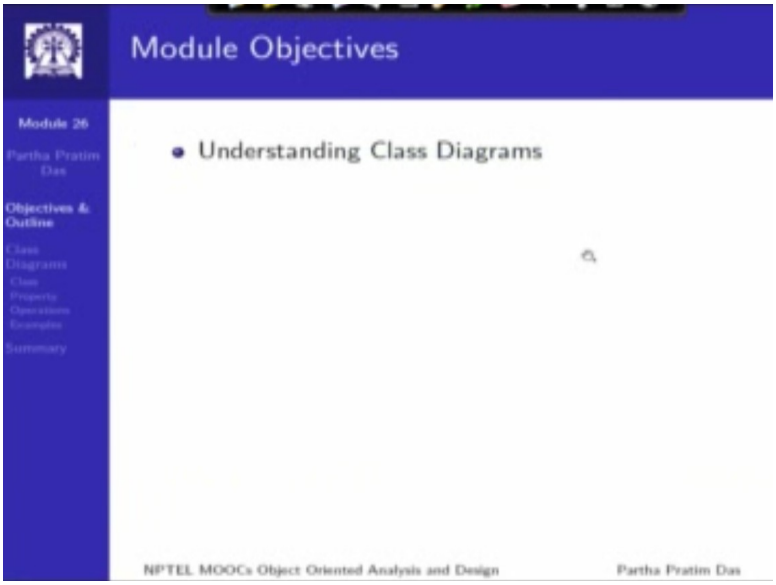
Lecture 38
Class Diagrams: Part I (Class, Property and Operation)

(Refer Slide Time: 00:26)



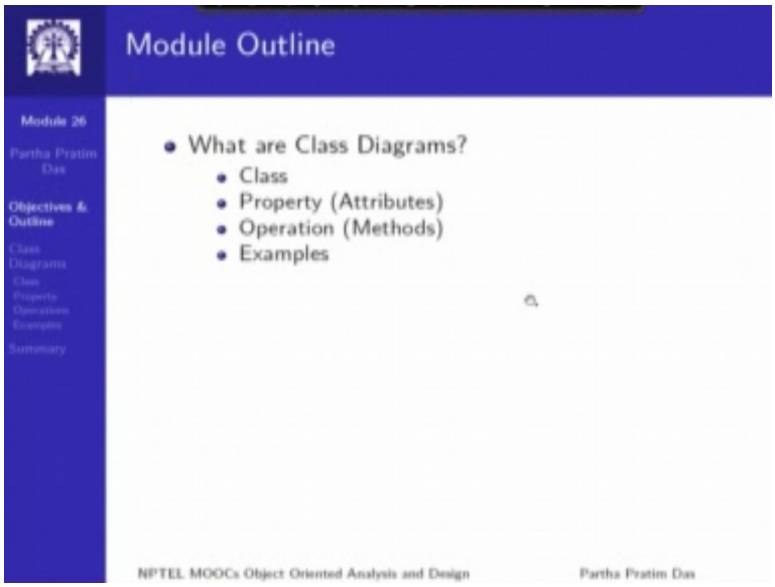
Welcome to module 26 of object oriented analysis and design. We have been discussing about UML diagrams, introducing variety of diagrams. We have already discussed about the use-case diagram at length. Next diagram that we take up is the class diagram which is possibly one of the most important diagrams in the presentation of a system.

(Refer Slide Time: 00:56)



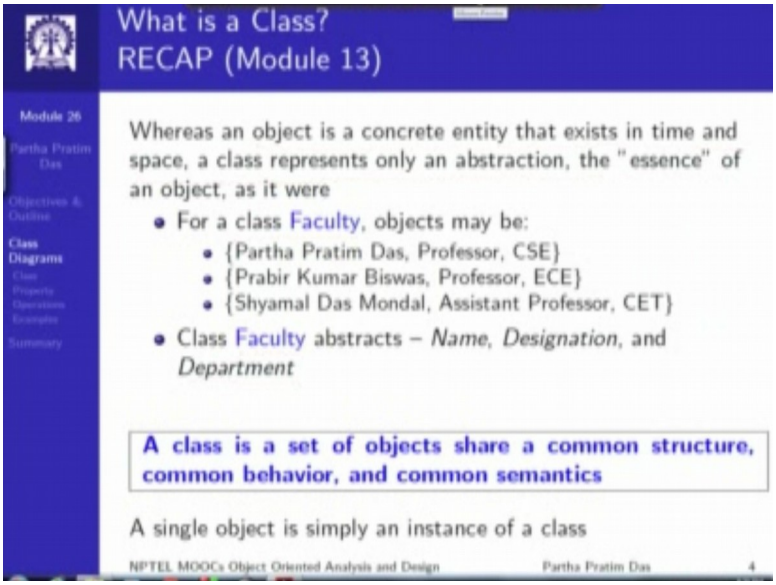
We will, in this module start the discussion on class diagrams and this will go over to the next two modules as well.

(Refer Slide Time: 01:07)



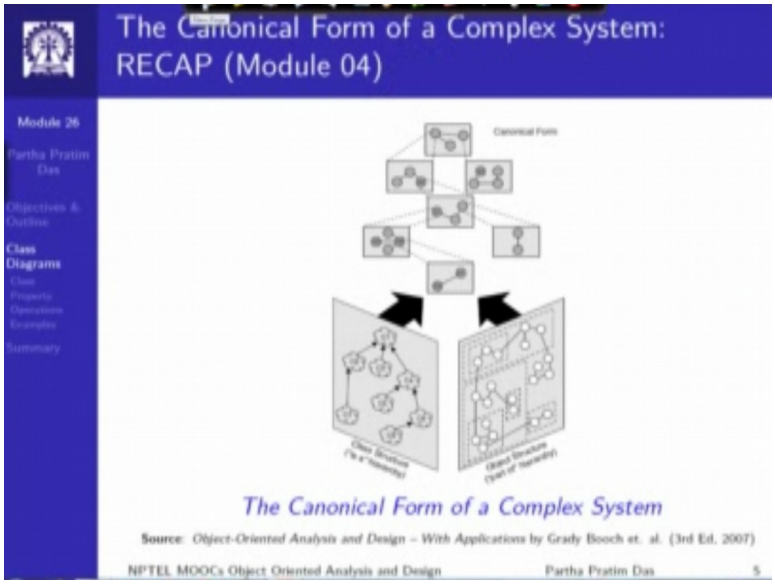
In this module, we just specify what is a class diagram and discuss the basic representation of class, property and operations. We will take up examples as well.

(Refer Slide Time: 01:26)



So before we get started let us quickly recap on what is class, so we have discussed the concept of class and objects at length, so a class represents the generic footprint or the generic layout of objects that exist. The concrete entity that exists are objects and we have seen that for the class like faculty, this could be the object instances and class as an abstract will have different properties or attributes that define this class.

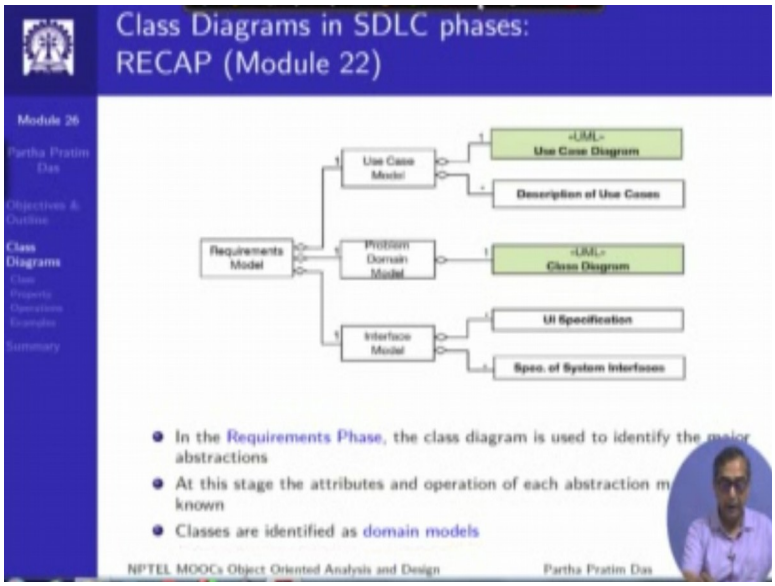
(Refer Slide Time: 02:14)



We have discussed earlier also about canonical form of complex systems, where we observed that a system could be looked at in two distinct views, one is the IS A hierarchy or generalization specialization view which is primarily the representation of the class structure. We have also seen that it can be looked at as a part of hierarchy which is the HAS A relationship which represent the object structure.

Class diagram is unique in the UML representation because it provides us the mechanism to define the part hierarchy the part of hierarchy or the object structure and the IS A hierarchy of the class structure together in the same diagram. So we will take a look in terms of how this is done.

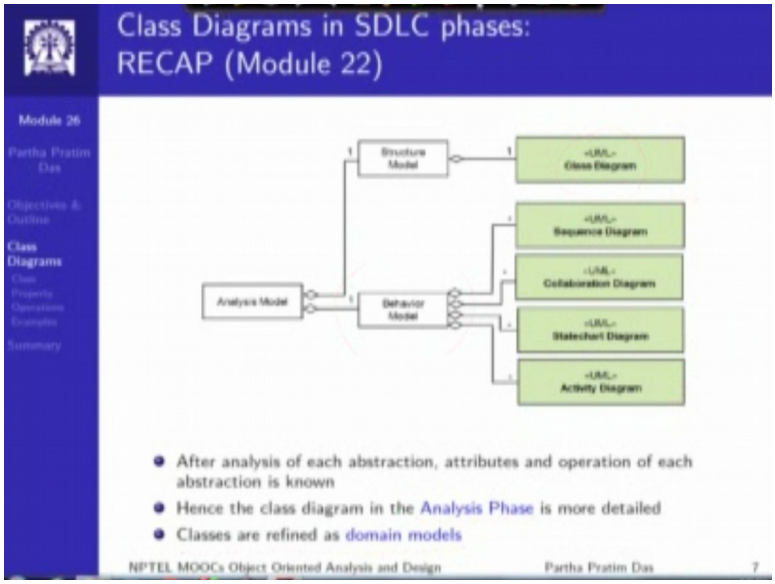
(Refer Slide Time: 03:27)



I would also like to remind you that we have seen that at different phases of the software development lifecycle or SDLC, different UML diagrams are involved and right from the beginning that is in the requirements phase, the class diagram along with the use-case diagram come in and start playing an important role, when you talk about class diagram at the requirements phase it is often that the attributes and operations, the properties and operations need not have been very clearly defined.

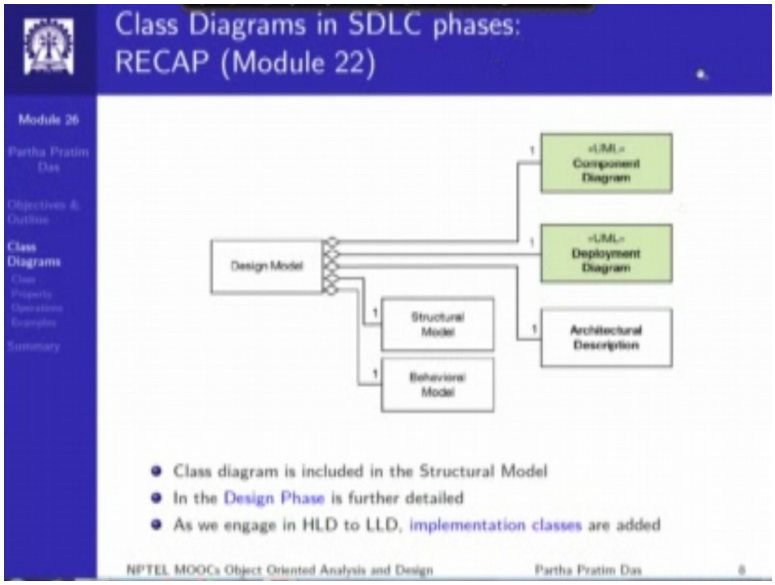
May not have been totally identified. It may be more that we have the abstraction, the concept of a class that we have been able to identify and we are still looking for, not maybe we have some of the properties and some of the operations but we may not have been able to complete that whole thing. So here the classes are being identified as domain models because we are trying to capture what the domain has. This is where the class diagram is supplementing the use-case diagram.

(Refer Slide Time: 04:53)



In the next SDLC phase which is the analysis phase, we analyze the class diagrams that we have captured earlier, tried to refine them further, continuing to represent the domain models, represent the information that the problem domain has and build up the UML representation of the problem along with several other behavioral models which we will discuss in subsequent stages.

(Refer Slide Time: 05:32)

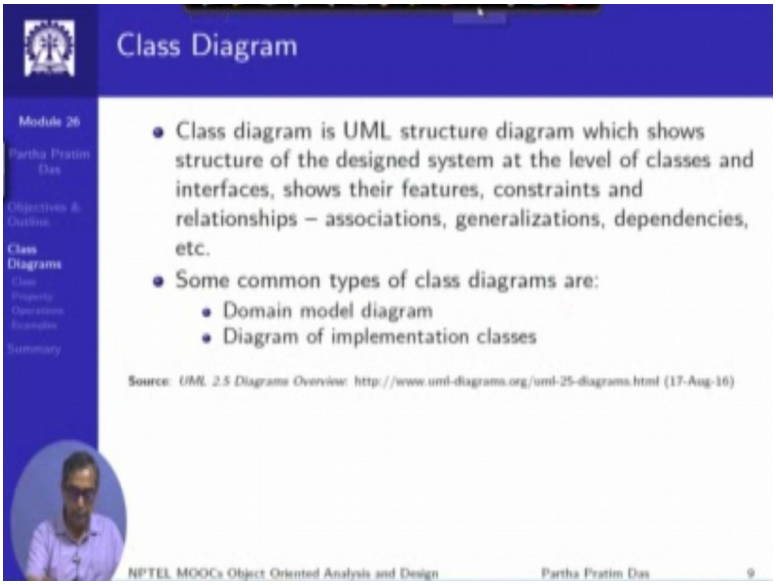


Further in the SDLC, the next phase talks about the design, called the design phase, where we have seen that the design is further detailed in terms of high level design and low level design and possibly at this stage we are making use of the class diagram. The class diagram belongs here, the structural model and these are the new additional diagrams that are coming in.

Now in the design phase the class diagram is a further refinement of the domain models and in addition you may have implementation classes. That is now, we are in the design process, so we are in the output of this would be actual classes that we would start coding possibly in C++ or in Java and so on. So it is likely that besides the domain models we will also have specific classes added to the design which will talk about the implementation aspects of the whole design.

So the class diagram as we see, play a role in all these 3 phases very critical. In other phases also it will be required. But usually the development of the class diagram will remain limited to these few phases alone.

(Refer Slide Time: 06:57)

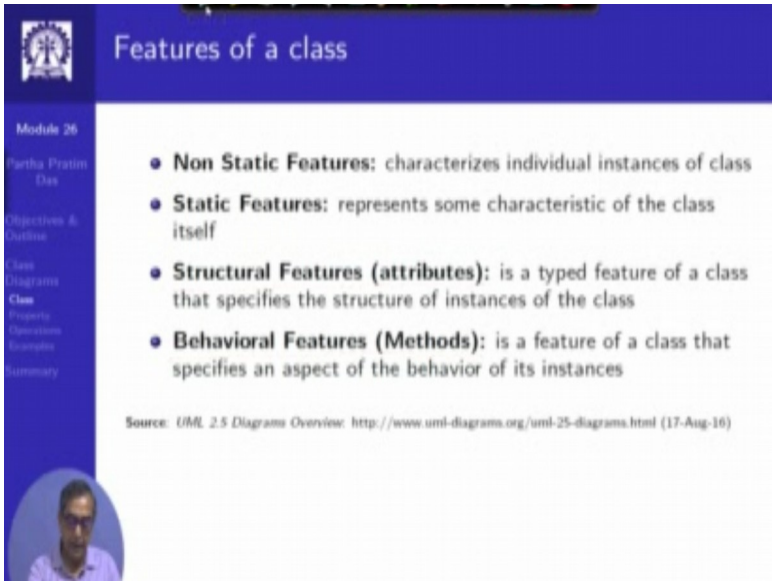


So given that a class diagram is an UML structural diagram. You will recall that we have talked of structural diagrams and behavioral diagrams. Class diagram talks about the basic structure of the design system at the level of classes and interfaces. Interfaces we will talk about more. Interfaces are are like classes where we may not know the details of the properties that are inside, but we know just the operations that we want to satisfy.

Interfaces could be generalizations of various different class concepts and so on. Besides that, class diagrams will show the features, the constraints, the relationship, this is most important and we have talked about variety of relationships in terms of association and generalization, dependencies and so on and we will see how all of these in terms of the class diagram can be represented. Typically, class diagrams are of different kinds.

Most of them are either domain model diagrams where basically I am capturing the domain information from the problem which will significantly happen in the requirements as well as analysis phase and the other type they are the diagrams of implementation classes which will be added further as we moved from the analysis phase to the design phase. So implementation class diagrams which represent implementation classes, naturally have far more details than the domain model class diagrams.

(Refer Slide Time: 08:45)

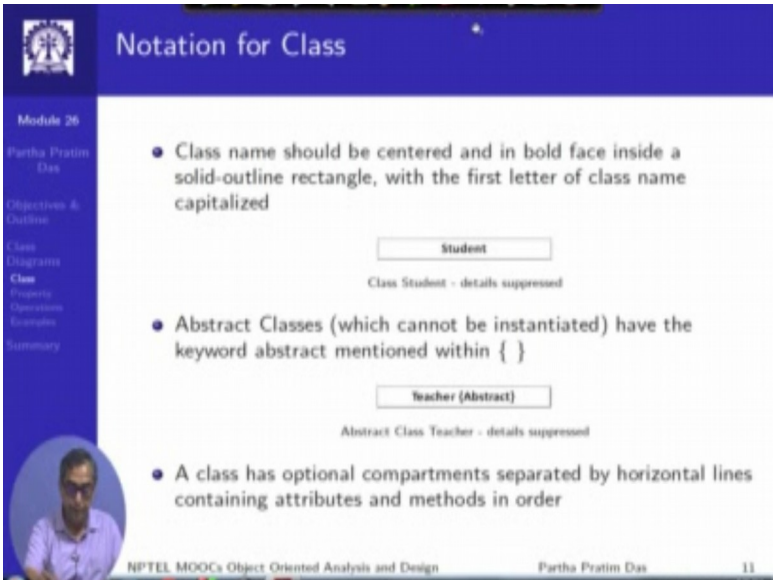


Now before we can start the representation of classes, let us remind ourselves that there are the major features of the class include that a class has non static features that is where the individual instance or every object instance of a class will have these features. So these are called non static features. We have static features where there is some characteristics, some feature may be property or operation which is a characteristic of the class itself, not of the specific objects.

For example, if we talk of that, we have a property to define count the number of instances of a class that we have constructed. Then this count will be a static feature because every object will not have a separate count because it is just one of those objects but at a class level we will have a static count property which will keep the count of the classes that have been the counts of the objects that have been construct.

Accordingly, we could have operations also which are static, for example the operation which will actually update this count when an object is constructed, will update it when an object is destructed or when you want to know how many objects of a class exist and so on. In other terms there are structural features or attributes and they are behavioral features or methods or operations that will be involved in defining a class.

(Refer Slide Time: 10:30)

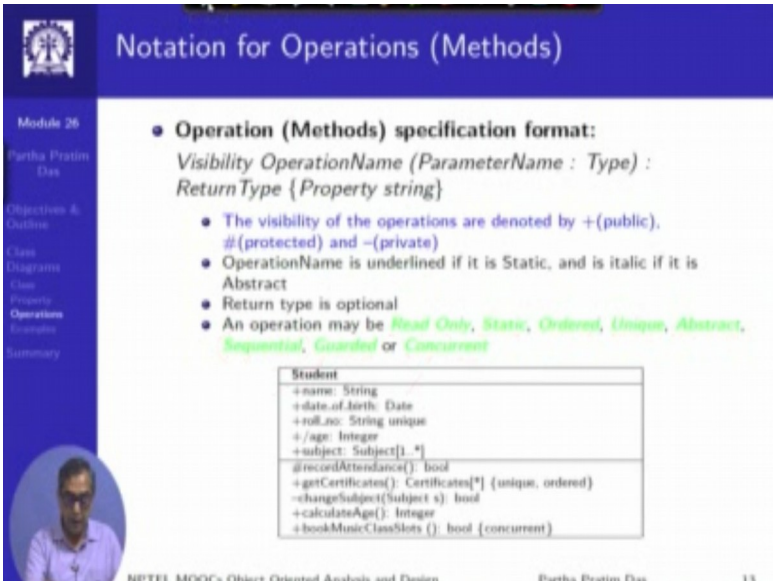


Now coming to the notation. A class will be typically depicted in terms of a closed rectangle which has a solid boundary and in between that, within that boundary, the name of the class, class must have a name and the name of the class will be written in bold letter with the first letter in the upper case that is a basic convention, so we will not write a class like this where the first letter is not upper case. This is the correct way of doing that.

Some classes could be abstract. If it if a class is abstract, if a class is abstract then it can be, we will write within curly braces after the name of the class that it will be abstract. What it means is this class only has a concept but in the realization we will never actually instantiate the class and create objects. Abstract classes in an alternate form could be represented by writing the class name in italicized font as well.

Finally, a class may have optional compartments, that is I have a class, say this is the class, the name of the class is a book, I may have further compartments. One compartment for the properties and another compartment for the operations. But these are optional I may have both of them, I may have only one of them. I may not have any further compartment at all.

(Refer Slide Time: 12:12)



Now coming to the description of properties, a property is written, so typically you can see that, let us say this is one class, this is the class name, this is the first optional compartment where we issued the properties and let us say we are talking about this particular property. So what it has, certainly this is role number is a property which has a property name. So after the property name, we put a colon to separate it from the type of the property.

So string is the type of the property. So roll number colon string says that roll number is of type string, where it is assumed that string is something which is known already. If it is not then that will also have to represent it. So those of you who may be familiar with C++ and Java will relate that we keep the similar information in a C++ or a Java class but it is just that we write them in the different order.

We first write the type name and then we write the variable name. Here we first write the property name and then we put the type of which it belongs. So that is this part. Then it is prefixed with a symbol which could be one of plus hash or minus, they mean the visibility of that particular property of that particular data member. If a property is public, it is meant that any other class can access that property directly.

If it is private it means that only the operations of that specific class can access that member can access that property but no external class, no other class can access that property. And finally if the visibility is protected then we have something in between this. If I have one class, say employee and if we have another class say manager, so that they are related by on a IS A hierarchy that manager is a employee which means that every manager satisfy all the

properties of an employee but may have some additional properties, some additional operations.

If such a specialization exists then for a protected member in the base class, the derived class or the class that specializes the base class can access this protected members. But other classes cannot access the protected members. So protected visibility tells us a different kind of access restriction where for a child class or for a specialization protected members are like public, they can be accessed modified.

Whereas for any other class which is not, a specialization of this given class, the protected members are like private. So as we define the property we also define the visibility of that property. Then we may have an optional qualifier that follow the type name which say certain specific things about that property. For example, here we are saying that your roll number is, the style is to put it within curly brace and these are the different qualifiers that I could have.

So if I say unique it means that of all the instances of this class, this particular property will always take distinct values or in other words, I cannot have two students, we are representing the student class, I cannot have two student objects which have the same value on the roll number property. So several such are possible, I can make something read only. If I make something read only, then that property can only be read, it cannot be changed, it is whatever is set at the construction time of the object will remain as such.

A property maybe optional. If a property is optional then I can have null values for it, that is it is not necessary to have a specific value at the time of instantiating objects. A property could be static so that it operates only at the class level. A property could be ordered that I can say that within the different values will have to occur in a certain order and so on. So the other way to say if a property is static is also to be able to underline that.

So if we underline that, that also means that the property is static. Now the property could have multiplicity. This is important that it could happen, I could have number of them. For example, here I have a subject, visibility is public, is of type subject which means that there is some other class subject, so we are saying that this subject property is an instance of this subject plus, but what we additionally have here is one dot dot star.

One dot dot star this defines the multiplicity which says that I can have one subject, I can have two subjects, I can have arbitrary number of subjects, of course more than one. So we can specify a fixed number here, we can specify a range like this, we can specify a range which is bounded and so and so forth. So that is the basic sense of the multiplicity.

Certainly the multiplicity by default is one, for example here nothing is written, so the multiplicity is one. So there is only one data part property here. So if we do not say anything it is one and of course that is a minimum value that you can have. Then some properties like this one is written with a forward slash before the name.

So you are saying, this forward slash means that this is a derived property, which means that this property by itself will not be stored but there will be some operation through which the value of this property can be computed whenever I want to access it. If you try to see the reason, we are saying this property is age. For several applications, I will need to know what is the age of a person, age of a student.

Certainly we cannot store the age because certainly with every passing day the age is potentially changing. So we cannot update that data. So instead what we do? We store the date of birth and we will possibly have some method, some operation like say calculate age which will be associated with this derive property age. So that whenever we want to access the age property, this operation will be invoked.

This operation will consult the date of birth, consult the date that is today and based on that compute the age and give us the value. So in terms of the view of the object we will see, as if there is a age, but there is nothing be actually stored. So this is the concept of derived property and we will see the use of several derived properties in the design. Moving on, the next that we need to specify is certainly operations or methods.

So they written in this form, say every method has a name and the fact that it is a method is shown by having a pair of parenthesis after that. It is separated by colon from the return type. So this says, this will, get certificate is a method in student and it will return certificate. Record attendance is a method, is an operation and it will return a bool. So again you can see that compared to the common object oriented programming languages like C++ or java, the style here of expression is little different.

We first write the method name and then we specify the return type and the return type itself is optional. So I may have a method which does not have a return type, which basically means that it is not expected to return any value. So if you think in terms of C++ then it's return type would be coded as void. Methods have similarly different visibilities and the visibility notation is same as of the attribute.

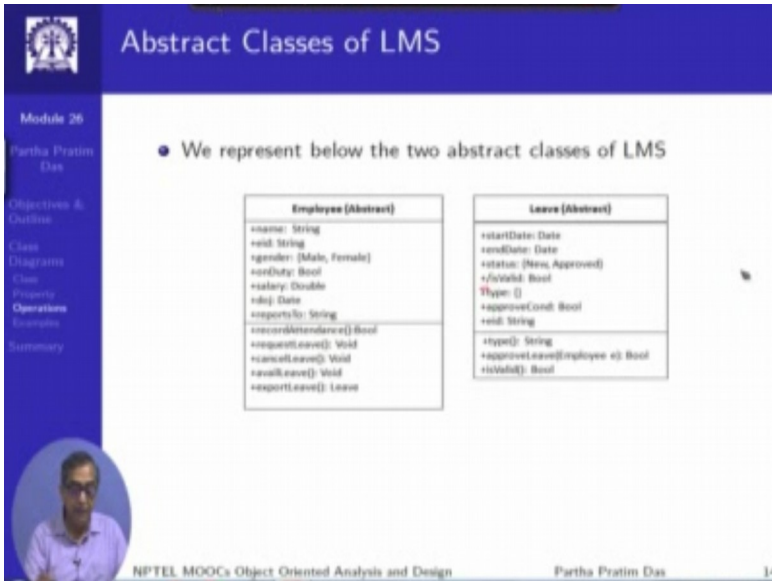
So they could be public, protected or private. Methods can be qualified by different qualifiers, like we are saying get certificate as a method is qualified by unique and ordered which means that this method will produce results which are a number of certificates. You can see the multiplicities star which means any number of certificates, zero or more. But what it says is that it is unique.

So if it returns a list of three certificates then these 3 certificates will have to be unique. They are ordered. So there is some ordering between them. For example, the ordering could be the level of education for which the certificate is provided, the 10th standard certificate, the 12th standard certificate, the graduation, the under graduation certificate and so on. And these are the various different types of qualifiers that are possible, ordered, unique, abstract, sequential, guarded, concurrent.

These sequential, concurrent, these are qualifiers to take care of concurrency in the system, so that there are situations where a particular operation maybe invoked by more than one object at a time. So if you say that a method is concurrent then it is possible to invoke it by multiple objects at the same time. But if you say it is sequential, then it necessarily means that you will have to do one and then you can do the next.

It could be guarded, operation could be guarded where in it means that the operation has a precondition, that is I can invoke this operation from an object provided certain guard condition is satisfied. So these are the typical ways to denote the operations or methods of a class.

(Refer Slide Time: 24:26)

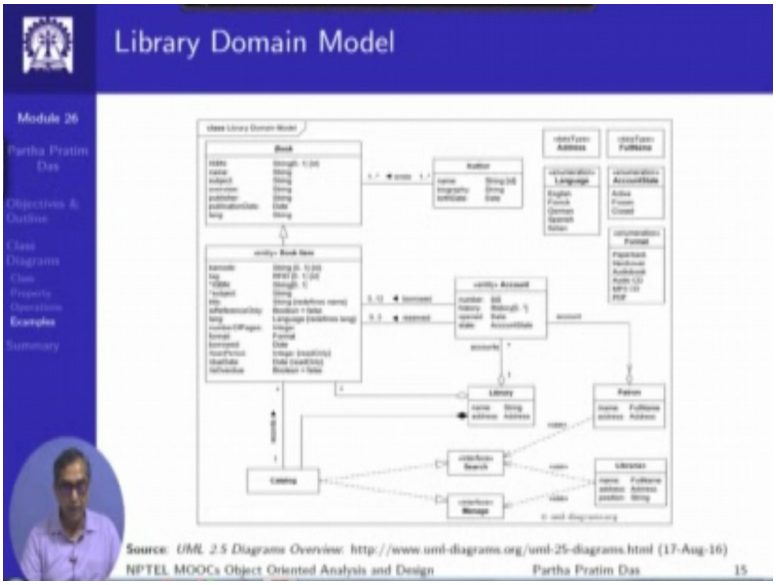


So if we just refer to our LMS example. So LMS as we have seen have two major abstract classes employee and leave. There are specialization which we have seen earlier. We are not talking about that. We are just talking about the abstract classes, so there could be several properties of employee name, employee ID, gender, salary, the date of joining and so on and there would also be several operations like recording attendance, cancelling leave, requesting leave and so on.

And we have seen how this work, how this can be extracted and here we are just making a representation for that. I would draw your attendance to this. For example, for leave we will have something called, Is Valid. So if Is Valid is true then that leave can be processed. If it is false, then that leave cannot be processed. Naturally we do not expect that while the leave is created, somebody is entering a value for its value.

But this will relate to some operations say validate leave which can have the result of a Boolean which will be set when I try to access this Is Valid property. Therefore Therefore, you can see this slash here which say that Is Valid is a derived property of the class.

(Refer Slide Time: 26:06)



I have also included here a couple of other examples which we can just take a look at, for example here is a library domain model, that is this is trying to capture the information of a library. So there is a book class which has all this different properties of the name of the book, the subject of the book, the publisher of the book, the ISBN number of the book and so on. There is another class book item.

So the difference between book and book item is, so when I say that we are going to follow the object oriented analysis and design by Graham Gooch at all then we are talking about a book. But then when I study, I need a specific book that I own. When you study you will have specific copy of Gooch's book that you own. So among, if 5 of us have 5 copies of Gooch's book, all of them will be represented by the same instance of the book class. But each one of these copies are distinct.

So there will be some more information like say barcode which will report on each copy to distinguish one copy from the other. There could be RFID tag and so on. And since this book items specific copies can be loan from the library, so there will be certain derived attributes like loan period, like due date.

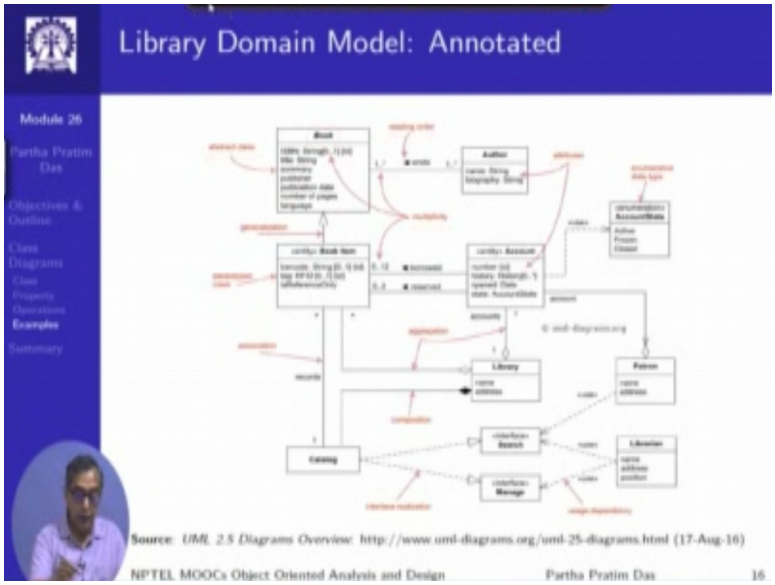
So you do not specify the due date but what you keep is possibly the date on which the book has been issued and you keep information in terms of what is the issue period that the library allows, based on that whenever I try to access due date there will be some method which will compute the due date property and give me that date. Similarly, there could be, so these are all different derived properties that we have.

You can see that this is this is in a sense a little bit simplified model because we are talking about the domain, for example it is not showing the visibility specification for this different attributes because that will get decided based on getting all the different classes together and when we go towards actual implementation then we can decide as to what is the what are the properties that we want to hide, what are the properties that should be publically available and so on.

But here since we are just capturing from the domain we have not yet done that exercise, so you do not see the visibility specification symbols. So similarly there are several other classes like, class defining an author which is a name biography, birth date of the author. The library itself is a class which has a name and an address whether it is the national library situated at Calcutta which is a central library of IIT, Kharagpur and so on.

So if there could be library instances coming in here, there could be librarian is another class which has different attributes and so on. So in this way you can see a whole lot of classes that are captured from the library domain to represent the, to build up, start building up the class diagram. You also of course see lot of other symbols and some of which we have casually talked of while we introduced different concepts, but we will come to discussing those, once we have actually talked of the relationships.

(Refer Slide Time: 30:04)

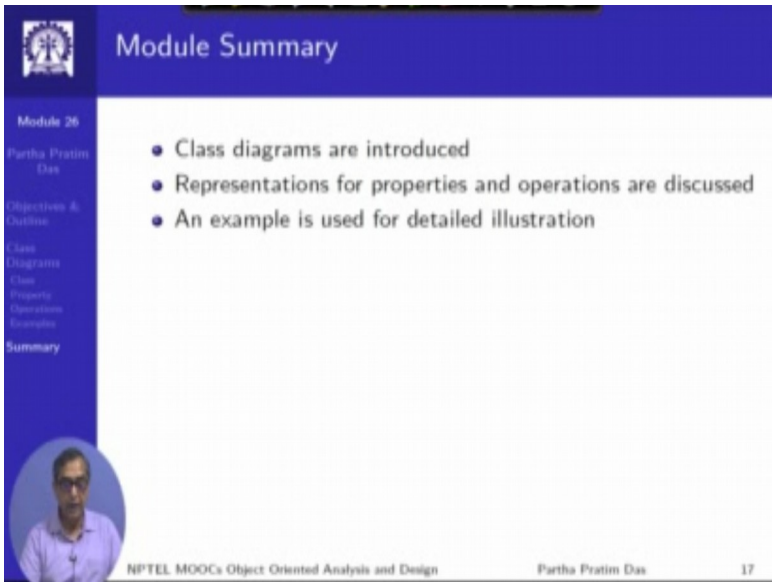


Finally, it is the same library domain model. The only thing that is provided here, that in addition to the actual model you have lot of annotations given, so these annotations, this will

basically help you prepare for your understanding, so that it specifies that this is an abstract class and you can see, how do you know this is an abstract class because you know that it is written in a italicized form.

So these are different attributes as marked here then rest of the stuff you have not done yet. So you can study this and understand and get familiar with the way the classes can be represented in the class diagram.

(Refer Slide Time: 30:56)

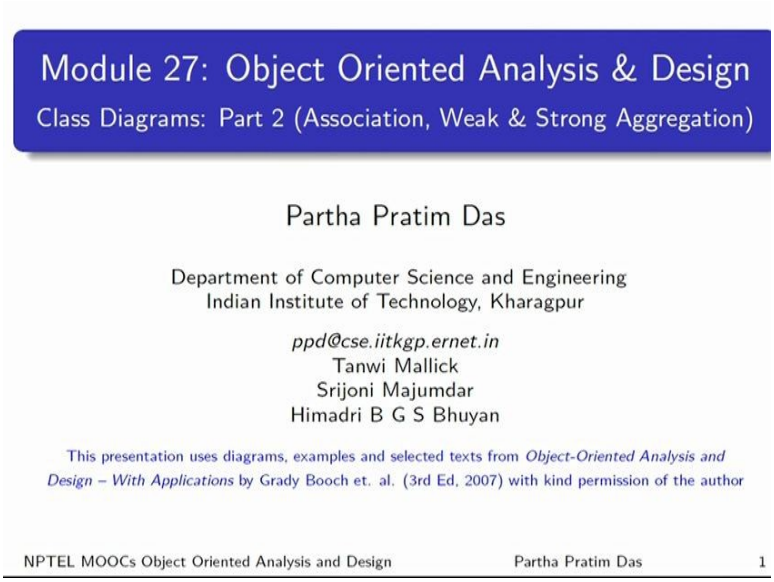


To conclude we have introduced the class diagram. We have shown the representation of properties and operations in a class and we have shown a couple of examples. In the next module, actually in the next two modules, we will talk about different relationships amongst classes that we had studied earlier, how to represent them in the class diagram.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology - Kharagpur

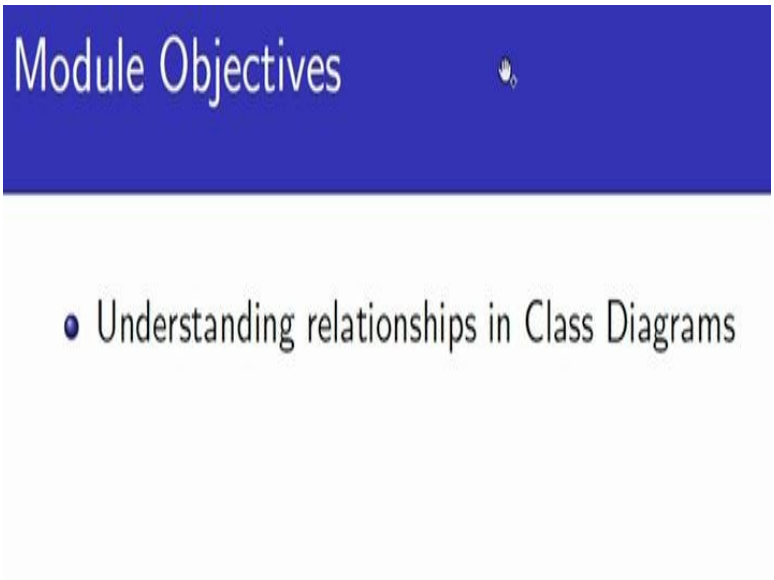
Lecture – 39
Class Diagrams: Part 2 (Association, Weak and Strong Aggregation)

(Refer Slide Time: 00:21)



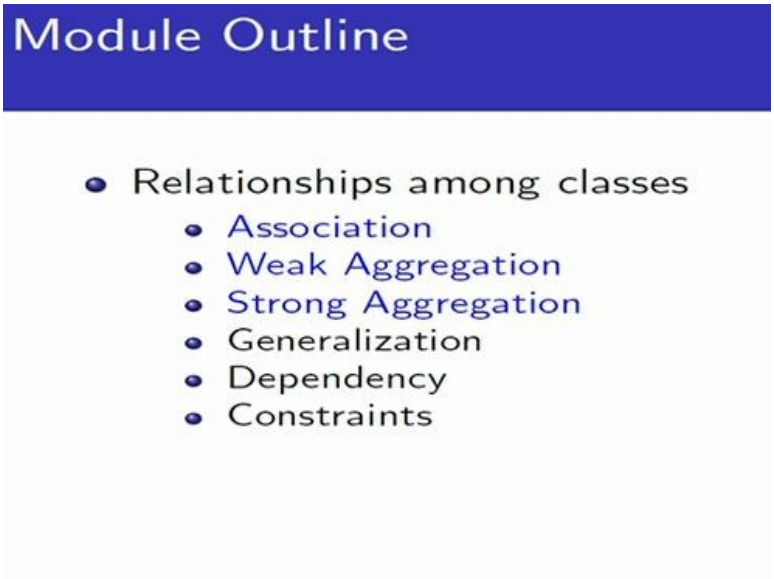
Welcome to module 27 of object oriented analysis and design. From module 26 that is the last module we have started discussing about class diagrams and in that we have introduced the notation for a class, abstract class and the notation for properties and operations. And we have seen different variations that are possible.

(Refer Slide Time: 00:50)



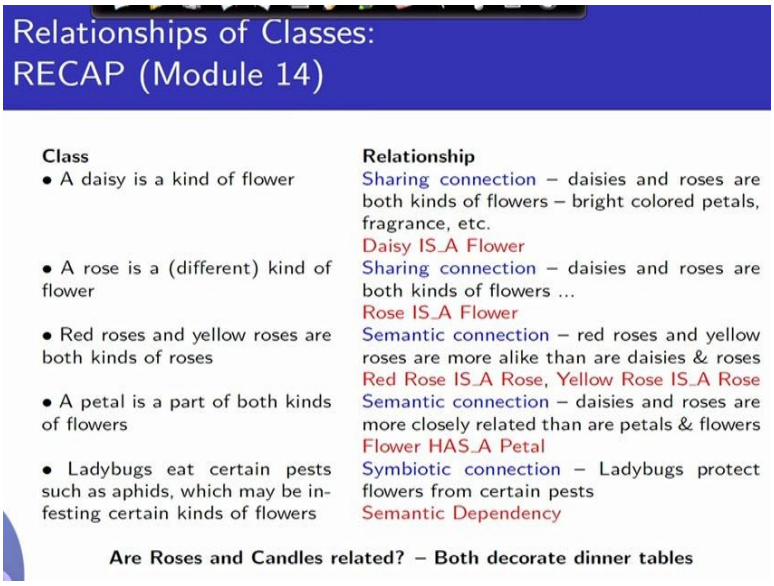
In this module and the next we will discuss about the representation of relationships in class diagram.

(Refer Slide Time: 00:55)



So we will introduce variety of representations and since this will be discussed in two modules one after the other I have highlighted the ones that will be discussed here association, weak aggregation and strong aggregation. The remaining relationships will be discussed in the next module.

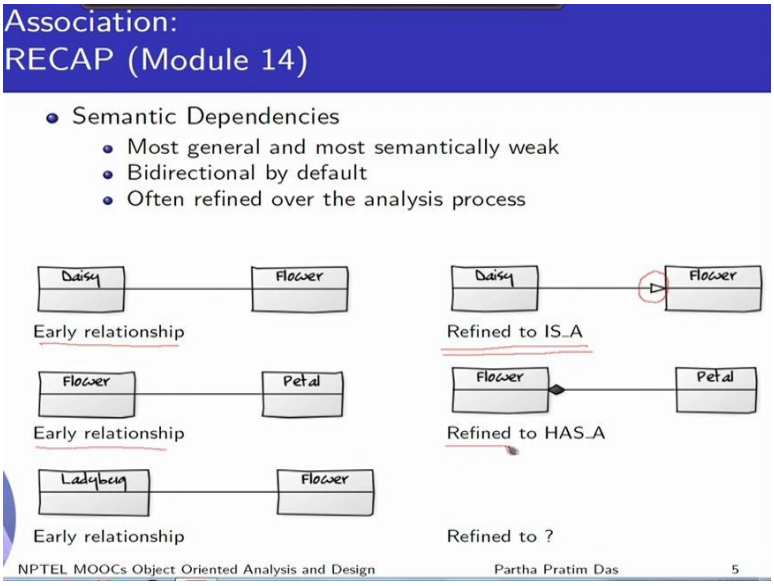
(Refer Slide Time: 01:22)



So just to recap this is referring back to module 14 where we talked about relationship among classes with this example of Daisy, Rose this kind of flowers. So daisy is a kind of flower we saw is a sharing connection which leads to IS_A relationship. We saw rose is a different kind of flower similar sharing connection IS_A relationship. We also saw that red roses and yellow roses are both kinds of rose so we identify that as a semantic connection which again leads to different IS_A relationship.

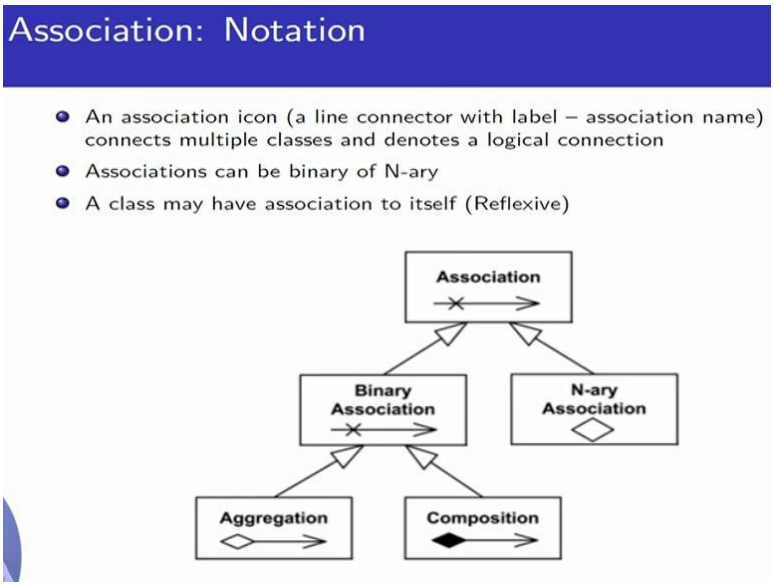
We saw petal is a part of both kinds of flower so this is a different semantic connection which gives rise to the part of or HAS_A relationship and we saw that there could be different other kinds of relationship also like symbiotic relationship as to how lady bug and flowers are related. Because possibly lady bugs protect the flowers from certain types of pest.

(Refer Slide Time: 02:40)



We have also seen that as the class, as the design go through different phases. Then initially we may just have an identification of an early relationship were we just kind of connect the two classes by a solid line and as we refine then we may be able to identify what is the actual relationship in that and may qualify that association. This is another flower and petal initially just an association and refined to HAS_A. So we will see these representations take part in the class diagram.

(Refer Slide Time: 03:18)



So the first relationship which is which is quite important is association which says that the two classes are associated in a certain way. The association has a name and actually it is not two classes alone it could be between multiple classes and if more than two classes are involved then

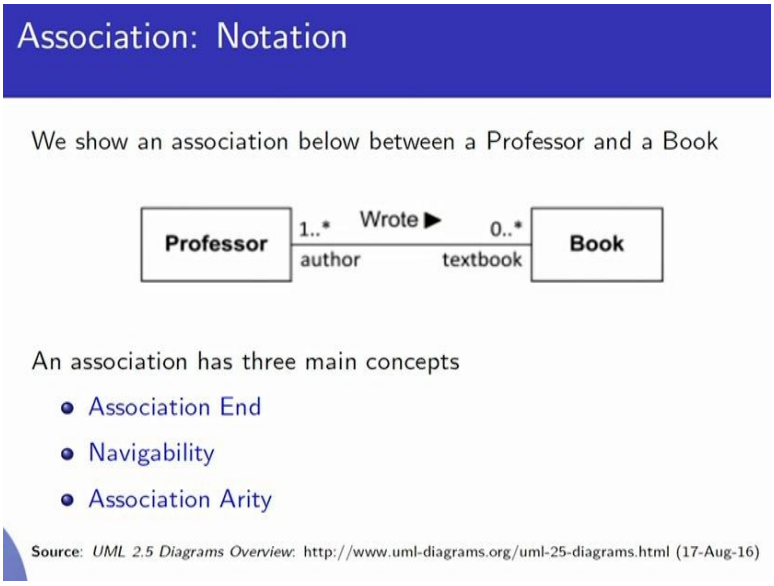
we say it is a n-ary or if three classes are involved we will say this is ternary relation/association if two classes are involved we say it is a binary association.

So associations could be of multiple different arity but usually more often we will see that variety of binary association is what dominates the representation scenario. And interestingly a class may have an association with itself. So I have a class person I may have an association like this. So it says that person is associated with person which will mean that basically there is some relationship between two different instances of the person.

For example, we will see is married say this association means married to so on this side we have husband on this side we have wife. Both are instances of person and but they are associated through the marriage. So this is a kind of a self-association or reflexive associations are also possible. And these are just different this is the kind of the UML diagram to specify the different kinds of association that we have.

So there is a this is the root concept of association and then it could be binary as I said or it could be n-ary. Binary is what is more frequent and binary association can be aggregation or composition. We have talked about these terms in earlier modules as well but certainly when we come across them in the class diagram we will again explain what these different binary association means. So this is a basic hierarchy of specialization in association itself.

(Refer Slide Time: 05:43)



So this is taking an example this how an association is expressed. So an association is the simplest form is a binary one so one side there is a class professor other side there is a class book. So if I start at the very beginning then I have professor, I have book, I join a line I say there is an association. But if I just say that the professor and the book is associated then only little information is communicated. So it comes a question of how the professor and the book is associated.

For example, it is possible that the professor is associated with book because professor reads book. It could be READS. Professor could be associated with the book because he has a book, he owns a book. It could be READS. Professor could be associated with the book because he authored a book he wrote a book. Professor could be associated with a book because the author of that book has mentioned the professor's name in the acknowledgement. So associations could be of variety of different forms.

Even when they occur between two classes so we put a name to the association. Which try to semantically specify that what is being associated with what and how? So the root is a name of the association and on the two sides we mean this is again optional. We mean see that when root associates professor with book then which role of the professor are we talking about. Because a professor could be an author, the professor could be an instructor.

The professor could be a mentor; the professor could be just acting as a faculty and so on. So here we are saying this association is referring to the author role of the professor. This association is referring to only the textbooks in the whole domain of books. So this is the basic structure of an association and whatever we write here at this two ends is specified in terms of the association end and we can have more information on this.

And the association end we could also have multiplicities which say how many instances of this class associates with how many instances of this class through this association. So what does it say what does it mean if we look at the multiplicities. On the professor side it is one dot dot any so it says that a professor is associated with a book one professor two professors, three professors any number of them which means that a book can have one or more professors as authors because they wrote the book.

This side we say that it is zero dot dot star that it could be zero book, one book, two books any number of books which say that a professor can write zero book it is possible that a professor has not written a book. So that is zero. He may have written one book. He may have written five books and so on. So in this way the different professors P1, P2, P3 different books B1, B2, B3 will have different kinds of actual relationships.

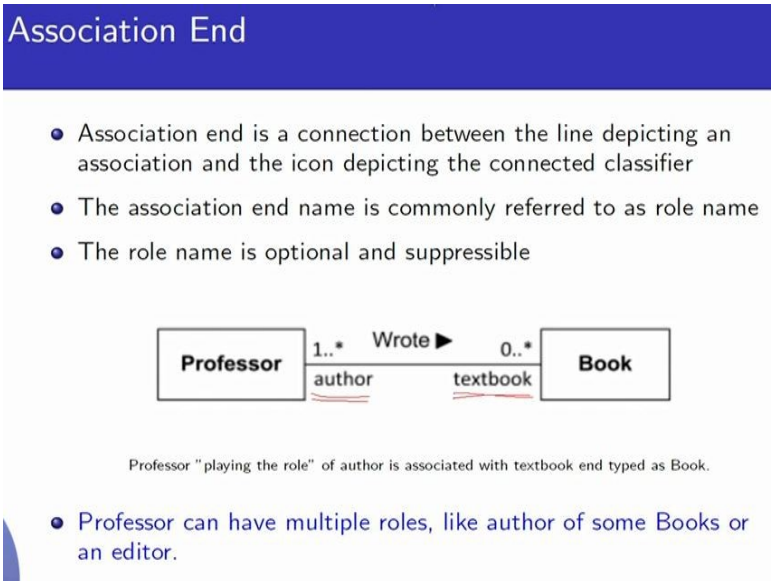
So it could be like this so it says that B1 the book B1 is written by P1 and P2. Professor P1 has written two books B1 and B2. Professor P3 has written B3 alone. So this could be all these different kinds of when there could certainly be a professor P4 who has not written any book so not associated. But we cannot have a book B4 which is not associated. Why? Because a multiplicity here says it is one.

So if I have a book it must have a professor on this side which is understandable because you cannot have a book without an author. There must be one author at least. But you can have professors who have not written books. So this is these are the different key concepts that an association has and we will see what this mean. The association end we have started talking about it.

The navigability as to how you go from one end of the association to the other and the arity of the association. By the way when you write the name of the association you normally put an arrow, filled arrow which is called the reading order of the association that is professor wrote books so he will not read this association as book wrote professor this arrow says that you have to read it in this direction.

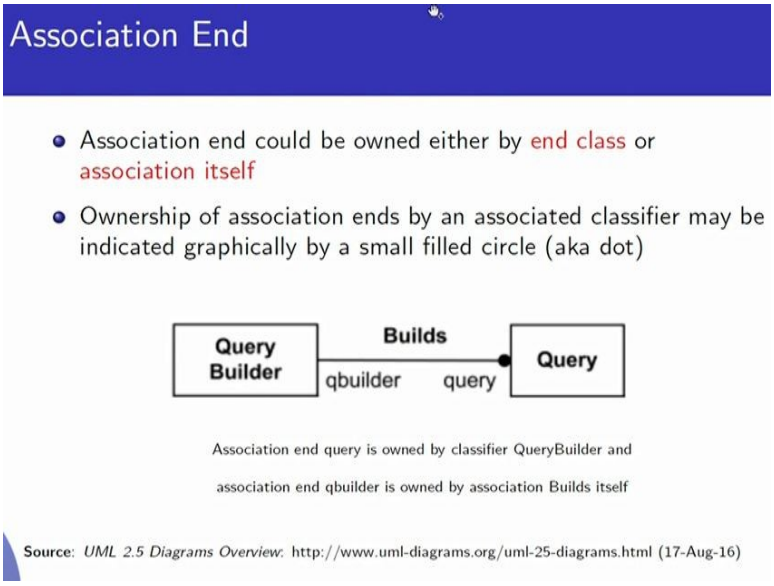
It is not again mandatory but it is usually convenient if have the reading order also marked in the association.

(Refer Slide Time: 11:17)



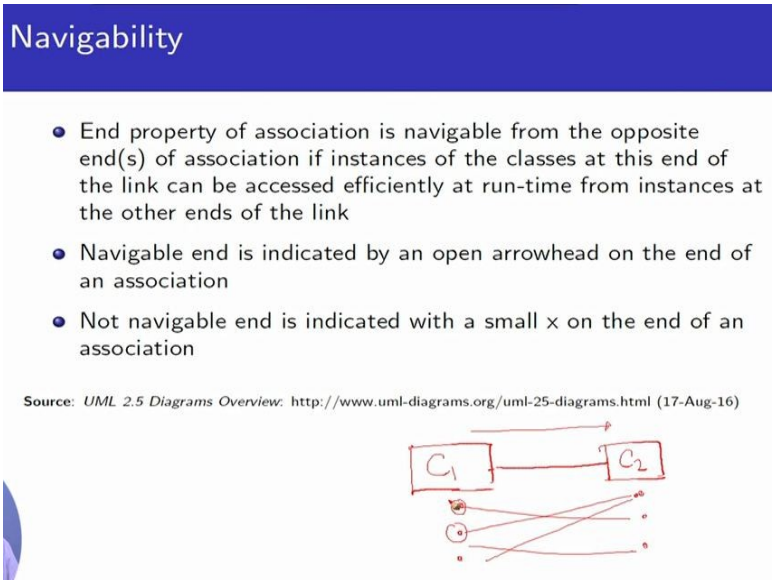
So this is a basic representation in the association end we have I have already talked about the role of the on two sides. The role of the professor, the category of books we have talked about. So this we have actually already discussed.

(Refer Slide Time: 11:38)



Now it is also possible the association could be end could be owned either by the end class or by the association itself. So that is what is representing by putting a dot at this end. So if we put the dot like this the name of the association is builds and so the association end query the name of this end is query is owned by the query builder. So query builder naturally owns that query is not the builds association itself so association end could be specified in this manner.

(Refer Slide Time: 12:24)

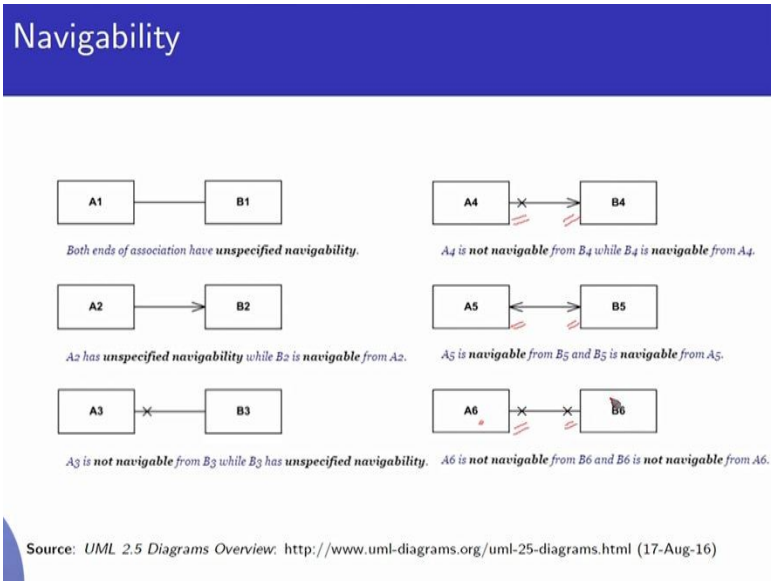


Let us see the question of navigability. That is if I have two classes associated then can I how can I go from one class to the other which means if I have objects of this class if I have the objects of this class the association or the relationship is between these objects. Now is it that given an

object here I can go to the other related object. Is it that given the object here I can go to the related object on this other side and so on so forth?

So navigability tells us which direction or which directions in which I can go from one object on this side to another object in this side. So navigability can be defined in multiple different ways.

(Refer Slide Time: 13:28)



So these are the different symbols that are used. See major symbols one is if I do not specify anything is just a flat connection then we are saying that we do not want to specify the navigability so this is called unspecified navigability. I have not been able to model as yet as to how it can be navigated. Or I can have a directed arrow so it says that B2 is navigable from A2. So I can navigate them so given some A2 object I can find the associated B2 objects.

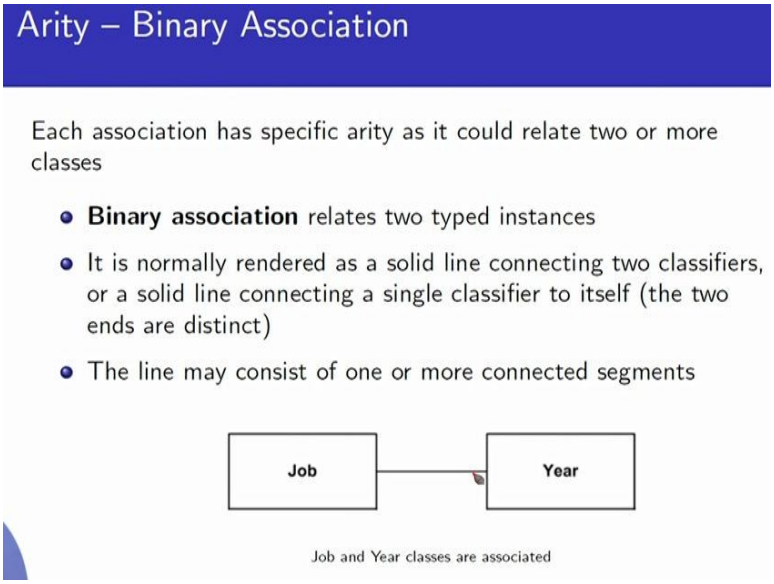
This is the basic idea. I could cross out a navigation. If I cross out on one end, then it says that A3 is not navigable from B3. Mind you this is different from not drawing anything if we do not draw anything then it is unspecified that I may be able to navigate I may not be able to navigate it is not known but when you cross out it says that it is not navigable that is given B3 objects I cannot come through the A3 objects related A3 objects to this association.

So here if you look into this this side is not navigable and this side is unspecified. Similarly, more examples are given here. This is not navigable; this can be navigated. Here it is a

bidirectional navigation that you can have objects on any side and navigate to the other. This is does not allow any navigation at all. So the object if you have an object here you cannot find the associated object through navigation of this particular association.

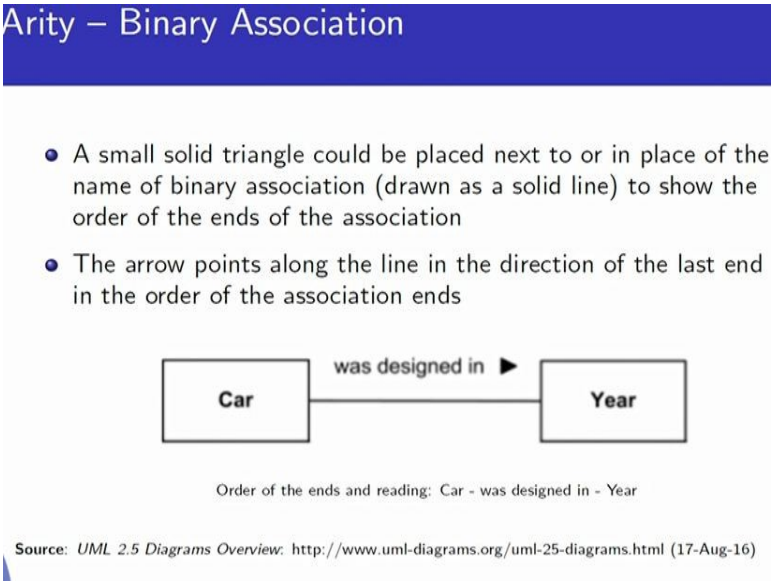
And the same is true from the B6 side coming back to the A6 side. So these are the different navigability that the class diagram allows.

(Refer Slide Time: 15:22)



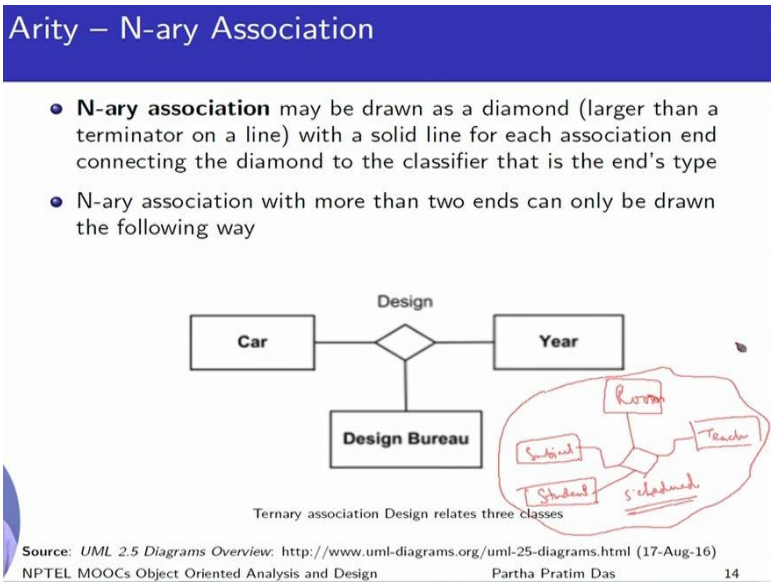
Next is the arity which is simple to understand. Most associations are binary and we have already seen how these are related to one line and along with that I could have all those navigability, reading directions, name and all of this.

(Refer Slide Time: 15:43)



So this is this example we have already seen. This is the name of the association this is the reading direction and it is simply creating it binary association.

(Refer Slide Time: 15:58)



If I have an N-ary association then we use this kind of a diamond an open diamond, a big open diamond and connect the different classes on different side and we give a name to this diamond which is the name of the association. Now certainly please do not think that since diamond has four corners you can use this only to connect four classes in an association. Association can have any arbitrary arity.

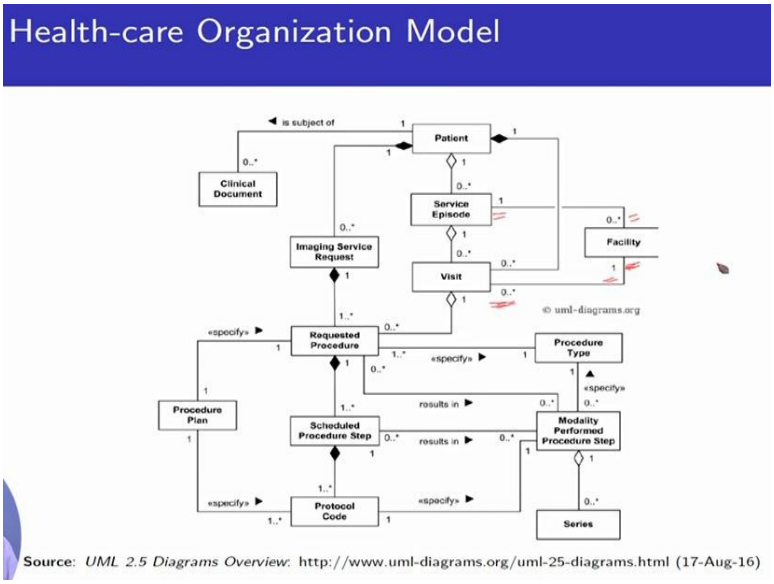
So it is possible that I can connect five, six, eight, ten different classes through an N-ary association. But of course it is true that as binary is the most common ternary is is seen to quite a times. But association of higher arity is more difficult to understand more difficult to model and maintain so usually will not see them very frequently. For example, one nice example of say the ternary kind of association is suppose you are working on a timetable scheduling classes.

So you need to know what is the room. You need to know what is the subject. You need to know who is the teacher. So between these three classes the room so I have room, I have subject and I have teacher. So I can have a ternary relationship which is say schedule that when is a particular subject scheduled in a room taught by a certain teacher. So this is a ternary relationship.

That will represent the information of the triplets having being related because a subject is held say in the second hour of every Tuesday and taken by certain teacher. Now if we try to along with this if we try to model the students who who attend this course. Then I could put an attending presence in the relationship and this will become a relation with arity four. But certainly as you can understand this makes it more and more complex to analyze and follow.

So normally, we will try to decompose everything in terms binary relation and at times use ternary relation.

(Refer Slide Time: 18:54)



So this is just to show certain instances of different class diagram. So this is like a health care organization. Of course these examples have have lot of details so I am just talking about some sampled aspects you should take time and try to understand these examples more thoroughly to understand what do they mean and will find that you will soon be able to start understanding how to read this diagram and get the information from them.

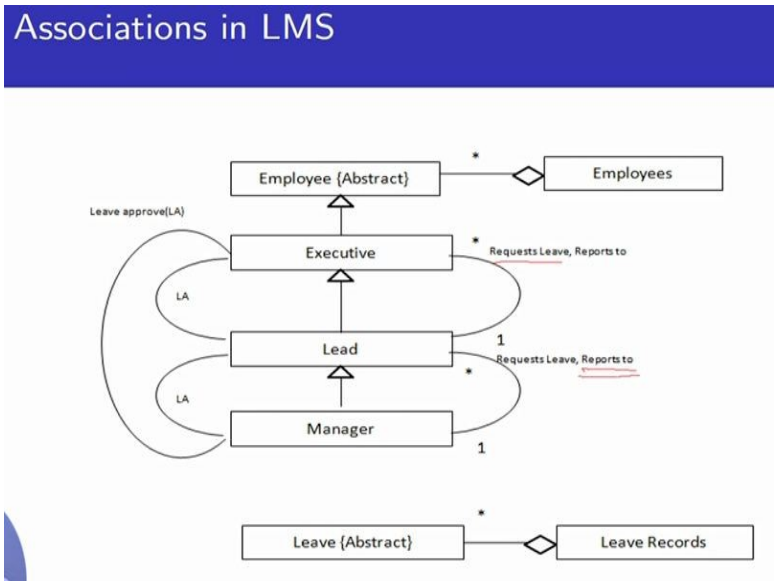
For example, we can see this is a requested procedure possibly for certain ailment and we say that there is an association with modality performed procedure step. For that procedure something you need to be done. For example, if ECG has to be done then every lead of the ECG must be added with certain liquid so that modality. So you are saying that there is a relation association binary association between this class and this class which is results in.

So requested procedure results in modality being performed in the procedure step and the multiplicity is zero dot dot star zero dot dot star which means that a requested procedure may not result in to any any of this a modality performed procedure step may not have raise from the requested procedure. But it could have any arbitrary number of associations as well. So you could see in this way that different kinds of binary association are involved here.

So I would just request and in in in this the navigability is not specified. You do not see any directions at the end of the arrows so it basically says that the navigability is unspecified. So for example the facility and visit have an association between them so a facility may be visited by several but one visit will always be a unique facility. You cannot at a time be at two facilities. So this side is one whereas this side is zero dot dot star.

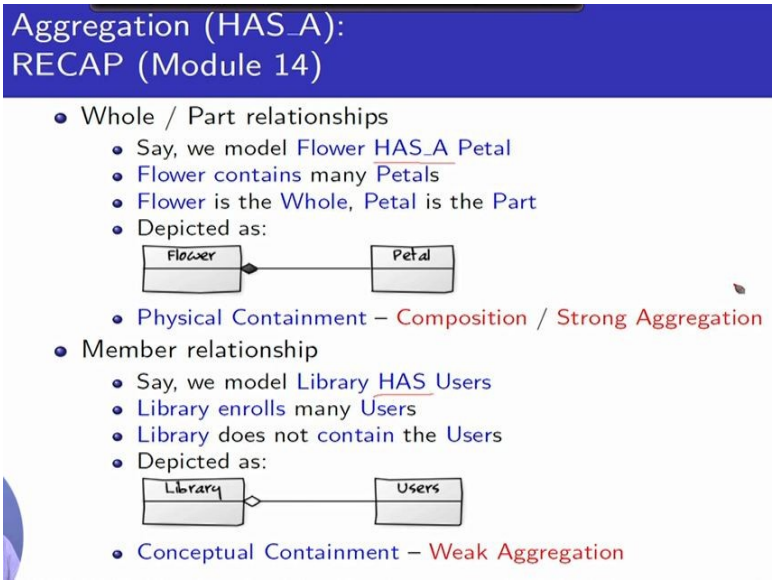
So please spend some time and I have also given the url pointer is to where I have taken this diagram from. If you go to go to this particular url which is the UML-2.5 diagrams overview discussion in that portal. You will find also the detailed description or more description of what this problem are and how this class diagram should be read.

(Refer Slide Time: 21:54)



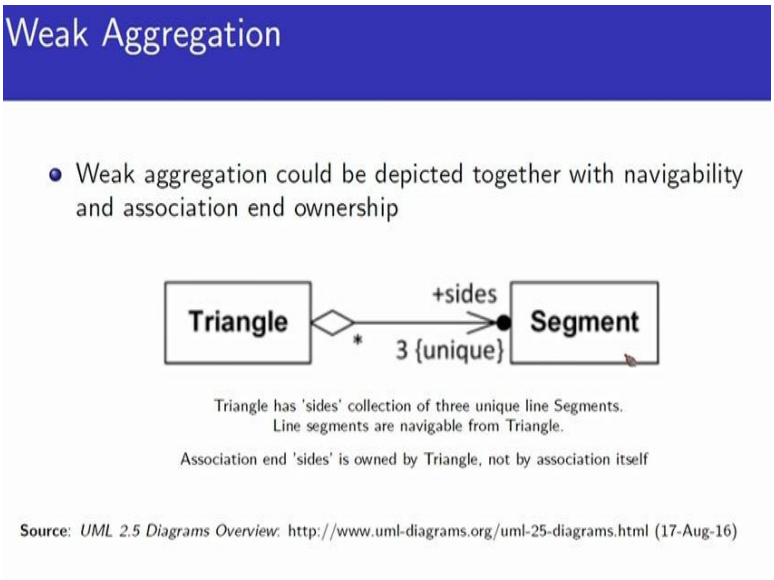
These are different associations in the LMS system particularly relating to requesting leave from reporting relationships and so on. So these are different binary associations.

(Refer Slide Time: 22:08)



The next is the aggregation which we know is HAS_A relationship and we have seen earlier that there is a difference between HAS_A and HAS which is a membered relationship. So flower has HAS_A petal because petal is a part and parcel of flower but library HAS users because library is associated with the users but users are not component part of the library.

(Refer Slide Time: 22:42)



So this association is needs to be represented. So based on this HAS_A and HAS there are two kinds of association which are specified. There are known as aggregation association. The first kind is a weak aggregation. A weak aggregation is represented by an open diamond and has two classes again. This is a binary aggregation and this is a HAS or membership kind of aggregation so it says the triangle has sides or line segments.

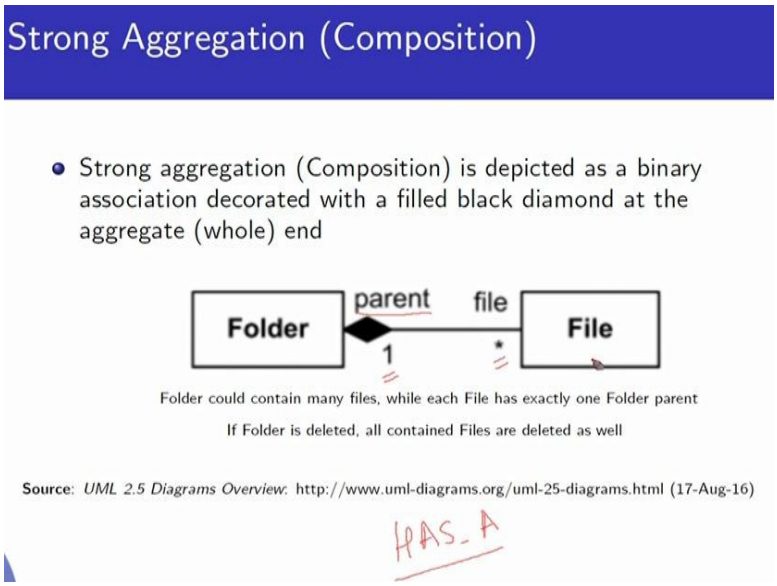
Now it is it is possible that you have a diagram like this you have a diagram where you have so a segment may be associated with multiple triangles. So and a when you look at from the segment side you put this aggregation to be at star which says that segment a segment could be by itself also not a part of the triangle. So we say that a segment could be associated with any number of triangles. But a triangle will be associated with three segments.

And then you give it a name and you say that this segments are called sides but the end you put that name. so whenever you have that situation between library and user and this kind you say this is a weak aggregation and along with that we can annoted further information so now we are saying the same relationship of weak aggregation this is this says there has to be three and now we are saying that this is unique.

Because certainly the if I have one segment here one segment here and one segment here this is not a triangle all three segments do not give a triangle they will have to be unique non-

overlapping segments so that is what is being said here. We have also specified the navigability that from the triangle you can go to the segment. From the triangle you can decide on that which is quite understandable.

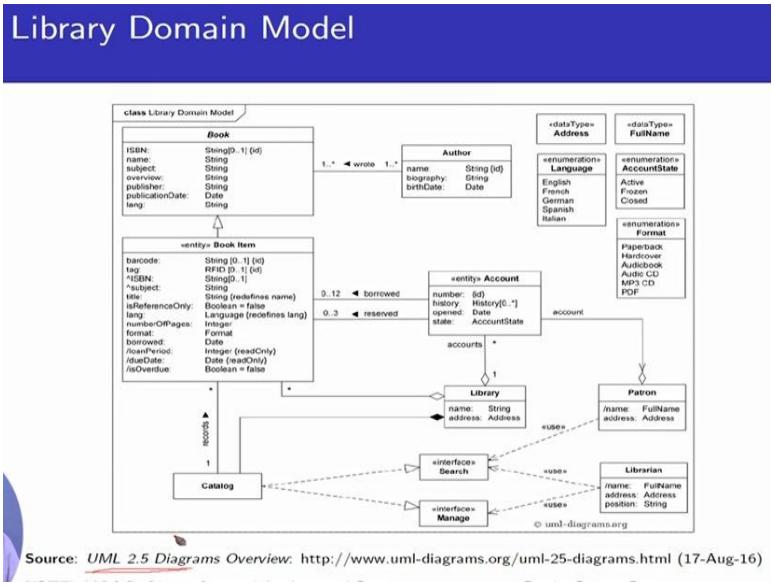
(Refer Slide Time: 25:03)



The other kind of aggregation is the composition where one is a part of the other so petal so if I have again go back to the flower and petal then we will have something like this and this is designated by the filled up diamond the filled up diamond. So if the diamond is filled up we say it is a strong aggregation so you are saying between the folder and the file there is a strong aggregation because folders contain files so if you delete the folder files will also get deleted.

So they are a part of the file. So if you if you destroy a flower the petals will also be destroyed. So whenever you have that so this actually is modeling HAS_A or part of relationship in terms of a binary association. Certainly you can put some name at this end and so on. So the multiplicity is clear because a folder can contain arbitrary number of files but a file can remind only in one folder.

(Refer Slide Time: 26:22)



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

So now if we go back and take a further look again in to the library domain model which we had seen in the last module as well so we can see for example you have accounts of users and we have library and there is a weak association between them. This is a weak association between them and it says that the account will belong to the specific library but a library may have multiple such accounts where the users are coming in.

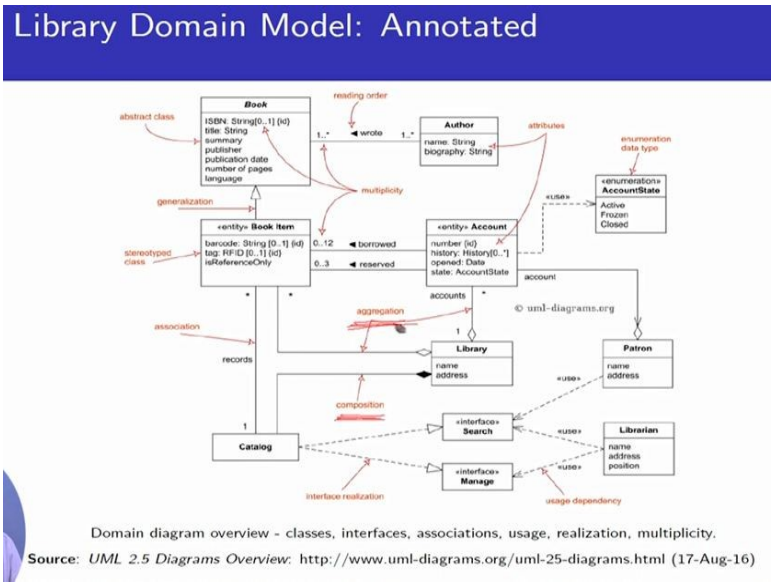
Similarly, it says that catalog which list the different books and the library are associated in a strong aggregation and nothing is specified which means that this is one-one that a library has a unique catalog and a catalog belongs to a library. Interestingly here for example the designer has decided to model the association between library and book item as a weak aggregation not as a strong aggregation. Why?

Because the scots are that a library is just in the collection but it is not that if you remove a book or you add 10 books the state of the library changes but basically a library remains a library but if you remove a catalog then the library will not be there. So this is being represented here in terms of a weak aggregation. We have multiples here which tell you that a library can have more any number of copies of a particular book.

There is no multiplicity given here it says it is one certainly a specific copy of a book can exist in one library only. Similarly, you have other associations here which is records, the catalog, and

books. One catalog, records any number of books. You can see other other different aggregation relationships also at other places and try to understand the diagram better. More specifications are given in the site. And you could check out those.

(Refer Slide Time: 28:56)



And this is again we also had seen this diagram, this is just same diagram but it is only that this is annotated now. So if you want to practice then you can just make reference to this. This is a composition of strong aggregation. This is an aggregation or a weak aggregation and so on.

(Refer Slide Time: 29:25)

So in to summarize we have several different kinds of association relationship between the classes in a class diagram. And these associations could be represented in very simple link terms

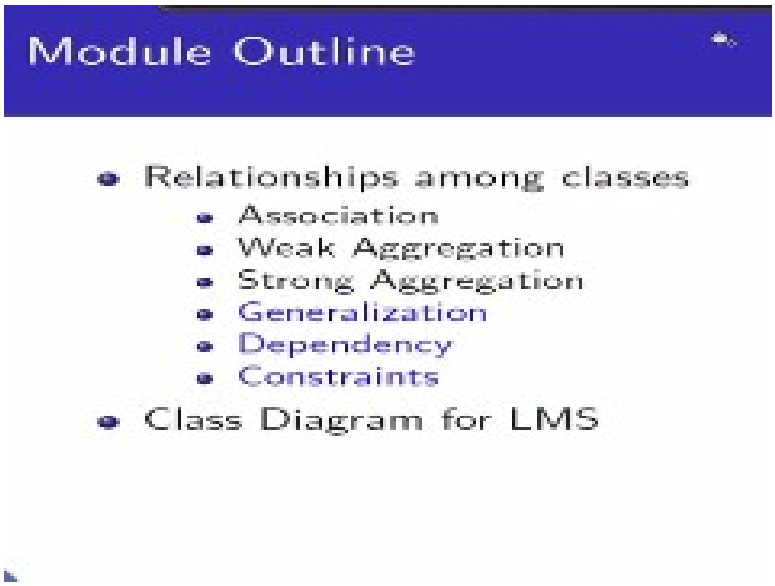
or they could be qualified. They could specify multiplicity have name, have reading direction, navigability and so on. And the arity could vary from being two to being any arbitrary number, the N-ary association.

And specifically we have taken a loop in to the aggregation kind of relationship where there are two kinds the weak aggregation and the strong aggregation. Weak aggregation represents HAS relationship and strong aggregation represents HAS_A or a part of relationship. With this we will conclude this module and the next module we will continue in discussing the remaining kinds of relationships that classes may have.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture - 40
Class Diagrams – Part 3 (Generalisation, Dependency, Constraints)

(Refer Slide Time: 00:36)



Welcome to module 28 of object oriented analysis and design. This module is the 3rd and concluding part of our discussions on class diagrams as UML diagrams. So in the earlier 2 modules, we have first studied the basics of class diagram, how do you represent class? And how do you represent properties of a class? The operations of a class, how to control the visibility of different members, the property that attributes and.

How to specify different details behind this properties and operations like if a property is read only? What if a property is derived? and so on. Then we started discussing about this was in the last module, discussing about the different relationships that may exist between 2 or more classes and we observe that there are variety of relationships, spanning association, which basically relates to 2 or more classes in a certain way and

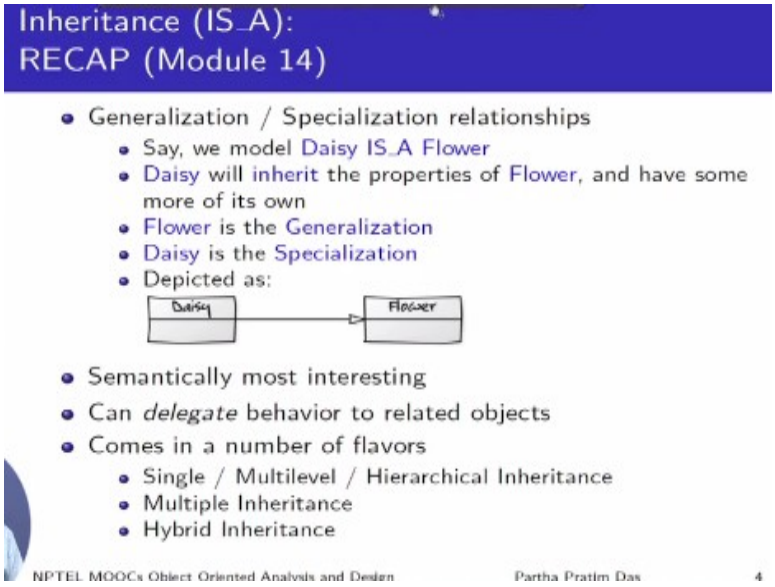
In any general form, a relationship can be viewed, first is an association; which has 2 ends that can be qualified, there is a reading direction, there is navigability in the association which basically specifies us to how does one class relates to another. Associations mostly are

binary but they could be ternary or N-ary in general. Then we take a look at different forms of aggregations.

We had earlier seen, 2 major aggregation structures possible among classes. One is the composition or component structure, where like car has 4 wheels, so wheels are composite the car, so this is has a relationship, which is in class diagram module has strong aggregation and we also saw a weaker version of the aggregation where you have certain class has 1 or more objects of some other class related to it, associated to it, to complete the functionality.

Library has multiple users, who use the library but they really do not, they are not really contained within the class in the sense of composition, so we designate them as weak aggregation and we have seen for an association, weak aggregation as well as strong aggregation, what are the notational details? The multiplicities, the MPN field diamonds and all that, so in view of that, in continuation of that we will discuss 3 other kinds of relationships.

(Refer Slide Time: 03:39)

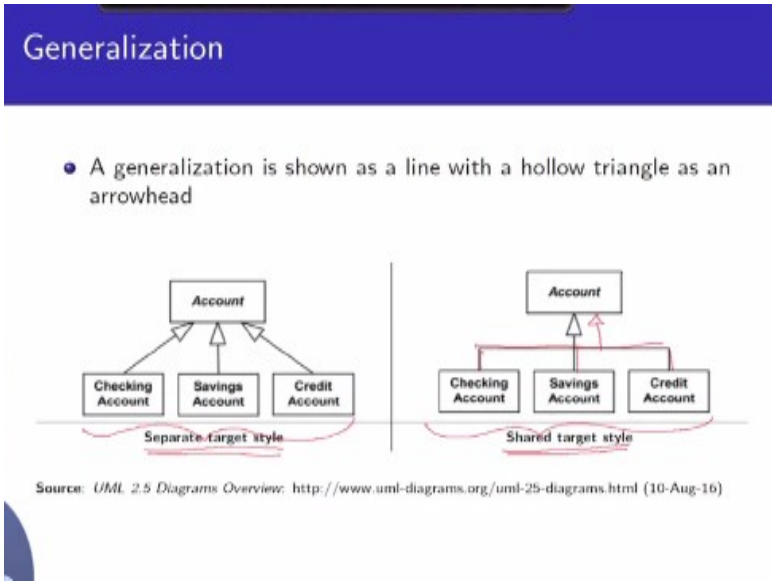


The generalisation, the dependency and the constraints, these are further relationships that may exist between 2 or more classes. Besides that, in this module, we will also quickly take a look into our running example of LMS and try to improve on the class diagram of the LMS that we have been constructing so far. So first we start discussing about generalisation, I would like to remind you regarding what we had talked of in terms of generalisation several times earlier.

So it is a generalisation, specialisation relationship between any 2 abstractions that are possible. So when I say daisy is a flower, I have a daisy class or a daisy concept, I have a flower class, flower concept and by inheritance or generalisation specialisation, I mean that if daisy is a flower, then daisy has all the properties that the flower has but daisy may have some more additional properties, we say flower is a generalisation and daisy is the specialisation.

We had seen without actually formal introducing class diagram; we had seen that depiction in terms of a diagrammatic form. So I would ask that before you go through further of this module, please refer to the earlier modules where inheritance and these relationships have been discussed, so that you can understand this in stronger detail particularly you should look into various flavours of inheritance that are possible.

(Refer Slide Time: 05:00)



All of those flavours can be represented in terms of the class diagram. So this is the basic representation of generalisation, if I have should formally say it is; this is this is the generalised class and these are the specialised classes and we draw an arrow from the specialised class to the generalised class and the arrow head is an empty triangle which represents that it is indeed is a relationship.

So we read it as checking account, is a account like this, savings account is a account, credit account is a account and with this, what in semantically it means that account has some properties and some behaviour, some operations, for example, account level properties, in terms of a holder's name, in terms of the account number, the account balance and so on, so

being specialisations all of these specific types of accounts like checking account, savings account or credit account.

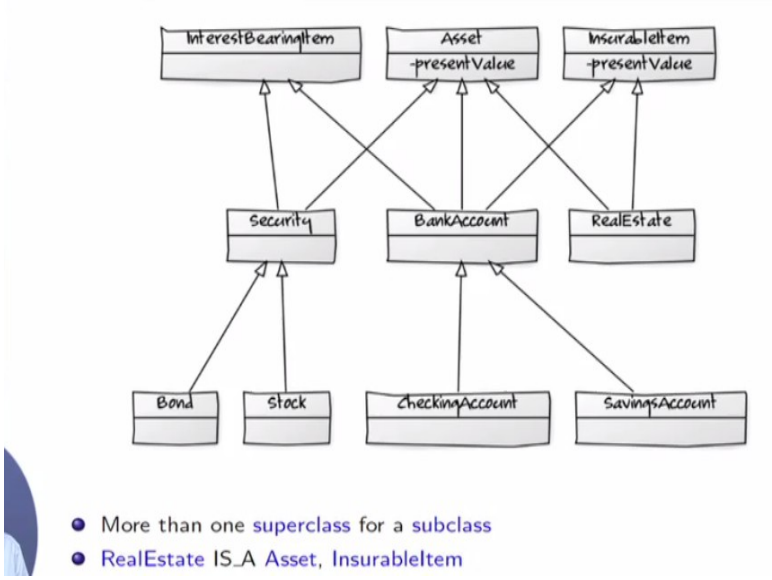
In in Indian banking we also have term deposit account, we have current account and all those, they will certainly have all these properties that is the holder, the number, account number, balance, but in in addition if I talk about specifically a savings account, then it will have certain further properties may be savings account has a restriction that I must have a balance of say 1000 rupees at the minimum in the savings account which may not be applicable in terms of a savings, checking account, it may not have that property.

Savings account may have that I can transact at most say 1 lakh rupees within a single day, but checking account may have unlimited options for transactions and so on. So these are different specialisations which use the properties that the generalised class has an adds further properties at their own level. So this clearly makes the depiction of the reality of real world.

So that whenever possible we would like to work with the generalised concept and this will certainly enhance the understandability as well as the reusability of the design. Now in terms of the diagram representation, you can see that we have 3 specialised classes of one generalised class and we can draw 3 separate arrows, if we do that then we say it is a, the drawing is in terms of a separate target style or.

We could draw a common bar line and let all of this specialised classes hang from there and this bar in turn actually points to the generalised class, this is called a shared target style. Now in class, UML class diagram, both these styles are available depending on the context, depending on what is easier to join represent, you could choose any one of them from time to time.

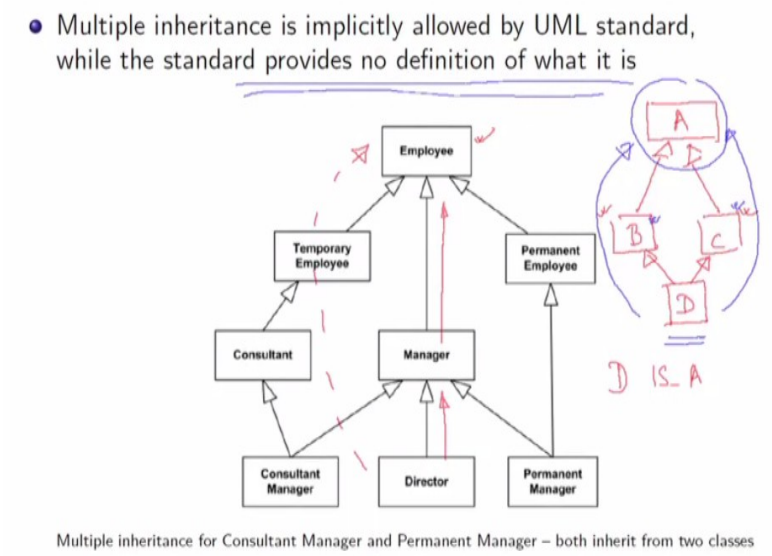
(Refer Slide Time: 08:14)



Naturally in terms of inheritance, you can not have multiple inheritance, this is again a slide from module 14, where we have drawn that a one bank account, this is a specialised class, this is specialised form interest bearing item, asset as well as insurable item. So it says that the interest bearing item has the property that I can earn interest on the deposit that I have.

Bank account has that property, bank account allows us to do that, so we have a specialisation on this. Similarly, asset is having certain fund value which you can pledge to somebody or which you can use to take loan and so on. So bank account can be used for that purpose with the balance that I have. So bank account is also an asset, so here we see that a same class is a specialisation of more than one classes.

(Refer Slide Time: 09:44)



Whenever that situation happens, then we say that we have a situation for multiple inheritance and as this example shows that you would have multiple inheritance from 2 or more classes, here bank account has 2 generalisation security, I am sorry; bank account has 3 generalisations, whereas security has just 2 generalisations and so on. So certainly, we can represent multiple inheritance exactly using the same notation as a single inheritance.

So we can see here that director is a manager, manager is an employee and we have consultant manager and permanent manager. The consultant manager is a manager on one site, and is consultant on the other. So it gets bore this property, so being a manager if you will get the rights of the manager, being a consultant it will get the flexibility that a consultant gets possibly it is not full time, possibly it gets paid according to how much his work and so on.

On the other hand, there could be permanent managers, who are managers again get the same manager rights, but is a permanent employee, so the permanent manager also inherits the permanent employees benefits like retirement benefits, like leave, like fixed salary and so on. So multiple inheritance is a very important feature which is often used in modelling different real life scenarios and place a critical role in terms of the class diagrams.

Now one point that I would like to highlight is the multiple inheritance as such is implicitly allowed by the UML standard, in the sense that since you can draw from one specialised class, you can draw multiple IS A relationship arrows to other classes, so the multiple inheritance gets represented in a class diagram, but as such standard has not try to provide any specific definition that if I have multiple inheritance and this is what the inheritance would mean.

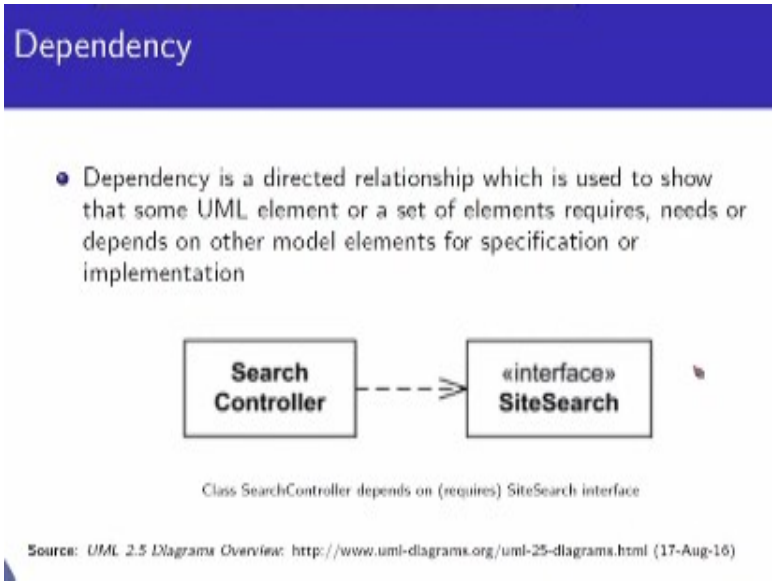
Particularly there are certain critical situations like if I have a class like this and I have 2 classes specialised from that, this is fine, but then I have one class which multiply inherits from these 2 classes, A, B, C and D, so D multiply inherits from B and C both. But B in turn is the specialisation of A, C in turn is the specialisation of A. The question is certainly if I use the transitivity of the inheritance, that is transitivity means, director is the manager, one generalisation.

Manager is a employee, the other generalisation, so there is an implicit transitive inheritance between the director and the employee that is director has inherited all the properties of an employee through the manager, so if I use that logic here, then D actually inherits the properties of A through B as well as D inherits properties of A through C. Now does it mean that some of the properties will be inherited twice?

Some of the operations may have been redefined in B, may have been redefined in C, then what will happen to those operations? How will D interpret those operation, because it put inherit the same operation from B and C, because they existed in A, the original route class, but it may be inherited in 2 different forms, so what should D do in terms of defining its own operations, own properties?

So there are issues in terms of how the multiple inheritance should be interpreted and you should keep in mind that while it is implicitly allowed, standard has not provided any specific definition further, so you will find that a multiple inheritance model as created in UML class diagram, when you try to take it to the OOP an object oriented programming language like say C++, then specific OOP language will had specific interpretation, specific semantics for multiple inheritance.

(Refer Slide Time: 14:12)



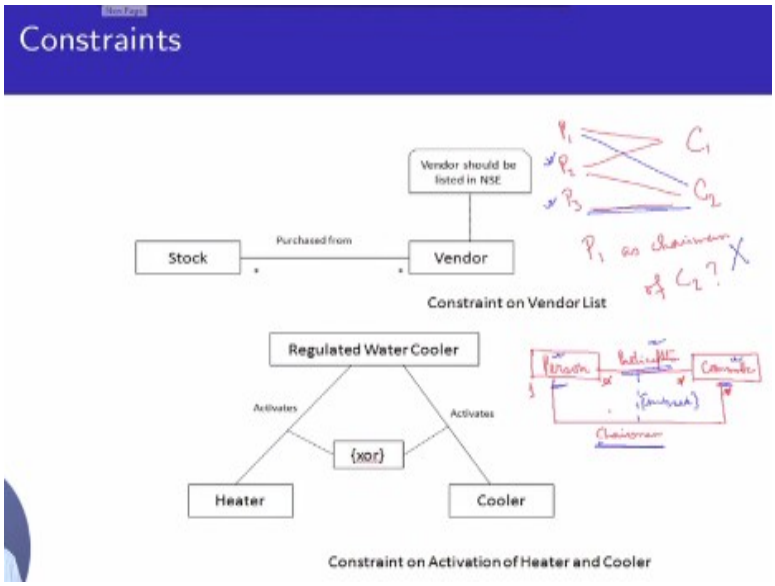
So you will further have to understand what is the semantics of the for multiple inheritance for a specific language and then use that in the implementation model, so some cleared is required to actually model with multiple inheritance and then take it forward in terms of

building up certain actual implementations. Another form of relationship that is possible is known as dependency.

Dependency is somewhat represents the little bit of a different concept. It just says that between 2 classes, this one class depends on the other, one class is affected by another, one class may be using another. So unlike association which tells you that specific messages at intended to be sent through association or when we talk about aggregation we are basically talking about strong or weak compositional aspects of the model.

When we are talking about generalisation, we have certainly talking about a very strong structural similarity between the classes. In dependency, we more or less just try to highlight that how does one class may be effecting another? Or may be dependent on other and that is the typically drawn with dotted arrow at the arrow head at the end and depicts that this class on the left requires the other class or depends on the other class.

(Refer Slide Time: 15:30)



Yet another form of relationship could be expressed in terms of constraints, so constraints for example, let say there are 2 forms that we have shown here, so we are saying there is a class stock and there is a class vendor. So basically it is a scenario of procurement, so stock is the inventory and vendor is the provider from whom we can buy and build up the stock, build up the inventory.

So we have association purchased from the direction, reading direction is not shown, it is clear from the way it is written, so the reading direction is like this, the multiplicities are

anything, so any item can be purchased from any vendor, you may not purchase from anybody also. Given this, we need to also specify that well we will not purchase from actually any vendor availability in the country.

We want to purchase only from the vendors who are listed in the national stock exchange, so we represent that by with the dotted line and this is kind these are typically called notes, I am sorry. Let me write it again. These are typically called notes with this corner cut, so this kind of a note attached here which say that well in this relationship the vendor may actually contained several vendors who are not listed in this.

They may be related to the other classes in the system and for that reason may be there, but for this particular relationship, the vendor would need to list in NAC, so you are putting some kind of a restriction, some kind of a constraint on that. Similarly, let us say we have a regulated water cooler, so it is a regulator water cooler means it is a water cooler which tries to set a certain some temperature and maintain that.

So internally it will have a heater and a cooler, if the temperature has to go up, the heater has to be put on, if the temperature has to go down, the cooler has to put on. So to represent that we have a association showing activates. So this activates relation is applicable for the heater as well as for the cooler, but the reality of the situation would be that you never conceive that you have a state of the regulator water cooler, where both the heater as well as the cooler is working.

You can conceive of this in terms of particularly car air conditioners, or room air conditioners usually do not have the heater path, unless you are in an appreciation air conditioning system, but in a car air conditioning, you will always find that there is a dial like this which can be rotated and it show that there is a blue part usually on this end and there is a red part usually on this end.

The blue part designates that your cooling, the red part designates that your heating. Obviously we will not do both of them together, so how do you represent that in the class diagram, so you, one profusion is you have an XOR relationship, exclusive OR, which means that the heater can, heater activates regulator water cooler is valid, cooler activates regulate water cooler is valid, but both cannot be valid at the same time.

This exclusive OR, XOR is exclusive OR, usually OR means inclusive, but it could be either this or this or both, but exclusive OR means that both cannot have. So these are these are different forms of constraints. For example, I could have another constraint say let us say I have a person and let us I say have a committee, so basically forming a committee from a group of persons.

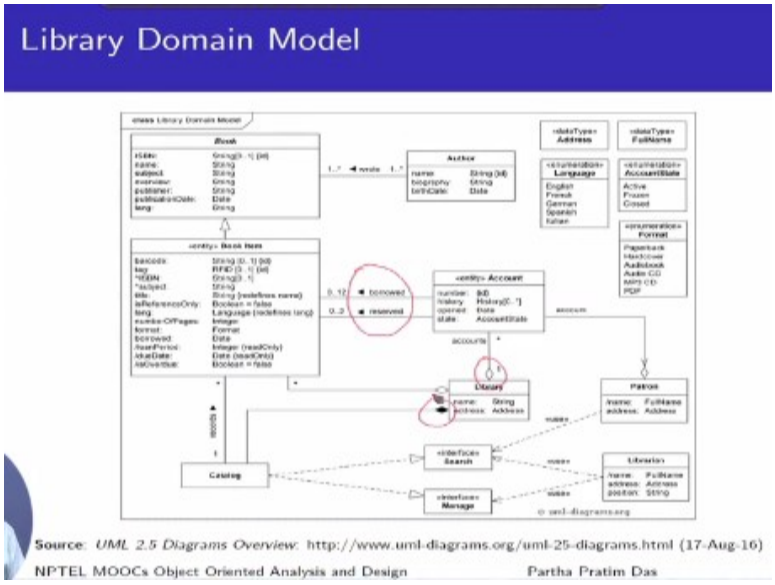
Naturally the association could be something like participation, okay, say I can have ***, that is any person number of person can participate in any number of committees. Then every committee, there is a chairman, so I can model that again I have a committee, I have a chairman, so my relationship is chairman. Naturally I would expect that every committee has one chairman and a person could be chairman of any number of committees including the person may not be a chairman at all.

Now if I just modulate like this, then it will be quite possible that if I have a committee C, and I have persons P1, P2, P3, I have another committee say 2 committee C1, C2, let us say that, the participation is like this. They participate here and they say participate here, now can I have P1 as chairman of C2, can I have that is, if I say my blue is the chairman link, can this association be possible? P1 is a chairman of C2?

An other to be absent, right, this person does not participate in the committee but how can he cheered the committee, so this infeasibility have to be represented and we again use constraint for doing that. For example, the way this can be denoted is you can join these 2 associations with a dotted link and write subset, so this is the subset constraint, which say that the chairman, the chairperson association has to be a subset of the participation association.

Which means that if there is a relation between a person, a person and a committee, as a chairman then that combination, that doublet of the person and the committee must be a member of the participation, so this will be ruled out, so if C2 has to be cheered, it have to be either P2 or P3, so if I say that this is what I have P3 is a chairman of C2, then that it will be valid, but P1 being the chairman of C2 is not valid.

(Refer Slide Time: 22:31)

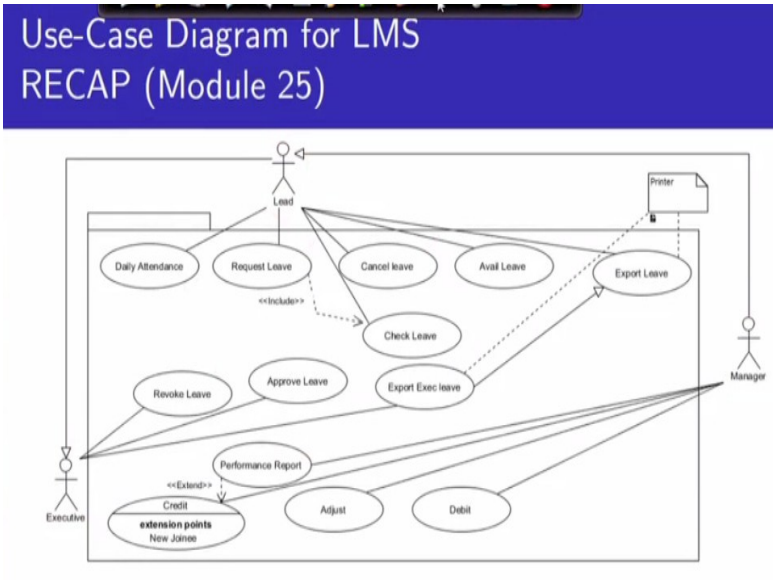


So in this way, different forms of constraints will have to be considered, so that we can model that reality in am more precise manner. So this is just you know take in a look into the Library domain module, we had earlier looked at this in terms of the let me go back to that in terms of the properties, now we can see the various kinds of other relationship for example, we conceive a generalisation relationship here.

A book item is a book we can see different this is say is the librarian users, librarian manages the library, librarian searches the library, the patron, patron here means a user, searches the library. So these are like these search manages these are like interface classes, which basically give you different operations that you can do as a part of that class, so how does a patron relate to the such interface is through this users' dependency.

So there is used dependency between so the patron needs to use the such interface to be able to perform the search activity. So this is how all this can be done, this we have already seen, these are the different associations, this is the weak aggregations, strong aggregation, I am sorry, yes; weak aggregation, this is the strong aggregation and so on, so these are all different types of relations that we can have.

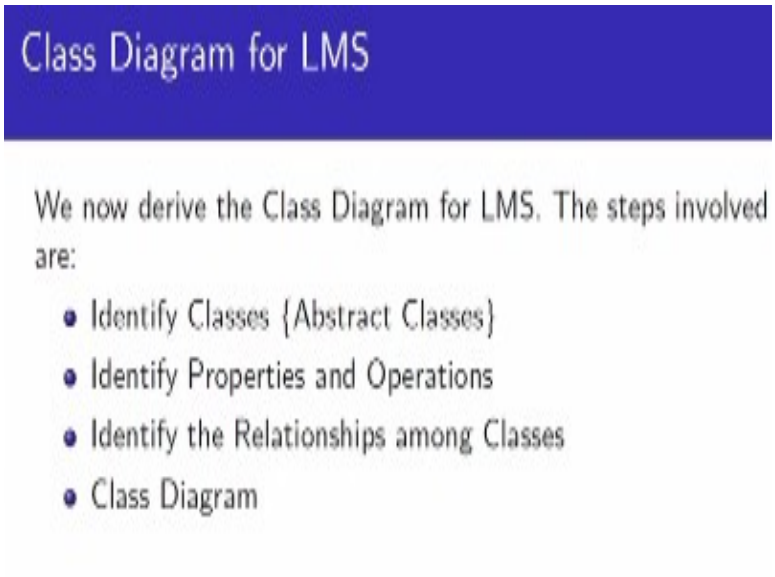
(Refer Slide Time: 24:20)



You can use this particular on the same example, here it is annotated, so you can clearly see the different kind of usage dependence is interface realisation which is another kind of dependency and so on. Now so that is all about the relationships among class diagrams so we can quickly go back to our leave management system and try to advance the class diagram, the UML diagram for the leave management system, by little bit more.

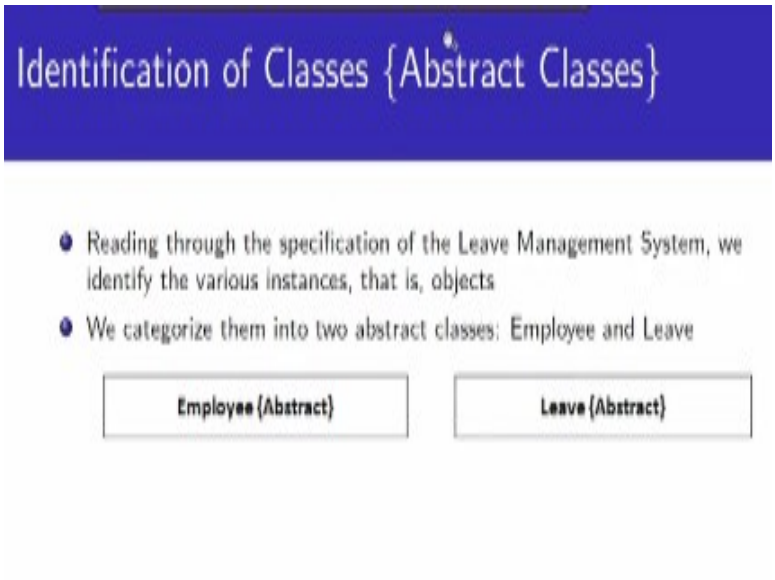
So I just pick up the use case diagram here done in module 25, so we know that these are the different actors we have already identified, so naturally they will transform themselves into classes in the design in the class diagram design and we have several different types of use cases of how we use and what is not explicit in the use case diagram is certainly all of these, or majority of this use cases need some other abstraction which is of leave.

(Refer Slide Time: 25:27)



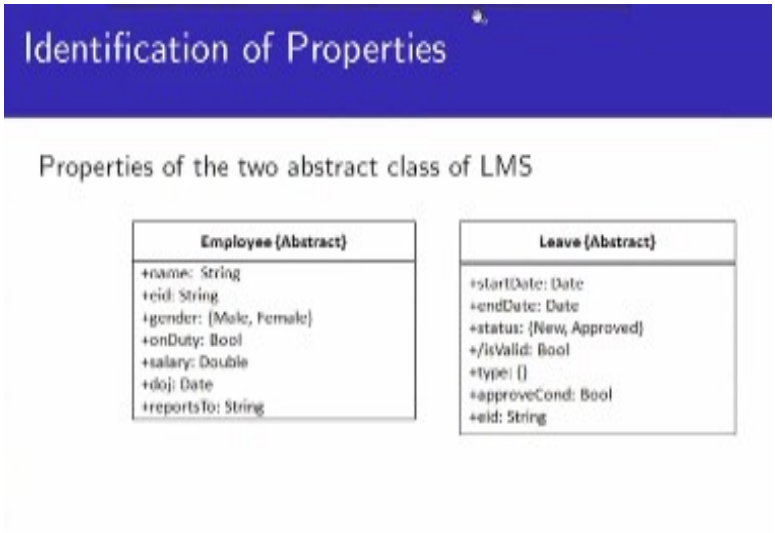
Because when we apply for leave, when we request for leave, or when we approved leave we need that abstraction. So now we will see how we get those in terms of a consolidated class diagram. So in terms of the class diagram, the tasks are 2, identify classes, often you start with identifying abstract class, identify the properties and operations, identify the relationships and finally combined them all together, in terms of the class diagram.

(Refer Slide Time: 25:55)



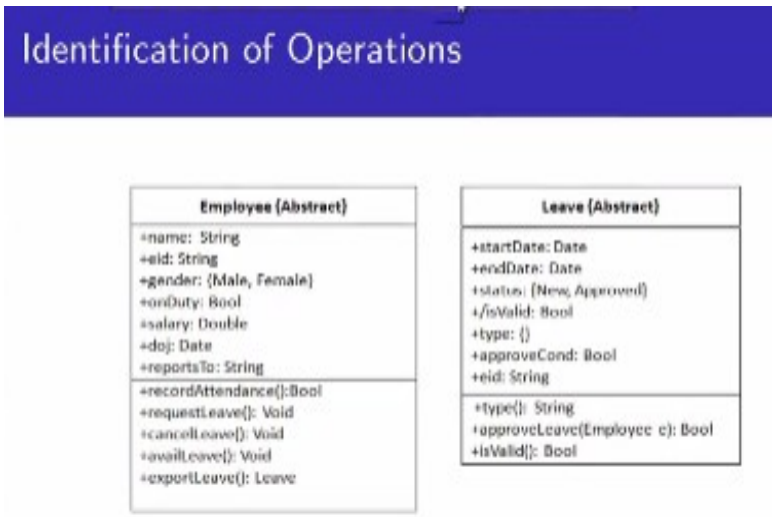
So these are task, which we have been performing all through the past several modules and so we can just quickly glance through the result of that, certainly in the notation of class diagram or main abstract classes, there would be any more, but main abstract classes are employee, which we say is an abstract class, leave is an abstract class, because we can see that there are several specialisations of this which we have separately discussed earlier that will result, so these are the 2 basic concepts.

(Refer Slide Time: 26:39)



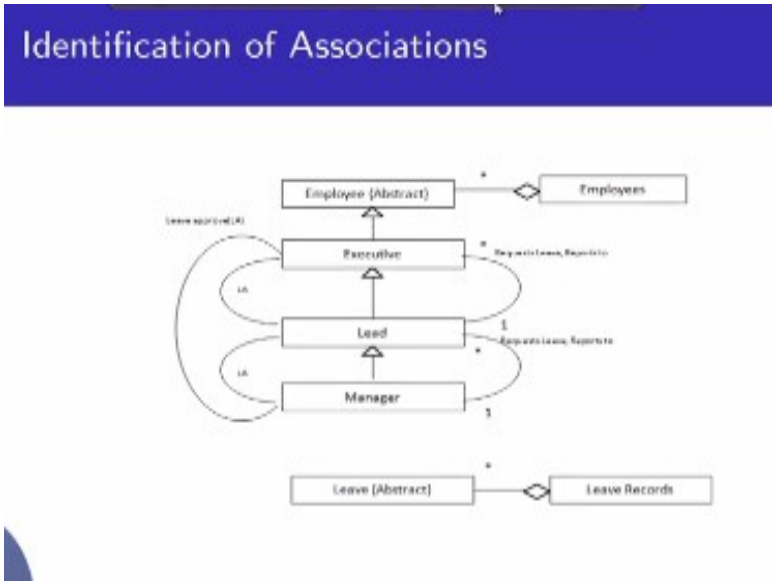
Beside that, there would be several other classes, like there would be a class for the leave store there is a class for the documentation, there could be class for tertiary actors in the system like doctors who prescribed different documentation and so on. So with that we will try to complete the or; try to make a complete list of the identified classes. For the classes that we identify, we have different attributes, and we had seen the extraction of these earlier.

(Refer Slide Time: 27:02)



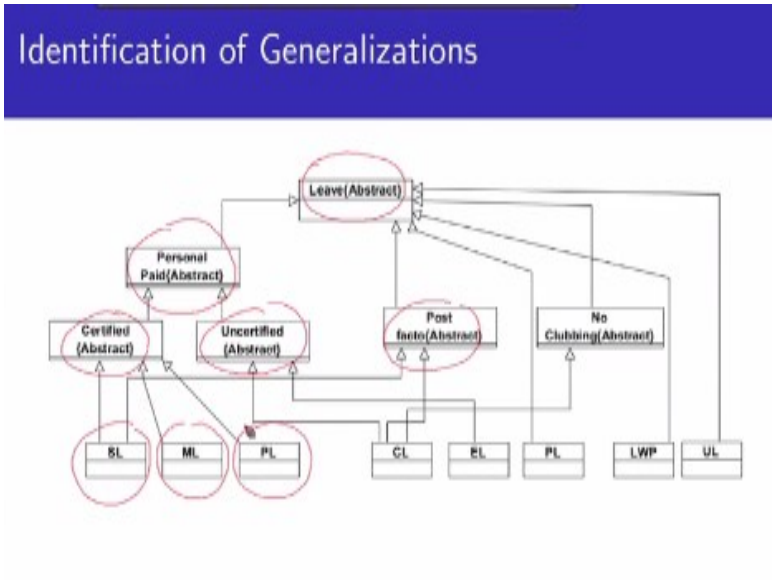
So for these abstract classes, we can add these attributes, these properties and we can keep on adding more and more, these are I am not saying that these are the exhaustive list and I am just indicative one, which you can see how to put together. Then you on top of this attributes or properties, you have add different operations that that class do, so we had started with out of these 3 compartments we started first with this, which is the mandatory.

(Refer Slide Time: 27:32)



The class identity, then the attributes, properties, then the methods of operations, so this becomes a more complete class representation for the employee as well as this one is for the leave. So some of the associations again we had earlier identifies requires leave report to so, the different use cases you will often see, the different use case is actually manifest themselves as in terms of operations in classes.

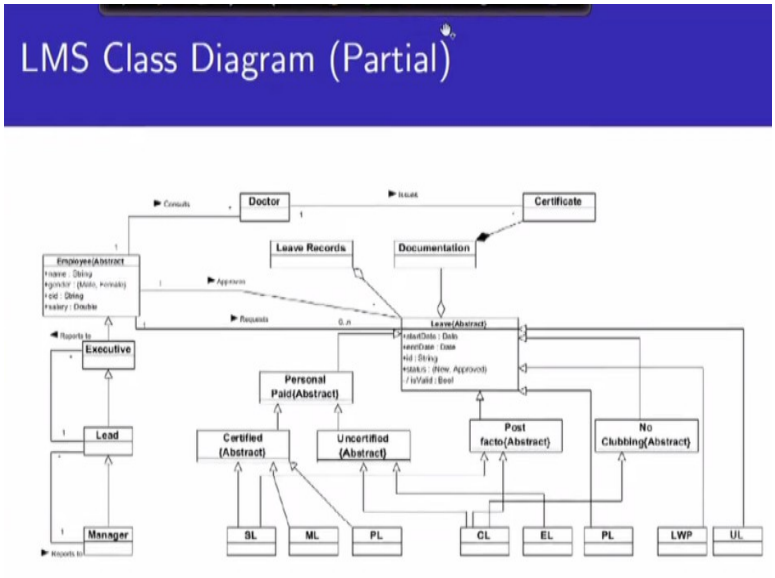
(Refer Slide Time: 28:04)



Because the classes, different classes will have to actually provide that service to other objects or they manifest actually in terms of different association that exist between the classes because that is how the classes interact. Then we have seen the generalisation specialisation so in terms of the formal representation this is what we have the leave abstract leave and we had earlier shown this.

We have personal or paid leave, we had post factor leave, certified leave, uncertified leave, all of these are basically abstracts because you the organisations does not as such have something called as the certified leave or you cannot directly apply for certified leave, what you apply for other at the leaf level, the different concrete class instances that are possible.

(Refer Slide Time: 28:46)



So this diagram now fits into our class diagram notation, so I if you put things together and mind you this is just a partial diagram, this is not a complete one, we will share the complete one as a solution of LMS, so here you can see the actors have come into one hierarchy, which was originally captured in the use case diagram but in addition to that there is a association identified.

Which is report to executive reports to the lead with the multiplicity of one on the lead side and any on the executive side which mean that many executives could report to a single lead and it is possible that at a point of time, a lead does not have any executive reporting to us. Similarly, the similar relationship exist association exist between lead and a manger. We could identify several others.

Here we have the abstract leave class and on this side as a whole we have this whole hierarchy, generalisation hierarchy from the leave which will all actually inherit the semantics of association that the leave makes, so critical is the employee extraction and the leave extraction, say HAS A association request, which is request leave. So we want to mean that the employee request leave, so employee has to initiate that and leave will get requested.

HAS A multiplicity of one on this side, and HAS A multiplicity of 0 to n, 0…n, on the other side. because if you see that any leave will have an unique employees who has applied for it. So on employee side, the multiplicity is singular but on the leave side, an employee could may not request for any leave, so the association could be backwards, or could request for 1 leave, 2 leave, different kinds of leave and soon.

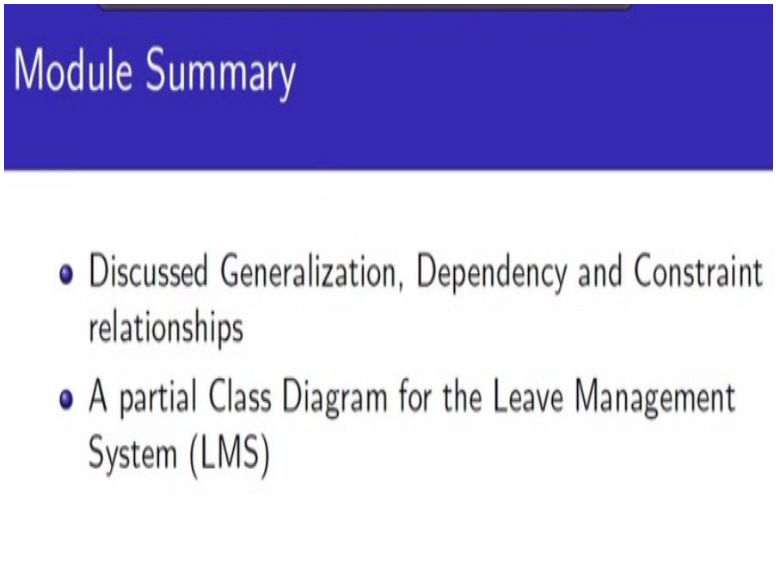
So this is a multiplicity on here and based on that it will reflect as you propagate down the relationship on the generalisation hierarchy, so this association on the generalisation hierarchy, will get different kinds of specialised form of this association but the most fundamental is this association between the 2 abstract classes and another association of approval between the 2 abstract classes.

When we say this you can see that between this approves and reports to between these 2 associations, you need to set some constraints. I have not done that, just left it opens so that you could take it up and complete the diagram further because certainly the leave of an employee is approved by another employee that is because it is an abstraction, so an executive is an employee, so a lead is also an employee.

So if you read it that way, then the approval basically means an employee is approving a leave which belongs to yet another employee. Now certainly what will lead to rule out, you lead it to rule out that in this approval process, it is possible that if the executive reports to the lead, only then the lead approves the leave for that executive. The executive will not approve leave for other, I am sorry; the lead will not approves leave for other executives who do not report to that particular leave.

So between whenever an approval has to happen then that reporting relationship will also have to be valid. So that is a constraint, so I will leave it, to discover more and more associations, aggregations, and generalisations, possibly generalisations has been extensively done already, the constraints and so on and try to complete this diagram, let later point of time will come up with the solution also.

(Refer Slide Time: 32:40)



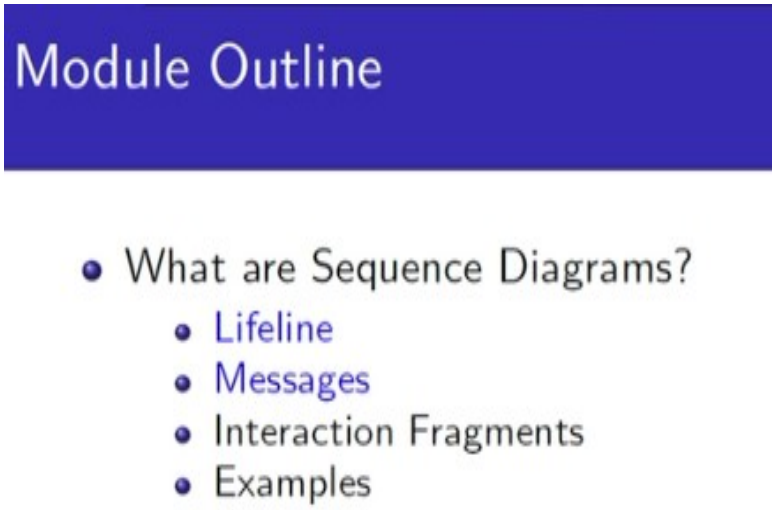
So this conclusion on discussion on the class diagram as a UML diagram and now you should be in a good position, to be able to represent any system prescription in terms of a class diagram. I would request that for that the assignment management system, the LMS system, you take it as your exercise to carry out the class diagram and get convinced that you are understanding all of those very well.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture - 41
Sequence Diagrams - Part 1

Welcome to module 29 of object-oriented analysis and design. We have been discussing about UML diagrams and we have already introduced the use case diagram, which is critical for the requirements phase and we have introduced the class diagram, which is the most used structural diagram in the UML diagram set. In the present module, we will talk about sequence diagram, which is one of the behavioural diagrams, specifically categorized as an interaction diagram.

(Refer Slide Time: 01:00)



This discussion will span the current module as well as the next module, so while introducing the sequence diagram, we will talk about the lifeline and messages and discuss examples in that context and in the next module, we will talk about the interaction fragments and some more examples.

(Refer Slide Time: 01:25)

Client-Server Computing Model: RECAP (Modules 05, 11)

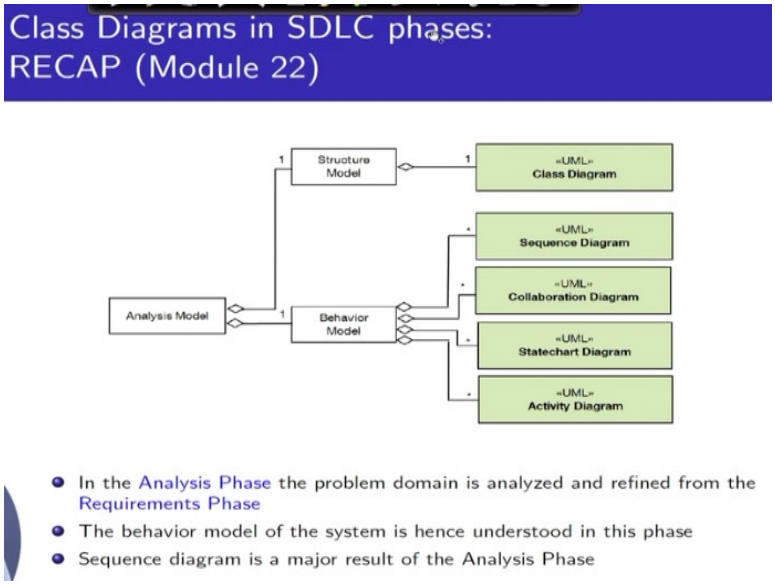
- No object exists in isolation
- Objects are acted on and themselves act on other objects
- Leads to the **Client-Server Model** of computing where
  - Behavior is
    - Services provided by an object
  - Services are requested by
    - Sending Messages, Invoking Operations
  - In Client-Server View
    - Clients request for Services
    - Servers provide Services
    - Contract between client and server ensures correctness

Before we get started in terms of what is a sequence diagram, I would like to remind you of the basic computing module. We have been talking about the structural aspect of it, but now we want to talk about the behaviour or execution aspect of that. So please be reminded that we have in the final module of our computation, we have several objects, so objects exist in the system. See these are different objects and these are the this you can think of as a time.

So objects exist on that time and then, they exchange messages between themselves that one object may send some message to another object, may receive a message from the object and when you get a message, you perform certain computation that is the computation associated with a particular operation, which has been requested through the message. If you recall, this was a basic client server model.

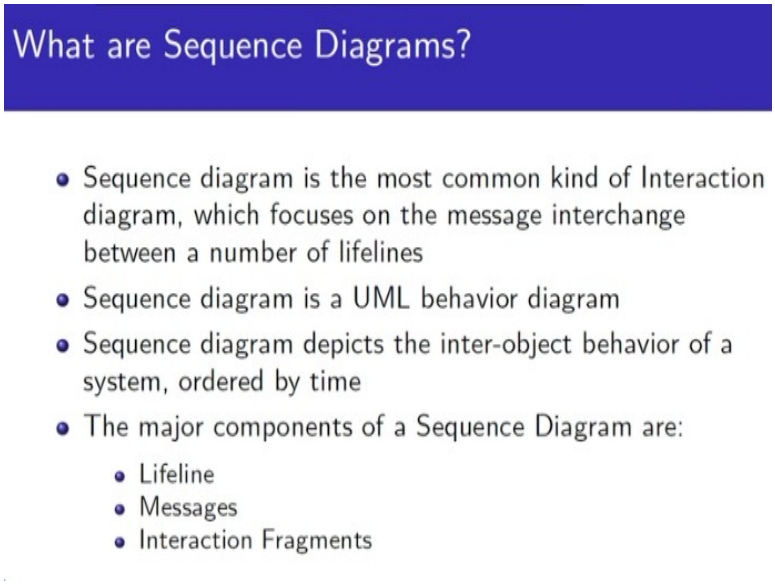
So sequence diagram is a is a first level by where we try to capture this behaviour of the services provided by the object and how messages request for certain service and how that service is responded. So we are going closer to realizing and representing the actual client server version of the system that we are designing.

(Refer Slide Time: 03:07)



Now, these are the different places where the sequence diagram I am sorry the sequence diagram can occur. If they can occur in the analysis phase, as we have already seen, they can occur also in the they can also occur in the design phase as a behavioural model. So these are the two places, they start featuring from the analysis phase, where we first try to capture the dynamic behaviour of the objects.

(Refer Slide Time: 03:44)



And in the design phase, certainly we would like to refine them more. Now coming to what is a sequence diagram, as I already mentioned, it is a most common kind of interaction diagram, which focus on the message interchange between a number of lifelines. So these are the two key ideas, concepts that we will try to introduce and detail out on the sequence diagram. It is a behavioural diagram, as you know and it tries to depict.

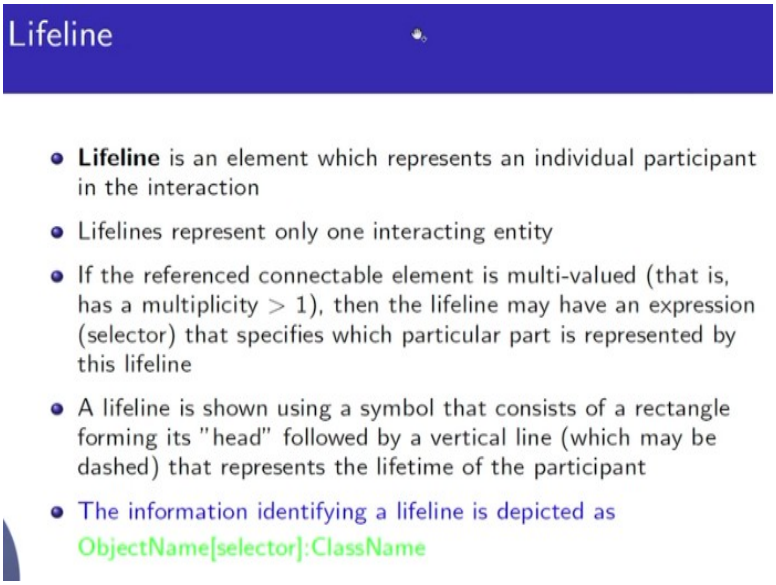
And this is the basic client server model, so it tries to depict the inter object behaviour of the system and it is ordered by time. This is very, very critical and that is why often we say this is a temporal interaction diagram. So it tells you in the order of time how things happen and when you talk about time here, we will see often that the time is not actually a clock.

That is time may not mean that so much specifics, so many seconds, so many minutes or hour has to elapse between two happenings, two events, but certainly it emphasises in the order in which things should happen. For example, if I want to check mail, then certainly I cannot do that without launching the browser. Now after launching the browser, I need to fine the URL of the Gmail. Only after I have found the URL of the Gmail, I can put in my login request.

Once I have put in the login request, it has to get authenticated by the login server. Once this is authenticated, then my inbox will be shown, so these series of things that we regularly do in terms of checking an email is a specific illustration of what a sequence diagram tries to capture. So this is the time sequence. So there is no specific time given between launching of the browser and the browser actually showing a page.

Or in terms of requesting an URL and actually getting that login page for the Gmail and so on, but the temporal ordering is most important. There are three major components of a sequence diagram lifeline, messages, and interaction fragments.

(Refer Slide Time: 06:18)

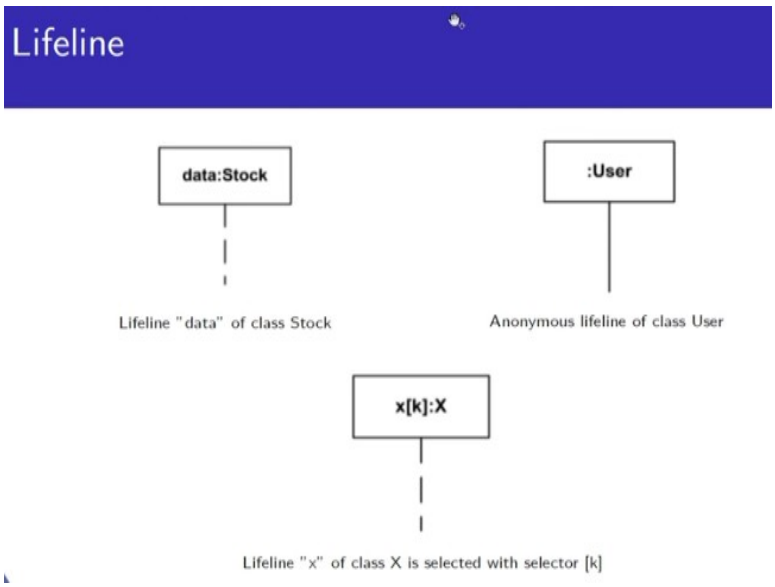


So first let us look at a lifeline. Lifeline is the basic concept of lifeline is every lifeline represents an object, an object in existence. So lifeline is basically the time through which the

object lives in the system. So that is generically defined as an individual participant in the interaction and it represents only one interaction entity. So if we if we have a collection of them, then I must use some kind of a selector to select which particular one I am talking about from the collection.

Lifeline is shown by a head, which has typically the name and then it has a dotted dashed line that goes below, the vertical line and the sense of time on that line goes from top to bottom is the advancing time. So this is where the object has started leaving and it continues till the object is annihilated. So let us start taking examples.

(Refer Slide Time: 07:28)



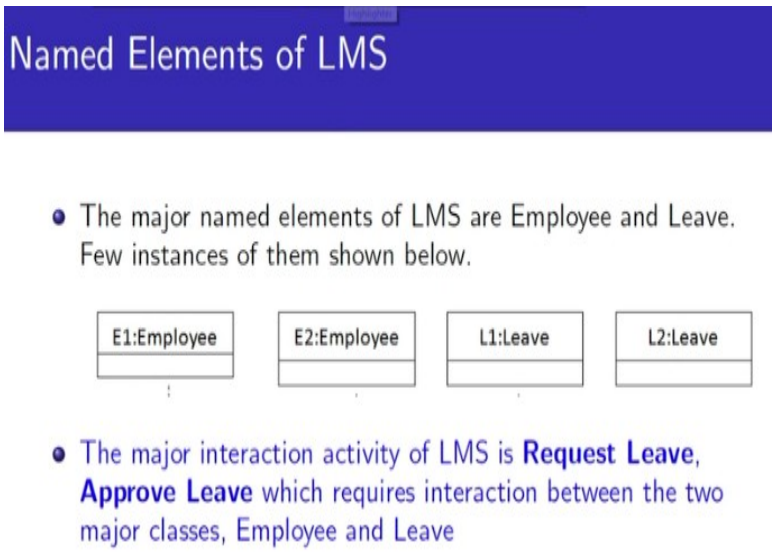
And we will be able to see. So here, this is this is a lifeline of data of class stock. So this notation is again important to note. These are name of a class and when I receive that with another identifier with a colon separator, then this means, this is the object. This is an instance of the class. This is the class. So data colon colon stock here in terms of the class diagram and in terms of the sequence diagram, the UML in general, would mean the data is an object of the type class.

So as I said, the lifeline will belong to an object. So this is the object and this is its lifeline, this is advancing time. The second one here is an anonymous lifeline. Here we have just have the class name, but there is no object name given. So the name of the object is not specified. So this is called anonymous object or anonymous lifeline. We could write something like this as well, where certainly this is the class and this whole thing is an object.

But just note the array like notation being used here, so here possibly there are several objects of the same type and we are using a selector, this is a selector, a selector key to specify which of these objects is participating in this interaction. So these are the typical lifelines that will be involved in a sequence diagram and in all these, actually I should go back, all these are these are called named entities in the UML. So you have a specific name given for classes and named entity.

We may have names for the lifelines as well. So these are some of the named elements, entities of the LMS.

(Refer Slide Time: 09:21)



So here it says that E1 is an object of employee. So E1 is a specific employee, E2 is another specific employee. L2 is a specific leave and so on and there could be the lifelines would run from these objects.

(Refer Slide Time: 09:47)

Types of Messages

- Message is an element that defines one specific kind of communication between lifelines of an interaction
- There are 2 major types of message in Sequence Diagram
  - Messages by Action Type
  - Messages by Presence of Events

Message by Action Type: A message reflects either an operation call and start of execution or a sending and reception of a signal

Message by Presence of Events: A message depends on whether message send event and receive events are present

Now coming to messages, the messages and element that defines one specific kind of communication between lifelines of an interaction. So usually a message will be between two lifelines, most cases and they are categorized according to two broad classes, two major types, one is by the action type, that is what kind of role the message and messages to achieve and other is by the presence of events.

That the message certainly is something that an object sends to another object in the generic form of the client server model. So a message usually will need to have a send event, which initiates sending of the message and will lead to have a receive event, which is the receiving of the message, so if we classify messages according to the presence or absence of these events, then the classification is called the messages by presence of events.

So we will take a look into each one of these types of messages, what are the different varieties that exist in every case.

(Refer Slide Time: 11:03)

Messages by Action Type

The various types of Messages by Action type are:

- synchronous call
- asynchronous call / signal
- create
- delete
- reply

So we go by action type, then various types of messages under the action type are these: synchronous call, asynchronous call or signal, create message, delete message, reply a message, so these are the different kinds of messages that are allowed in a sequence diagram.

(Refer Slide Time: 11:21)

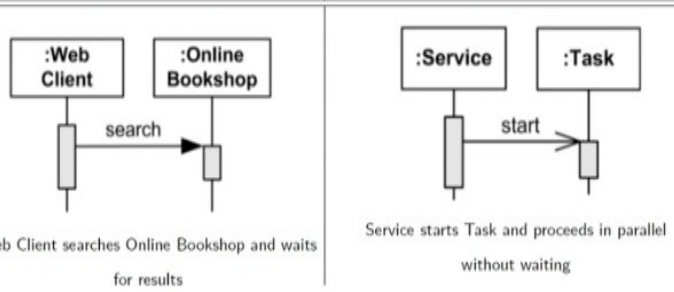
Messages by Action Type

Synchronous call typically represents operation call - send message and suspend execution while waiting for response

Notation: filled arrow head.

Asynchronous call - send message and proceed immediately without waiting for return value

Notation: Open arrow head



So these are synchronous call, so let me just explain the basic concept. So there is a object here, there is another interacting object here and this is this means something is in execution, some method of object is in execution. So you send a message, a search message and to this particular interacting object. And as you send that, the search method or the search operation in the online book shop object gets activated. So that starts executing.

You do a similar thing here as well. Here there is a service object, there is a task object, you can note that in all these cases, the objects are basically anonymous and you are executing

some method of service. You send a message to task, which is a start message, so basically the start operation in task will start executing. That is the basic structure of sending a message.

Now you do note that there is a difference in the arrow head, which gives the difference in terms of their basic behaviour. We say this is if the arrow head is filled up, we say this is synchronous call, which means that once the request has been sent, the requesting object, that is this particular method, which was executing will be put on hold till this requested operation is completed and that response comes back, some way to indicate that this has completed.

So between the requestor and the requested, only one will work at a time. When the requestor has sent the request, then this is executing, this is on hold at this point and only when this has completed, this will start executing again, that is synchronous. Asynchronous is where the requestor sends a request, this is initiated, this continues, but requestor also keeps on continuing, requestor does not stop. So requestor has sent the task, just do that task.

I tell somebody, that this is the letter, please go to the post office and deposit that letter, send that letter, but after sending that person, I continue with my own work, so that is an asynchronous call. It is there It is not the execution of one here is not dependent on the other. Here the execution is dependent because till it completes execution, this cannot start further, so this is one of the key ideas, key properties of a message that you must identify when you are modelling.

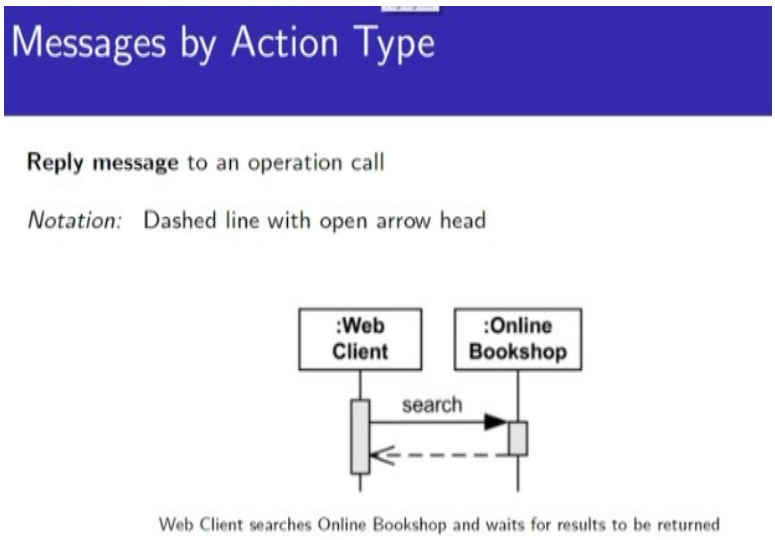
As to you wantd to deal it synchronised or need it asynchronous. Of the other types are various, you know, certainly the basic hygiene kind of messages as I said, is a create message, so you will need to create objects at different points of time, so this is designated by a dashed arrow with arrow head and the head of it is not an execution box, but is an object itself. So this says online book shop is now creating an account. So this is called a create message.

Similarly, this account exists, an online book store is sending a message, you can see the name of the message is destroyed, which does invoke, I mean if you know C++, Java, it invokes the destructor and the object eventually is destroyed. So beyond this there is no

lifetime for this object, nothing exists after this. This object has was created here is terminated at this point, so this is known as delete message.

So that is the basic object creation and destruction that will occur. So we have seen synchronous, asynchronous, create, delete, the only other message that is important is a reply or response message, which may be sent.

(Refer Slide Time: 15:23)



It is in many cases, it is optional, it is typically shown with dotted line and with open arrow, so a web client sent a search request to the online book shop and online book shop in return has sent a reply. So this is a reply message or is often called a response message also. Now, it is not that always there will be a response, certainly for example, for asynchronous call, the reply message may not be there because the requestor is already executing in some other state.

But in a synchronous call, he will need a response message because you remember that the requestor who are the sender of the message is waiting for the requested task to be completed and therefore that completion will have to be reported in terms of a reply message. So these are the major types of messages in terms of the actions that they perform.

(Refer Slide Time: 16:25)

Messages by Presence of Events

The various types of Messages by Presence of Events are:

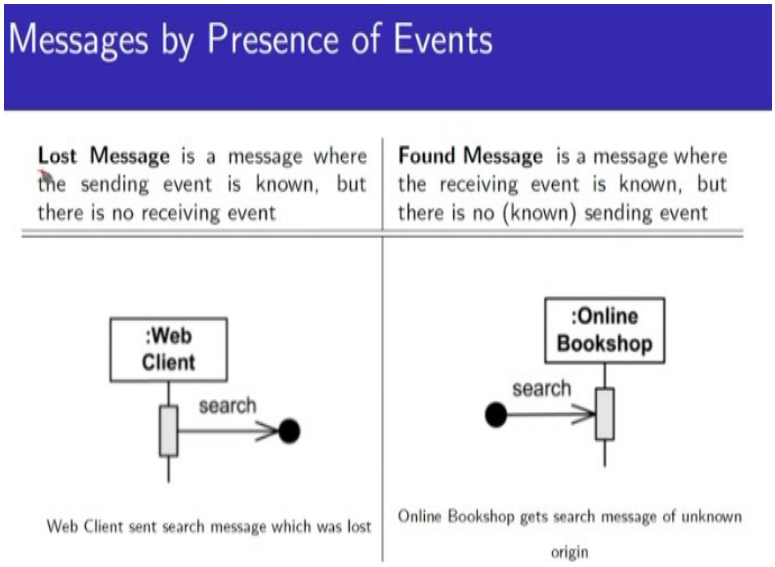
- complete message
  - The semantics of a complete message is the trace <sendEvent, receiveEvent>
  - Both sendEvent and receiveEvent are present
- lost message
- found message
- unknown message (default) – both sendEvent and receiveEvent are absent (should not appear)

In terms of the events as I said that most messages are complete message that they have a send event and a receive event. A request has been made and by reply this has been this has been completed in terms of that. So when I send a message, this is the point where I am doing the send, so I have a send event like this and when the message is received by the receiver, I have a receive event at this.

So normally a complete message would mean that one that has been sent by the send event and that has been received by the receiver. So because you would expect that why you are doing messages because you want to interact. So in the interaction, there will have to be sender, then there will have to be a receiver that is to be received. So if that is satisfied, then we say we have a complete message.

But there could be lost messages or found messages. As we will see, the lost message is one where you have just the send, you do not have a receive or a found where you just have the receive, you do not have a send and you could have unknown messages where both the send and receive events are not present. So that is that kind of default, but certainly in a diagram in a sequence diagram, unknown messages usually would not mean anything and you should not have.

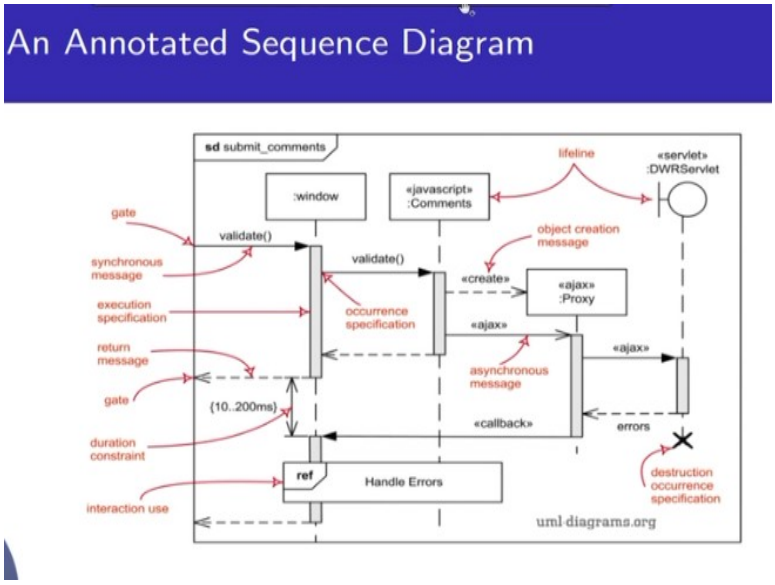
(Refer Slide Time: 18:10)



So in terms of two special kind of messages, the lost and found, lost message is one where the request was sent, so here the send event happened, but it never reached anybody, which is very typical when we go to web search and all that, we have sent the request and it does not reach the web browser, it gets lost, dropped in the whole process and we denote that in terms of this big black dot. So this says that the received message received event was not there.

On the other hand, it could be found messages that the online book store gets a search request, so this is the received event, it has received that but there is no identified sender on the other side. It is not known where did it come from. So the send event is not known. So you say lost in comparison to lost, this is a found message. So these are possible message in terms of presence of events, but certainly we would primarily work with complete messages, which have both the sent as well as receive events.

(Refer Slide Time: 19:19)



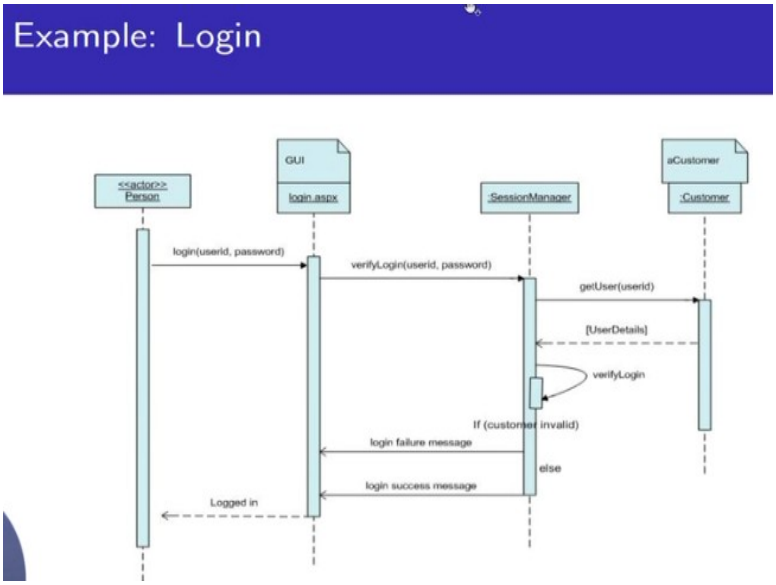
So this is this is just for you to see and get familiar with different messages, so we can we can see here. There is a lot of red in this diagram, then we will use blue. So we can see here. We can say that this window object is sending a validate message to the comments object and what kind of message is this? This is on one site, a synchronous call message. Because we have a what kind of by presence of event? This is a complete message. You have both the sent as well as receive.

So we can see different such types, for example here creates, so you have a create message, which is creating the proxy, the comments is creating the proxy. So you have a create message, the new lifeline starts. There is an existing lifeline of servlet and a message has been sent to do some tasks, which felt and because of that the object is destructed. So this is a delete message that you have in place.

You have a reply message here a reply message. There are several other notations. Please do not get worried about them. We will slowly introduce those. And if you particularly look at this message being sent from comments to proxy, after this was a create this was a create and this ajax message is being sent from the comments to proxy. You can see this is again a call message, but this is an asynchronous call.

So after sending this, the comments will continue to work when this execution starts. So these are the different kinds of messages, just illustration of the notation that you would think of.

(Refer Slide Time: 21:19)



So you could go through some examples quickly. For example, I want to do a login, how do I model that. Certainly, here is the initiating, this is here is the person who is initiating who wants to do the login, so the corresponding object sends a login request and within parenthesis what we write are known as message parameters, message could have parameters that it carry, certainly if you have to login, you need to specify the user ID and the password.

And the message is sent to the GUI because that is where we write down the user ID, password. So the message is sent to this. The message is synchronous because unless the login is completed, I cannot do anything else. Next, the GUI what it will do, it will need to discuss or consult the session manager of the particular session as to check if you are already logged in, if you are in the session, if your login is valid.

So it sends in turn it sends a verify login message to the session manager, this again is synchronous, so which means that when this verify login message is received at this point, at this event, then this as well as this, both of them are hold on hold in execution. They are not executed. Now the session manager has to check that, so the session manager looks at that customer database to check if the user exists, so it does a, sends a, get user details, and sends the user ID again in a synchronous manner till it gets a response of the user details.

So in this path of execution in the customer object or which is basically a customer store, so this is customer class and a customer is a say as if the name of the particular customer object. So the customers based on the user ID, the user details are returned. So once the session

manager gets the user details, so it knows the password that has been given, that has come from this message. It has got the user details.

So it has got the password of the user as present in the system. So at this point, it has the password from this site, it has a password from the database. Now it has to verify that whether they are same, so that is verifying the login again, but this message interestingly you can see is a self message that is the session manager actually sends the message to itself and this smaller one denotes the verify login as a self message.

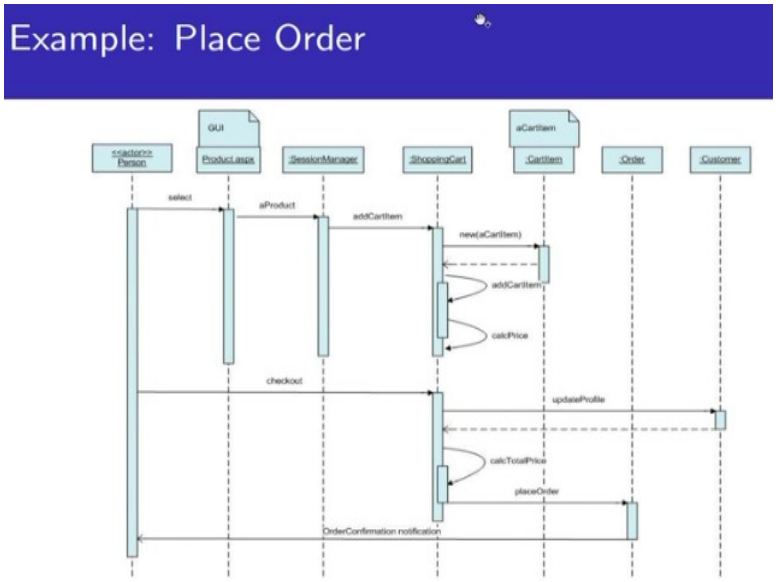
Because it is it is important to verify that because password is normally not kept in the way they are typed. They are kept in some encrypted form, so the password that you have got that needs to be encrypted and then it is to be compared with this encryption and so on, so that is the verify login. So you can see that the session manager on one got the verify message from the login GUI and the other its gets a verify login again from itself and it computes and all of these are asynchronous.

Because the whole process has to go in step by step, one step after the other. And after this has been done, at this point what will you have? You will have either the login is successful, the password has matched or it did not match, so you have a branching. So this is how it is typically you will write branching. There are other ways of noting that, so you say if customer invalid, then you will do a response message, which is login failure message.

That say that the login did not go through else you sent a login success message, so you can see there are two messages that go back from the session manager to the GUI, both of them are of both of these are of reply message kind. Because they are responding back, because they are replying message type. But only one of them can be sent depending on what happen in the execution, is it valid login or is it invalid login.

Whichever comes across, then based on that the logged in information is given back to the person, you either say that you are logged in or you will say that your login has failed. So this is just a simple sequence diagram to show that the typical way we login, how that can be represented in terms of the different lifelines and messages that pass between them.

(Refer Slide Time: 26:07)



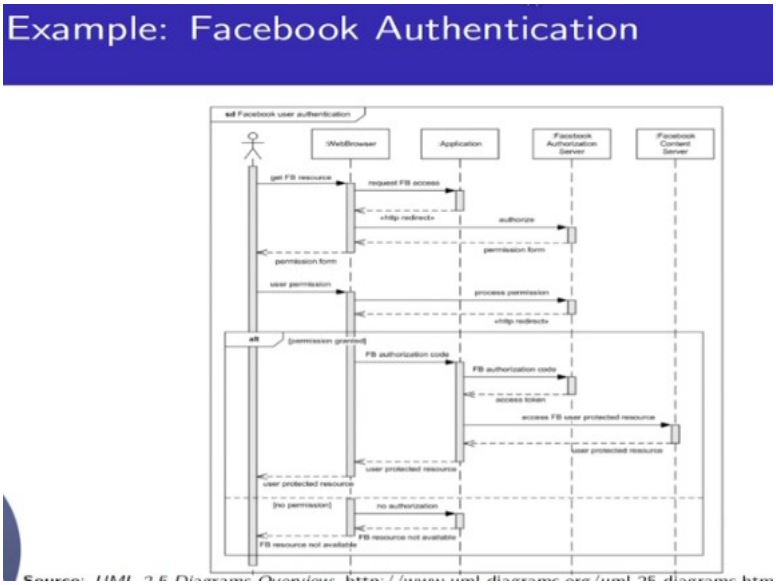
I have another illustration here, which I leave for your own analysis and understanding. It is on the similar line. So if the person has already logged in and as if this is basically some kind of an online store, so after login, the person wants to place an order, so the person is here, the GUI where the products can be seen, the session manager is there, the shopping cart where we elect the items that we have selected is there, specific cart items are there.

Then we have the order in which once, we have done the collection in the shopping card, we would like to actually place the order and the customer, so these are the different objects, which participate in this and these are the different messages that go through them. So Please go through this diagram, we can see there are several ,several of them are actually synchronous calls.

Actually all of these are synchronous calls because in the in the shopping process, online purchase process of any kind, certainly things will have to go step by step and only one step is done completed, the next step can happen, so that is a synchronous situation broadly. There are several like cart item or calculate price, which are self messages, which you can use and then finally you have the order confirmation notification that comes back to the person.

So I would request that you carefully go through each and every message and try to map it to your own experience of having purchased anything from the online store and you will find that how we are representing those behavioural aspects of the process in terms of the sequence diagram.

(Refer Slide Time: 27:59)



There is yet another example. This is as if you are doing a Facebook authentication, so this part this is the web browser through which you are working this application. This is Facebook authentication server and this is the Facebook content server. So the idea is working from the browser, you have to first approach the Facebook application, then the Facebook application whether you can access that depends on what the authentication server will tell us.

And once you are authenticated, then you can actually access the content from the Facebook content server. So these are the four major objects or four major timelines that participate in accessing an element in the Facebook and the if you if you again look carefully at this part, this is mostly synchronised message and response, which consults the authentication server through the application to validate if you have permission for access.

And once you have permission for access, then you do this part, which is actually accessing the content, so your requests are going till up to the content server. So again again you read through these sequence of messages and try to understand how this is happening and try to understand in depth. Because if you understand couple of these sequence diagrams, in terms of your personal experience.

Because I am trying to pick up examples, which I believe that most of you who are attending this course would have personally experienced either purchasing something online or accessing certain message, certain photos, certain video on the Facebook, so these are the typical kinds of sequences that we will need to go through and this is how they can be modelled.

So there are there is of course a final part where permission is not there, so you do not actually in this part below the dotted line here, you do not actually are able to go up to the content server and access the content because your permission has been denied.

(Refer Slide Time: 30:03)

Messages of LMS

The messages for the major activities of LMS are given below:

- **Request Leave**
  - Request Leave() from Employee
  - new() Leave
  - isValid() Leave
  - return(ifvalid == true)
- **Approve Leave**
  - Approve Leave() from Employee
  - Approver()
  - Reportingto()
  - return(Reportingto)

So that is mostly about the lifeline and messages and in view of this, if we look into the LMS, then our typical message would be a request leave message or approved leave message, so it will come from an employee to the leave and then we will have to see what should be the different processing that should happen. So I leave it as an exercise to you to think over as to what should be the basic sequence diagram structure for a request leave or approved leave that we have discussed for the LMS system.

(Refer Slide Time: 30:39)

Module Summary

- Introduced sequence diagram to capture the detailed execution flows of objects, their interactions and lifeline with a temporal ordering among events
- Discussed lifeline and messages in depth with examples

In summary, we have introduced sequence diagram to capture the basic interaction of different objects and their behaviour under the client server model. We have explained the modelling of objects lifetime in terms of lifeline components and discussed about the various messages, different kinds of synchronous and asynchronous, these are two key kinds of messages and the response.

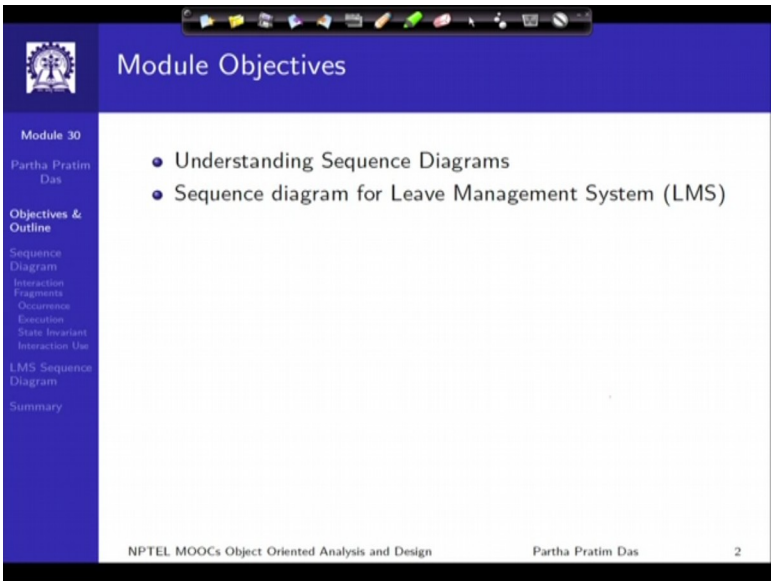
These are the three key types of messages, which will need most of the time to deal with any situation of dynamic behaviour besides the create and delete messages. So please I will ask that you please try to go through the example diagrams that we have shown and try to become confident about understanding the lifeline and messages before you take up this next module, where we will talk about the interaction fragments and further on the LMS example.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 42
Sequence Diagrams Part 2

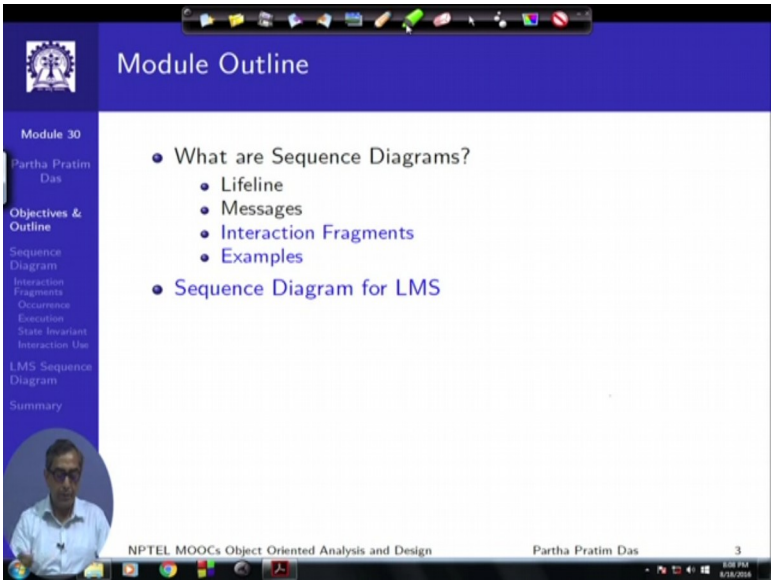
Welcome to module 30 of object-oriented analysis and design. We have been discussing about sequence diagrams.

(Refer Slide Time: 00:29)



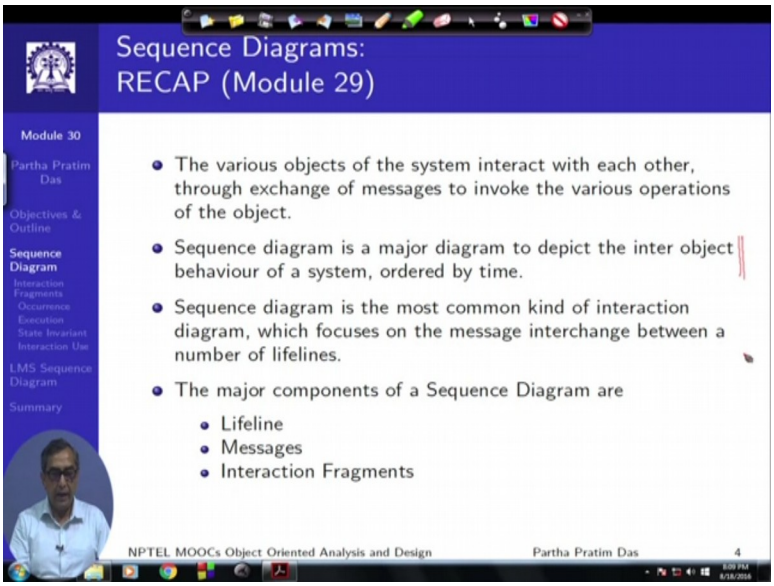
So, we will continue that discussion complete the understanding of that sequence diagram and we will take a brief look into building the sequence diagram of our lms exercise.

(Refer Slide Time: 00:44)



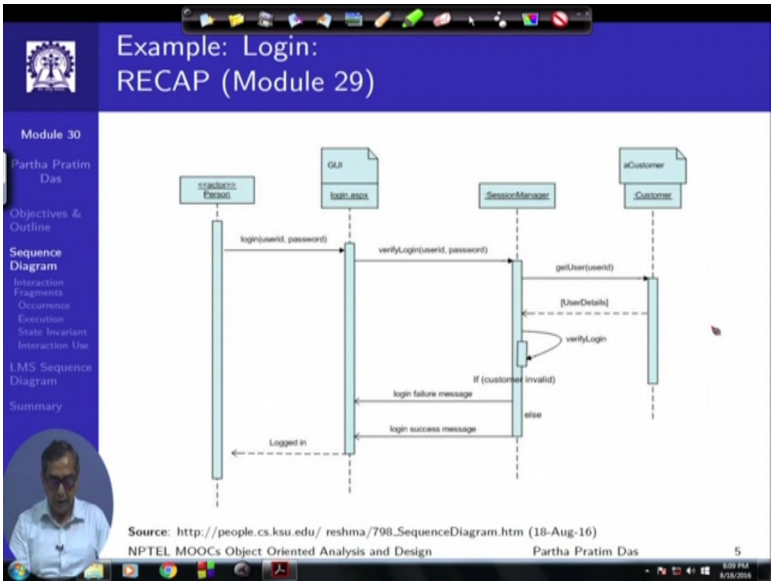
So, in terms of the sequence diagram we have already looked at the basic concepts that of a lifeline which is how does a object exist in the system over time, how it can be created within this diagram, how it can be deleted and how it can exchange messages and in that context, we have seen various types of messages synchronous, asynchronous, create, delete and reply messages and we have seen their classification by response type as well. We will continue in this module to introduce the concept of interaction fragments and look at some more examples.

(Refer Slide Time: 01:33)



So, this is what we have already seen that in the sequence diagram is a major diagram which depicts the inter object behavior and it is a kind of behavioral interaction diagram and these are the main components. So, this is just to bring you back to the same point.

(Refer Slide Time: 01:56)



Now this is one of the examples that we had talked of. So am sure you have already revised this. This we have left as an exercise last time. So, this is like a sequence diagram for a place order kind of use case in an online store. So, if we take a closer look this time, we have a person object o who is basically initiating this order activity. So, there is a gui to choose the product, so naturally the person sends a select message to the gui for the selection.

You can look at here, this is a synchronous message. Once this message has been received then in turn the gui will send a message, a product, a product that has been selected, that message to the session manager. The session manager in turn will add it to the will try to add it to the shopping cart. So, it is like you have selected the item, now you go and say that add to cart. So, once you have selected before it can be added to cart, naturally the system needs to check in the session manager in terms of what is the present status of that item.

Whether the item is available, whether the item has a so what is the current price of that item and so on. So, the session manager sends this message add cart item to the shopping cart and once the shopping cart gets that so what shopping ui here shopping cart has received the add cart item, then the shopping cart has to actually create a new cart item. So, it does not knew because this cart item was not there, it will have to be created and put in the cart, so of the cart item class, a new cart item is created.

And that cart item creation once this is over then a reply response message goes back to the shopping cart who then invokes the add cart item on itself. So now the cart item object has been created, so now it needs to be added to the cart. So it is a self-message, once that has been added then you calculate the price of the resultant price because of this addition. So that is also a self-message and the item has been added, it is ready for ordering now.

Now in this way the person can actually add several items to the cart and once she is ready for a checkout that is checkout as you know is a process by which we say that the items that are put in the cart you confirm that you want to buy them and therefore are ready to make the payment. So, you send the checkout message to the cart and in that checkout message to the cart, then the cart in turn has to update the profile of the customer because it has to check whether you are whether you had earlier orders.

Whether you have some default payment terms, whether you have already have some credit with this

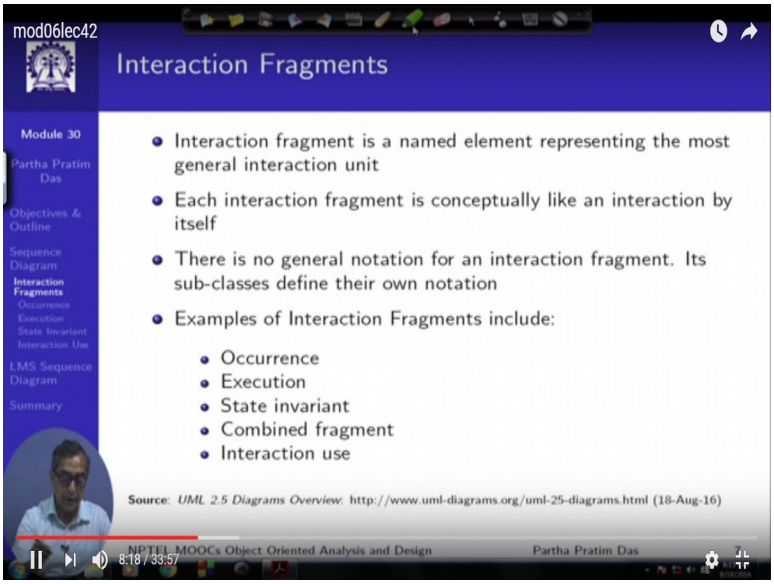
store and so on. So, it will send a message update profile to the customer instance and get a reply from that. Once that has been update then the shopping cart again sends a calculate total price, calc total price message to itself so that all items that have been added to the cart, their total price will be computed and on completion of that, a new message place order is initiated to place the order on the order object.

And once that place order execution is over; the order confirmation notification is sent back. Naturally this is just an overall flow of how the order is placed, of course there are several details for example there are several failure situations that finally will need to be modeled for example when we are trying to add a add an item cart item and calculate price, it is it is quite possible that actually the product that was selected, that item may not be available to be added to the cart.

So, what will happen in that case will have to check for that. When we update the profile of the customer here, it is possible that the customer has a profile where the sale order may not be a valid one. Certainly, there is a whole issue of a payment which is not shown here and it is quite possible that the placement of order may not go through in that case, we will not have a order confirmation but we will have a order regret kind of notification.

But leaving that aside that if we just work on the kind of as we say the happy paths as to what is likely expected path of flow behavior then this is kind of a diagram which show how place order will work. So, a task often in terms of sequence diagram would be to actually think about the total flow of events in the system in terms of the classes, the named entities and identify what message need to go from which object to what object so that in that order in that sequence we actually achieve the targeted requirements.

(Refer Slide Time: 08:16)

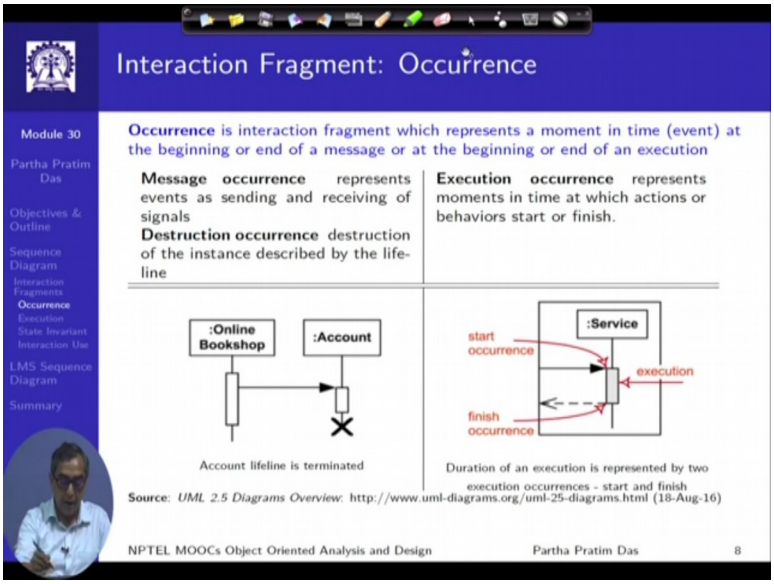


Now let us discuss about the interaction fragments. Interaction fragment is a named element representing the most general interaction unit. So, it is like the main idea of interaction fragments is a the sequence diagram as a whole is showing you several different interactions every there is a unit interaction between one class with another as one message is send and there is a sequence of messages sequence of synchronous, asynchronous calls, creation, deletions, all these giving you a total complex picture of interaction.

But if we just take a part of it then that part itself could be something meaningful and if we see that that part needs to be performed several times in several different places in a sequence diagram or in several different sequence diagrams then we can actually take those out and kind of make them as a as a reusable interaction part. So, interaction fragments have the general objective of talking about interactions at different levels which can be taken out and reused at different places can be referred at different places and so on.

As such there is no general notation for interaction fragments there are different kinds of fragments and each fragment has some implicit notation of its own. These are the common types of fragments we will now take a look into what these interaction fragments mean.

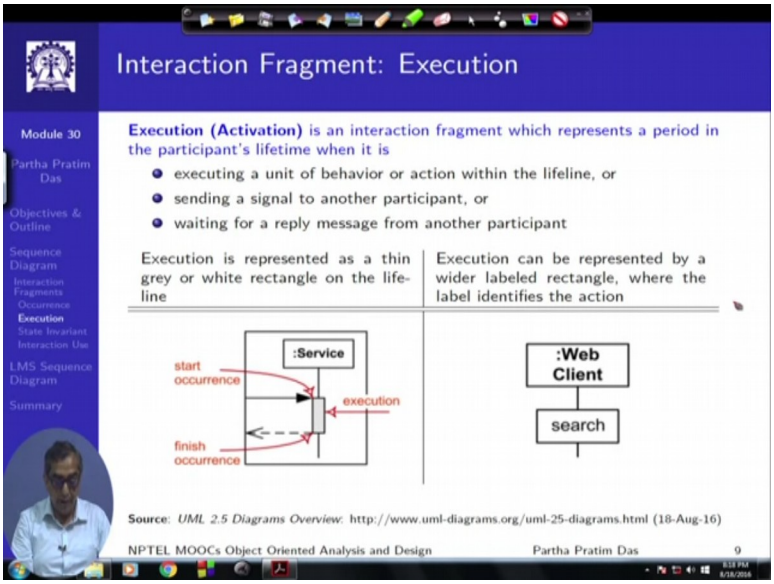
(Refer Slide Time: 10:03)



The first is the interaction fragment of occurrence. The occurrence is an interaction fragment which represents the moment in time or event at the beginning or end of a message or at the beginning or end of execution. So, if a message is sent from say online book store shop to account then this message has an event at the beginning, it has an event at the end. So, these are the occurrence specifications, similarly an execution is happening so there is a beginning of the execution and there is a end of the execution.

These are called the occurrences, so those occurrence specifications can be used to particularly associate different conditions different constraints at those. So, we typically have message occurrence which relates to the events at the sending and receiving signals, we have execution occurrence which is in terms of the beginning and end of execution and we may have a destruction occurrence as well which relates to whenever the objects are destroyed. So, this is the first kind of interaction fragment that is frequently used in the sequence diagram.

(Refer Slide Time: 11:27)

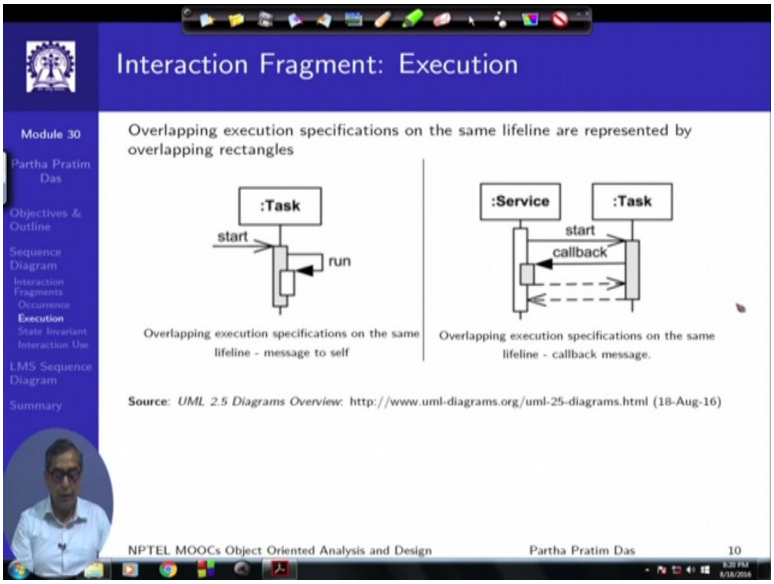


The second is an execution fragment, it is often called the activation fragment also. In an interaction, it is an interaction fragment which represents a period of time in the participants lifeline when it is actually executing. So execu it is executing a unit behavior action, sending signal to another participant or waiting for reply message from another participant. So, a an execution like in here hav has started based on some message.

And then during this execution one part is it is executing so that is the one sequence of things that can happen. Besides it may need to send some signal to some other participant, message to some other participant. If it is synchronous then it will have to wait for receiving the response on the signal that has been sent. So we depict these in terms of the execution fragment, so the execution is typically drawn out in terms of rectangular boxes, we have been saying this already so you have understood.

They are filled in grey or they could be just left blank or alternately we could also use a bigger kind of a rectangle and write the name of the execution so here search is an execution fragment. So, this is an execution fragment to so that basically means that it it is kind of an interaction, it is a kind of a small piece of interaction which happens at different times in the sequence diagram and builds up the total sequence of activities and we designate that with the start occurrence and end occurrence and the actual execution and signal link.

(Refer Slide Time: 13:28)



Interestingly it is possible that the interaction fragment of execution could overlap. For example, here we see so here you can see one in grey and then another in brown. So basically, here the task has had started an execution and at within this the task finds that it needs to send a message for getting some service. It so happens that this object the task object itself is a provider of the service. So, it sends a message back to itself, if I sends a message back to itself then in that same timeline it is actually doing another execution.

So that is what is shown by the overlap. So, when you have overlap, that means basically the same object is having multiple related executions going on at the same time. Here this is a synchronous one so when the grey one, the kind of parent one sends the run message to the to start the execution of the white part, this part then naturally the grey part goes on to fault. Because this is on a synchronous call. It can also be a different way that it can overlap is if we have some situation of a call back.

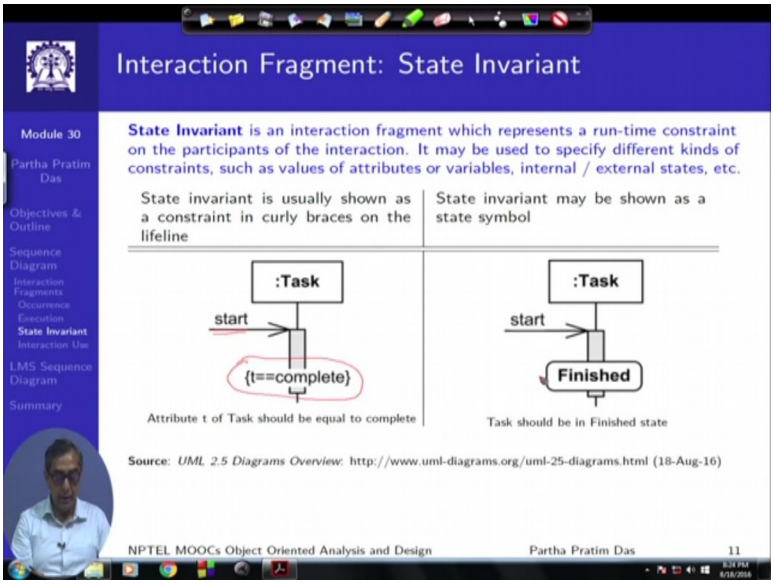
So, call back is like this here you have seen that the basically the message is back to itself. But it could also be that the message has been sent from the service to the task, the start messages it is an asynchronous one, so the start is continuing to execute. Then task in after doing some work in the execution find that it need some service it needs some execution to be done by the service itself. So, it sends a call back message. Why is it called call back? Because service had started the task execution and task is referring it back to the service.

So, it is kind of a loop finished. Here the loop is on the cell, here the loop is through another object. Now when it does that since this was asynchronous, the service is still responsive, because the service

is executing other ways it would have been on hold and then those those would have been nobody to receive this call back message so as a call back is received, a new execution overlapped on starts. And since this is synchronous at this point the execution of the task is on hold but the execution has fresh execution has started in service overlapped with earlier execution.

And when that is over you send the response back, when the response goes back again the task execution continues and finally when the task execution ends then you send the reply back. So, there are 2 situations as we had seen where overlapped execution specifications are possible one where there is a self-message and one where there is a call back.

(Refer Slide Time: 16:49)

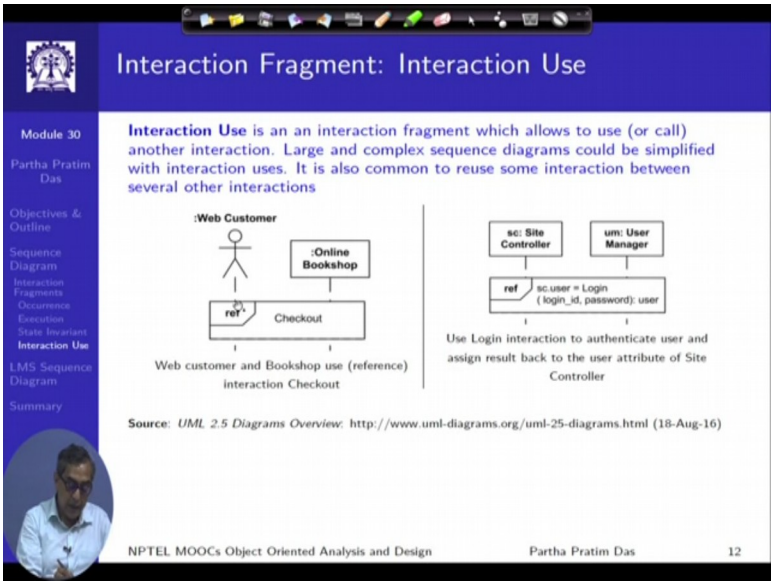


There are few other interaction fragments which are used, one is called a state invariant which basically says if you look at this, those executions which started here with this start message and here we write within curly braces that t is equal to complete. So, which means that there is an attribute t in the task and we want to say that when this execution will end, then t value of t should be complete. So that is what is known as a state invariant so we are saying that whatever may happen.

Whatever parameters that the start message could have but I need that whenever the execution ends, t is equal to complete. So, the state that it reaches must be invariant. So, it could be either it in terms of this kind of a condition statement or it could be written in terms of actual state values. So here what we are saying that when this execution ends, then the state must be finished. So that is that is the name of the state, it could be any any other state also.

So, this is known as a kind of state invariant so you as you can see this are more like constraints that we can put to ensure that the correct state has been raised or correct situation has been raised in terms of the sequence computation.

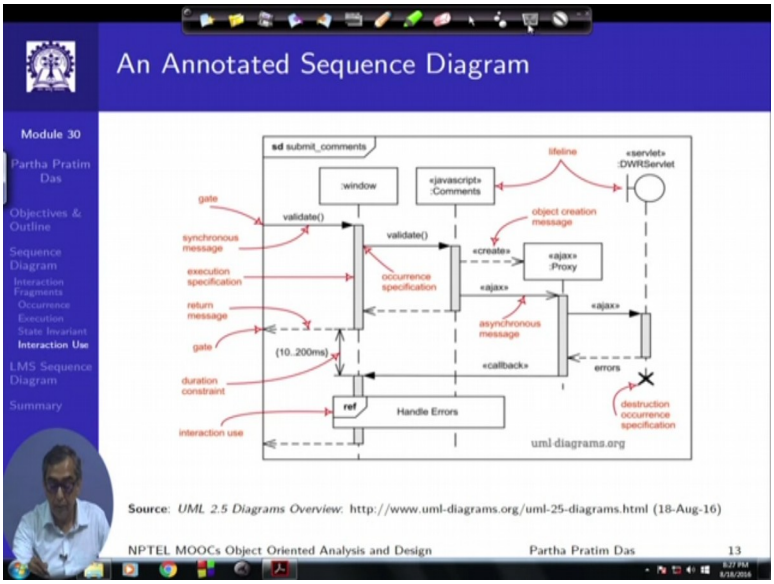
(Refer Slide Time: 18:25)



Finally, there is an interaction fragment called interaction use which is kind of a sub routine we can say. So, these are so if you have defined a an interaction somewhere say the checkout interaction that we just observe and then we can say that that checkout interaction part I can refer here which is ref means that I can actually give a name to that set of interactions, the sequence of messages and the execution and all that and at a different place I can just refer to it without actually again including all the details of this checkout interaction.

We could also instead of just doing a flat include, we could also actually pass messages, the login and password messages. So, login assuming that login was earlier specified as a interaction fragment, I can refer to that passing the new parameters here I do not need to detail out the whole of what this interaction is. When you do that we say this is an interaction use.

(Refer Slide Time: 19:43)

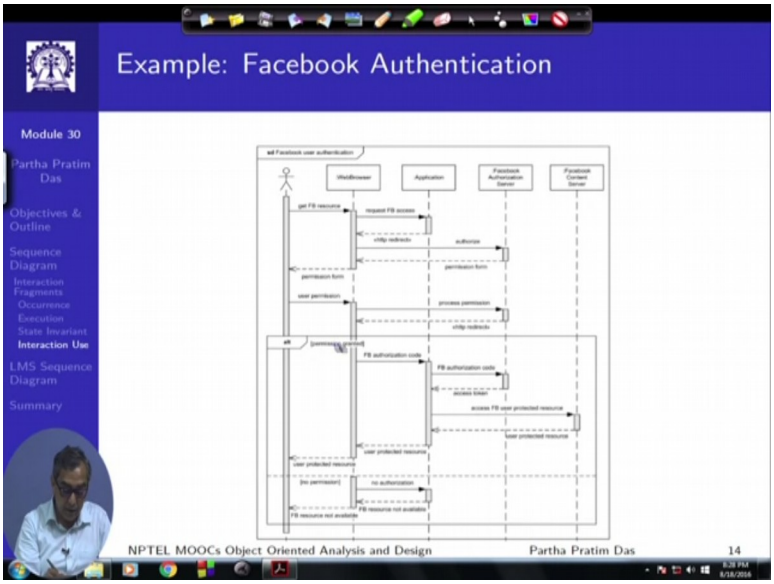


So, these are the different kinds of interaction fragments that are that are commonly available for. So, if you look into this annotated diagram, you can see different places where this interaction fragments are can be seen. For example, this is a occurrence specification, there will be an occurrence specification here as well. There will be in terms of the interaction there is a interaction use which is trying to use the handle error which has been defined suppose at least somewhere else.

Then there is a separate occurrence specification for destruction. So, you have now you have already understood the different lifeline and message details, you are understating the interaction fragments also. So now you are more or less equipped to understand the whole diagram and for example this is a there is a execution fragment with the execution specification, there is a start here, there is a end occurrence here and so on.

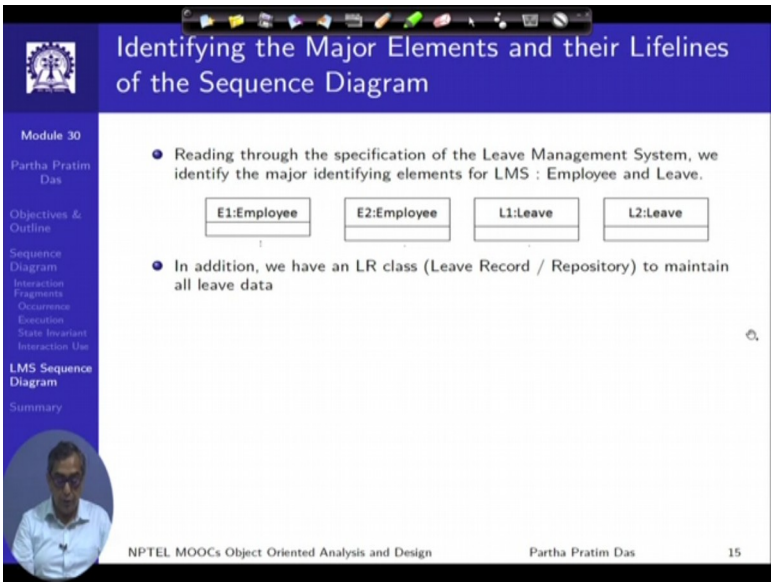
So, you can go through the we have been referring to this set of similar diagrams according to as we develop, as we learn more and more concepts in the sequence diagram so you can refer to them all here and get a better understanding.

(Refer Slide Time: 21:22)



I will leave this as an exercise for you to again try to follow. The Facebook authentication, you can see a fragmentation being done here I think there is no other no other interaction fragment used explicitly here of course you will have the occurrences executions I mean all ov all over the place. So, you can study them better.

(Refer Slide Time: 21:52)

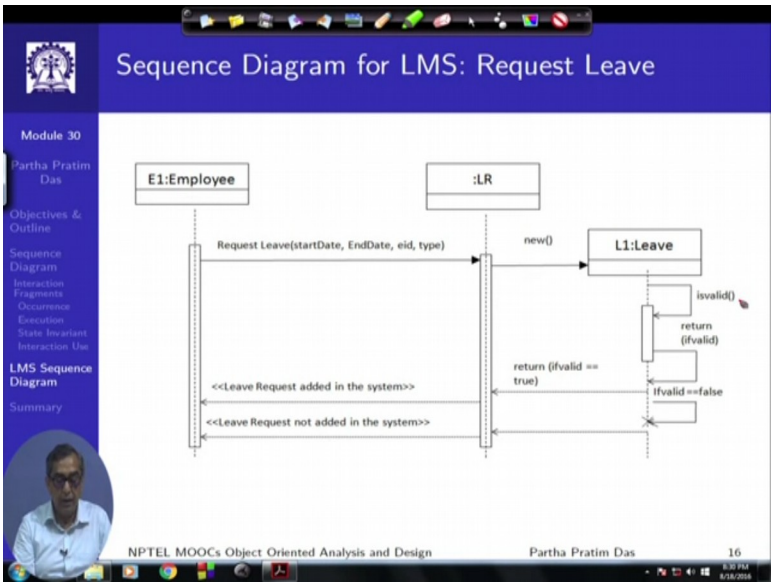


Now let us move on to using these in terms of our lms example in terms of the lms exercise. Now certainly we have been as we have been repeatedly coming through we have seen that there are 2 major named elements, identities in the lms the employees and the leave. So, while we want to build the sequence diagram for lms these will be the typical objects. So e1 colon colon employee which means that of the employee class, e1 is an object instance.

It gives another object instance. L1 is an object instance of leave and so on so forth. So, we will have these timeline lifelines on those. In addition, actually what we have not what is what was not explicit in terms of the specification of the leave management system but which is kind of a implementation class that we have to deduce is if there are so many leave requests resign in the system with the different you know state information has to whether the leave has just been requested.

Or it is been approved or it is been regretted and so on then we need some kind of a store to keep this leave information. So, we say we will have a LR which we can say a leave record or a leave repository where all these collections of leave will exist and so LR will also provide a major part in terms of the lifeline.

(Refer Slide Time: 23:30)



So, let us say if we just this is not a not a very detailed one as as I said that I would like you to detail out each and every you know diagrams that we are doing for the lms system but this is just to show you how to go about things. So, here is the 3 major lifeline items, E1 anonymous LR and L1. So how will it start, it starts with the employee sending a request for leave, so for the request for leave there will have to be several parameters.

So, this message will have parameters like start date, end date, the employees' id, the type of leave that the employee wants, there will be more. But some basic parameters s\will have to be there and whom would you send the request. Now the leave does not exist so the request have to go to the common repository store who is kind of providing the services of managing the leaves. So, this is kind of a managing the leave that will assume. So once the LR gets this, then LR knows that this is a, so it has

started this execution LR has started this execution.

Employee was already in execution so we are not showing the earlier parts of how the employee has logged in and authentication and all that. That will be a separate diagram again. But now LR has received the leave request so the LR has to create a leave record. So, you send a create message, here we have marked it as new and this leave record, this leave object gets created. Now we know that all kinds of leave are not valid in the system some are valid, some are invalid.

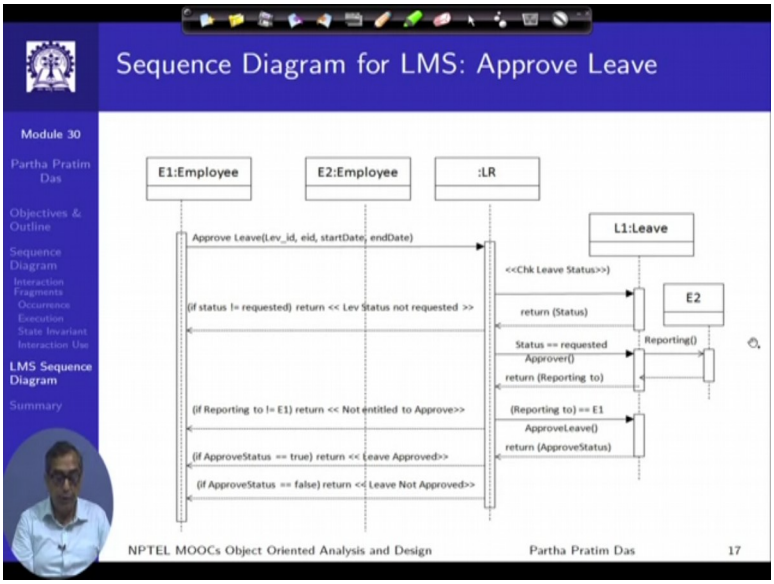
So, after creation this leave record sends some self-message saying is it valid. So, it invokes a message that it is valid so which is basically starts an execution to check if the leave is a valid leave or it is not a valid. So, it is expected to return some kind of a Boolean value that say if valid. So, when this returns if got this is valid so it returns this so it is a possibly that if valid may be false with the possibility that if valid is true. Certainly, if in case if valid is true we infer that a leave object has been created with the given parameters and it is a valid leave.

So, the response goes back to the LR at this point saying that we have got a correct leave record now. So once LR gets that naturally LR will again send a response back a reply back that is leave request is added in the system and based on that again execution here will proceed. But it is possible that the leave itself is not valid so if valid is returned as false, if it is returned as false then what will need to do we will need to do 2 things, one is we will need to have a delete send a delete message to itself because this leave is not attainable one.

So, it has no reason for this to be maintained and all these returns if valid being false, the return will come back to the LR again and when it comes back to the LR, LR now knows earlier it knew that it is true now it knows that the if valid is false that is the request to create the leave has not been successful. And that is why it got a false message. So, when it gets a false message then it will again send a response back reply back to the employee saying that leave request not added in the system because it is not possible because the request was not a valid one.

So just a just a simplistic diagram as yet but it gives you the basic flavor of the activities that has stated in the specification in terms of what is expected by an employee in terms of the request.

(Refer Slide Time: 27:55)



Now let us look at the next step if a leave has been requested then at a certain point of time it will need to be approved. So, we perceive that there is an employee E1 who has to be a leader, a manager starts and approve leave, sends an approve leave message to the LR to the leave repository which has all different parameters. So, there will be further details to that because we can always question that how does the employee know that such a leave exist.

So, will possibly assume that there is a list which is blindly shown to the leads and managers and they can pick up any one of them with the leave id of the leave that is created, naturally associated by employee id, start date, end date all those leave properties. So once the approve leave is sent to is been received by LR at this point, the LR has to first check that the leave inquisition actually has an appropriate status so that it can be approved. So, if a request has come for a approval on a leave which is cancelled.

That is not certainly acceptable. So, it sends a message to the leave, the leave object now to check the leave status. Executes and what is returned is the status of the leave. So, once it gets the status of the leave naturally there are 2 possibilities. One is the status is not requested leave, that is it is in some other state, it is cancelled or it is regretted or something. So, if the status is not requested then LR would immediately return a message ss send a response back to the employee saying that the leave is not in requested state and therefore there is no question of approving it.

In the other case if the status is same as requested which means that the leave has been created but yet to be approved or regretted then it will send a approval message to the leave which says that we want to

the LR as if is telling the leave that I want to initiate the approval process for this particular leave and I need adequate information further that. So, what is the requirement? We did a say in terms of the modeling constraints in the class diagram which we will use here.

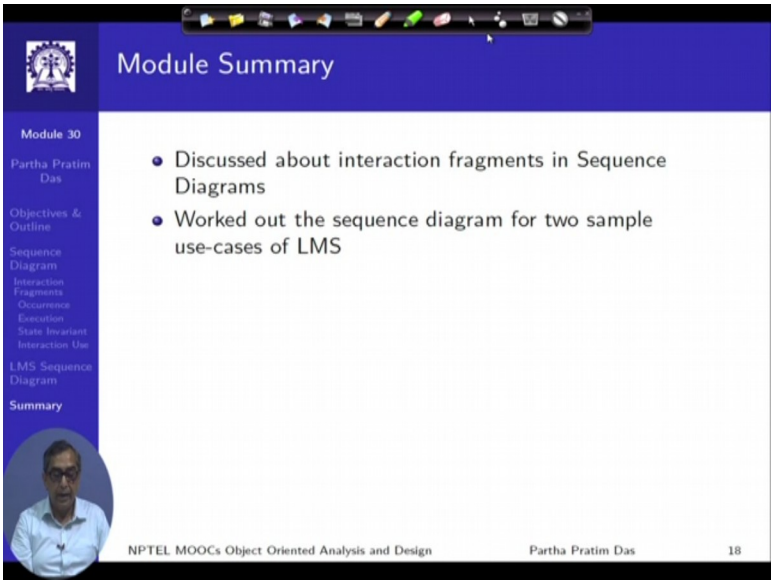
In the sequence diagram is approval can be done only by those to the only by that particular employee to which the leave applicant reports. So certainly, this execution of approval process has to invoke another message and that message is goes to E2 which will, so E2 is the employee who had requested for the leave. So, leave will send the message there and get back response from the employee as to who is the reporting person, reporting manager for E2.

Now once that has been received here, the reporting to, the information of the reporting person then the LR will have to check whether this is same as employee E1. Because employee E1 has initiated the approval so if it is E2's leave that E1 is planning to approve then E1 needs to be the reporting manager for E2. So, if the reporting manager is same as E1 then certainly the LR proceeds with the approved leave process on the leave L1 and that is possibly approved or it may not be approved because of other reasons, documentation reasons, duration reasons and so on and then send that status will return.

So, in case the reporting to is not E1 then certainly this part cannot proceed anymore so the LR will send a message back that reporting is not equal to E1 so not in entitled to approve the leave. Whereas if it is successful then the approval request will go and at the end of this execution from the leave object will get the approval status then depending on whether the approval status is true that is the leave can be approved then you say leave approved or if the approval status is false then you said leave not approved.

So, based on these conditions one of these 2 response messages will be sent back to the employee. So, this is just showing up the glimpse of how we can visualize the approved leave dynamics behavior so I would request that like this you know work on the other lea use cases and build up the sequence diagram for the cancelled leave, regret leave, reject leave, revoked leave and so on and that way try to complete the sequence diagram for the LMS.

(Refer Slide Time: 33:34)



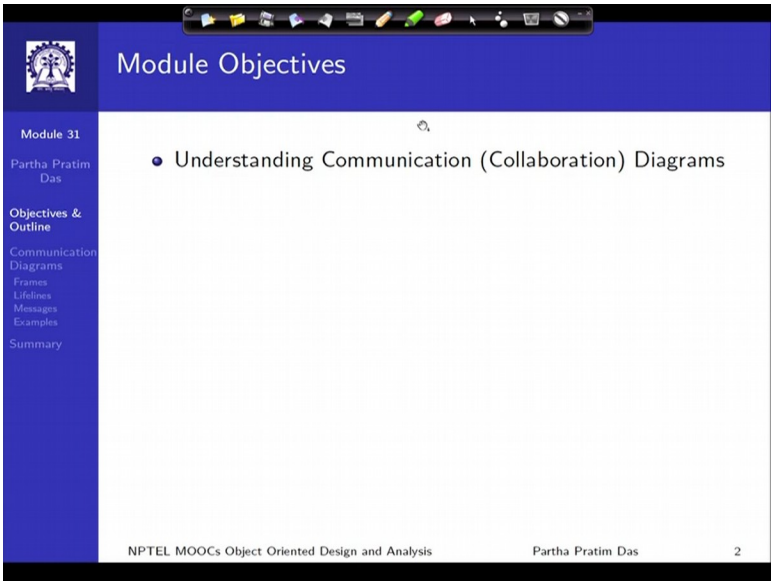
So, in summary here in this module we have completed a discussion on sequence diagram having talked interaction fragments and then we have taken a quick look into the sequence diagram of the LMS and for 2 use cases we have outlined how the sequence diagram will look like.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 43
Communication Diagram

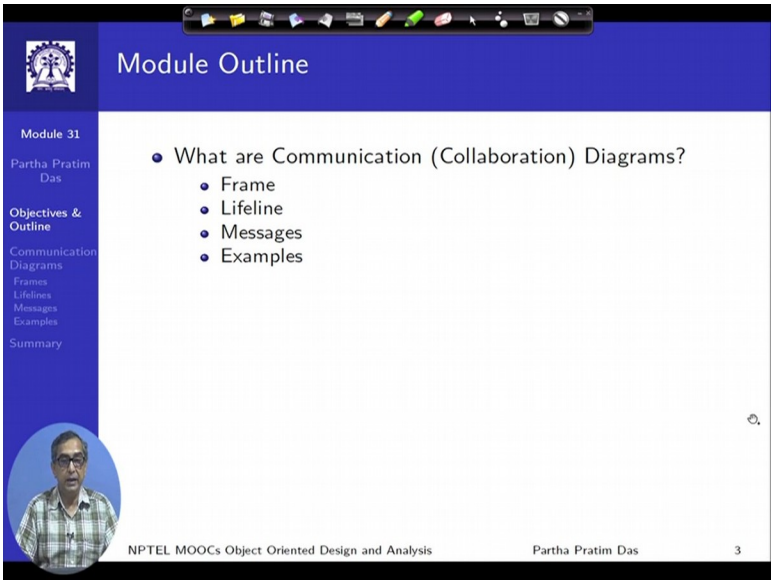
Welcome to module 31 of object-oriented analysis and design. We have been discussing about uml diagrams, we have covered 3 major diagrams the use case, the class diagram and the sequence diagram. In the current module, we will talk about communication diagram which is also an interaction diagram like the sequence diagram.

(Refer Slide Time: 00:47)



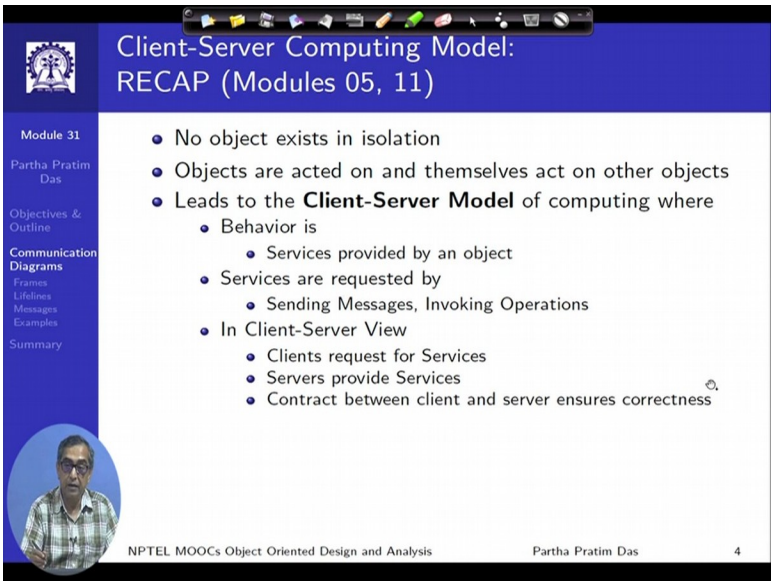
And it has a lot of relationship with the sequence diagram as well.

(Refer Slide Time: 00:52



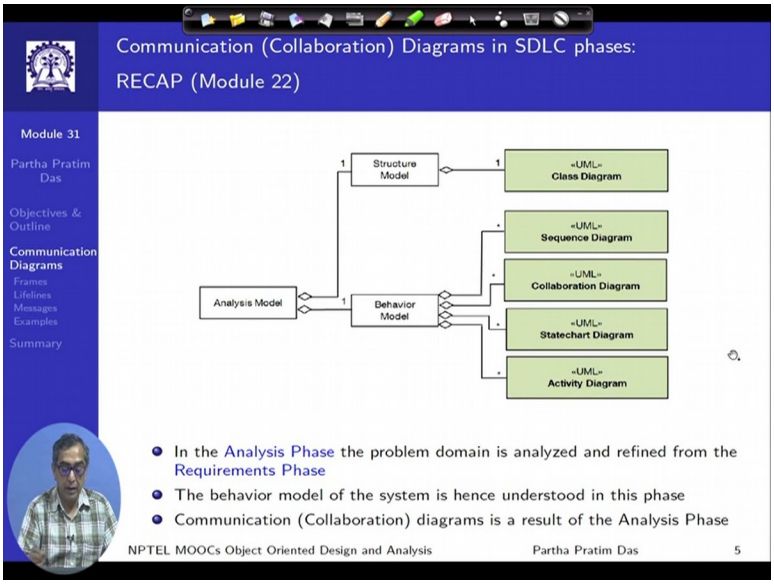
So, will talk about what is the communication diagram and what are the different components of that diagram

(Refer Slide Time: 00:59)



I would just like to remind you that for understanding this module well, you should be well-versed with the cli client-server computing model which we have discussed earlier.

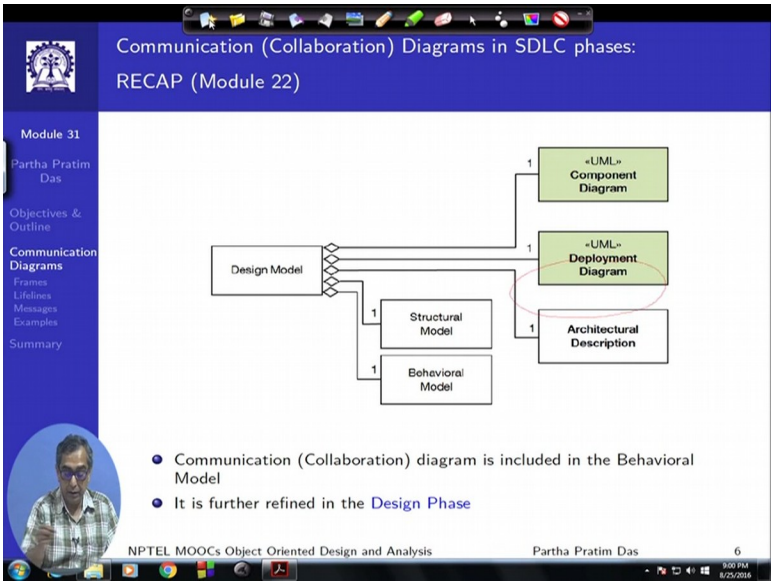
(Refer Slide Time: 01:10)



And the are communication diagrams and the they just note that we are often referring to it as collaboration diagram also because as history goes, there in the uml version of 1.x somewhere in the middle, this diagram was introduced as a collaboration diagram. Later on, when modifications were made and we got into uml 2.x currently that same diagram is been renamed as a communication diagram.

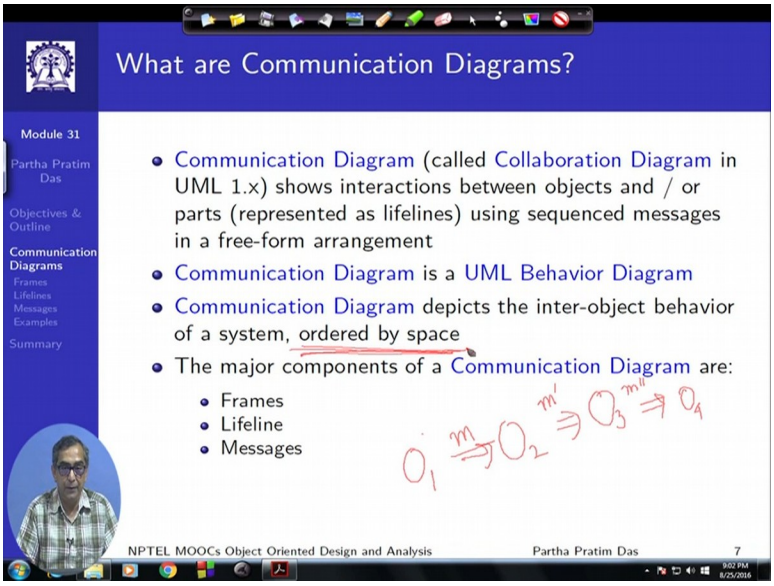
So, in the community both terms are interchangeably used. So, this diagram, this collaboration or communication diagram offers as a part of a behavioral diagram, the interaction diagram in the analysis stage that is where it gets generated and then it gets refined in the design phase as well which we had seen this earlier.

(Refer Slide Time: 02:07)



So, in the design phase it is actually bundle in terms of this particular behavioral model.

(Refer Slide Time: 02:17)

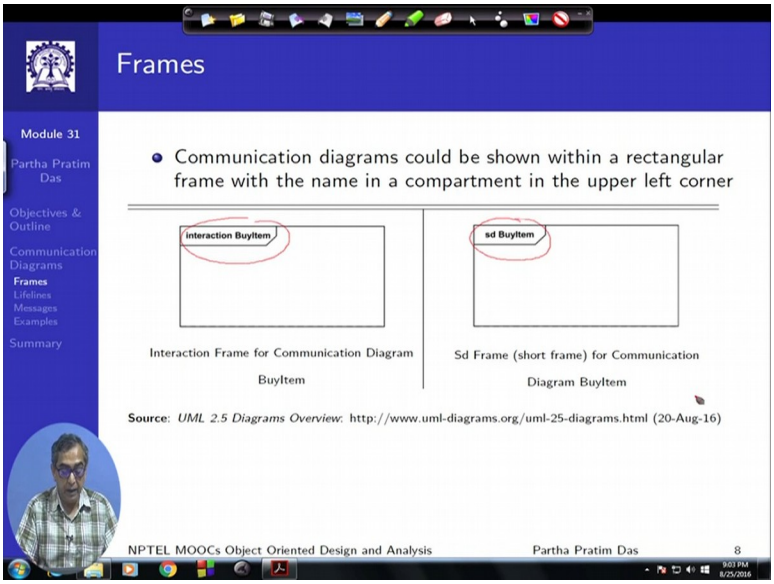


So now what are communication diagrams, the communication diagrams is basically interaction, as the name suggests it saying communication diagram. So how does objects communicate between themselves. So, looking at the client server model of basic computation is message is through messages, one object has to send message to another and in the recipient object has to react to that. So, in the communication diagram, we try to check this interaction between the different objects which are known as lifelines.

The messages are sequenced and we will see this more what the sequencing mean. do not get confused with when we said the sequence diagram. The sequence diagram talked about the sequencing which is strictly temporary in nature. So, we were always concerned in the sequence diagram in the terms of the looking at the messages in the order in which they happen in time. In communication diagram, in contrast we will try to look at what is saying as more in the order of space.

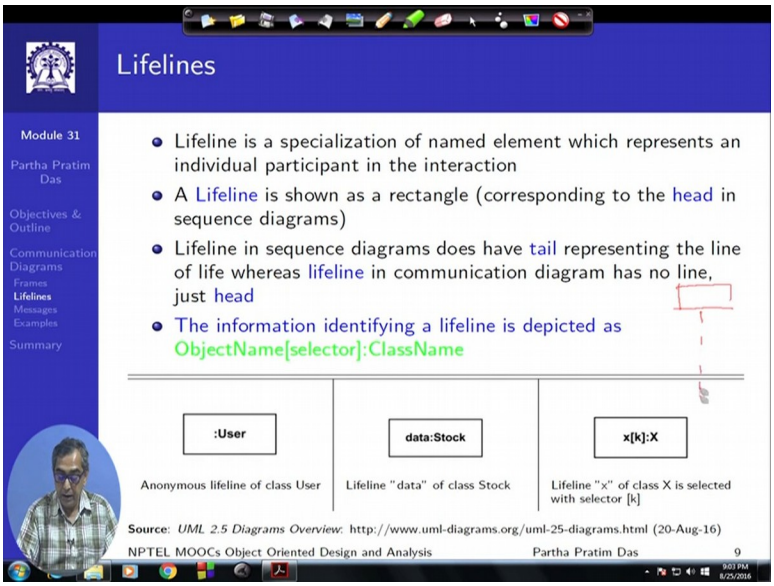
That is we would be more interested if that object o1 has sent a message m to object o2, then eventually when will object o2 send a message to object m3 and when object m3 will send a message to object o3 will send a message to object o4 and so on and it is not necessary that they will be in consecutive time bound but what we are trying to see is basically what is covered by space that is how objects actually receive and send messages, how do they communicate and that is the basic purpose of the communication diagram.

(Refer Slide Time: 04:18)



Typically, this diagram is drawn in terms within frames which are rectangles and that the corner, we have the 2 styles of naming, either you say interaction and then say the name of the communication diagram or you say SD and then you say the name of the diagram. SD is little bit of a misnomer it derives from the sequence diagram term but this actually means a communication diagram. So, this is how you will know that this is the communication diagram here.

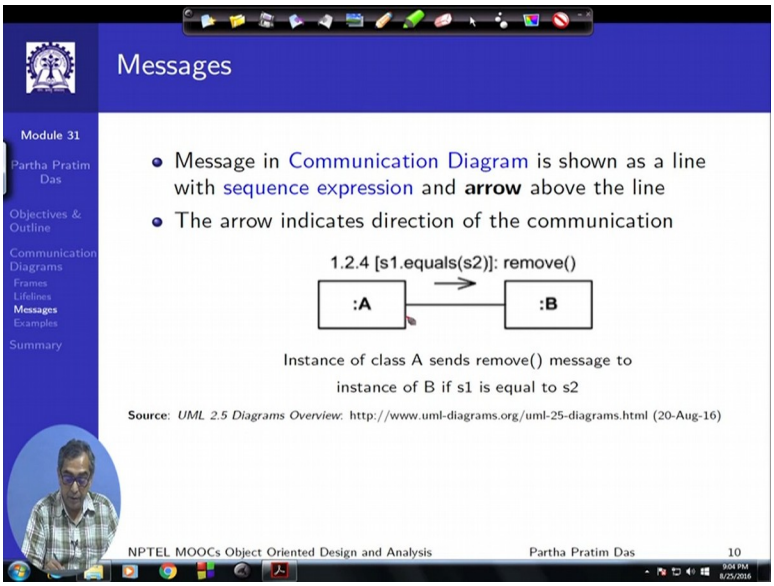
(Refer Slide Time: 04:45)



Then we have lifelines as as we had in the sequence diagram in you recall this a lifeline is an instance of an object and we had this that it is a box within which the name of the object is given and then we have a vertical actual line over which the object has a life. The lifeline in communication object just has a rectangle or the it is so called head of it, it does not have the tail. Its tail part is not there, it just has the head so that is called the lifeline in terms of the communication diagram.

In sequence diagram, you have the head as well as the tail, so these are typical heads that you have, we had seen this similar head in sequence diagram, this is an anonymous instance of the user class, this is a instance named data of the stock class and this is a collection data where am we are talking about the k th lifeline of the x collection. So, these are the basic terminology.

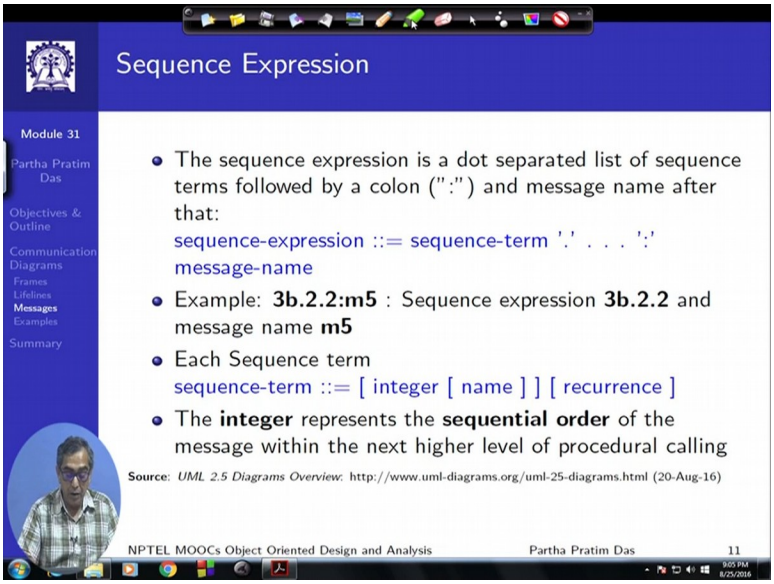
(Refer Slide Time: 05:42)



Then we have messages. Messages are shown in terms of a connection line and you show a arrow designating who is sending message to whom. So, this is an instance of class a, this is instance of class b and this is the message that being sent. You will see that we will just next go into what does this number mean, it is called the sequence expression and this the condition which says s1, if s1 is equal to s2 then the remove message will be sent.

So that is the basic structure of a message. Now the how the messages are sent or in what condition what message is sent is guided by the whole of the sequence expression which you see here.

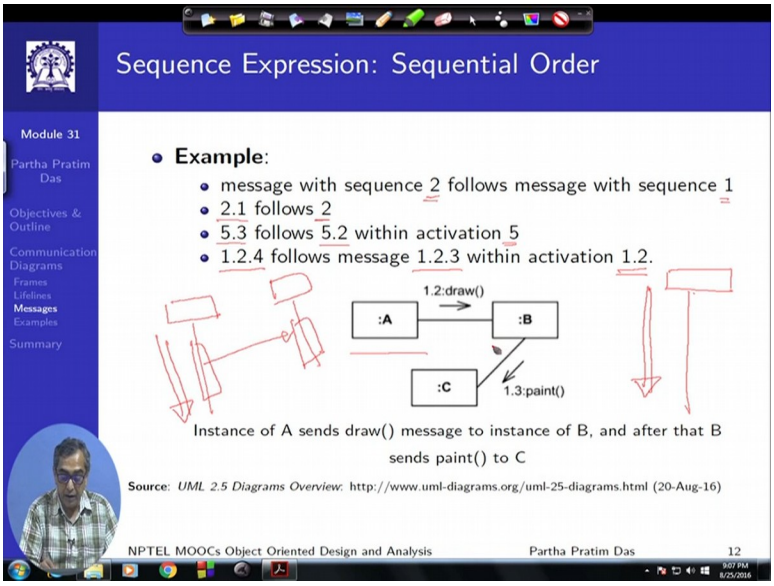
(Refer Slide Time: 06:30)



So, you will now next go into notion of sequence expression. So, the sequence expression, this is the this is the formal you know formal definition of the sequence expression or will just look at this. So, in the sequence expression you have 2 parts one is a actual expression and the other is a name of a message, they are separated by a colon. So, this is this tell me what is the sequencing of the message and in what way and this tells me what is the message.

So, these are 2 parts, sequencing and the message separated by colon. So, this integer basically these integers basically tell me the sequential order of the messages.

(Refer Slide Time: 07:11)



So, let us look into some examples. So in this sequential order, it is strictly integer order. So, a sequence 2 follows a sequence 1 and then in effect of 2 if you are doing a new message next, then you

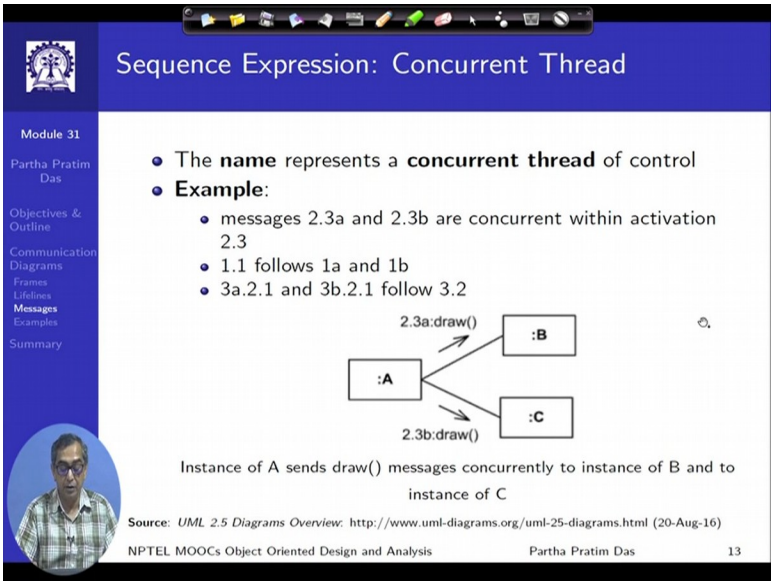
mark it as 2.1 which will follow the action 2. So, 5.3 will follow 5.2 in the message when the activation is for 5. 1.2.4 will follow 1.2.3 when the activation is for 1.2. so, it keeps on going like that. So, you see the messages in space but through the sequence numbers you can relate as to what is their ordering in the time domain.

So, the purpose like like in in sequence diagram if you if you recall we had the lifelines going like this and going like this and and basically, we wrote the messages on this on a on an order. So, this temporary order was a important thing, then of course you could see the special one because the different lifelines, I have one lifeline here, I have another lifeline here and I can say that this sends a synchronous message to this lifeline.

So, we could see who is sending what message to which other object but the interestingly the sequence diagram is looking into this time order. And interestingly the communication diagram is to look into the special order that who is sending whom and the sequencing information is encoded in the sequence expression. So here we wa if we read this sequence diagram it says that the instance of a is sending a message draw message to instance of B which is, so sequence order is 1.2

And on receiving that, B sends a message paint to an instance of C and therefore its sequence expression turns out to be 1.3. so that is how by reading this sequence number you can figure out in which order they are happening

(Refer Slide Time: 09:21)



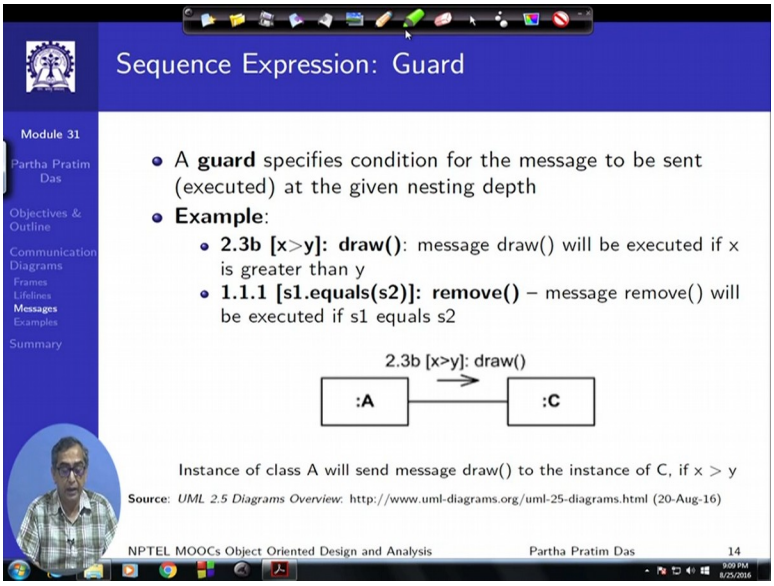
But the communication diagram primarily has to specify what is the special order, what is the object

interaction order. at times you are you can use a name along with this number, it is a you typically a symbol alphabet. If you use that then it gives concurrent messages. So here what is meant is there is an instance of a class A, it is sending a message draw to an instance of class B, it is sending a message draw to instance of class B and these are happening at a sequence order of 2.3.

So, in this, that is in the activation of 2, first 2.1 has happened, then 2.2 has happened, then in this the 2.3 A and 2.3 B, these are happening in concurrent one, that is between these 2, between these 2, there is no sequential order. They will happen at a same time, so this is what you specify when you write this named part along with the sequence expression. So, if I have 1a, 1b that means 1 who is taking place concurrently in terms of 1a and 1b and when both of these are ever then we will have the sequence expression 1.1.

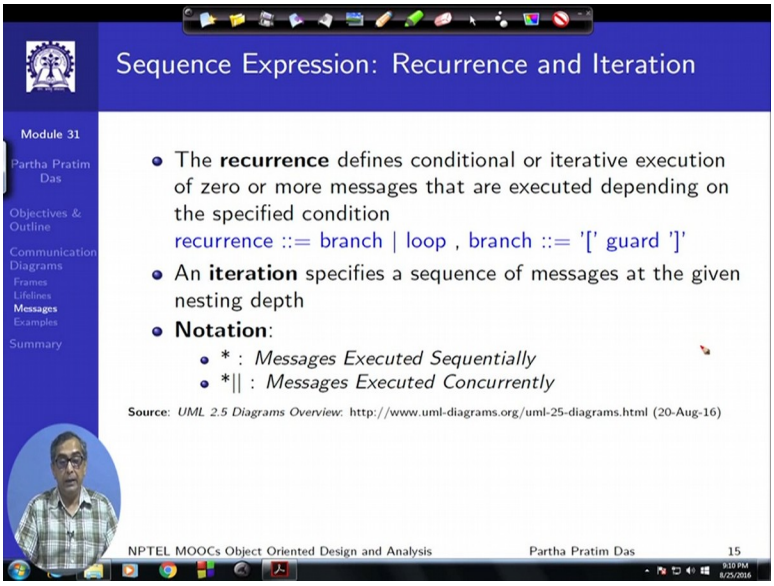
So that is the that is the basic so you you first resolve without the name and then once you have resolved without the name, then if the integer expression is same, the names may that they are all concurrent messages that will happen.

(Refer Slide Time: 11:03)



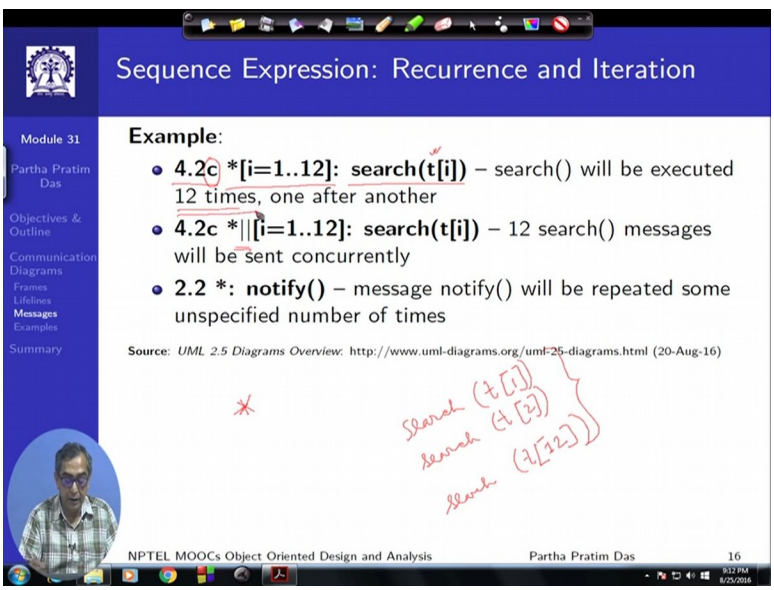
Messages can have guards, guards are like Boolean conditions, so here we say that this is a am sorry, we say that this is a guard, so guard is x greater than y. so it says that at 2.3 sequence at the concurrency b so there is some something else which is concurrency a. if x is greater than y then send that draw message. So, it will be executed if x is greater than y. similarly here is another the sequence expression is 1.1.1, at that level if s1 equal s2 then you send the remove message. So, this is these are the guard conditions that you can specify in terms of sending the messages.

(Refer Slide Time: 11:50)



Certainly, the messages could be sent in a recursive or a iterative manner. You can keep on sending number of messages to number of objects from the at the same level and when you send number of messages to number of objects you can have 2 protocols again. That you could send those messages in sequential manner, one after the other or you can send the messages in concurrent manner that is all together.

(Refer Slide Time: 12:23)



Example let us say am saying 4.2c so 4.2c means it is at a it is the second activation, second message in the activation 4. It is the c concurrent thread of the execution and in this am saying star I 1to 12 send the message, search it which means that the fact if I just write this means sequential. So, it says that from 1 to 12, you keep on sending search messages. So, you first search, search t1, then you send

search t2 and so on. You execute them one after the another.

But when this whole thing if I introduce to parallel bars that means it is a concurrent message which means that all these 12, search t1 to search t12 all these concurrent messages will be sent together, they will be sent at the same time. In this case, it will be sent 12 times one after the other, in this case they will be sent at the same instance. So, this is how these are 2 different kinds of iterations, sequential or concurrent iterations that you can specify.

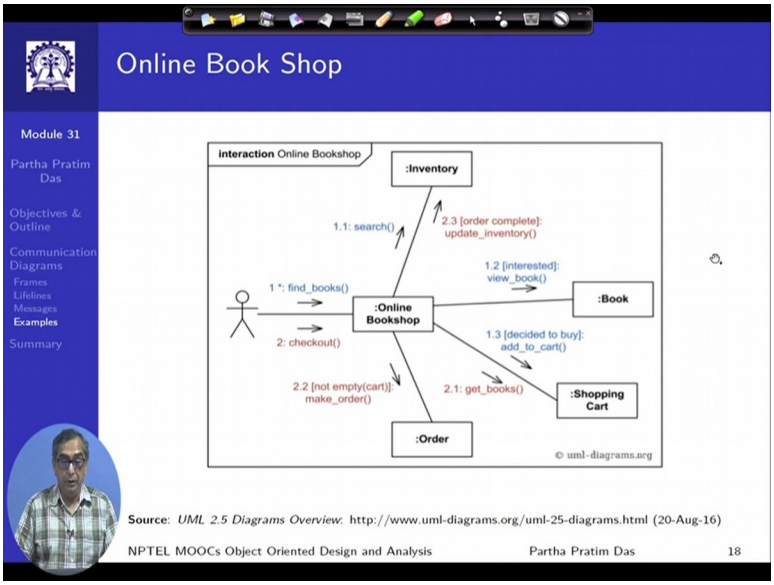
You may not specify how many times a message will be iterated, so it could be like just the star, so it says that at 2.2 level the notify message will be sent to, you do not know how many, star means it could be 0, it could be any number. It will be repeated arbitrary unknown number, unspecified number of times. It may be governed by some other factor.

(Refer Slide Time: 14:20)

`

So, this is these are examples of the iteration, for example we have already explained this is a going from these are n different requests being sent one after the another, this n different request being sent concurrently at this runtime.

(Refer Slide Time: 14:43)



So, let us look at an example to understand all that we have. So just to show you, I will, there is an annotated diagram also which has bring up for your practice later on. So, these are the different lifelines we have, just the heads, the online bookstore, the inventory, the book, the shopping cart, the order all the kind of different objects that we need if you want to buy some book from an online

bookshop.

If you look at these you can certainly see that these are the different messages that are that are involved here, just for convenience of and better understanding, I made the level 1 messages as blue and level 2 messages as red, otherwise they do not have any specific meaning in terms of uml. So, from these numbers you can figure out certainly the first message is this, where the actor this you know is the actor, sends this message to the online bookshop and what is that message, find books.

What is the nature of this message, you can see from this star that it is a repeated message, it is an iterative message. What kind of iteration, this is a sequential iteration. So, what does this message being sent from the actor to the online book store instance mean to start with, it means certainly as you can understand if the actor is trying to find books repeatedly, one after the other. Now but what what what happens once once the actor has sent this to the online store.

Online store in turn has to look up the inventory that the books are there. Because the online store is store will have to see whether the book is available, so based on one of these messages in this sequence, it will fa in fact for each one of these messages one by one in the sequence, it will be 1.1 that is a next message to send which is a search message which is to go to inventory and then inventory will find it not find it, whatever. Based on that then 1.2 is so now you try to perceive what is happening.

The the actor is sitting in front of the online book store window, trying to find books so there is some specify some keywords, some author names something, to find books which is taken by the online book store, then a message is sent to the inventory for the search and you have got a set of lists possibly and then the question is whether you select a book depends on if it is interest. If there are 10 books come up, you do not look at all books. So, you do a view book of a particular book object provided you are interested.

So, you model that by saying that the online bookshop sends a message to the instance of the book which is guarded by the fact that if the person is interested and that message is view book. And look at this was 1, this was 1.1, this was 1.2, so logically this is the next this that will happen. If you look into this in the sequence diagram then you will see this 1 first, then you will see 1.1 and the and the different life lines you will see see them in this order. But here you are seeing them in the special order of who is talking to whom for the task.

Now certainly once this has been done, the interested book has been booked up then in some cases, the person would be interested to buy, would be would decide to buy. So, 1.3 is a next message which is guarded by decided to buy. If the book the persons was interested in, the person is deciding to buy that, if that condition gets satisfied then the book is added to the cart. So that you say add to cart. So, this becomes in this sequence this becomes a next message.

So here you can clearly see that 1 is a overall activation of finding a book and that has been achieved through 3 sub activations of 3 messages of search from online bookshop inventory, view book from online book shop to book and finally add to cart from online book shop to the shopping cart. So, this kind of complete one activity and then certainly then what will happen again this is this is got a star, this is a repeated iterative process, so this process will keep on happening every time you send this out,

Send the find book out, all all messages may happen not all may be all messages will not, this this may be false, this may be false, so in that case the message will not get send. But in this order, this will keep on happening, keep on happening, resulting in this shopping cart I think filled by the books that the persons has decided to buy. So, what is the next activity that will happen? What is the next thing that is going to happen is marked as sequence 2 which is a checkout process.

So, you said. Now the actor sends a checkout message to the online bookshop. The online bookshop knows that now the checkout has to happen so why will that checkout will happen for, naturally from the shopping cart. So, at this level the next sequence message is 2.1 and that is to get thee books from the cart. Certainly, there is no guard condition because if you have been asked to checkout whatever there in the cart will be blocked for the consideration of the person.

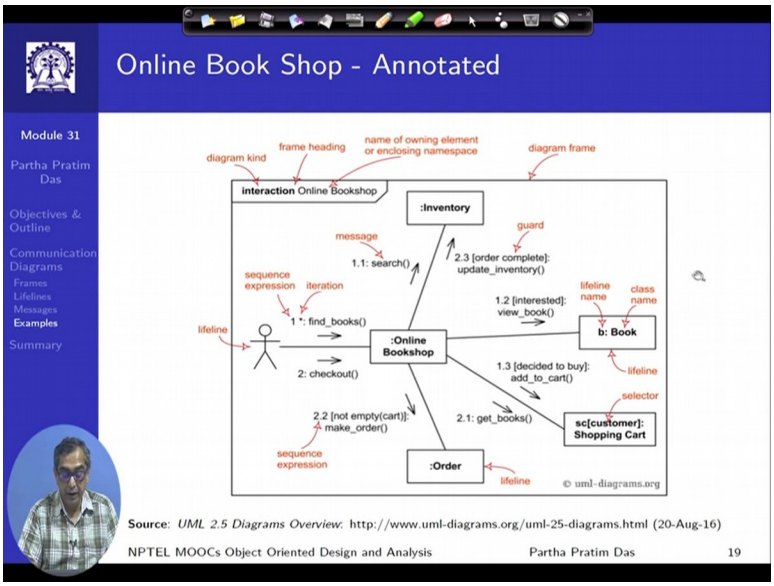
Of course, you can refine and make some you know some hygiene conditions as guards for example, get books will apply provided you have the some books in the shopping cart. Now once so that you will you will see subsect widely so once that has happened then you have got the books that there in the cart so this is the next message in the cart is empty or not empty. Certainly, if the cart is empty and there is no order to make so this message cannot be sent.

But if the cart is not empty even if it has got one book then we will proceed to make the order which is a message that the online book store, bookshop will send to the instance of the order activities order

class. So once the order is ready then so with they will they will be several several other examples here. For example, we are not showing that so the order will will have the process of demand and all that authentication all those will come up, assuming that all that has gone through

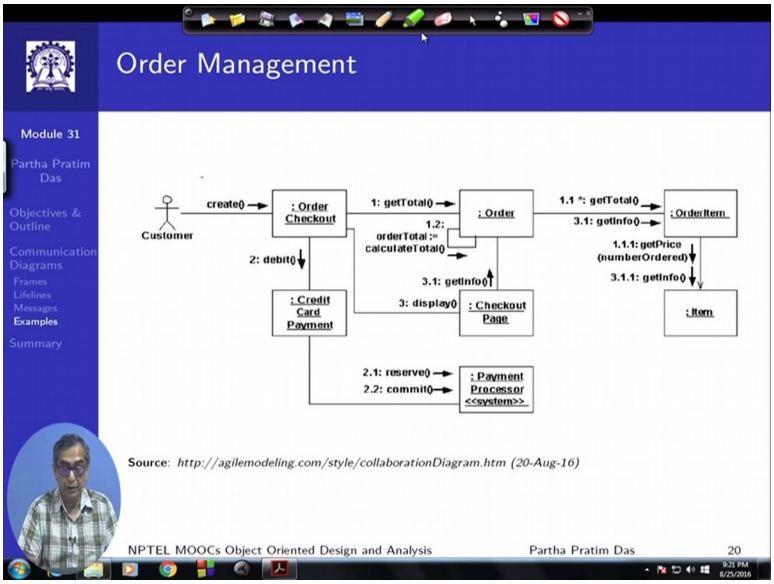
and the you reach a state where the order is complete, this condition becomes true, then you have the next sequence in this in the activation 2 after 2.2 is 2.3 where you say if the order is complete that is the guarded one, then you update the inventory because now they will have to be shipped. So, this is just to to show and explain to you as to how the em different actions in the online bookshop can be represented in terms of messages and how these messages are sequenced for the purpose of representation.

(Refer Slide Time: 22:38)



So, this is the same diagram only thing here is all the different items are annotated, so you can use this diagram to clarify if you have had any doubts and make sure that you understand the all different syntax of the communication diagram clearly.

(Refer Slide Time: 22:59)



There is another example here for order management which have primarily included for your practice. So, your customer who places an order so that is for order checkout part primarily and so this is this does not have a sequence expression which means this is what starts the whole thing then these are first action which is getting totaled. So, if you have to check out the order means ehe you have to actually start considering how much you have to pay.

So, if the order checkout sends a get total message to the order instance. You can see that in compared to the earlier one this is little little differently for example here for every message there is a function operator notation which is used. So, these are all optional stuff and after this has get total has happened, naturally the order object need to know what are the items that exist in the order. So, it sends after this 1, it sends the next sequence is 1.1 which is get total sent to the order item.

This order item needs to tell what is the total because it depends on the item. Now the item cannot say that it needs to the order item cannot say that it needs to actually go to the item in the inventory so it sends 1.1.1 which is getting the price depending on how many have been ordered. So that is the parameter that you send that these many have been ordered and these are next message. So once that has happened so this then within this within this

Then you go to the next stage in the processing which is after 1.1 next stage is 1.2 that is you get the total of the order because you have now got the price for each and every order item so your 1.2 is the total of the order which the order sends to itself. So here you can see that these are self-message kind of, we have seen self-messages earlier in sequence diagram also so this is the self-message which order

sends itself to find out that what is the total of this order.

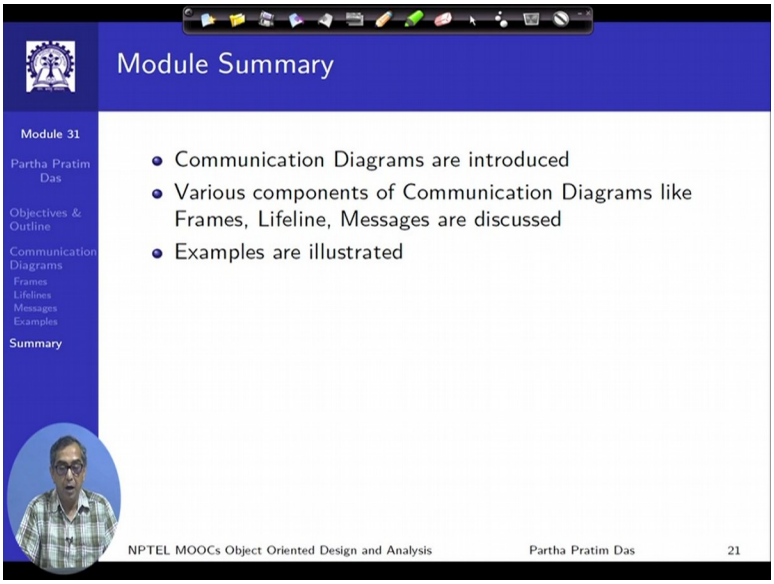
So now after having done this, you have the total of the order, so next step would be to so you have done with the messages starting with 1, so next activation is to which is that money the order total money has to be paid. So, you send a debit message to the credit card payment that within that activation you have to reserve block that with the payment processor which is the payment gateway or something.

After it has been blocked and your card details, identity has been authenticated, then it has to commit that is actually debited from your available balance in the card if the commit message happening and that will complete this part which is debiting the order total. So, the action 1 was activation 1 was done, now activation 2 has been done. Now what you expect, now you expect that all these information will come on the display and the details on the items that you have actually purchased now through this order will be known to you.

So, the third activation is a display which the order checkout again send to the checkout page. Checkout page needs to know what the order is, sends out 3.1 to get that info to order which in turn goes to the order item which in turn goes to the particular item and through this 3, 3.1 and 3.1.1 you get the list of items which are displayed on the checkout page. So, this is just another kind of so all these sequencing you can you can see that here dominantly you are showing

What are these different lifelines which are interacting between these with this messages to achieve the different purpose that they need to serve and at the same time you are following certain sequence expressions which show the original temporal ordering of their behavior. So, the same thing drawn in the sequence diagram would look very different because these are based on these numbers the messages will become order on the tail of the lifeline and these objects will not be placed according to their interaction but they will be placed just arbitrarily as vertical lifelines one after the other.

(Refer Slide Time: 28:08)



So, this is the basic fundamental to the communication diagram. So, in this module we have introduced the communication diagram, we have ref we have made references to how closely it relates to the sequence diagram and what is their interoperability, the sequence diagram is a temporal behavioral diagram for interaction and communication diagram is a special behavioral diagram of interaction and both of them are closely related but they define 2 different aspects of the dynamic behavior,

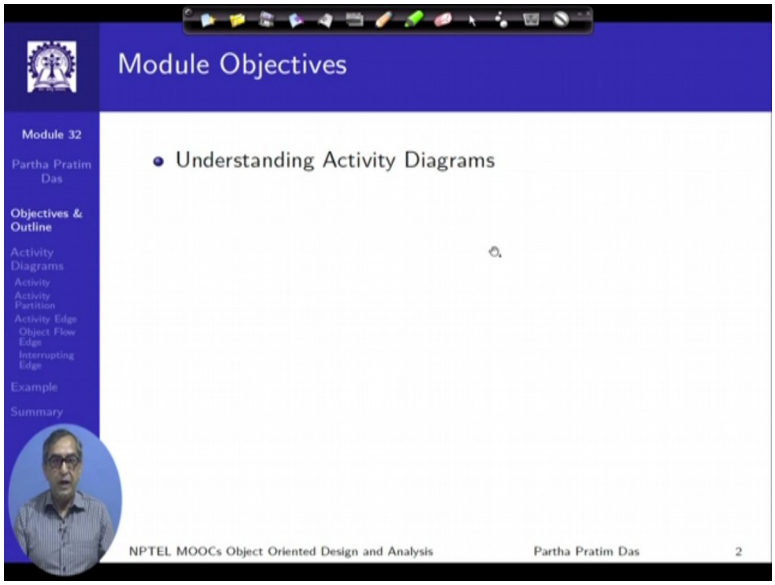
We have discussed also couple of examples and I would em encourage you to we have not covered it in the presentation here but I would encourage you to try to build the communication diagram for our leave management system exercise.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 44
Activity Diagrams Part I

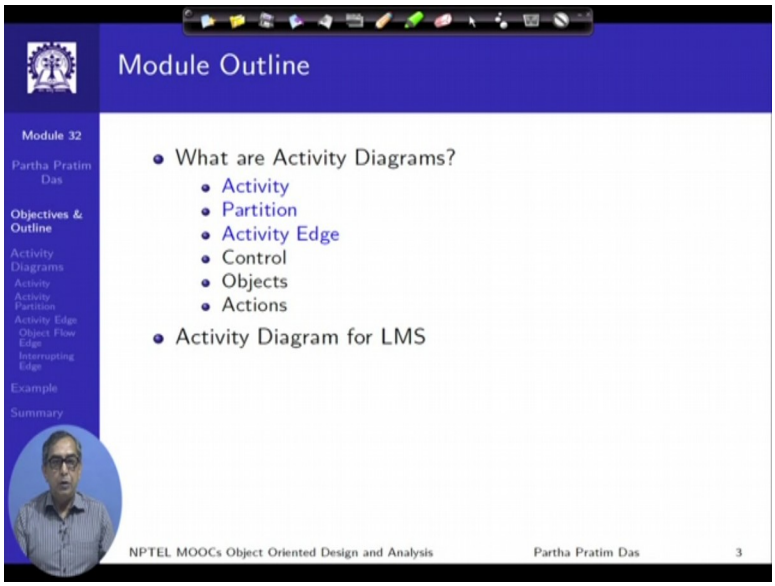
Welcome to module 32 of object-oriented analysis and design. In this module and in the next 2, we will continue we will discuss about the activity diagram of uml.

(Refer Slide Time: 00:34)



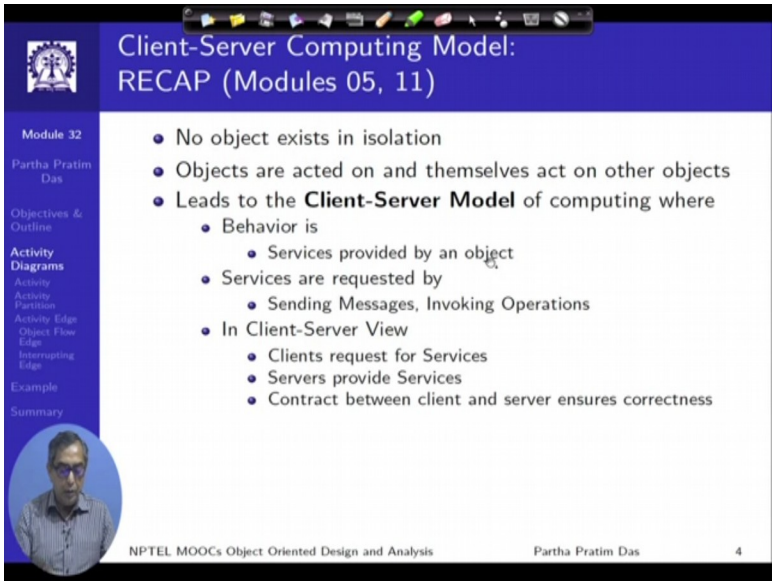
So, the objective would be to understand the activity diagram.

(Refer Slide Time: 00:41)



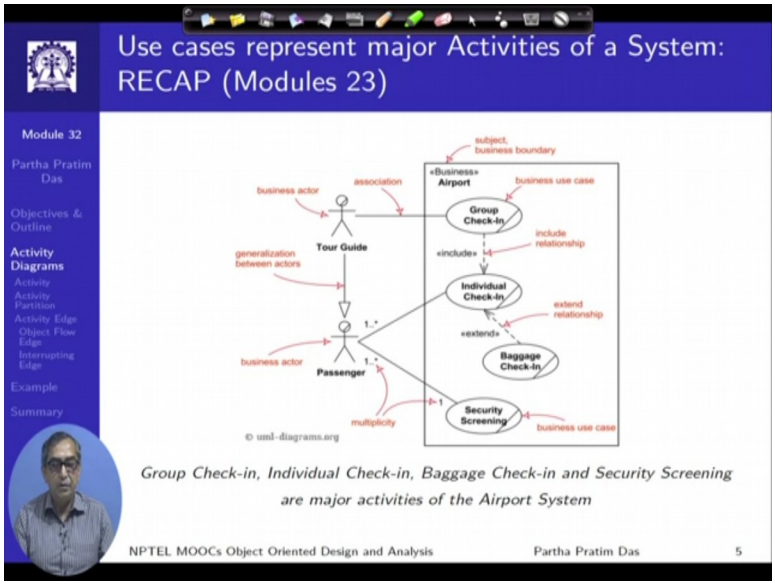
Since this will spread 3 modules, so this will be the overall outline to understand activity diagrams and in every module superficially we will show those items in blue which we discuss in that module. So here these are the first 3 items that we will focus on and this outline would be available on the left of your screen all through the slides.

(Refer Slide Time: 01:09)



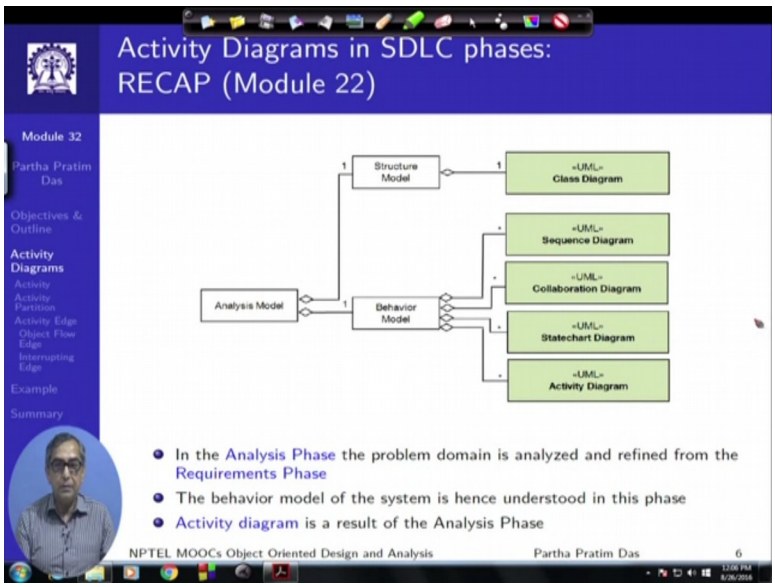
Now we will quickly I will quickly point you to the earlier items that you should be certainly familiar with and now be master of actually, the client-server computing model which we have been regularly referring to. Activity diagram will have a deeper bearing with the realization of the client-server computing model.

(Refer Slide Time: 01:34)



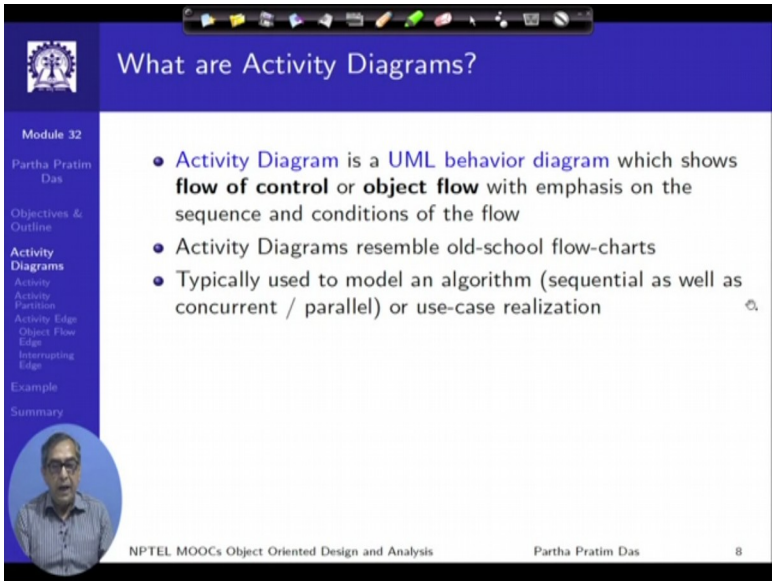
We have say a major reference to the use case diagram and the use cases because the use cases represent the major activities that happen for a system.

(Refer Slide Time: 01:49)



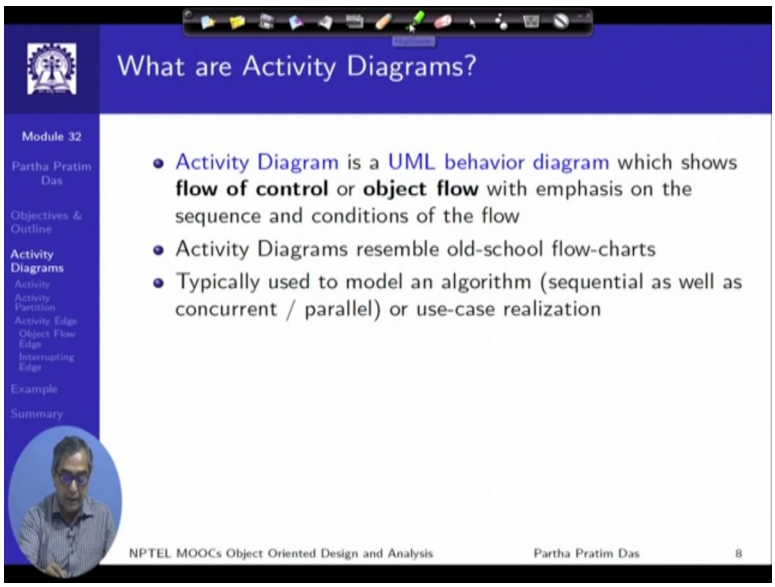
And for in terms of the sdlc phase this is where the activity diagram lies, so we are talking about primarily the analysis phase after the requirements phase where the basic understanding has been made and here where the activities will have to be captured in terms of the activity diagram and that will continue through into the design phase where it is a part of the behavioral model where it is further refined based on the better understanding or the basic approach to the design.

(Refer Slide Time: 02:20)



So that is a the context in which we will discuss the activity diagrams and I will urge you that the previous diagrams that we have already discussed particularly the use case class and sequence diagrams which have a frequent bearing on the activity diagrams particularly the use case and the sequence diagram. So you should be you should clarify your understanding on those diagrams before you go through this video.

(Refer Slide Time: 03:00)



The activity diagram is a behavioral diagram am sorry, activity diagram is a uml behavioral diagram so it shows the flow of control or the object flow emphasizing the sequence and the conditions of flow that is the basic point. So now you will find that a quite a part of the activity diagram would actually look like what earlier we used to call the flow chart. So, they are kind of refinements or advancements on the flowchart representation which we used eeh a very early stage as a model to represent certain logic.

So, they are typically used to model algorithms, so so far, we have only talked about the the objects the intended use cases, the properties and operations and in terms of the operation we have just talked about identifying the operation or in terms of use cases, we have just talked about identifying the use case. We know there is a request leave, we know there is a approve leave and so on, these are use cases and then they made their inroads into the corresponding class diagram.

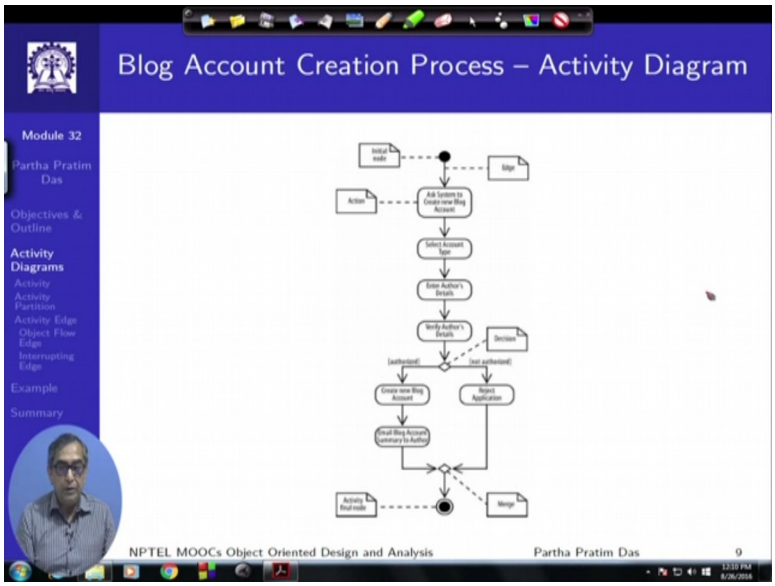
But here we would be more interested to really look into the algorithm that go in for performing the request leave, for performing the approved leave and so on. And importantly in today's system many at many a times, the algorithms are not necessarily sequential, they may be concurrent algorithms or they may be parallel algorithms. The basic difference being that a concurrent algorithm is where multiple threads of control apparently execute at the same time while I may actually have only a single, even a single processor.

So, concurrency basically talks about multiple things happening at the same time, we have talked about concurrency earlier too and in fact we could have multiple cores in the target system or multiple processors. So, the concurrent tasks could actually happen in a parallel mode so that they are not time sharing, they actually have separate executions possible. So, all of these should be

representable in terms of the activity diagram and so we will use all that to realize the use cases.

So that is the basic you know requirement of the activity diagram, that's the basic space in terms of the uml representation that we are trying to fil in.

(Refer Slide Time: 05:40)



So just before I go into the neatly bity I just put up a simple activity diagram. This is like a describing a process to create an account for blogging. So, these are the main things you can see that there is an initial node where everything starts, we say is a start node where everything is starting for that process and then you see a series of boxes which describe the different steps that you do starting from their access, ask system to create new blog account.

Certainly, that is the first thing we will have to do, you will have to select what type of blogging account you want, then you enter the details that is entering authors details and then the author detail will have to be verified in terms of the email access and you know other required items, if payment is involved and all that. So, these are the different steps, these are different activities that this whole process will involve and then in some cases like in here.

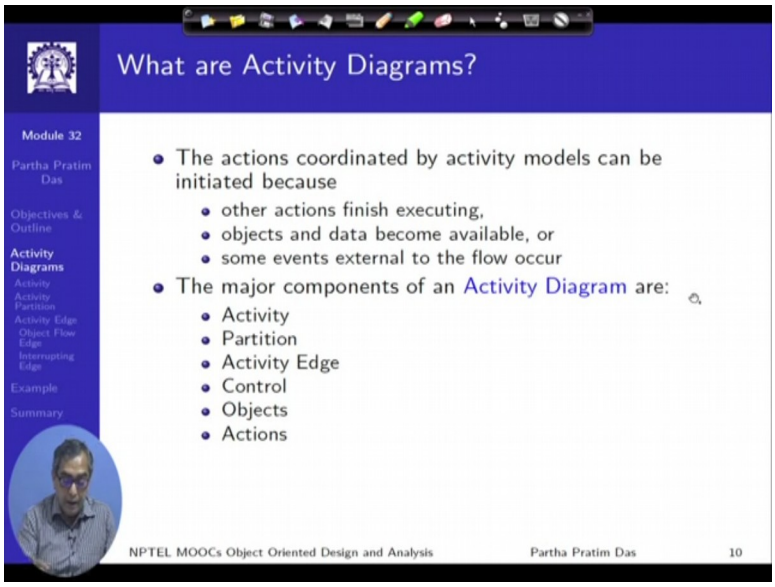
The outcome of verifying author details could be that we have an authorization that the verification turns out to be ok or the verification fails. So, based on that we will have a branching in terms of the flow so it could be authorized or it could be non-authorized. If it is authorized, then you have further actions of creating the new blog account, email the blog account summary to the author and so on. On the other hand, if it is not authorized then the application gets rejected.

And based on either of these you know flow paths being taken, finally we reach a a common merge

point where which in this case denote the completion of this particular process in terms of an end node, final node. So, if you look into this I have taken particularly taken an example which looks pretty much like the flow chart, we will see that how we will have various different aspects so that it will get more enhanced from the flowchart.

But in this way activity diagram would be used to define describe different processes that correspond to the algorithms or correspond to the use case realization.

(Refer Slide Time: 08:00)

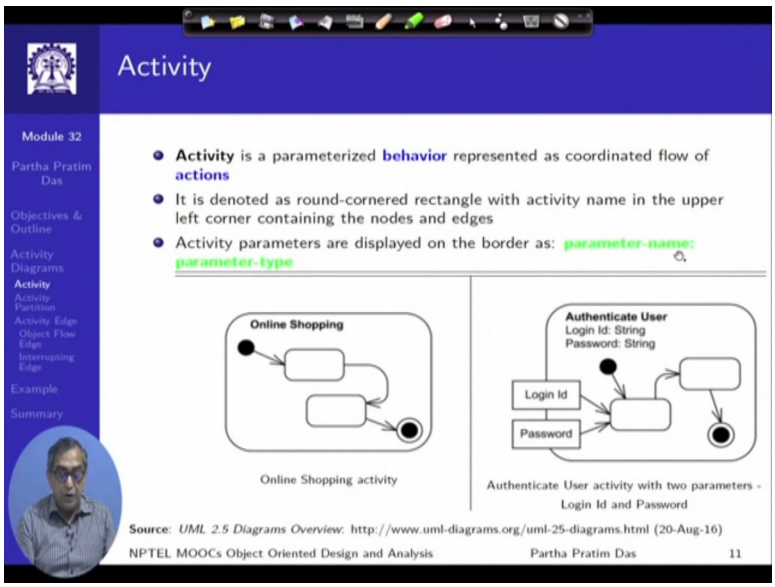


In terms of the so in formally speaking, the activity diagrams have actions which are coordinated by activity models that we are going to describe and these models can be initiated because some other action has finished. So, as we saw in the earlier, once you have filled in the author details then it goes for verification. So once one action finishes the other takes up. It may be initiated because certain object and data become available.

For example, you are logging in so you provide the user id and password. So, once you make that available then the action for verification of the login starts or may be because of some other events external to the flow is happening in terms of receiving a message token from someone or having certain system conditions and so on. So, activity diagram will show you all these actions and how they are interrelated. In terms of the components, these are the different components and I will not go through them right now.

We will take up these one by one and show how they are represented, what do they mean and how finally they help us achieve the activity description.

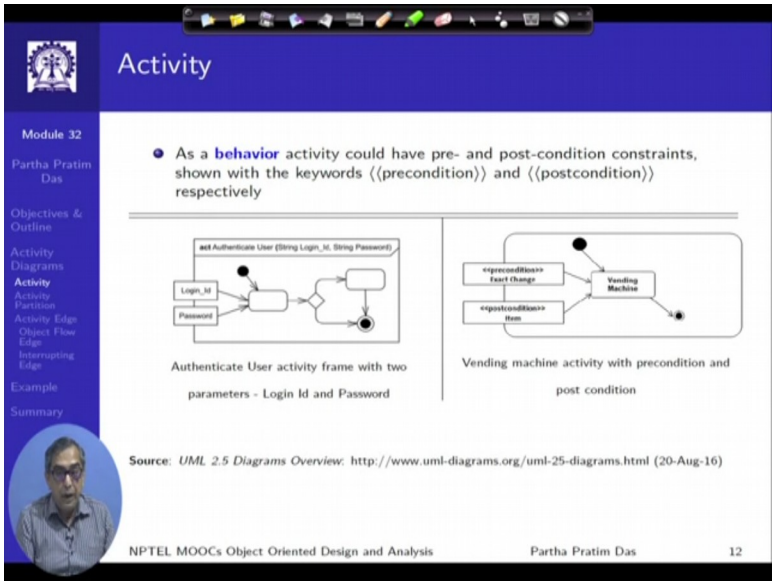
(Refer Slide Time: 09:19)



So certainly, the first is an activity which is certain kind of actions. So, here we have an online shopping activity which again in turn have certain sub activities in that. So online shopping these are not described here, we will come to the descriptions but it could be that this has sub activities but what is usually unique in terms of activities that there is a start and there is an end. So, the object there is a unique point where the activity starts and there is a unique point where the activity ends.

It could be parameterized also, so this shows another activity where authenticate user activity which needs to take these 2 parameters login id and password does some sub activities and starting from the start node, which is the final node. Look at the particular type of rectangle that we show for describing activities, this is not a cornered rectangle that we typically use, this is kind of a rectangle with rounded border. So, when you see that you know that this is a activity there is a parameterized behavior of an activity that is being talked off.

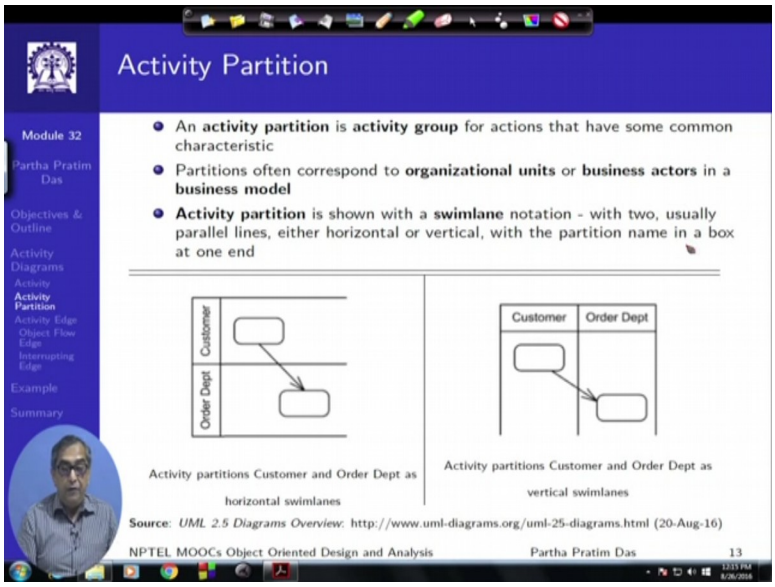
(Refer Slide Time: 10:29)



So, you could put a total set of activities, flow of activities in terms of a rectangular box also, provide it a name as the activity action and provide the parameters for that. There could also be preconditions and post conditions on certain activities. For example, there is a vending machine, vending activity where the that is you that is a machine where you go put your coins and you can get a a drink or a news paper or something like that.

Certainly, the pre-condition is that you must be able to provide exact change in terms of points and the post condition is you actually receive item. So, this is the basic unit the basic activity that will build up the activity diagram.

(Refer Slide Time: 11:20)



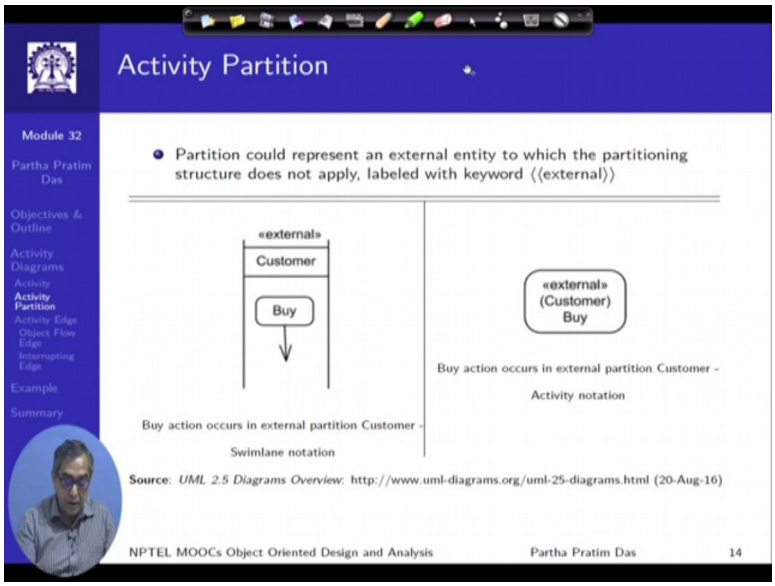
Often will find that there are different agents and typically if you relate back to the use case diagram these agents are actors and activities will relate to certain actors. So, when that happens then we try to partition the activity based on which actor is involved in this activity. So, it is a ordered department, there is a customer, so it could be in terms of actors or it could be in terms of other partitioning, it could be based on location.

It could be based on organizational structure and then you kind of draw swim lanes within which you show the activity so what it means is if I show this swim lane for order department then all activities that are shown in this partition shown in this swim lane are supposed to happen in the order department are supposed to be acted on by the order department here, it is for the customer and you can draw the swim lanes in a horizontal manner.

As you can see here these are horizontal swim lanes or you can draw them in a vertical manner, that clearly the swim lanes will clearly show you that how the activities can be organized in

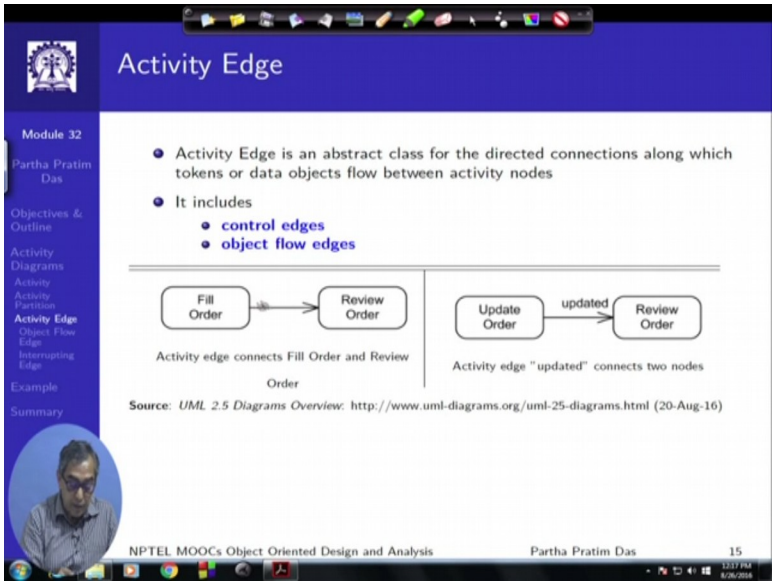
different parts of the system.

(Refer Slide Time: 12:33)



So, with activities and partitions you could also have actually external partitions, external partition is one where it is not a part of the system that you are build in but some certain things that you expect the external agent ex actor to really act on and prove take part in the activity diagram as a whole.

(Refer Slide Time: 13:02)

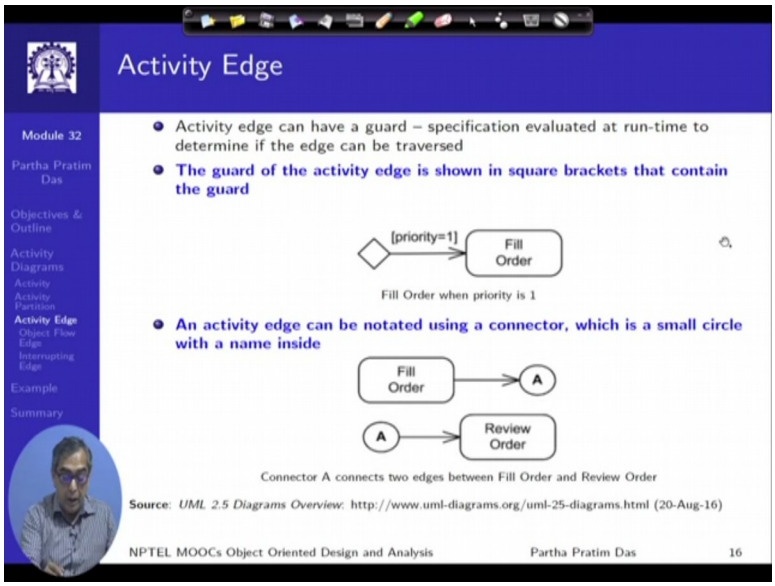


Activities have edges so there is activity fill order here, there is a activity review order here and there is a directed edge between them. Certainly, this shows that from the final node of this activity to the initial node of this activity there is a flow that is happening and we will see that typically this edges would be control edges or object flow edges where objects could also flow tokens could also flow from one activity to other through these edges.

They could the edges could have certain annotations which say that what is the effect of that edge

and so on. Activity edges with the activity nodes really make up the diagram.

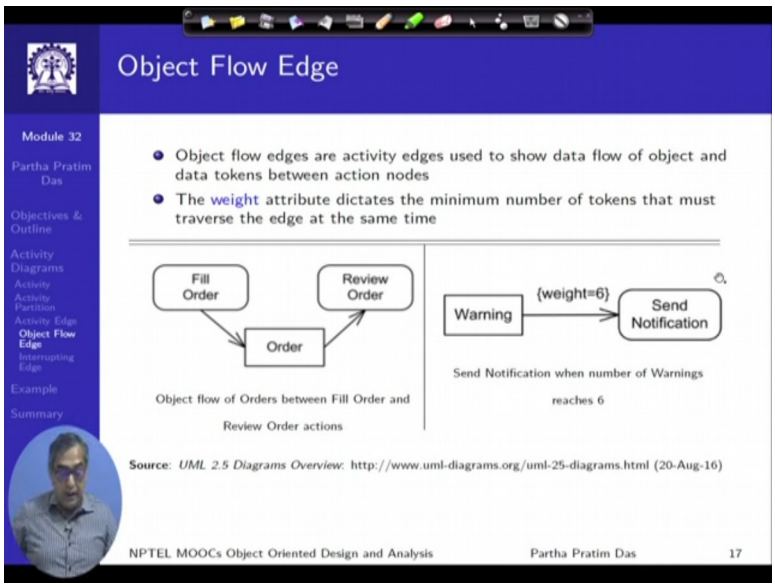
(Refer Slide Time: 13:49)



In further refinement, edges could have guards, we have been talking about the concept of guards in different context. So, what does this mean this we have not yet explained this, this is this basically is some kind of a decision or merge node, we will see later. But from this end of the activity you fill order activity the if the control has to flow in that then we are putting within square bracket a certain Boolean condition.

So, this means that this edge will be taken provided this condition turns out to be true. You could show the edges I mean in a complex diagram it may not be always possible to draw all the edges, it might just become a clutter of lines going from here and there. So, you could also you know divide a flow in terms of showing certain level markers. So, you say that the flow order this this a is just a continuity marker, so it goes up to a and wherever else you have a, you know that there is basically the same continuity. So, it you are showing this edge in 2 parts.

(Refer Slide Time: 14:56)

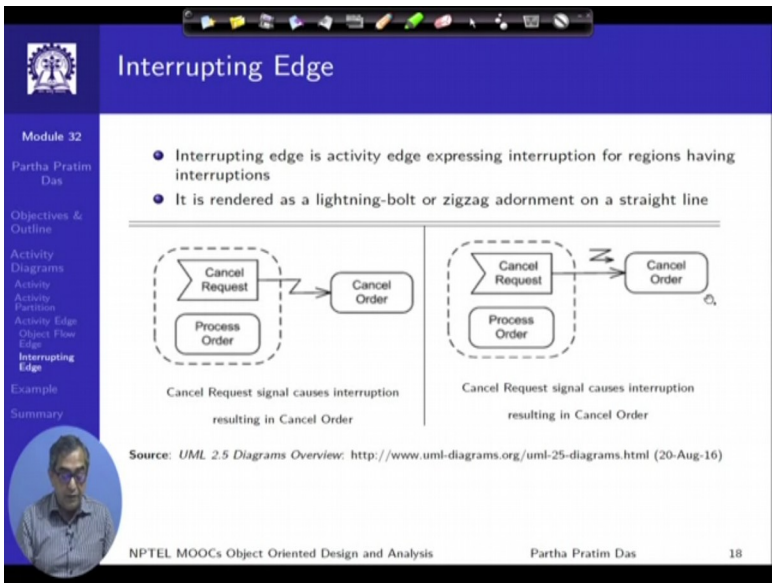


So, these are your activity edges, then there could be the activity edges could have objects that is this is how an activity, so there is an activity edge here as we saw and in that instead of directly showing the edge, we are showing an object. You understand that it is an object by the basic type of rectangle this is a regular cornered rectangle which means that actually as the fill order goes to the review order as the this edge is taken, this particular order will move from fill order to the review order.

So, whenever you have objects identified which move from one activity to the other along an edge, you will say that there is an object flow edge. It could be certain cases objects could be implicit also for example here, we say that the warning object will flow to the send notification provided you have this condition, weight is 6 which would mean that there are 6 instances of this object so that the send notification will happen.

So, the situation that it will describe is if warning is happening, then for every warning you don't send a notification to the user but if there are 6 warnings that have accumulated then you send a notification. So you can model such things here, you just note that this is actually an object and this is an activity.

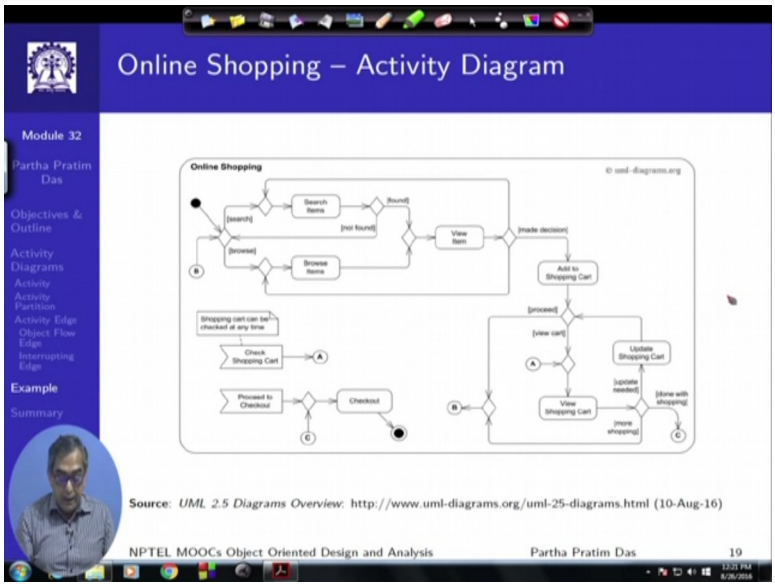
(Refer Slide Time: 16:26)



There could be other kinds of edges also. One important is the interrupting edge, interrupting edge is activity edge expressing interruption or regions having interactions. For some reason, the normal designed flow designed action cannot happen, so you have an interaction. So, for example you have a cancel request here. If the cancel request is done then naturally the order needs to be cancelled so the normal flow will get disrupted.

So, this is how this through this zigzag, you show that this is a interrupting edge, you could have the edge itself zigzag or you could have a straight edge like in here and have a zigzag adornment symbol port on top of this edge. So, this will represent the interruption taking place in the activity diagram.

(Refer Slide Time: 17:14)



So, with this let us try to understand there are other items here as well in this diagram but let us just try to follow a online shopping activity diagram. So, this is where we start here we are based on 2 guards, we are making a decision, we will talk about decision more. So, you could what is

happening if you want to do a online shopping, you can either search for an item or you can browse through the item through the list.

So, if you browse through then you have the browse item activity which is the process of browser browsing. If you choose to search then you have the search item activity which is a process of search and if you if you try to search then it is possible that you have found the item or you may not have found the item. So again, we have a decision where if you found then you proceed, if you are not if you have not found then you come back and possibly try to do the search again,

We will see what these nodes would mean otherwise if we have you could have either searched and found an item or you have browsed and come to an item so then you go into the details of looking at the item and in that process, you you might decide that you would like to buy some item where you have made the decision you proceed further and if you not then you would basically go back in terms of either continuing on the search or continuing on the browse.

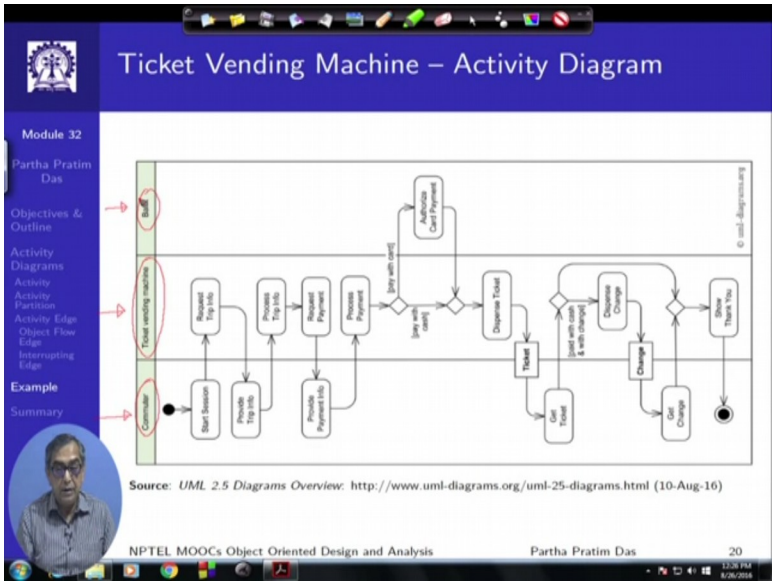
If you have made the decision then you do the next activity which is adding to the cart where as you add to the cart, you could be then you could view the cart and view this and then go forward or it could come to this whole process of viewing the cart from a completely different activity which is shopping cart can be checked at any time. So here as you are looking for searching or browsing for an item similarly you could just go an look at the status of the shopping cart seeing what are the items that you had selected before.

So, the check shopping cart item takes you to the next activity which is labeled at A which means that it actually comes here so you could either come through this process of search or browse, making a decision, adding to the cart and then viewing the cart or you could directly come from here viewing what you had seen earlier in terms of the cart and then either of the cases you actually move to viewing the shopping cart and after you have viewed you could select to add

And update the shopping cart or you could just decide to continue, do more shopping so you go back to B, just find out where is B, the B is here. So, you have added that item and now you come back and you just do not want to immediately go for checkout so you want to come back and start searching and browsing again. Otherwise it is possible that you said okay I am done with shopping I have no more items to choose and I put to the cart but I would like to check out pay for and actually buy.

So, you go to this label C which is what you find here, so which is the check-out process, you could come to checkout directly also. So, this is the kind of flow so here I have taken an example where you primarily see the flowchart kind of activity is only we will soon see the other kinds and you can find out how in a very compact, nice as swell as expressive form, you can represent the whole set of actions that you would like to do in terms of online shopping activity.

(Refer Slide Time: 21:07)



This is another example I am sorry about the orientation of this diagram because it is a in putting vertically it becomes too small and what am trying to show basically the activity partitions so you can see there are 3 swim lanes here. The commuter, the ticket vending machine and the bank. So, it basically it's about as if you have gone to a rail station or a bus station and there is a vending machine into which you can say where you want to go

And then put appropriate money through card or cash and then the ticket is issued so it starts this is your certainly the start node, you start the session naturally that is, so if you start here, it is done by the commuter. So, you have the start session activity and that certainly will request for a trip information as to I want to go from plus x to plus y, I want to go from Hyderabad to say Mumbai and so on. Then so that goes to a different swim lane of ticket vending machine.

Now this activity will result in the providing the trip information which is again for the commuter to look at. So, you can you can see in this bunch of activities here, the edges keep on crossing back and forth between the 2 swim lanes because it shows an interaction between the commuter and the ticket vending machine happening one after the other. So, after the trip info has been provided, you can confirm that I want to do this trip so you send it back and send a go into the next activity which is processing trip info.

Then the ticket vending machine will request for payment which again comes back to you for providing payment information goes back to ticket vending machine for processing payment and you have a choice to pay with card or with cash. If it is with card then it goes to a further activity which is now in the third swim lane of bank which has to decide the validity and all those. So, I am not showing all those details and if it is with cash then you directly go in.

Then the once the payment has been met then the ticket vending machine will dispense the ticket that means it will issue that ticket, then this is this you can see is object flow edge. So now so fat it was just a information now an actual object has to come out which is the ticket. So, the dispense ticket activity goes to the get ticket activity and that brings in an object here. So, dispense tickets as generated an object which flows in to the gate ticket activity.

And so, you have paid with then there are subsequent actions like if you have paid with cash then there may be some change that you are supposed to get. So, if you are supposed to get changed then that activity of dispensing change will happen and the change as an object again will come back to the gate change activity but if you have paid exact then this is not required. So, you bypass and come directly here which is the next activity which is basically termination ba the machine says that thank you for buying the ticket and so on, blah blah blah.

And you end up with the final node which is you have ended this ticket vending process. So, this kind of a process like we clearly show here that it is it's still a flowchart in the sense that it is giving us a one single sequential algorithm for doing the whole thing but representing in this kind of swim lanes, this will clearly tell us that who are the responsible persons the actors that are involved in different activities and it will also tell us.

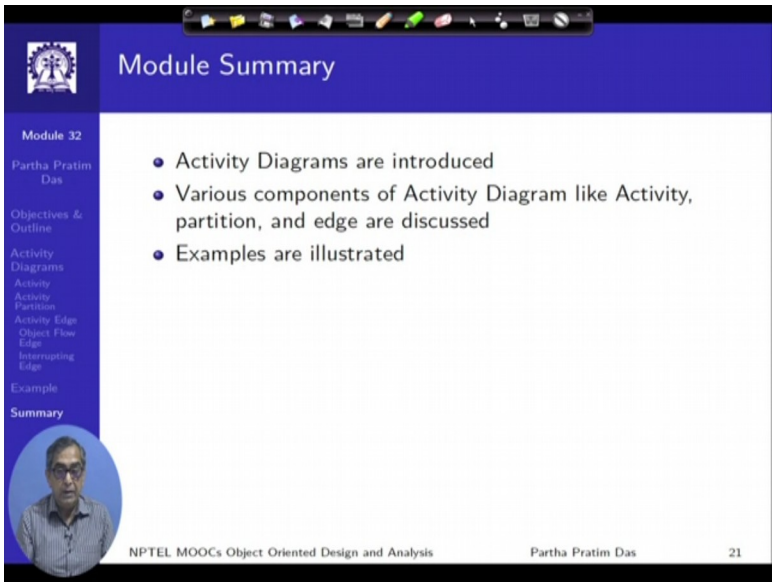
So, you will find that you will be able to get to this swim lanes very easily if you look into how the use cases were looking like. So, these are different parts of the use cases that you had in this corresponding ticket vending machine if you build that use case diagram and these are the different actors and you accumulate the activities under the respective swim lanes so you know that in terms of the commuter, what are the things that need to be done.

Commuter possibly is more more a manual person, more a person whereas in terms of the ticket vending machine, all the activities that you have here are parts of the core system that you are trying to build in. so you will have to really design deeply through that and in terms of the bank swim lane,

possibly bank will have their own authorization processes. So, it will already be existing so all that you need to really think about are the edges that happen with them that is how you send the information to the bank for authentication, what the bank sends out in return and so on.

Of courses, there could be more detailing of this activity further down if you look into from the bank's point of view. So, the swim lanes really help you isolate as to what activity should be where and what you should primarily be designing for and what are the activities which will have will need interfaces with the external world, so that you just primarily you need to work on the protocols and make assumptions in terms of how those activity edges work with the system.

(Refer Slide Time: 26:37)



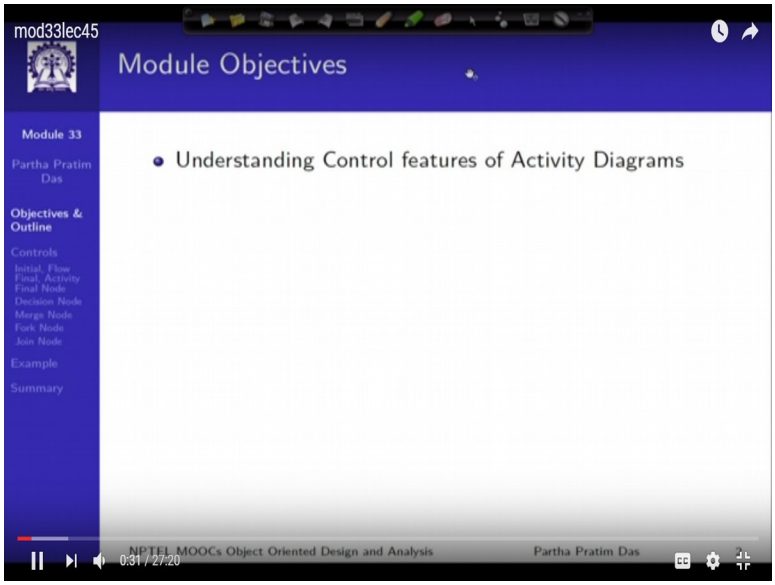
So, to summarize in this we have introduced in this module, we have introduced the activity diagram and as I said this is first of the 3-part discussion on activity diagram. We have talked about the basic components of activity diagram which is the activity partitions and the edges particularly including the object flow and interruption edges and we have explained through some illustrative examples which will continue in the next modules as well.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 45
Activity Diagrams: Part II

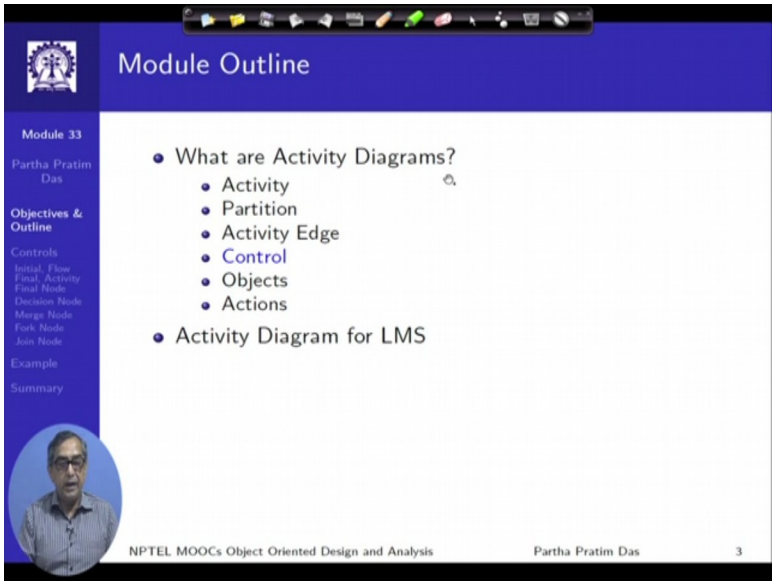
Welcome to module 33 of object-oriented analysis and design.

(Refer Slide Time: 00:30)



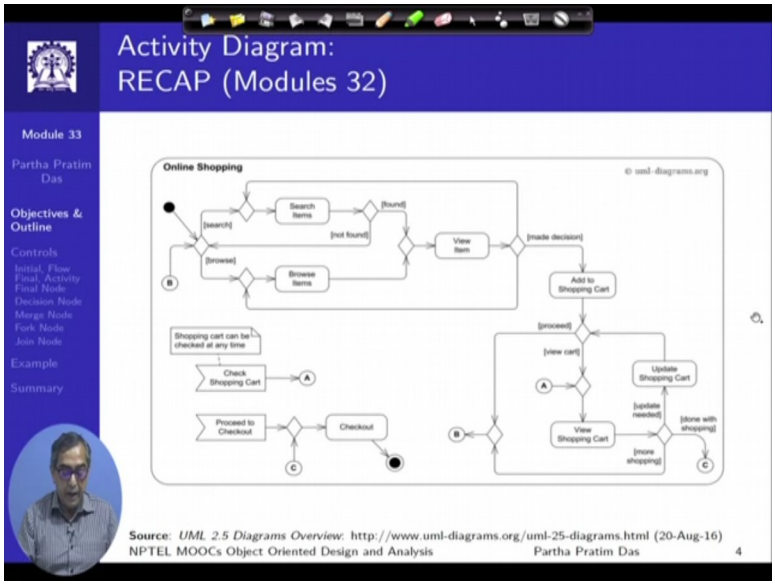
We have been discussing about activity diagrams and we have introduced the basic notions for it. So, in this module our objective would be to specifically look at the control features of activity diagrams.

(Refer Slide Time: 00:42)



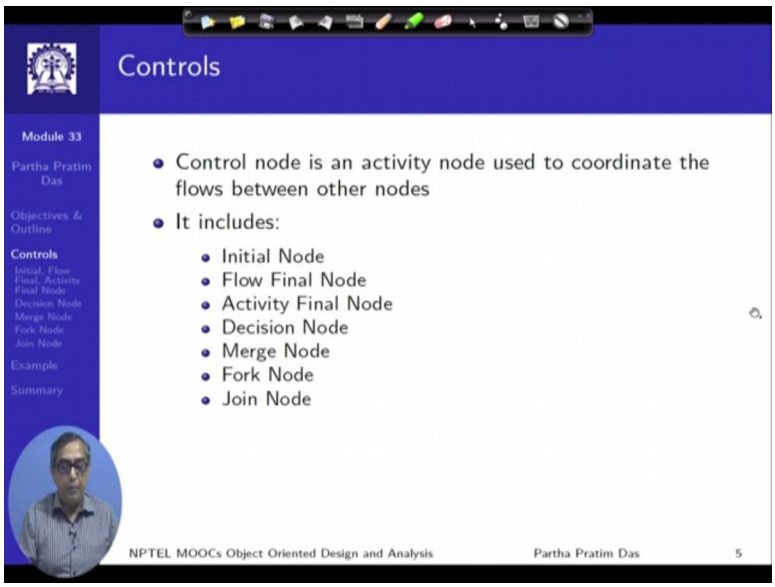
The outline as I had mentioned in the last module comprises this whole set that define the activity diagram and control which is shown in blue is specifically what we covered here.

(Refer Slide Time: 00:52)



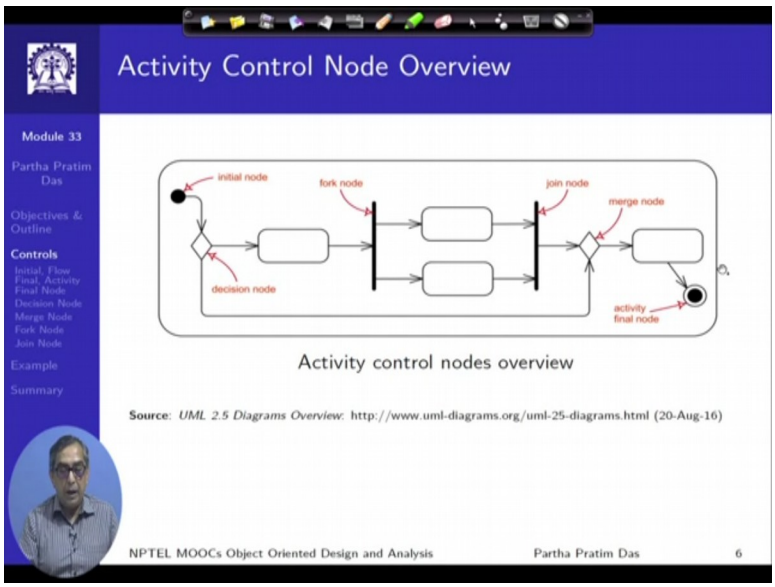
So, we would quickly recap, we saw the activity diagram defining activities, the partitions and ages including the object flow ages and we looked at guards and all those other related points and we specifically discussed about this particular example of online shopping.

(Refer Slide Time: 01:26)



So, before you go through this module I will request that you would revise on that online shopping exercise that we have discussed at length. Now in terms of control besides while certainly activity diagrams have activity nodes and have some other nodes but significantly they have a whole lot of control nodes. So, we have defined the edges between activities but we will see that actually there could be edges between variety of different nodes in this graph kind of structure.

(Refer Slide Time: 02:11)



So, these are the different kinds of nodes that are possible in terms of the control behavior, in terms of the control node. So, we will try to understand what they mean before I get into the exact definition of each and every node, let me just first give you an overview in terms of the symbols that are involved. So, we will explain each one with the behavior subsequently. So, in terms of the symbol let me use blue. So, this as we have seen a solid glob is an initial node, that's the beginning of an activity.

So, this is an initial node and if that is circled within another mt1 then it's an activity final node. So, the whole activity is a flow from the initial node to the final node. We have seen that these are are different activity nodes, these are the different activities. So, an activity will have sub activity, a sub activity may have sub sub activity and so on dependents on the granularity of how how much depth you want to go in. next we will see control nodes which are decision nodes.

A decision node basically we will take a decision and will have at least 2 outgoing edges based on the outcome of the decision. The decision could be binary, it could be so here it is a decision node which shows one possible outcome and another possible outcome. Then there are merge nodes which are shown by the same symbol of a blank diamond where more than one edges actually come and then one single edge go out which show that how the different flow that was coming from this side and that was coming from this side actually flowing to the next one.

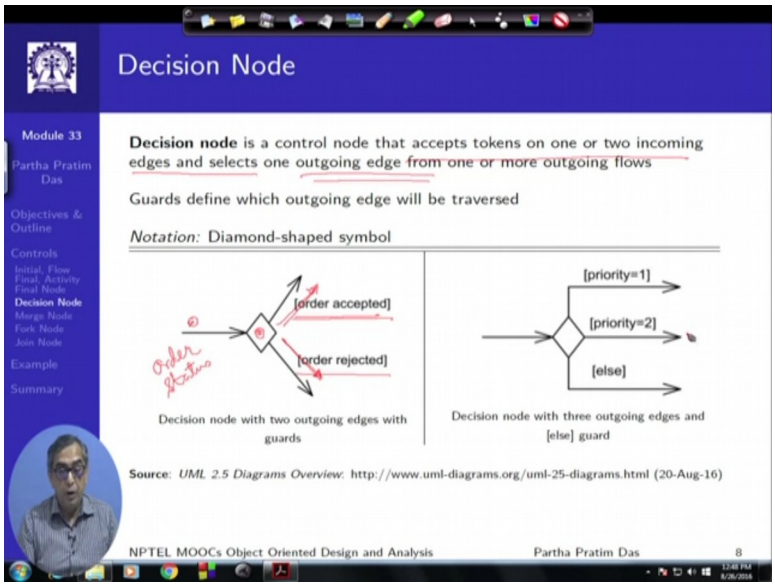
So, this is the merge node, so if you look into these components then this is pretty much like the flow chart notation that we are using. What is new are these 2, the fork node and the join node, the fork node basically is introduced for concurrency. So, the basic meaning of fork node would be that the flow is coming from this edge through this edge after this activity has been done and then

concurrently both these edges actually activate that is both these activities will start at the same time is a concurrent operation.

On the other hand, a join node waits for the different flows that had started concurrently to reach a common point of completion and then only this will proceed. So, this is known as a join node. So, these are very typical of activity diagrams which show you the concurrent part of the algorithmic behavior or the use case behavior where more than one activities are happening at the same time which is very very common in most situations.

Those of you who have done some course on operating system would be familiar with this notion of concurrent behavior, you would also be familiar with the fork and join operations that typically you see in Unix processes and so on. So, these representations basically model those kind of fork, join behavior where concurrency can be modeled. So, this is the basic overview of activity control nodes and how those controls will look like.

(Refer Slide Time: 05:40)



So, if we go into depth the initial node is a control node, let me revert back to red, initial node is a control node which where starts the flow starts at this point. It is sho shown with a small solid circle as we have seen. Final node is one with a small circle with a cross inside. So, this there is a difference between we have already seen this which is an activity final node, this is a control final node that stops all flows in an activity which we have already seen, this notation.

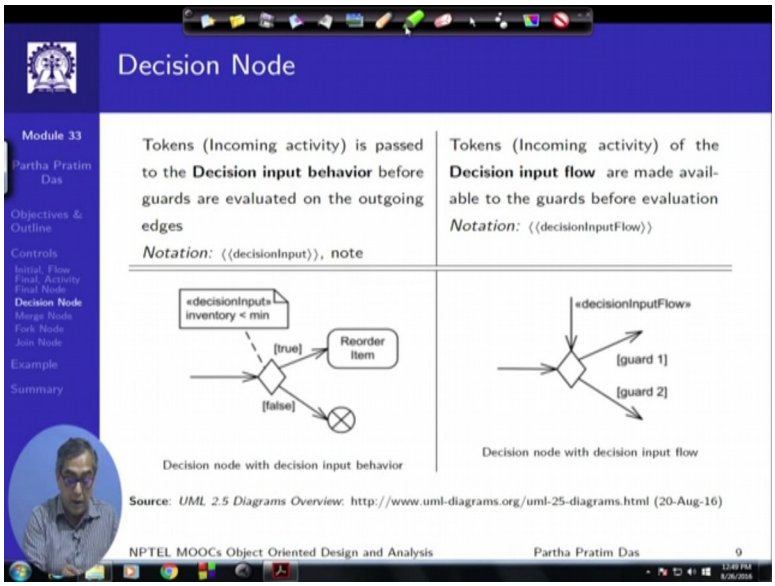
But there is a flow final node also which basically say that this not only this activity has completed but it is a total flow stops here. So, there is nothing further to happen on this. So, you put a flow final node for this, in some diagrams you will find this notation happening.

Moving on these are decision nodes as mentioned that accepts a token from one or two incoming edges, there could be more also and there and selects from the outgoing edge. So, there is a token coming from here, some token coming from here that is some information coming through this and this decision node based on that token decides either to take the control this way or to take the control this way and typically you show guard conditions here showing what is the value that the token has to take so that the decision is made.

So here there is some order status so the token coming in here is some kind of a order status and if that has a value, order accepted you take this flow, if this order rejected, then you will take this flow which is typical of in in flow chart we we typically just have if kind of binary decision, we just say true, false or yes, no but here it could be multi way. So, it is important that you specify the guard conditions there.

For example, we are showing it here, this is where you go if priority is 1, this is where you go if priority is 2, this is where you go if priority is something other than 1 and 2. So if the token coming in here is a priority value and based on that a decision, a multi-way branching is made.

(Refer Slide Time: 08:09)



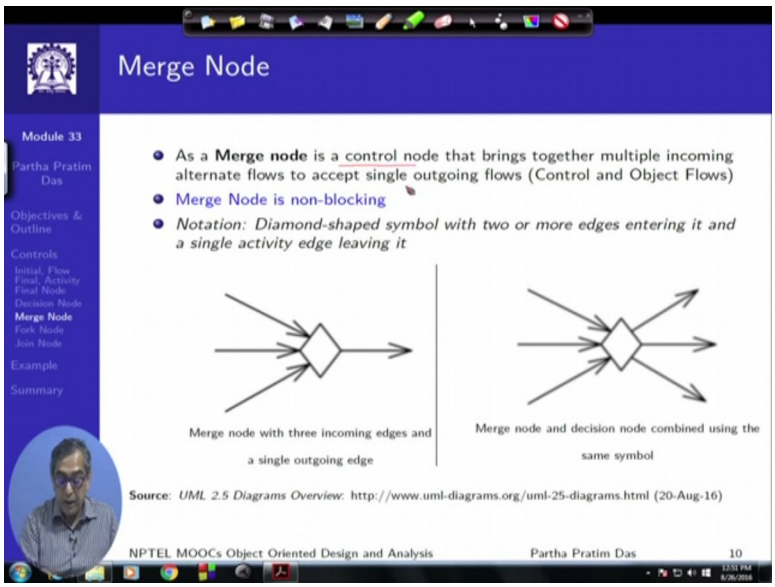
So, these are the decision nodes decision nodes could have decision input behavior or decision input flow. The decision input behavior says that you can this is marked in terms of a keyword like this, just note how the keyword is. It you you must have noticed that in uml very regularly we put things within this. So, whatever you put in within this corner brackets, double corner brackets are basically keywords of uml.

So, this is this says this is a decision input, so that decision input is nothing but the condition that that particular decision node has to satisfy. So, if you look at the decision input here is inventory is less than minimum that is here what comes in is the inventory. So, prior to this there must have been some activity which has changed the amount of inventory all as affected the amount of inventory and then in this is this decision node, your decision input is if inventory is less than certain minimum value and your guard conditions are true or false.

So, if you come here, then it is true which means that the item can be reordered was in this case say and if it is false then you get get a flow final termination that is in this flow, there is nothing else that need to be done, pursuit. In contrast, there could be input decision flow or incoming activity, so it is input decision flow.so where the ba condition basically comes from this decision flow, here the condition is fixed by the decision input, here the condition also comes from the decision input from this edge.

And based on that ehe condition the different values of those condition you make this branching. So, this is little bit more generic in nature but so if it not known as to what could be the different values or what could be the different conditions on which you branch, you may need to design based on a decision input flow, otherwise you design based on decision inputs.

(Refer Slide Time: 10:18)

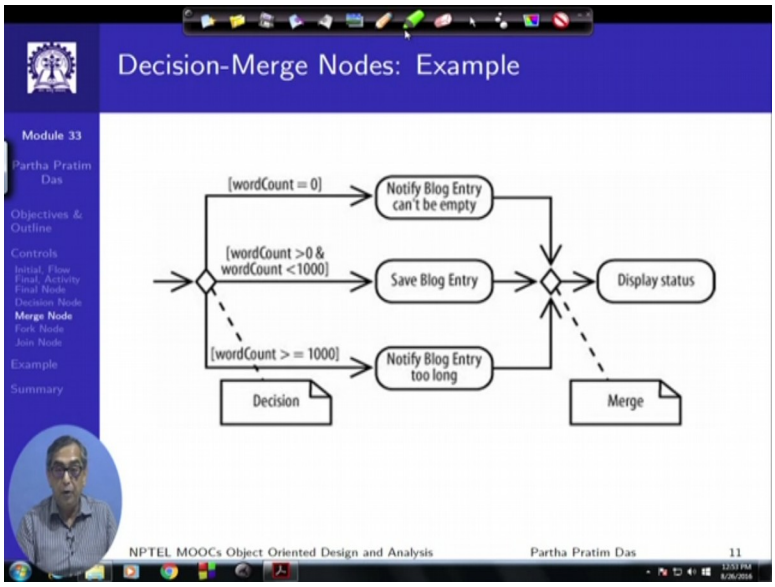


Similarly, you have merge nodes. The merge node is a control node again and brings together multiple incoming alternates flow into to accept a single outgoing flow. So, there are as you can see the symbol is same, so there are multiple ways through which the control can come here and whichever way it comes, it flows out from here. It is non-blocking, so even though there are multiple inputs it just the flow continues as soon as any one of them has arrived.

So, if if there is a token here that immediately gets transferred here that is the flow continues in this manner alternately if something first something comes here then the flow will continue like this. So, this is where and certainly it will have only one outgoing edge, so to make it a merge node. But you can combine the decision and merge node in terms of a structure like this were here 3 flows are merging in this node. At the same time, there is some decision input based on which the flow is branching out in 3 different ways.

So if if it is that you have a unique condition through which this branching can be decided and this is a point where so other way another way to show this could have been that you do this, you show this, this, this as a merge, then you have a outflow and then you have possibly a bank activity, this is nothing that needs to be done or or actually you could just let us let us do it differently, the flow is coming in here, the flow is coming in here, it goes out and you have another diamond,

(Refer Slide Time: 12:24)

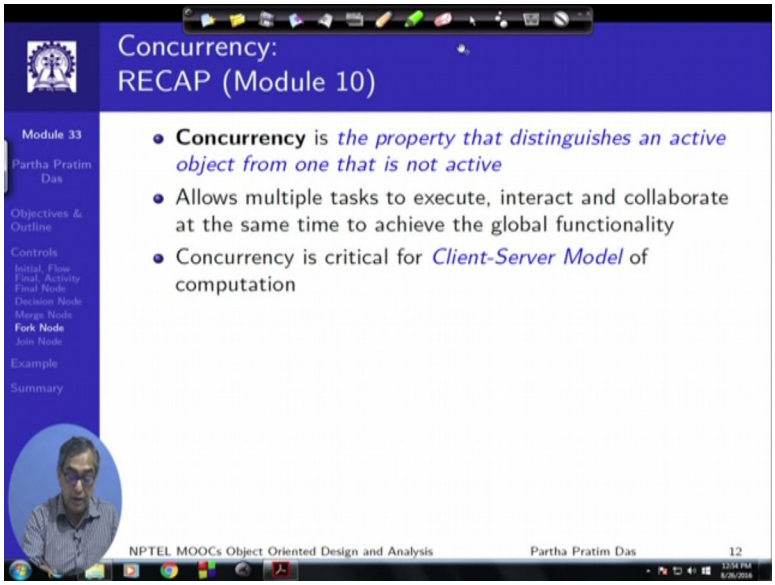


So, this is the merge node, this is the decision node and so do these 2 together could be combined and shown in a combined fashion. So, this is a merge decision node. So, these are some examples, for example here we are trying to just refer to the blogging example we had started off in the last module, so we are blogging, so if so some blog has been created. So, in if the word count in the blog is 0, that is a gourd condition that then the activity is notified blog entry can't be empty,

Certainly, you cannot have an empty blog. But if it is greater than 0 but less than 1000, this is a guard condition then you save the blog entry if it is more than 1000, the you notify again then the blog is too long and it cannot be put up. So, these activities are or different and based on this condition 3 conditions a decision is made at this point. But whatever that the decision is, you finally merge them all together, merge all the flows together and display the status.

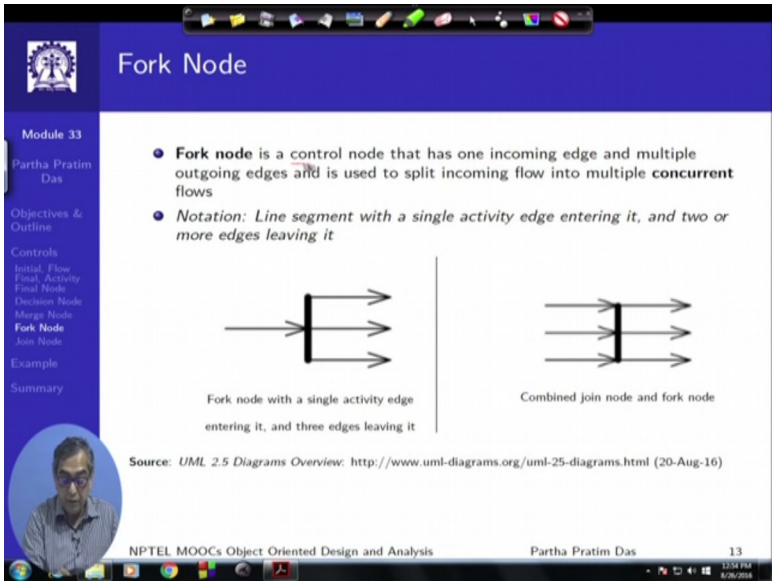
So, this activity is common irrespective of whether you have gone through this or you have gone through this or you have gone through this. The typical flow chart logic.

(Refer Slide Time: 13:24)



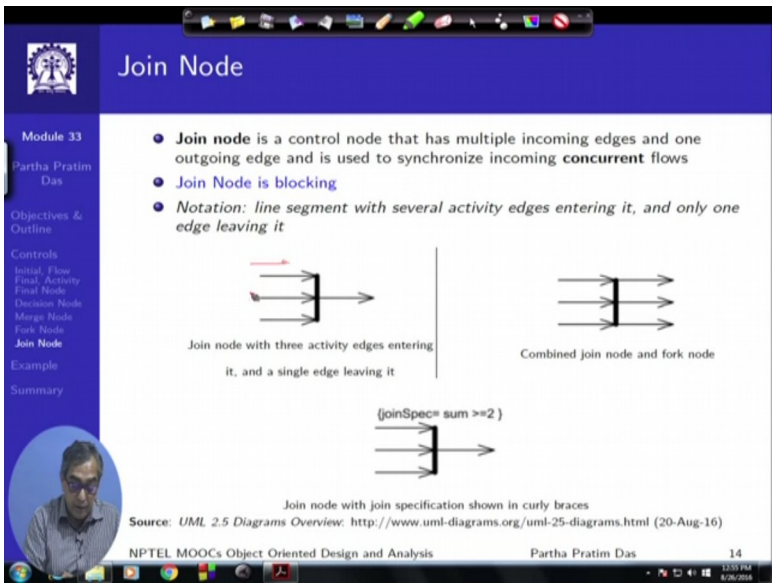
Concurrency we have discussed about that so now we will just again discuss in depth about the concurrency node. So, I have just explained it at the beginning.

(Refer Slide Time: 13:35)



So, fork is the first kind of control node for concurrency which has one incoming edge that is activity that was going on is flowing it through this edge and it has multiple outgoing edges. So, it has to have at least 2 to mean concurrency that is used to split incoming flow into multiple concurrent flows. So, this is a sequential flow and these are all concurrent flows. So that is what you have to keep in mind that once the control reaches here, this token reaches here actually as if 3 tokens, 3 separate control start happening at the same time.

(Refer Slide Time: 14:20)



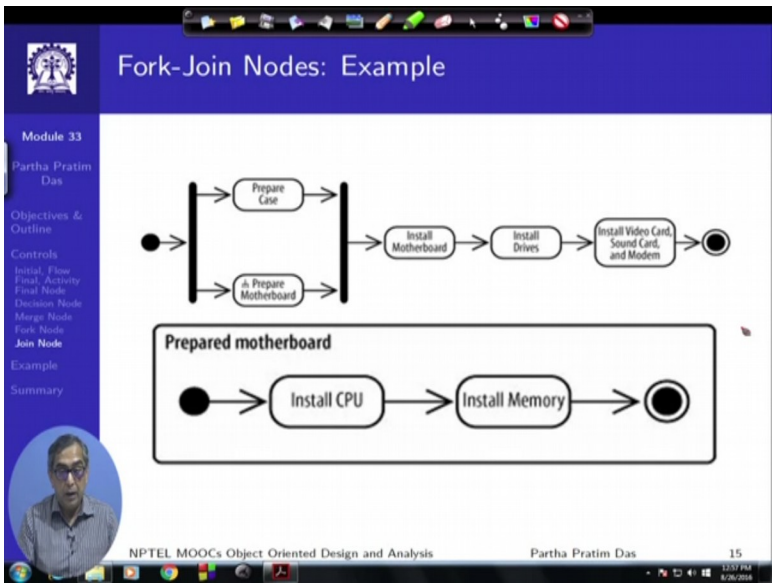
So that is a fork node which is clear and then you will have the other side which is joined. So here you have 3 controls coming in and these 3 controls when you put them in the join node, then the basic difference is this is combining through blocking which means that if the a token comes through this that is if the control the prior activity to this is over, completed so the control comes here, the control will not start from here.

It will not be able to start it it will have to wait till the token comes here and the token comes here all 3 tokens arrive, that is the blocking part of it, that is a critical. So, you will have to understand the crucial difference between a merge node which is this and the join node. In the merge node, this is the merge node if the token comes from any one of this, that is if any one of these flow has come through any one of these edges, edge 1, edge 2 or edge 3, the flow will immediately start on the outbound edge.

But here if the flow comes from one it will have to check whether the flow has completed on 2 or 3 only when all these 3 are completed then the token will go there then that will. So, if we say this is kind of a or condition on control flow, this is kind of an and condition and control flow certainly this is related to concurrency because by fork we have made that from one single flow we have created 2 or more flows happening concurrently. So, we we have to recombine them back into a single flow through the join node.

So that is the important part and as you can merge decision and merge node you can also merge the combine the join and fork node together. So, it is you you fork here there are some activities which must be obvious from the context and then then you wait for all of them to complete together.

(Refer Slide Time: 16:34)



So, we are just am just showing an example here this is a as if you have to define the activity, define the process for preparing a say a desktop computer to be dispatched. So, you start here and you fork because preparing the computer means preparing the case within which it is housed and preparing the motherboard and possibly 2 different teams work on this. So, they need not happen sequentially one after the other but they happen concurrently.

So as soon as the indication come that the computer has to be prepared, the prepare case as well as the prepare motherboard activity start and only when both of them are over, certainly you cannot install the mother board you you see what is the next activity, next activity is installing the motherboard. So, you cannot install the motherboard unless the motherboard is ready and also the case is ready. So, both will have to be ready.

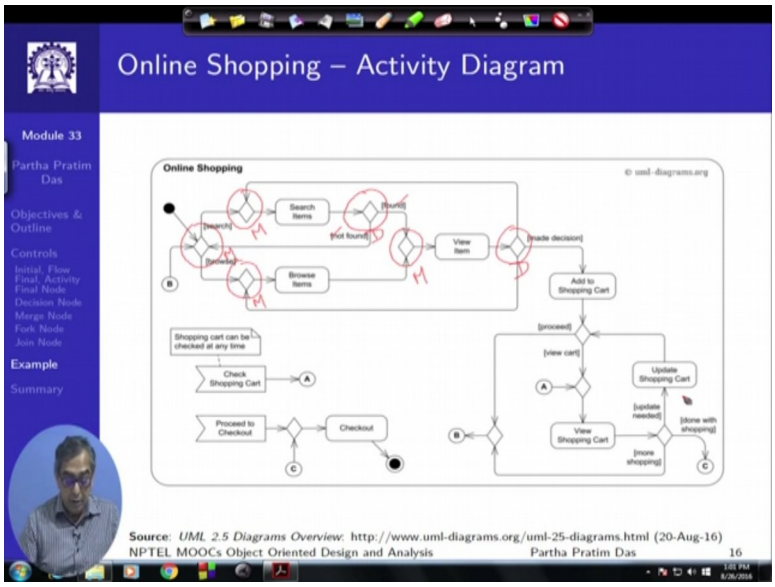
So, this is a that is why here you had to fork because independent activities have to happen at the same time and here you have to join because the 2 activities that were happening independently both have to complete and only then the following activity can happen and rest of the activities follow sequentially, install the motherboard you install the drive, installed video cards exetra and you are done with the final of the activity so that is the basic purpose of doing fork and join

And you can also see here we are I am using something which is a very special symbol, a three-way this inverted three-way symbol here, just note that this one, this means that the prepared motherboard activity is actually a another more complex activity and this have been defined somewhere else were more detailed activity diagram. So, it is like I mean conceptually if we think in terms of programming languages this is like a function call kind of your making.

So, you are saying that this is detailed further so we will look for where is the prepare for motherboard activity and there is a prepare motherboard activity. So, this actually means this whole activity so in place of having just a single activity you have this whole activity of again you have a start for this activity of an activity final node, so you install CPU, install memory and complete the prepare mother board activity.

So, this is often how you show a more complex activity as a part of some other activity and certainly this enhances the reuse in terms of the activity diagram because this may not be the only place where you need to prepare motherboard, you may need to prepare motherboard in terms of a repair operation also, repair process also and you will be able to simply reuse this activity to do that. So, this is how you you make use of the fork and join to decide on.

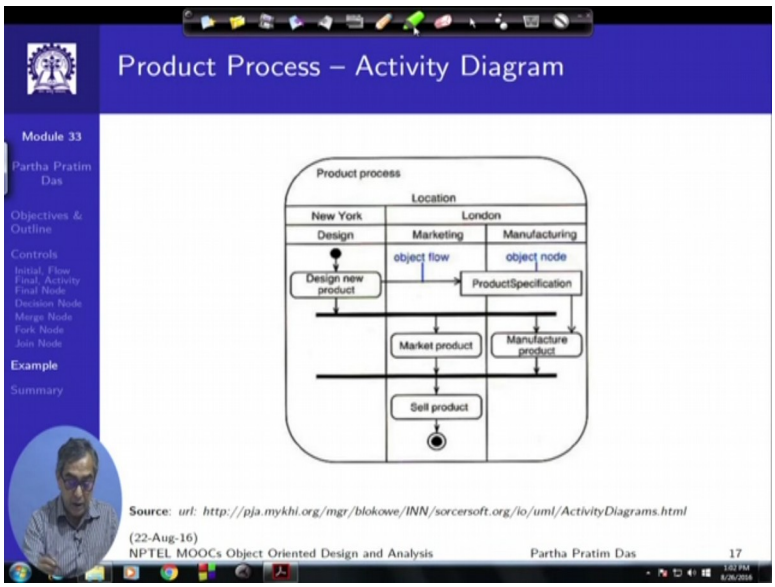
(Refer Slide Time: 19:41)



So, in terms of the online shopping example, we we have already discussed the example so I will not spend more time on this. Just we will identify that here this is this certainly is a decision node because 3 different inputs coming and there is only one output. So, this again is a decision node. this is a am sorry am sorry there are 3 come in and one go out. So actually, this has to be a merge node, this also has to be a merge node because multiple flow come in and one flow is going out.

So multiple flow are getting merged whereas this is a decision node where one flow come in and 2 the waist and found and a non-found this again is a merge node this is a you can see is a decision node and so on. So here we are doing again a merged. So, the through this, these are the only special control nodes that you have in this diagram, the previous one had that fork and join. So, this is how you can understand the control flow in terms of a activity diagram.

(Refer Slide Time: 20:52)



This is a yet another example of a product processing so here we are trying to am trying to show the fork and join kind of control along with the swim lines. So here you can see that there are 3 different swim lines, here and here in terms of design, marketing and manufacturing, the 3 processes of a product and then we have some combining information that both of these happen in London that is a locational information

And this happened in New York though this may not actually be a something that will seriously go into the software system but it is very important to capture this because you know that if if this is in London and this is New York any activity that happens across them, any edge that go between them would need to have a kind of external internet based kind of protocol to be to be followed. So, if we look into the steps of this then we start here first is always to look at the start and the final .

And this is designing a new product where it starts and that on this gives a object flow to the product specification. Why it is it an object flow? Because when the design is done you cannot just go to the next activity of manufacturing the product. You have to produce the output of that, so this is an object flow which gives you this object which is the object node. We will see what object nodes are in the next module further but am just showing you the flow.

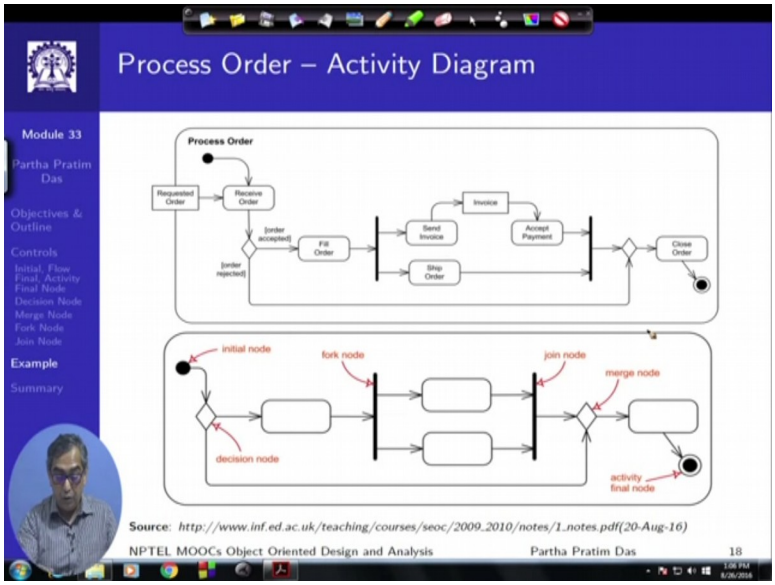
So, after the design I can come to after the design I can come to the manufacture with this object flow edge. Then it goes into a after the design, we go into a fork, this is a forking what does it fork ? It forks in two things that is the manufacturing of the product and the marketing of the product certainly are done by 2 different teams and must happen together. You cannot market the product say that we have this and manufacture it later you cannot just manufacture the product and keep it and then later on expect that marketing will find out customer.

So, both of these need to happen concurrently, so we have a forking here, so you can see a very interesting structure where the this activity will get will need to get initiated from this product specification reaching here and this happening together this control reaching here. So, this designates that the manufacture product activity will need this product specification object when the control comes here and when it initiates in concurrently a market product activity also starts.

Look at so this was the forking and you can we can again look at the merge part of it, certainly when can you go and do the next activity of selling the product. Only when it has been marketing, marketed that the customers know this and only when it has been manufactured. So, both of these edges have to reach this merger, I mean this this join point I should write it as a join, it is not merge. So, reach this join point and this will be blocking so if market product is complete this activity is complete.

And product has not been manufactured I cannot go and sell it. In the other way if the the the manufacturing is complete but it has not completed marketing going to the customer then also I cannot sell it. So, this will be a blocking join which will wait on both these activities to be over and when the token arrives on this as well as this, the token will go to the next cell product and the product will start skating sold. So, this is another kind of activity diagram that we have seen.

(Refer Slide Time: 24:56)



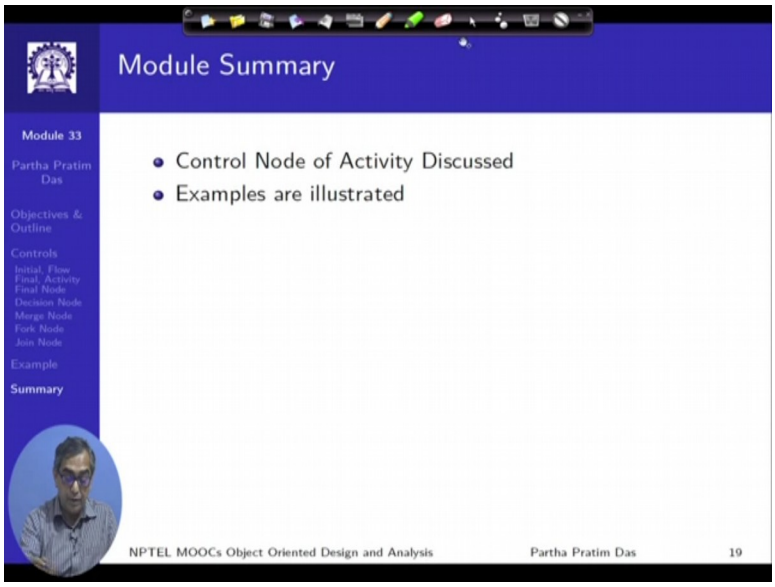
So, finally we are back to the overview diagram that we started with and if we just fill-up fill up some of the details of where it is taken from, this is a processing order. So, you receive the order and this is a decision that is made as to whether the order is accepted or the order is rejected, if the order is rejected you go straight and close the order, that is the end of this activity or if the order is

accepted you fill out the order and then you start 2 different activities in concurrent, sending the invoice and shipping the order.

Sending the invoice is required for getting the payment from the customer, shipping the order process is required to getting the inventory ready. So, sending the invoice will have the next activity as accept payment naturally it involves an object edge flow because the invoice object has to go to the accept payment and only when the payment has been accepted and the order has been shipped, so this is where you again see join is happening only when both of these has happened then the order gets can closed.

So that flows into a merge node where you can so this is a successful order flow and this is a rejected order flow, whenever they merge and then you go to the so you can see very closely as to here by join here anyway merge here and the join needs both of these controls to come in and the merge its either of this control to come in to move to the next activity.

(Refer Slide Time: 26:32)



So, this is these are the different examples, so I would summarize to say that we have discussed particularly different kinds of control nodes in an activity diagram and specifically you should make good understanding of the fork and join the decision fork and join control nodes, because I am sure decision and merge nodes you are more comfortable with because they are the typical flow chart nodes to understand in terms of fork

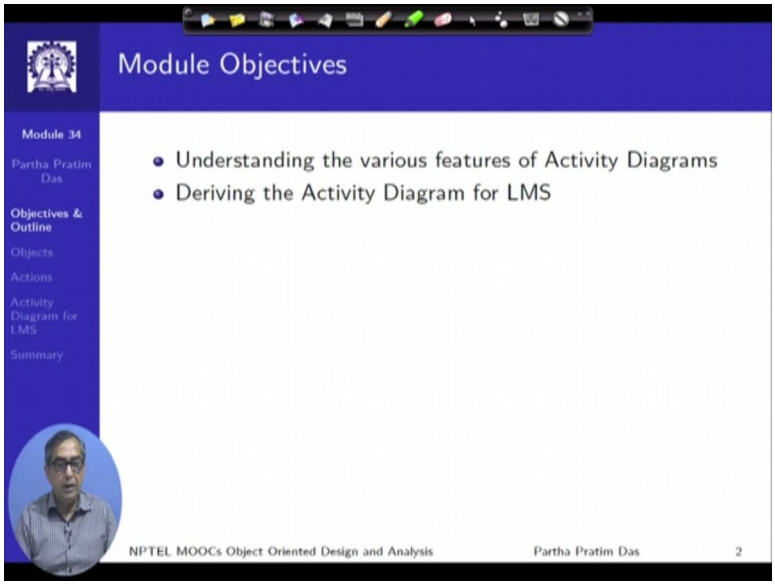
And join that how activity diagrams can be used not only to represent sequential processes but also to represent concurrent and parallel processes and we have discussed some examples which you can review again and try to build you own solutions.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 46
Activity Diagrams: Part III

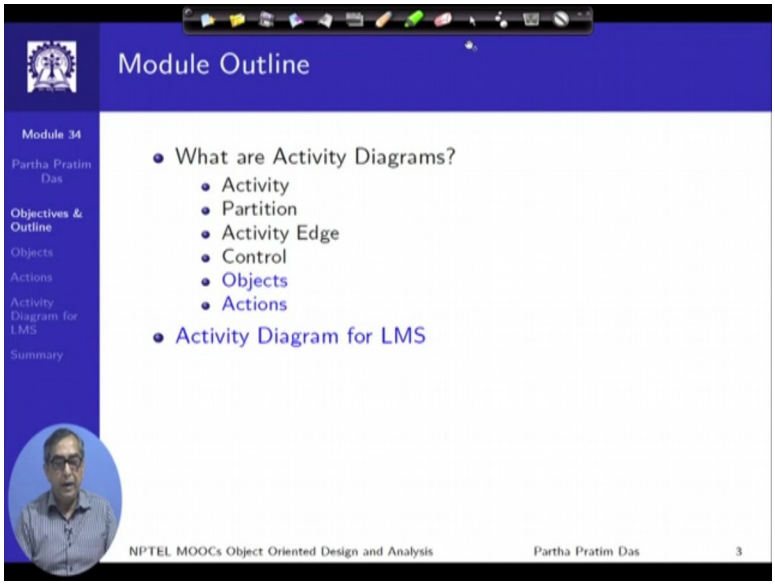
Welcome to module 34 of object-oriented analysis and design. We have been discussing about activity diagrams in uml in the last 2 modules.

(Refer Slide Time: 00:34)



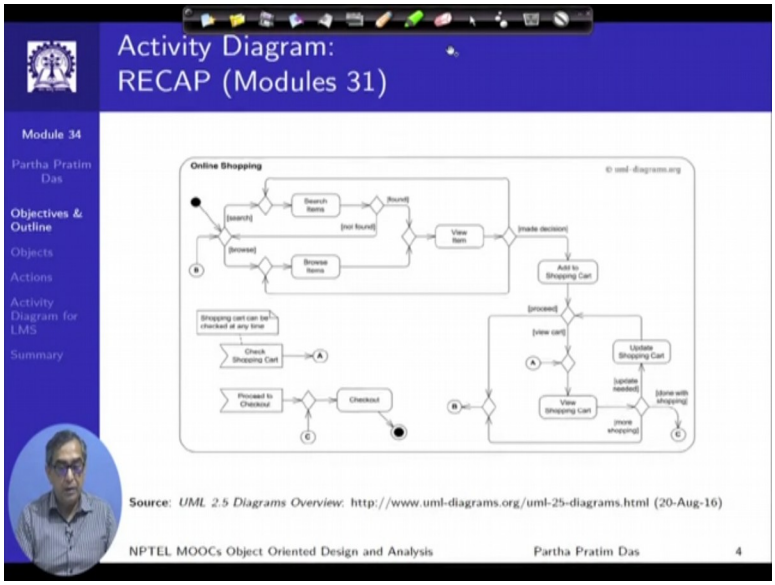
We have talked about several of the features of activity diagrams including the activity themselves the partitions, wedges particularly object wedges and then in depth about control nodes and how what role the control nodes play and we have seen some examples.

(Refer Slide Time: 00:55)



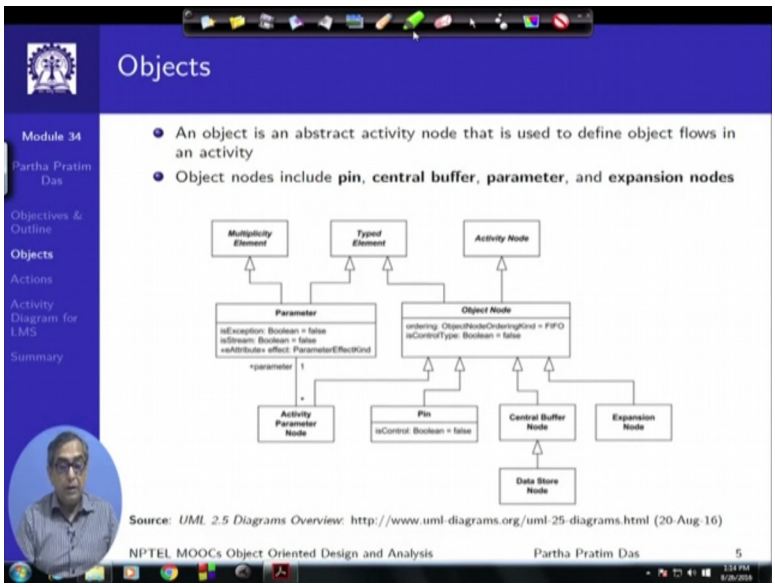
In this module will continue further and we will specifically take look into objects and actions and then we will take a brief look into the leave management system in terms of the activity diagram.

(Refer Slide Time: 01:11)



So, this is one diagram you can you can recapitulate on in terms of what we have studied earlier.

(Refer Slide Time: 01:21)

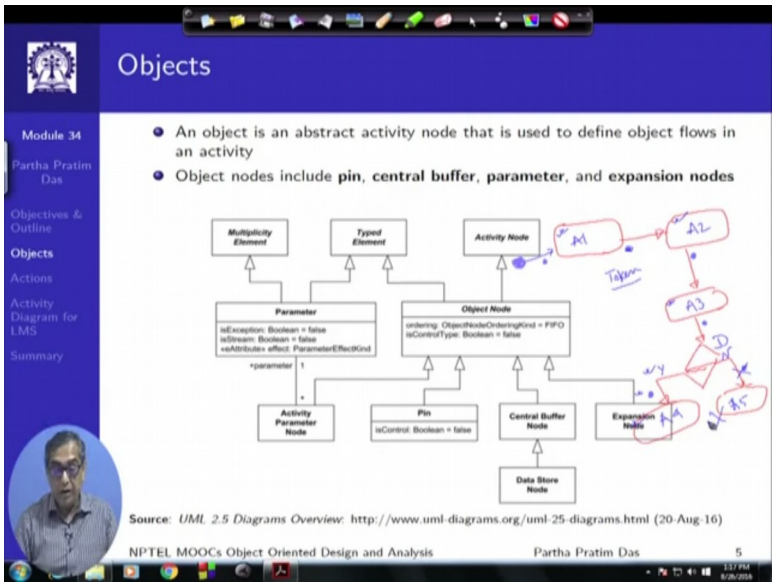


Now the next concept that we have in terms of activity diagrams here are objects. The object is an abstract activity node that is used to define object flow in an activity. See the way you can you can conceptualize if we let me just go one step back in terms of the conceptual model of activities and their diagrams, say suppose there are 2 activities here and there is an edge here, now there could be that's a there is a there is another activity here having edge here,

There is a decision here, there is a decision here, giving you different activities on this. Now as an active this is a representation of the flow now how do you really understand the dynamics of the flow; how do you really understand that how the flow is going. So, the way you conceptualize is you think about certain abstract objects, these are typically called tokens, that move from one activity to the other. So initially when when if we represent we have the initial starting mode, the

token is here.

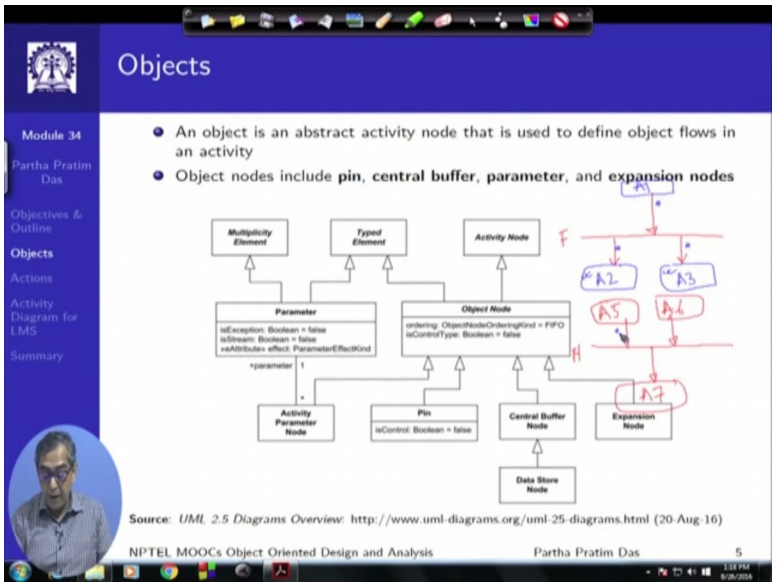
(Refer Slide Time: 03:05)



Then this activity happens say activity a1 then the token has reached at this point. When it reaches this point the token gets transferred to the edge and activity a2 starts. Once this is done, the token comes here, so it is as if the actual dynamics is as if we are tracing here tokens then you have a3, so when does an activity start? When all its inputs token have been received. This is these are simple flow so they have only one token so as soon as that token comes, this activity happens and once it completes, the token is goes to the out.

So, when you have a decision node naturally after this there is one token.

(Refer Slide Time: 03:47)



So after this decision node either the token goes here or the token can go here, it cannot go at both. So, if this is yes so this is no and if yes is taken, then this token comes, this token is not there. So, this token has come, the activity a4 will get fired but activity a5 cannot get fired. That's the basic

concept of activity flows in in general. So, you can you can understand that very well in terms of a fork situation that we have one flow coming and let it let me write clearly.

So, one flow coming here as 4 and you have 2 flows going out. So, what happens is is in terms of this you had some activity here, you have some activity here as well as some activity here. Now when this activity a1 has got completed naturally a token is placed on the output edge that goes to the phone. What the phone does? Phone basically duplicates this token on both the output nodes so that both these activities have a token in the input and they will fire, they will start fire.

Similarly, when we have a merge the reversing happens. Have a merge so I have 1 control coming in here, 1 control coming in here, 1 control going out, there is an activity here, there is an activity say say A5, A6 just arbitrary names and another activity A7 here. So, in merge what happens is whenever this activity A5 say will get over, it will place a token on this. But merge will not transfer that token here, it will wait for the other token also to come.

So, the merge node basically waits for all tokens to come, merges them together, places the token here and only then this will go on. So, this is the same logic that I have said but the only difference am am asking you to conceptualize if you think of a token traveling along the edges of the activity diagram with the further assumption that if I have an activity then naturally it may have multiple edges coming in and the activity will fire.

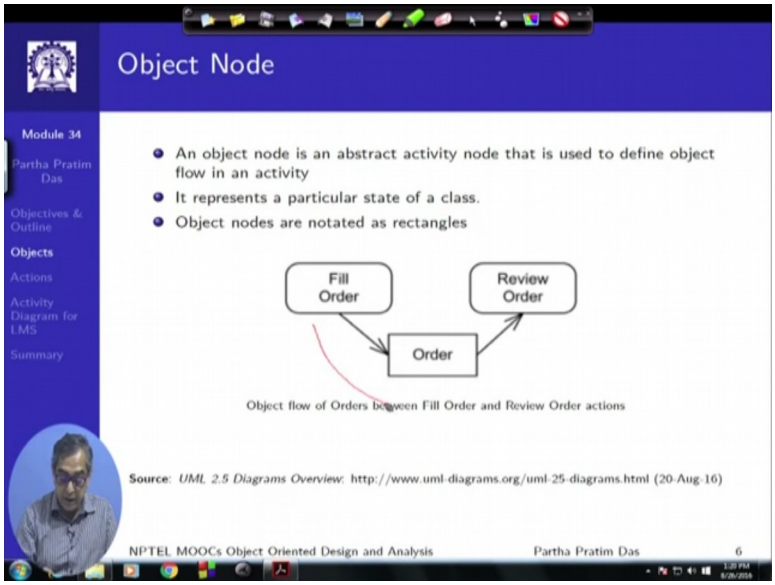
So, if multiple edges have to come in you either have a merge node or a emm or or kind of join node to represent that but only when the token is available on all of them, will this activity fire and once it fires it will place the token on the output. If you think about this then that becomes much easier to understand. So, whenever we are working with the activity diagrams and 2 activities A1, A2 we put an edge between them, we do not merge this token, this is implicit.

This is the basic behavior so that A1 when it will complete it will generate a token that goes to the edge and A2 has a token and starts working and when A2 gets over, the token comes to this. Now specifically what objects the role that objects play is if at places you want that beyond this generic behavior of tokens to manage the flow, if you want some more packaged information to go along with that, you put it in terms of an object.

So, it’s a same token but its this here the token is not an anonymous one without which is just to control the flow but it will also carry some information. And it could put the object in variety of

different shapes and sizes and different purposes to satisfy different requirements. So, it's best to just to have a quick look into those.

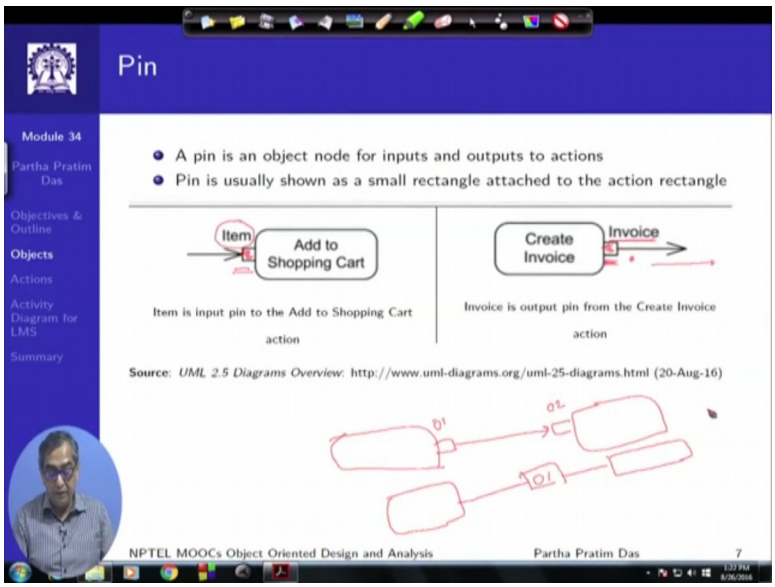
(Refer Slide Time: 07:43)



For example, we have already seen this. So here if you just think in terms of the simple model it is an edge of from fill order to review order which will carry a token. Now we have refined it saying that I have order object here. So, fill order which when it generates the token it actually generates a qualified order object and that order object goes in here. So, on the edge you see the order happening and this.

So basically, these objects will always happen on the edges because they are basically refinements of the token that has to go through that edges.

(Refer Slide Time: 08:18)



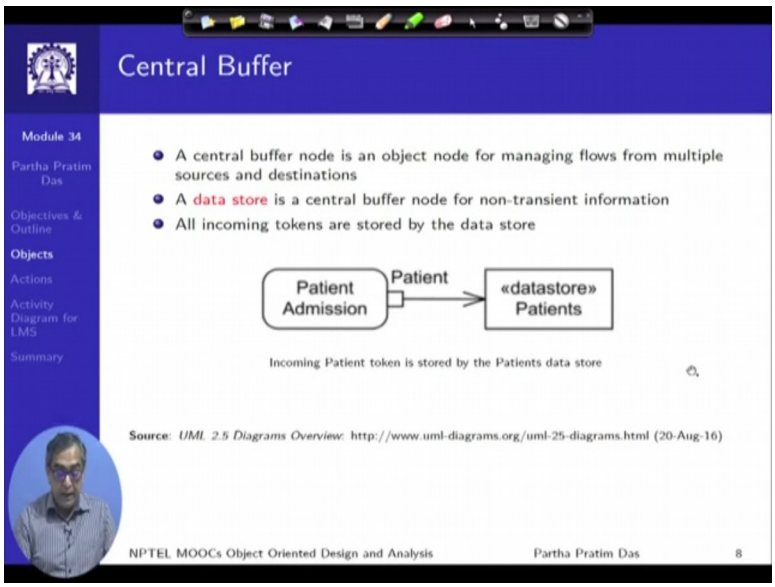
So, another concept is a pin object where where you show that well this is an input pin, this is an output pin. So, this shows that when the token arrives to add to shopping cart, you are showing that

it's just not an anonymous token you expect and the token in terms of an object. So, it should have other properties also and you are specifying what object you expect here. That's the item object. Here it's an output pin, it's an out-pin token, so when that create invoice completes the token is to be placed here.

You say the token is not a simple one, that is an object and that that object is identified as an invoice. So, wherever this goes wherever this goes it will get that object along with that token. So that's a you you can touch that the objects to edges or you can attach them to the pins so if if I if I connect them end to end so then this will look something like this. .so, so there is an object here, the object here and so on. So, this is how the pin objects will work, the they look like very similar semantics.

Because I could instead of going this I could have just done an O1 object and could have sent that to the next activity, the reason you you bind it to the pin because it you may have different kinds of flows. For example, if if you have the same 1 object generated which goes to the multiple of the different activities then it is easier to show them in terms of pin or if if you have that different activities can generate an object of the same type which we expect at the input of an activity, then it's easier to show it as a pin.

Also pin provides for certain transformational objects in the process. Will show an example.
(Refer Slide Time: 10:11)

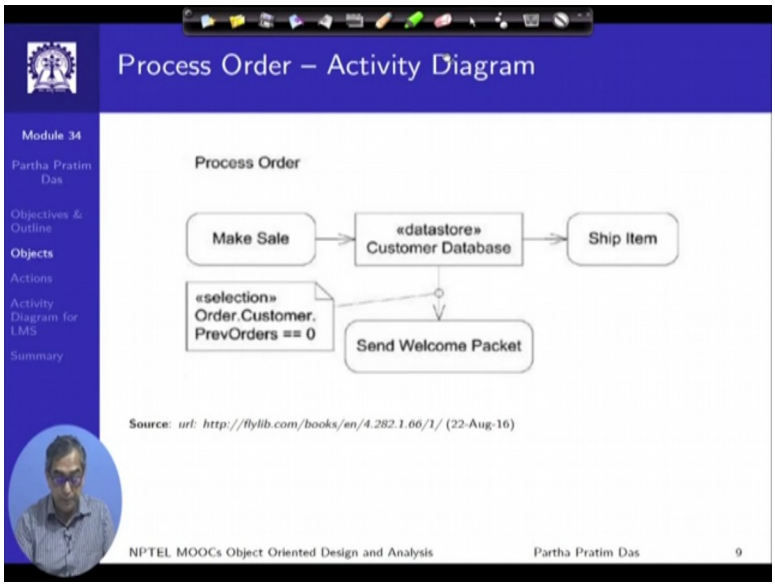


There are cases of what is known as a data store, often you will need a data store to maintain the collection of objects and so when you talk about the data store. So, data store is again shown in terms of a keyword. So, there could be several data stores which you can relate would later on

become possibly tables or in some cases databases in your central database system. So, this data store we are saying that the patient admission this activity ends, the token is generated, it is a outbound object patient and that's delivered to this object datastore.

So here kind of you do not have another activity following it, so this is kind of an object which just works as a repository. So, objects could be used for such purposes as well.

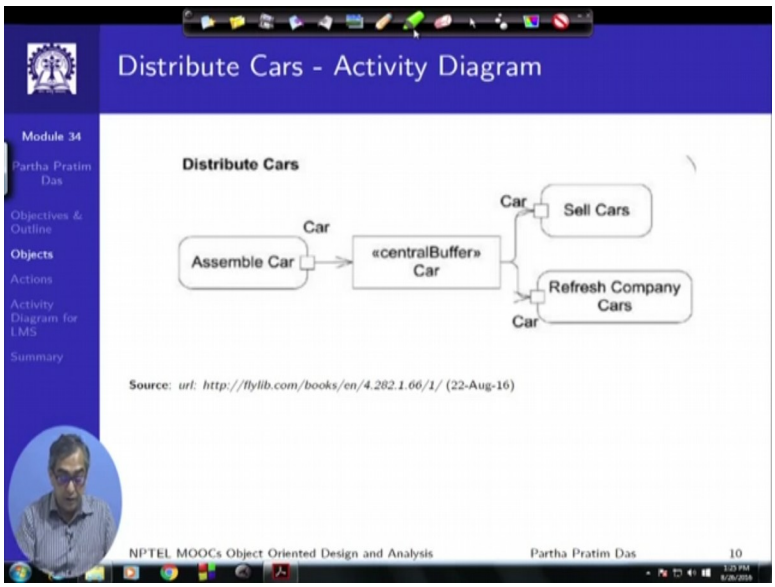
(Refer Slide Time: 10:58)



So, with that you can have variety of these there is an activity here which sends an object which on completion sends a token to the data store which is a customer database. So, making a cell that basically says that you have updated the customer database and then from this the activity, next activity ship item which means that based on this update shipping of the item will happen and the the outcome the transformational outcome in this is send a welcome packet.

So, where what you are saying that when this object goes to the edge then you send an outcome welcome packet based on certain selection criteria. So, objects are very objects as you know qualifiers of the edges are an important concept in activities and can be used in such multiple ways.

(Refer Slide Time: 11:55)

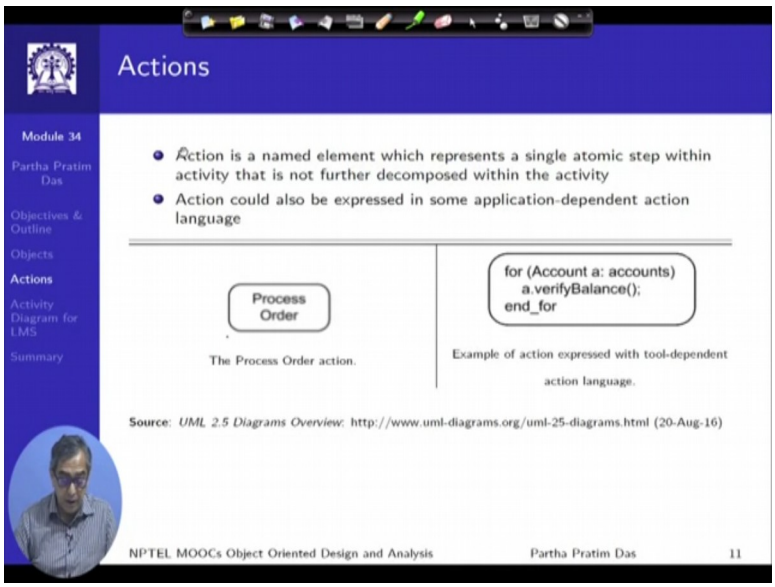


This is one another example of distributing a car, another activity. So, you assemble a car, this is an activity, on your out pin you have a car, because you have assembled it. That goes to the central buffer so that's kind of the garage where the assembled cars exist and from there you could either sell or you refresh the company cars. These are the different activities that can that can happen. So, this basically is is a is a multi-way activity that is happening and each one of them get a car from this store.

So here you can see that this is being a central buffer, this is different from if I have just said that from here to here I have a car. The difference being that if I had just shown an activity assembled car and sell car connected with an edge car so that you assemble the car and sell car that would mean that you assemble the car and that particular car you go send for selling or you assemble the car and that particular car you send for the refresh company which is not the case.

So, the central buffer gives you a different semantics where you assemble car and you accumulate them here which is some separate process and here you have from that central repository you pick up cars and sell them or refresh them and so on. So, this is how you can objects can be used to add more semantics into the activity diagram.

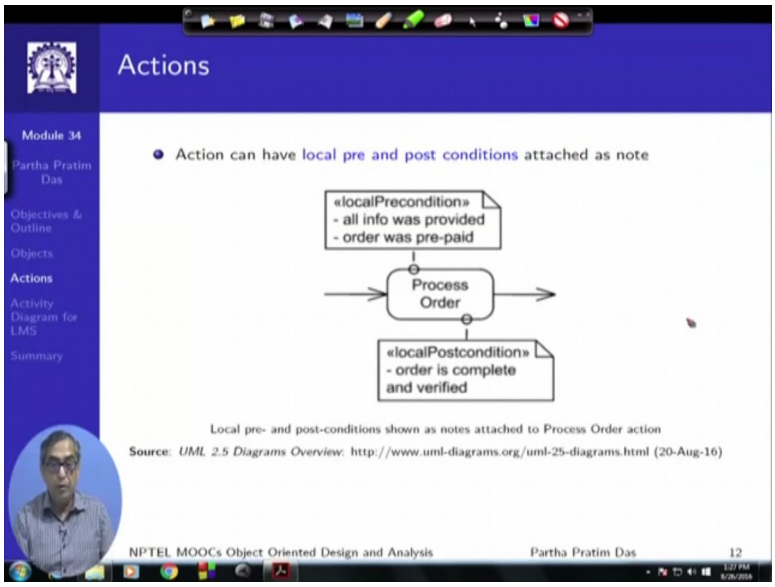
(Refer Slide Time: 13:18)



Then you have other than objects you have actions am sorry actions, they are they are also named elements. So, the action is a that the process orders action. So what action mean is basically it means there could be you know these are activities are algorithms at a certain level but you could you may want to actually have a more direct algorithm or programmatic representation of that. So, you could also define actions which are expressed in certain programming language.

So, actions not necessarily are generic in nature you could have actions in and uml activity diagram which just specific to the uml tool you are using so that you can give more and more details, for example you could have a c code attached as an action saying describing that how it will be realized and so on. So, actions give further details and low-level details in terms of that.

(Refer Slide Time: 14:23)

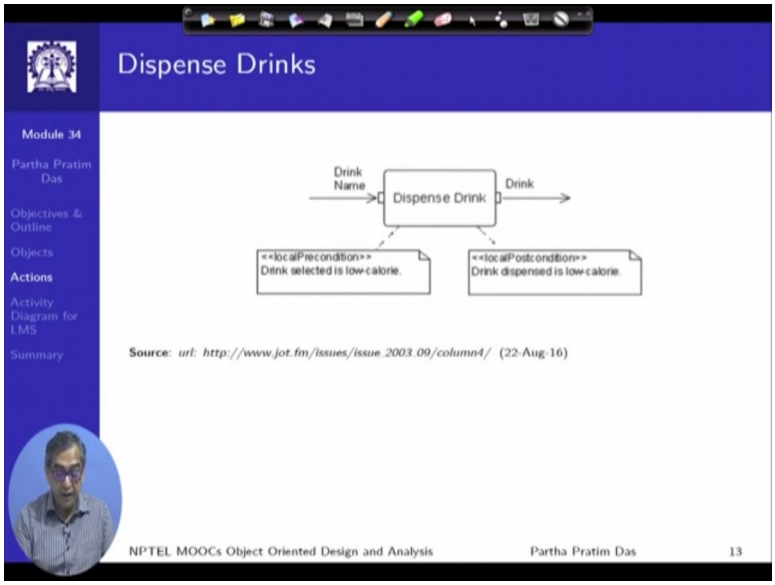


And certainly, since you want to model the overall activity. so, actions could have local precondition and local post condition which say that if this action has to happen then what are the conditions has to be satisfied for this action to take place. That's the precondition and what will

happen after that. So, if the pre-condition here of processing order is all information was provided and the order has been prepaid, the payment has been made, only then the order can be processed.

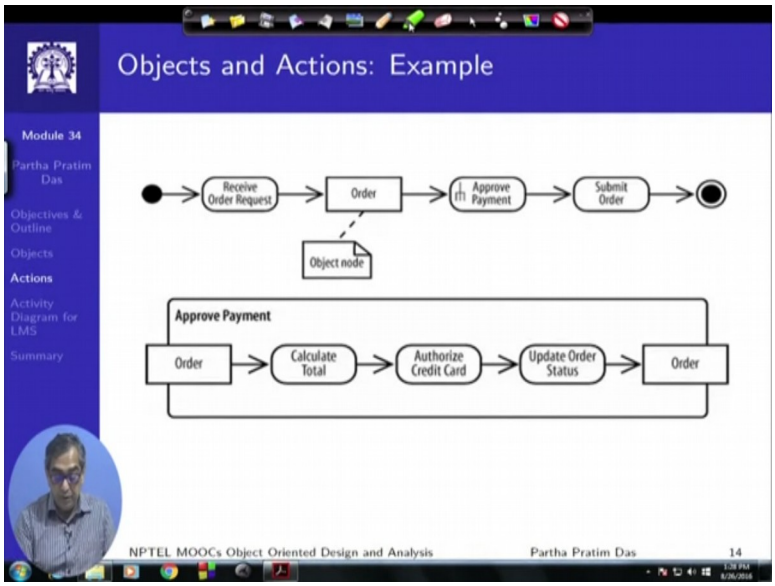
So that's the precondition to this and the post conditions certainly is the order is complete and verified. So, actions take place in this way.

(Refer Slide Time: 15:00)



So, this is another just showing dispensing of a drink. So, the precondition the action of dispensing drink has a pre-condition that whatever the user or the customer has selected that is drink is low calorie say and the post condition has to make sure that the dispensed drink is low calorie, again see pins here the input and output pin. The input pin here that gives you the name of that drink so that the dispensed drink activity can actually pick up that drink so the output pin gives you another object which is a drink object itself.

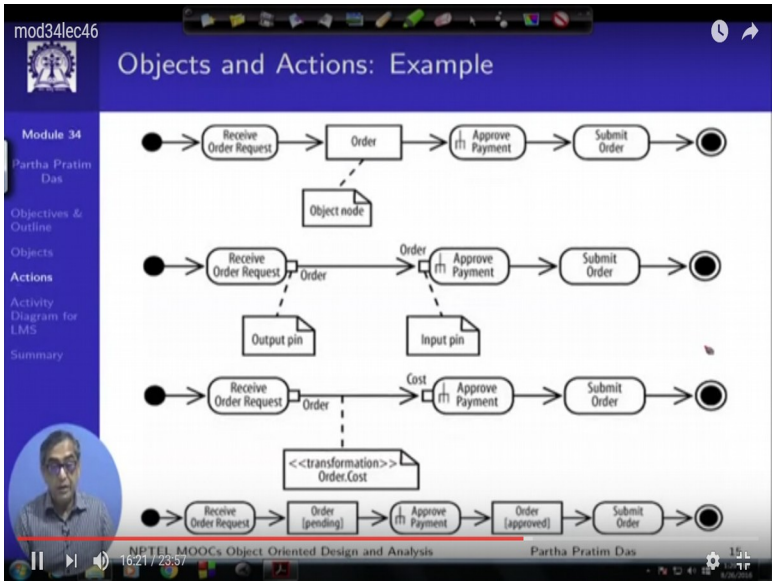
(Refer Slide Time: 15:40)



This is another example of processing order so you you receive your order request and you send it

for approval of payment, you are showing that order object in the middle which is the object node and then this goes to the submit order in this activity. You can see this notation of sub activity here, so approve payment is another detailed one which takes the order, calculates the total, authorize the credit card, update order status and finally makes the finally updates the whole thing in the in the order itself.

(Refer Slide Time: 16:20)



So, if we look into this flow we can see that in the same flow we can look at the object in multiple different ways this is the first one that we saw, the receive order request and send it for approval of payment you have an order object here. We can show the similar information this is just am just showing different variants of the same diagram but with certain sudden changes in the semantics. So, this I can also show in terms of a pin, this is an output pin in the receive order request which actually has generated an order on the output pin

And the approve payment in the input pin gets this order on the input pin and rest is same. Once we change from this representation to this representation, then I can make further changes here, I say that okay what the approve payments needs to know really not the order, what it needs to know is the amount of money that needs to be collected. So, when the output pin object could be an order, the input pin object is actually a cost that am interested in.

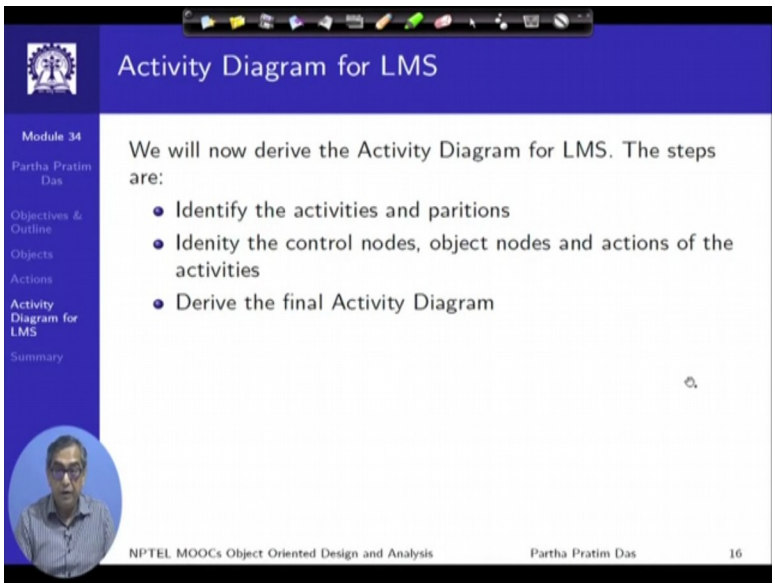
And so, I can tra have different kinds of objects at the 2 ends of the activity edge if I use pin and I can bind some transformational algorithm for converting one kind of object into another kind of object. This is represented in terms of the transformation keyword and which here the it is simply do doing a selection which is taking the order object is just checking out the cost component of it. So, it is order dot cost which gives in here.

There could be several detailed algorithms, there could be an action involved here and then you proceed with this whole thing and at a different level I can also represent this in terms of this request that are receive payment the object order object goes to the approve payment but am not getting into these details. All that I know that in this object is in a state which is un appending because this order is gone for approval.

So, this order is an order object but this is a pending status whereas once the approval has been done in in the earlier cases if you look into these cases we dint show we dint show any object flowing, it was the default token. But here we show that as if an object is this order object is flowing from the approve payment to the submit order, the same activities. Only thing that has changed is the status or the state of this order object. So, this is a modeling which is based on the state change information, later on when will talk about the state charts.

Will see how different such state information of objects could be combined in different places in uml. So here I just wanted to show you the the power of you know finally modeling the activities not only in terms of control flow which is what we have been doing in the last 2 modules in terms of activities, partitions, edges and particularly the control flow diagrams. But here what I illustrate is on top of this you can actually make a very strong object flow on the activity diagram and the tokens that move can carry whole objects with them defining a complete semantics for the diagram.

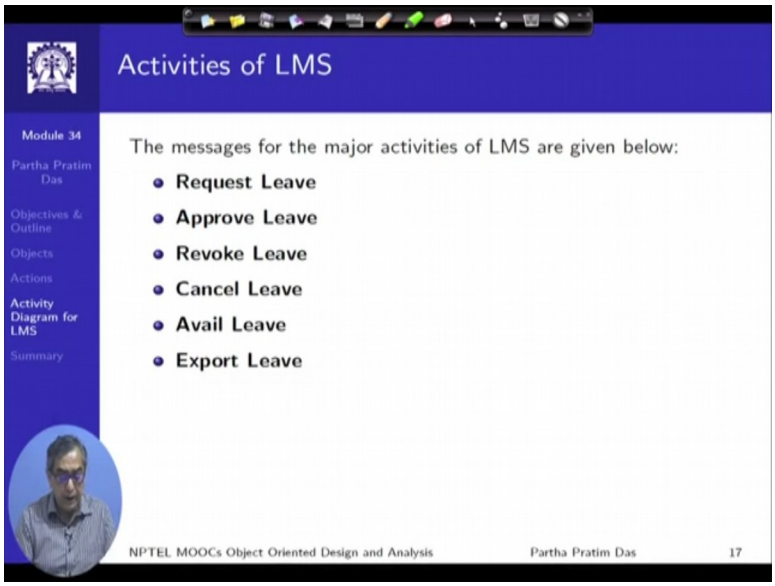
(Refer Slide Time: 19:51)



So that was basic overview of activity diagram. So, if you look at lms are running example then we need to identify activities and partitions, the control nodes, the object nodes, actions for activities and derive the activity diagram. Of course, what I show you here is very elementary form of the

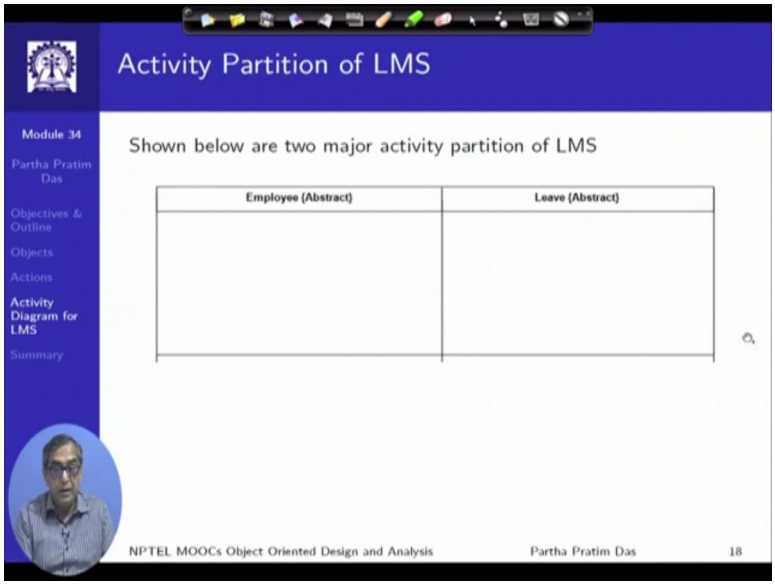
activity diagram. I will expect you to work this out at home so this is just to show you the outline.

(Refer Slide Time: 20:19)



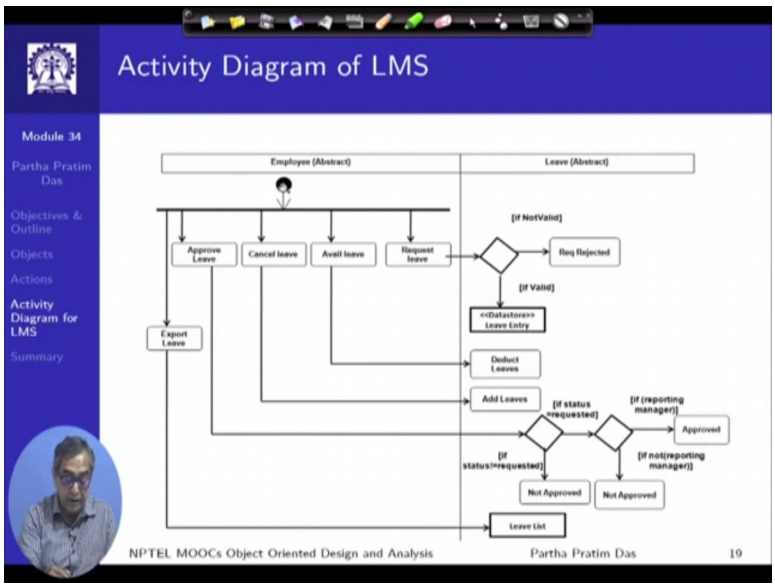
So, the different activities that we have here would clearly get in in case of lms, they will clearly ba be identified from the use cases. So, these are the different use cases that we can say will become major activities in the activity diagram. Managing leave in multiple ways, request, approve, revoke, cancel, avail, export and so on. So that's simple based on the use case diagram that we have met.

(Refer Slide Time: 20:48)



In terms of partitions in swim lines at least 2 swim lines we can immediately think of based on the 2 major agents that we have that is the employee, there is I will be activities the employee has to do and there will be activities on the leave abstraction that needs to be performed.

(Refer Slide Time: 21:13)



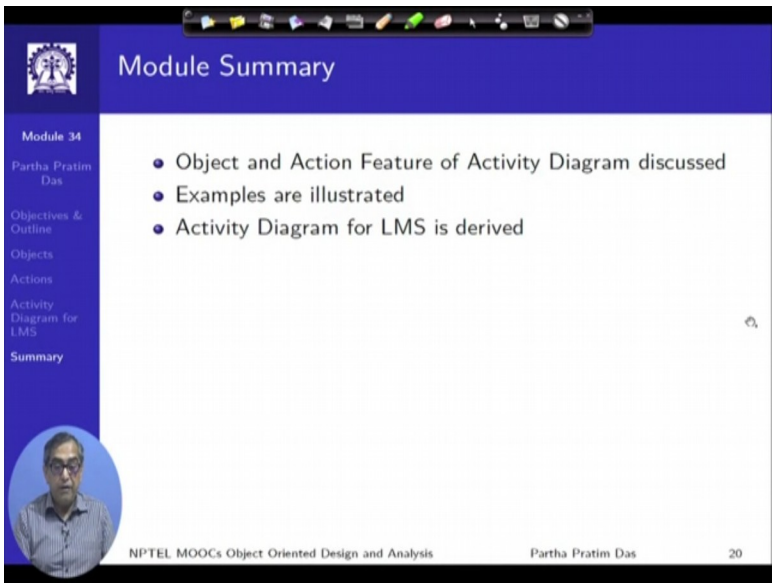
So, if you draw it on this swim lines then we can say that well we start here. .and when we start then it is possible that different employees will take different actions, different activities of requesting leave or approving leave or availing leave and so on. They all start with a fork because they can start concurrently anywhere and then it goes on with the different activities and if you request for leave, it process over to the leave swim line and depending on whether it is valid or not valid, you have different decisions, outcomes of rejected leave or leave entry.

So, this is the central buffer store object where if it is valid then the leave is entered in the store then avail leave then it will be deducted certainly there will have to be further if you cancel leave, there will have to be further activities that will come in. if you approve then you will need a decision to check if the leave is indeed requested. Only then you can go for approval, if it is not then it is not approved, if it is requested then you need to check if the person trying to approve the leave is actually the reporting manager.

So based on that the leave can be approved or otherwise the leave will be, of course these are there are further and further details and what is significantly shown missing in this diagram is we just have the fork but we do not have to join, that is finally it is not combining into some common place, so we need to figure out how to do those and complete it in terms of a complete activity diagram of lms, when we are at the end we will when you provide you the uml solution for lms, will give you a more detailed diagram.

But I will expect that meanwhile you work on this and try to build up the activity diagrams for lms as well as the ams and may be some of the other systems that interested.

(Refer Slide Time: 23:16)



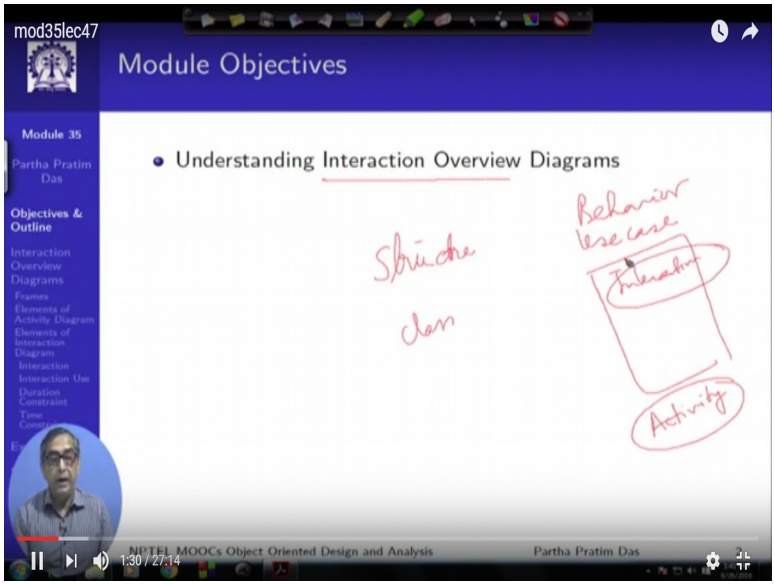
So, to summarize we have discussed about specifically about objects and actions in the activity diagrams and overall this concludes the three modules through which we were we have exposed you to different aspects of the activity diagram, how to model in them, how to use that and we have taken up several examples to illustrate the different features and the use of activity diagram and have given a brief over the activity diagram of the leave management system.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 47
Interaction Overview Diagram

Welcome to module 35 of object-oriented analysis and design. We have been discussing about variety of UML diagrams.

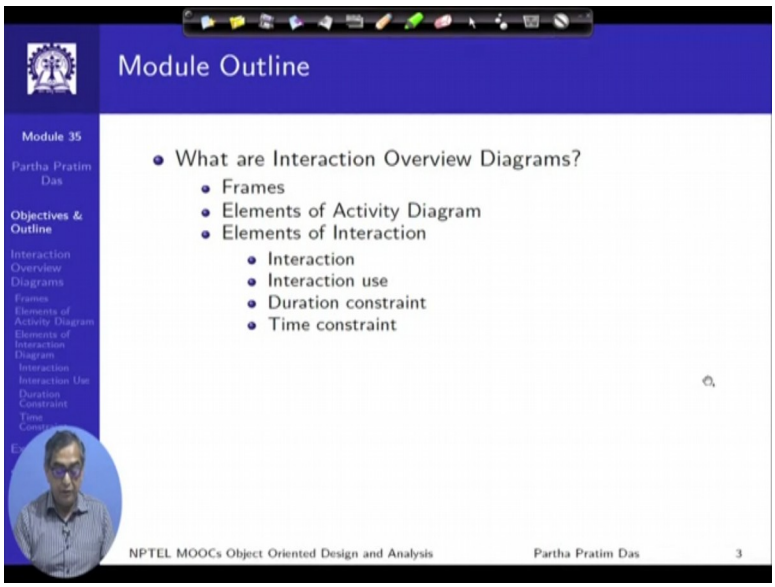
(Refer Slide Time: 00:42)



And we have specifically looked into 2 kinds of we have looked into structure diagrams and behavior diagrams and in the structure diagram we have primarily looked at the class, in behavior we have looked at use case and the whole bunch of interaction diagrams and in the last 3 modules we have looked at the activity diagrams. So, this is what we have seen so far. So, in this context we will try to understand the interaction overview diagram which is kind of a it's not a significantly independent diagram.

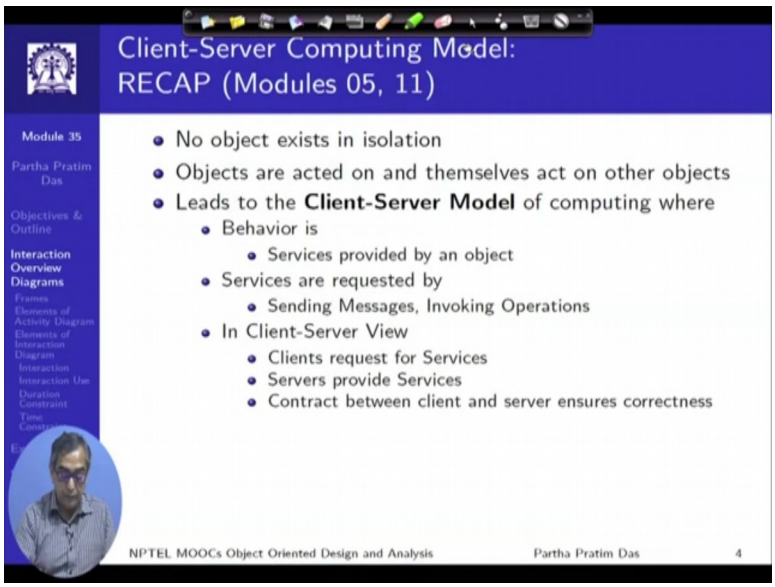
It is kind of a mix of activity and certain part of the interaction diagram. So, we will try to understand what does it.

(Refer Slide Time: 01:37)



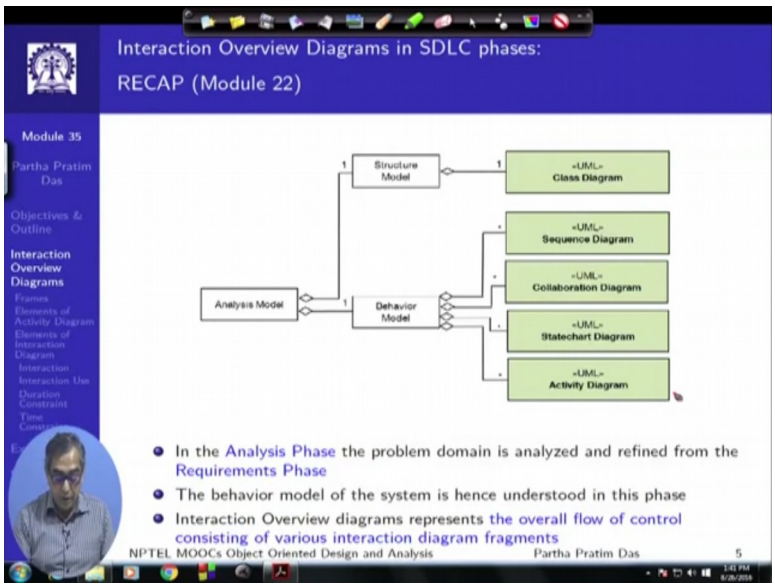
This is the module outline which will be available on the left of your screen all through the presentation.

(Refer Slide Time: 01:49)



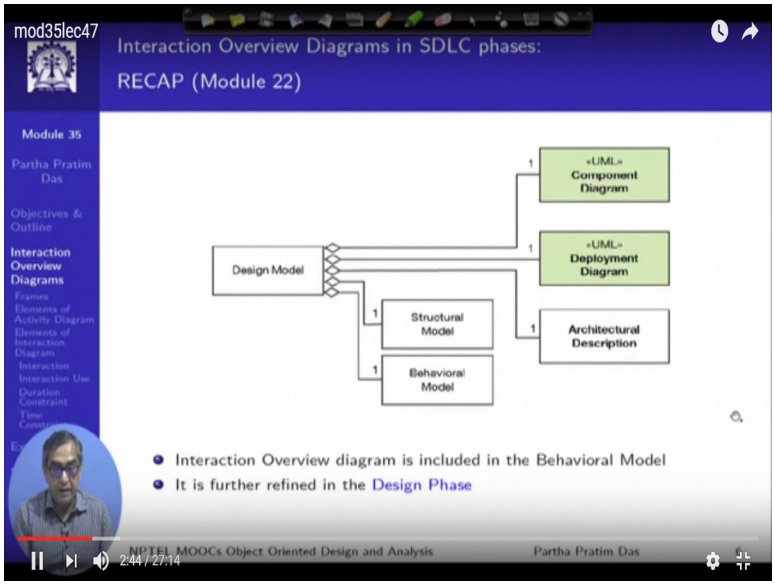
Client-server model you must be master of this now.

(Refer Slide Time: 01:54)



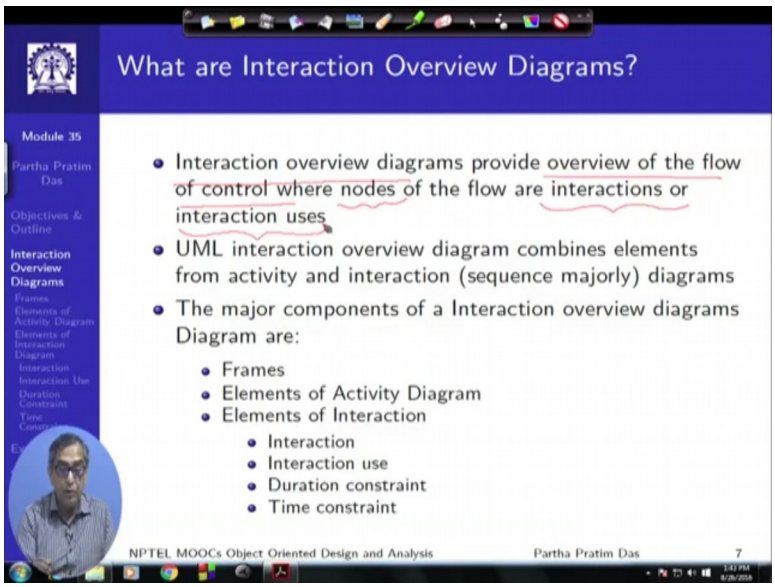
Interaction diagrams overview diagrams as I said is not separately listed here because these are the different these are the different kinds of interaction diagram that you have, the structural diagram that you have and interaction overview diagram represents the overall flow of control. It gives you a view of the system at a relatively higher level than the individual diagrams that you are looking at. So, this need to be captured very carefully starting from the early phases, starting from the analysis phase and in that way, it plays an important role.

(Refer Slide Time: 02:42)



Certainly, like all other diagrams, it gets refined in the design phase.

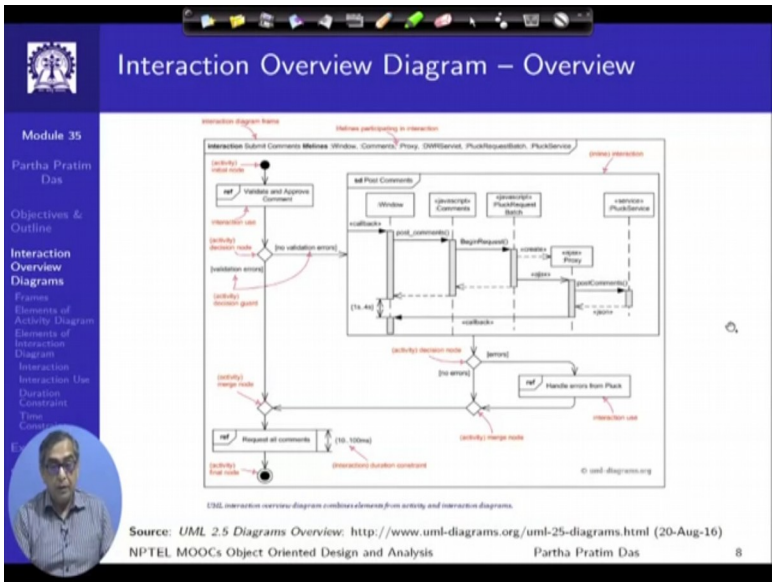
(Refer Slide Time: 02:48)



So, what are interaction overview diagram. Interaction overview diagrams provide overview of the flow of control. An important thing is the notes are interactions or interactions uses. So, they are not just elementary notes of activities or objects that we have. If we look at where the diagram is placed, that is the reason I was pointing you to the classification of diagrams is, it is one diagram which combines the elements of activity diagram as well as the interaction diagrams.

Of course, amongst the interaction diagrams significantly uses the elements from the sequence diagram. Therefore, you will find that the major components of the interaction diagram as listed below are borrowings from the activity diagram or the elements of different interaction diagram or the sequence diagram to be more precise.

(Refer Slide Time: 04:06)



So just I would like to give you an overview of the interaction overview diagram. So, if you look into this diagram and try to initially we you may not be really very concerned about what is the system that it is trying to describe and so on. But if you just look at the diagram and try to look at

the different icons and primitives being used, something should strike you. What should strike you? Is that in this diagram we are using the icons nodes of multiple different diagrams, for example if you look into this part, what are these?

These are basically lifelines, the heads of lifelines, these are tails of lifelines and so on. These are executions, these are executions, these are messages, these are different messages and so on. So clearly you have a sequence diagram. If you look into this part, these if you look into these, these are different kinds of control notes, we are certainly talking about an activity diagram. So, this is one diagram where we are trying to use both of these diagrams at an appropriate manner so that it can have a total view of the system at a different level.

So, let's lets get little deeper than this. So, this is this is where it starts, greenish yellow. The first you say is the interaction is validate and approve comment. So, let's assume that the situation is something like this that there is some webpage on which users are allowed to put comment and once the user has put a comment then you want to check whether comment can be, is a valid one and you approve it to be this pattern. So that's the that's the basic activity.

So, this is what you would like to do. So, this is a reference showing that this is an interaction which is detailed somewhere else. So, if you use this diagram, again I would like to remind you that UML diagrams are not pictures. These are these are actually active items, so when you use a proper UML tool like rational rose or visual paradigm, any of these, then these diagrams are basically active on so. You you would be able to say double click on this ref which says that you are referring to some UML diagram to validate and approve the comment process.

So that is kind of the first activity that you do here which is known as the interaction use. And then so you are not really in this particular instance, you are really not bothering about the details of how you do that validate and approve. You know that if you click on you can go there and see what are the details or the interaction there. But you just want to use that interaction and the outcome of that interaction which goes into an activated decision mode.

So, you are here now it is an activated decision mode as we have said and having 2 decision outcomes of, validation is ok there is no errors or validation has errors. Now if this is the outcome, let's say first we take the error case. If the validation is erroneous then you come here which is as you can see is a merge node, guessing if validation was not ok then you go on here, certainly the control will flow through this and you refer to request for all comments and you put a time window

you could not yet introduce what a time window is, it's a duration constant.

So, within this you put the comments and then you close this activity. So, this is this is you get as you can say this is one flow that you can see and in this flow, their advantage that you are trying to take, the reason is this is called interaction overview diagram, you really do not want to get into much details. So, if you get to this failure this is one reference here, another reference here, 2 decision nodes and you are quickly into the completion activity.

Now certainly if you really want to know what is validation, really want to know what is requesting for all comments, you can click and go to that details. On the other hand, if you had a at this outcome from the validate, that is there is no validation error that if you had a success then your control o flows to, according to activity diagram it should have flown through an activity mode. But here we use a total what is this, SD post comment which is its basically sequence diagram interaction that we are showing.

It flows into the sequence diagram interaction and this whole thing as if, so this whole thing in one way is an activity where the control flows and the control come out from here. But you do not show it simply like the activity rounded box, rather you show it with a detailing of the inside sequences that born, that is once you do not, you have no validation error then this is where you have wretched and these are the different lifelines, these are at a at a other further details of java object.

So am not getting into them, they how do they interact and exchange messages and actually do all that and finally tell you whether after the validation whether the comment could be posted. And this is the total posting process for the comment which you are showing in lot more details in terms of the sequence.

If so when this whole perceived activity for which you are actually showing the sequence diagram details, when this that would complete and the control will come out, you have possibilities of having an error or being successful. So you put a decision node again in the activity diagram and if there are no errors then you put a merge node to merge with the error flow and come back here and complete this.

So, this is the another major flow that you can see from here, starting from here, validation was okay, you went through the posting, the posting was ok, you come through, so the second major activity flow you can see. The only difference being that you have this particular activity expanded

a lot and also what can happen is after the post comments have happen, you may have some errors so that the comments could not be posted. So, you could have errors under that condition, again you want to handle the errors which you for which you refer.

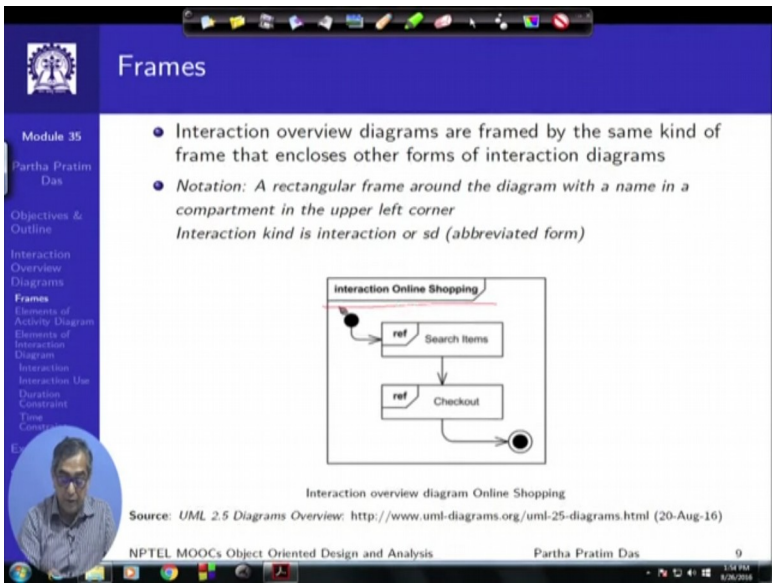
You refer to the handling error, is again may be a very detailed activity. So, you do not look at that interaction, show that interaction here, you just refer to it as a as a use of the interaction, so you can come through this. So that gives you the third major pathway, so this flow which is, so you have basically if you look at if you if you look at in the in the overall conditions then you will find that the comments was received. So, based on whether it is valid and approved, we have one flow.

If it is valid then whether it can be posted or cannot be posted, we have another 2 flows, on that total 3 flows you are showing here. And if you leave aside this part then rest of it is actually activity diagram. So, what you are trying to say is here this particular interaction headline is interaction submit comments lifeline then you have all these different lifeline objects. So, this interaction submit comment this particular overview diagram,

in this overview diagram, you want to get in overview of what happens when you submit a comment at but you are specifically interested in the sequence details, interaction details of the actual posting operation. These of the remaining activities are kind of together preconditions and post conditions that should happen so that this can happens both way. So, the interaction overview diagram has to appropriately choose as to which part it for example in other other way could be you you could have met this as a reference also.

You could have met this as a reference, references post comment instead of having this old detail, you could have just say that go there and find out what happens in post comment. But you do not want to do that because this is where you want to show that this is where the major system than a mix the major happy flow of the activity exists and you want to focus on that and you want to focus on that in terms of actual interactions whereas the remaining interactions you want to abstract out and you can see if 3 places where you just refer them but you do not actually get into them.

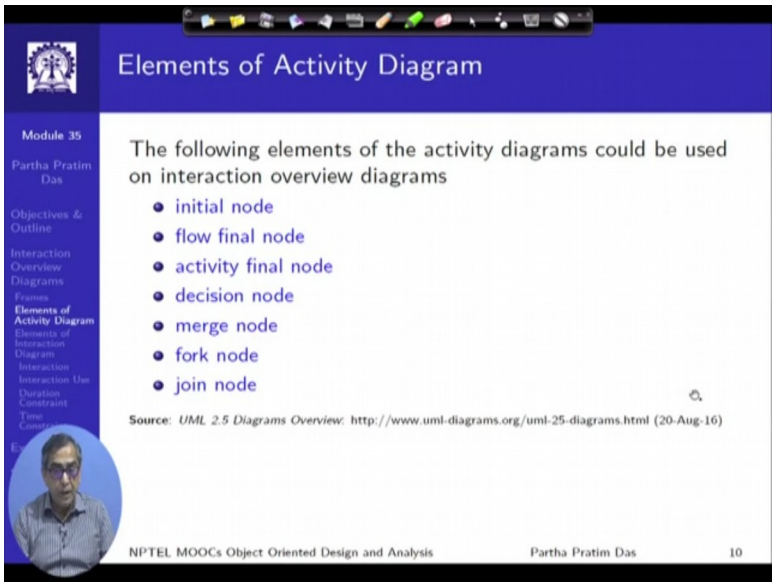
So that is the basic that balancing is the basic objective of the interaction overview diagram
(Refer Slide Time: 14:23)



And therefore, there are hardly any new you know concepts or nodes coming into this interaction diagrams naturally have frames like like earlier before, the interaction frames. So, they are typically with interaction word and then you put a name and it could refer to as I have already mentioned it could refer to other frames that are that are already available. So, if you click on this so online shopping it it says that in terms of an overview what is online shopping is you search items and checkout which is at the top level.

If you do search items then it will expand to another interaction overview kind of diagrams or connectivity diagram which will show you how to search items. So that's a being able to abstract at multiple levels and provide overview is a major purpose of this diagram and this is how it is framed

(Refer Slide Time: 15:19)

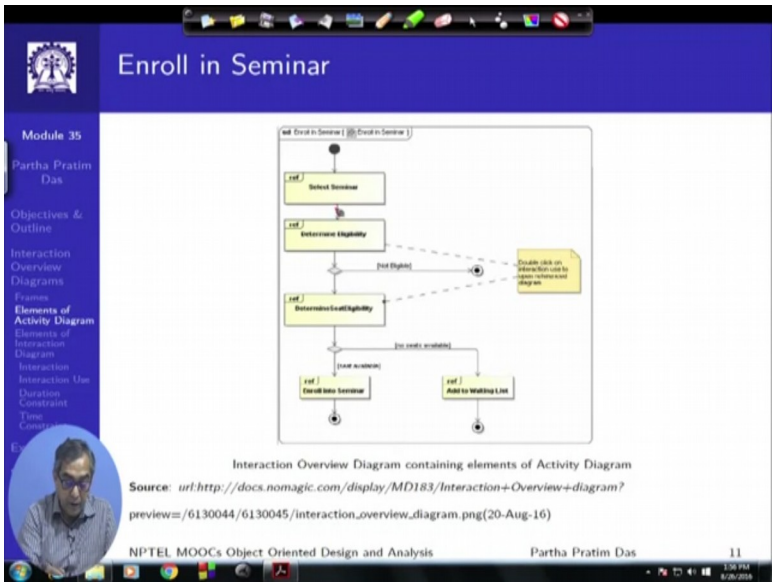


Then for representing this interaction you borrow we borrow from the activity diagram, we borrow different elements and these are the typical elements that you borrow, so you can see a whole lot of control nodes, the initial, flow final, activity final, decision node, merge node, fork node and join

nodes are which is taken from the activity diagram and you can use them and we have already seen in the last example, we have seen the use of our decision node.

We have seen the use of merge, initial, flow final, these kind of, now we will not see flow final, these are activated final, these kinds of nodes we have already seen, these are borrowed and just used in the same semantics.

(Refer Slide Time: 16:09)



Example another could be that I want to enroll for a seminar, so enrolling for a seminar so enroll for a seminar these are interaction overview, is select the seminar, determine the eligibility, determine the seat eligibility, enroll into the seminar, add to the waiting list. So, we these are the different interactions that need to be done and with those interactions what we show here as an interaction overview diagram is a total flow of the activities given that these interactions are detailed summary.

It is a how do you enroll for a seminar if I select the seminar, then how do you what do you do next is certainly determined whether you are eligible to attend, if you are eligible if you are not eligible decision node of activity diagram if you are not eligible, you reach the final node of the activity but if you are eligible the you see whether seats are available for you to join so determine the seat availability, if seats are not available you check whether they would allow you to waitlist so that you will be notified when seat is available.

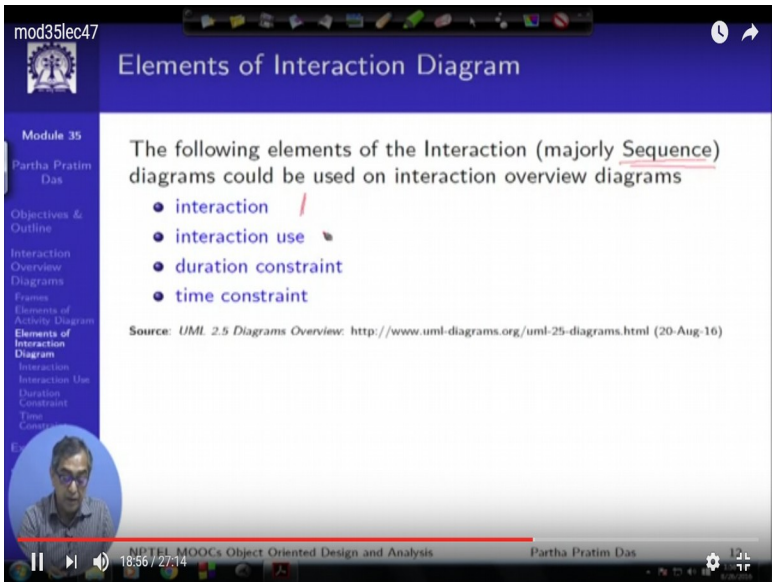
So that is another activity and which will finally take you to the activity final node. Otherwise if seats are available then you have to actually do the enrollment whatever that need the print, the price and so on so. So, this now naturally each one of these if the select seminar would have a detailed activity diagram interaction all that so it will have the sequencing of operations how do you

select what how whether you search, whether you browse, what conditions you apply and so on.

So, you do not want this top-level view to be cluttered with all these details of selecting the seminar, determining the eligibility and so on. At the same time if you just put all those diagrams together then it is very difficult for us to conceive as to what will be the total flow is being talked of. So, this interaction overview diagram basically gives you an ability to combine the interaction diagrams and the activity diagrams in a multilevel hierarchical manner so that you have all these in this top-level diagram.

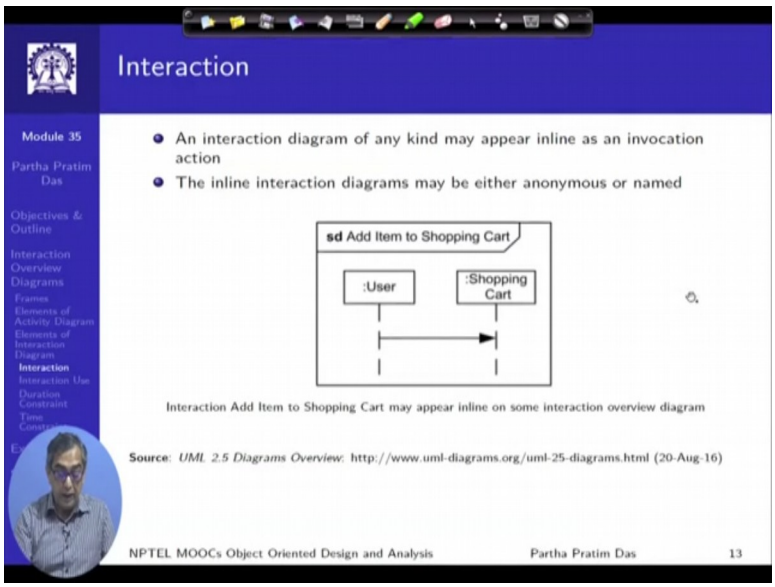
And if you just you just read the notes here, so if you double click on any one of them, then that will expand and show you the interaction overview within that, that in turn may have other references which again can be expanded in that way.

(Refer Slide Time: 18:44)



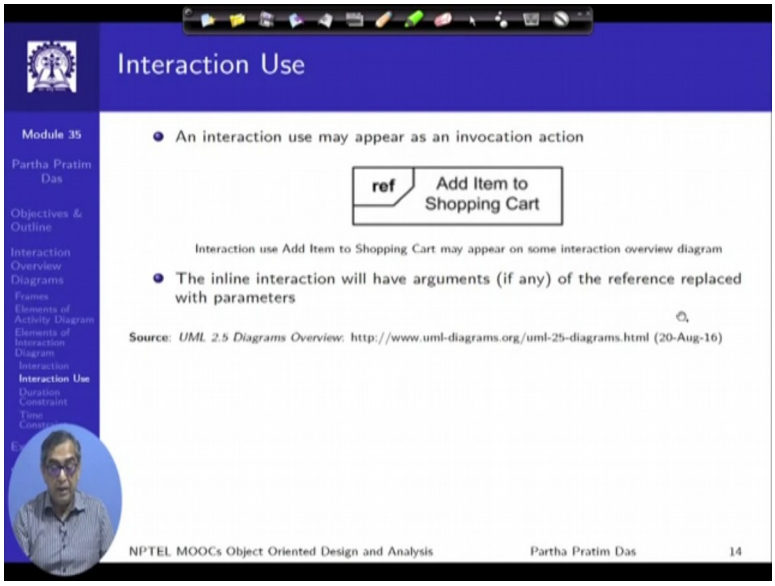
Besides that, activity diagram elements, it also takes elements from other interaction diagrams, primarily the sequence diagram so you have seen interaction, interaction use and so on. So, all those elements will be used again in the same context.

(Refer Slide Time: 19:03)



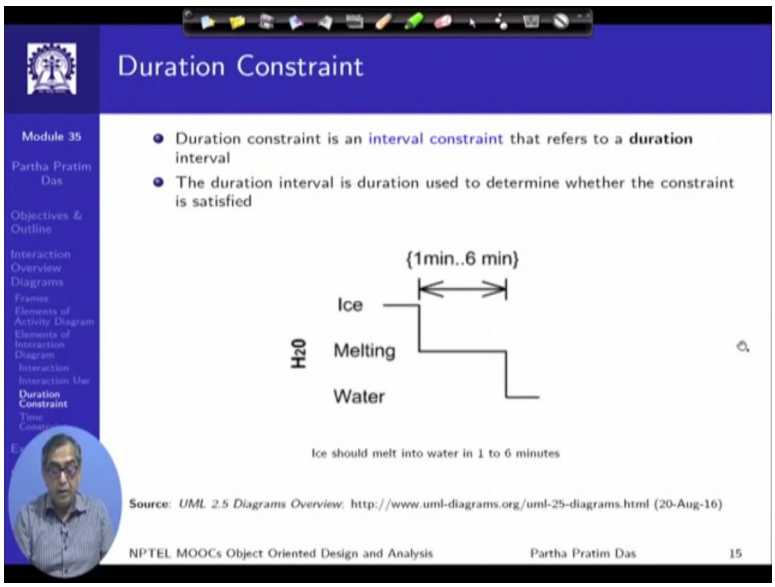
So Interactions are are shown in this manner, the SD and then you put that put the name of that and show that interaction explicitly as we had shown in the example or you could have instances of interaction use.

(Refer Slide Time: 19:17)



So, in the earlier example you recall that for actually posting the comment we had shown the interaction with all the details total flying sequence details and for the others of validate and approve and so on we just used references. So that's a when you say reference here, you basically mean that the interaction is being used so this interaction is available elsewhere and can be opened in the application.

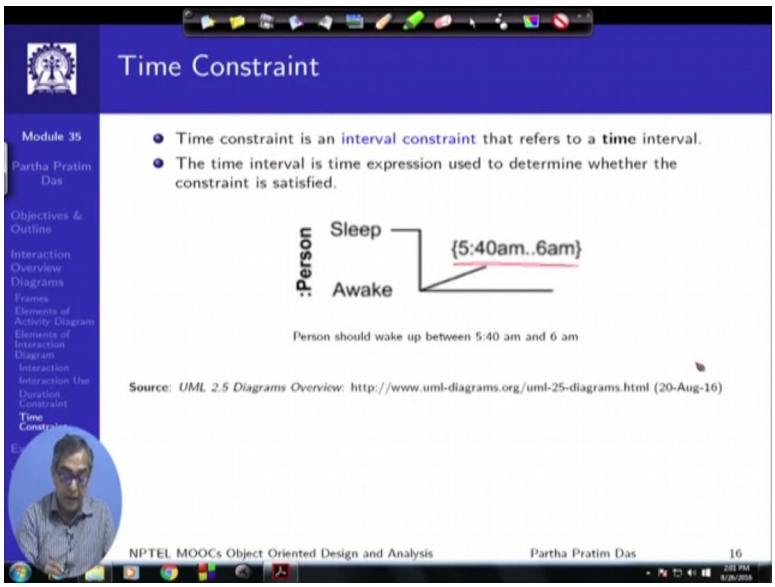
(Refer Slide Time: 19:47)



Besides that, in the interaction overview diagram you can provide constraints, particularly duration constraints so this is a this is a notation that you show, these as we will later that these are again borrowed from the timing diagram, we have not yet done the timing diagram. So, we haven't seen this but these are basically from the timing diagram. So, it says that this is a diagram showing that there are 3 states, of water H2O rather, ice, melting and water and this this is all the state change and it remains in this state for this duration.

So, whenever you have to show duration, you just show this kind of and then you write what is the time duration that is. Here you can see that the duration is also again in turn shown as a range so which reads that. So, this diagram will read that for the ice to melt into water, the melting time is anywhere between 1 minute to 6 minutes. So, such duration constraints can be regularly put.

(Refer Slide Time: 20:54)

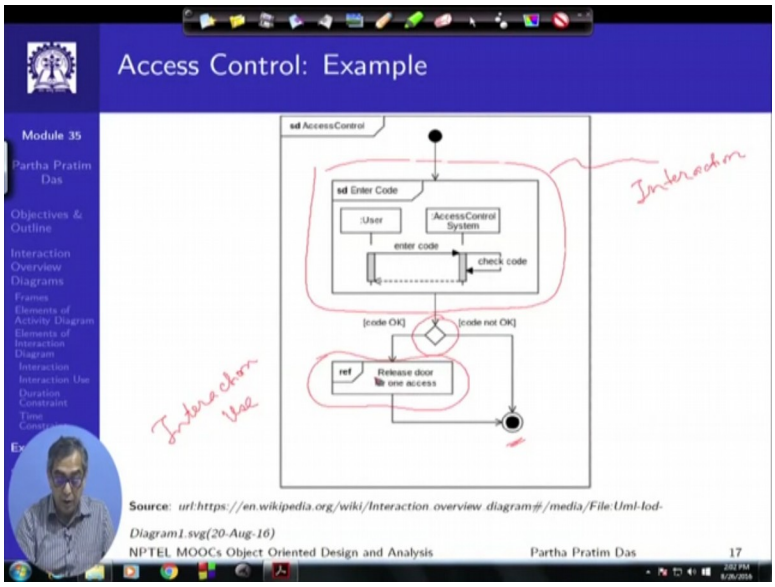


The other is you could put a time constraint. A time constraint is is basically an explicit time or explicit range of time that you associate with an event. So, you are saying that here I have a lifeline

of a person and you have 2 states sleep and awake and I associate with this event of transition from sleep to awake, I associate a time 5:40 am to 6 am which will mean that the person wakes up from sleep between 5:40 am and 6 am.

This is this is specifically talking about a particular this is also an interval constraint because it gives you an interval but you are talking about a specific time in this case.

(Refer Slide Time: 21:45)

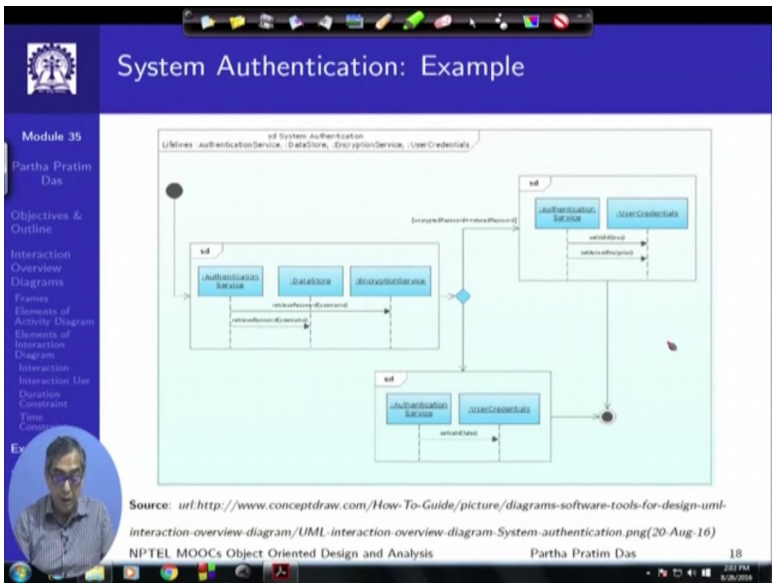


So, with this we could take a look into some examples again. So, this is another example I think we have already discussed quite a lot of this. So, this is another example we start here naturally you end here. And this is you are showing an interaction here with the details of what the user does to the access, it's trying to get access to the system. So, it enters the code and then access control system, message is sent to the access control system, access control system uses a self-message to check the code and finally provide the response.

So, if I look at the whole thing is a this is an interaction that am using and this is sequence diagram interaction and then I have the again the decision node coming from the activity so if the code is not okay, you immediately reach the final activity final state node after it but if it is ok then am doing a reference so this is when am doing an interaction interaction use. So here we have not shown how to release door for one access.

There may be several logics involved in that, several activities involved in that. This left says that there is it is defined somewhere and we are just using that interaction so that's a reference to this.

(Refer Slide Time: 23:14)



So, this is typically how the interaction diagrams will be drawn. You could look at I would leave live this primarily for you to go through and understand better from the slide so we were saying the SD is system authentication lifelines involved are all these authentication service, data store, encryption service and user credential, authentication service you can see here in this, the data store is here, encryption service is here and user credentials are here so that that basically describes at the top level.

Now if I look at the structure of the interaction overview diagram we will see that there is one interaction here, there is another interaction here and there is a third interaction here. This particular interaction overview diagram does not use a does not have a ref that is interaction use. So, it looks at these 3 interactions and then builds in terms of the activity diagram with these notes. So, we have a start here you first do this interaction, then you have a decision mate to either this interaction or this interaction and finally you merge at this node.

So, there is a little bit of shortcut the merge node is not explicitly shown, it is shown that you both of them will actually end. So, in this interaction again we have the sequence diagram showing the authentication service and say retrieve password message and from the data store password is retrieved and then you check whether the so you retrieve the password and then you put that retrieved password to the encryption service am sorry you retrieve the password from the data store using the encryption service so you get an encrypted password.

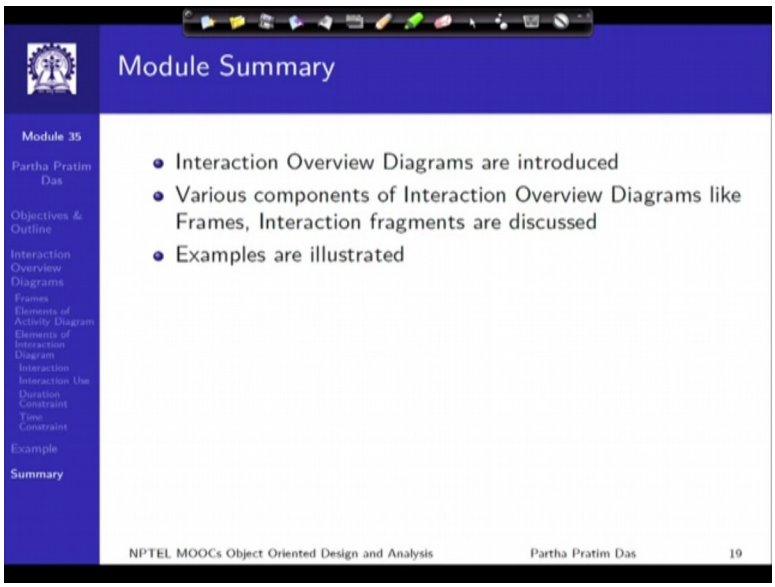
And this encrypted password and the stored password will have to match. if they match then you do this interaction, if they do not match then you do this interaction. So, if they match then you get the other user details, the validations and you have the success, if they do not match, you have a failure

and each one of them finally ends up with the activity final node. So, this shows another way of am just another way of showing that interaction overview diagram.

So, the advantage of these diagrams are this is I mean if you just do sequence diagram these are 3 different sequence diagrams. If you just do the do activities like the rounded boxes and draw that this is where your initial comment, this is where you have a decision which goes here and goes here and then you have this is the final and you have this, if you just draw this activity then you have a very abstract view, we just do not know what is happening.

So on one in the activity we have a very top level abstracted view may of a of this kind of a system, in terms of sequence diagrams you have 3 disjointed sequence diagrams which you do not know how they interplay and that is a place that the interaction overview diagram take which show you a mix of these details and the activities with the choice that you you have a choice as to whether you want to show an interaction in depth in the diagram itself or you just want to use that interaction and show the details somewhere else. Giving you a much better scope to comprehend and represent the diagrams.

(Refer Slide Time: 26:48)



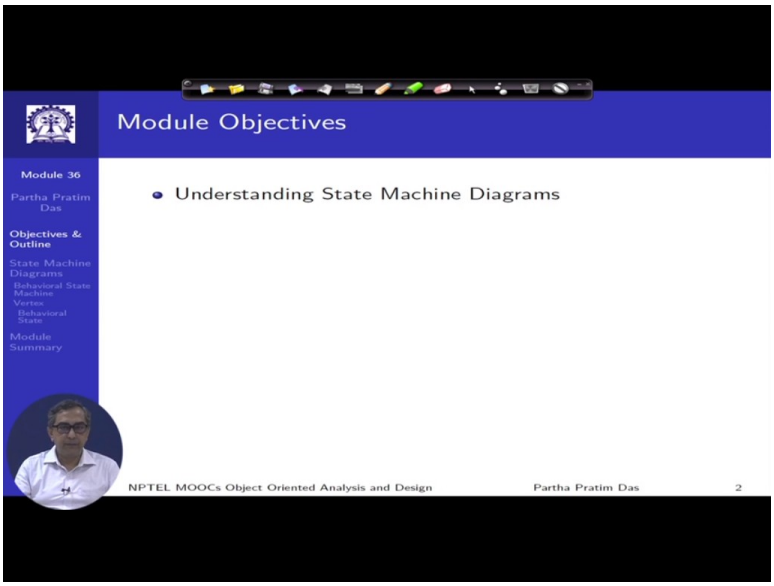
So, to summarize we have discussed about interaction overview diagrams, we have introduced that we have seen that it is basically a combination of appropriate combination of activity diagram and interaction diagrams primarily the sequence diagram and we have shown a couple of examples discussing how interaction overview diagram can be used effectively for building UML models.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 48
State Machine Diagrams: Part I

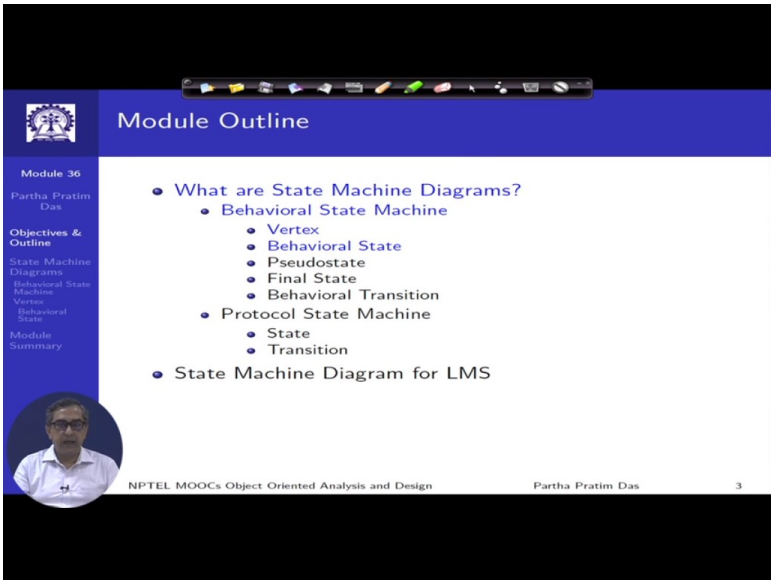
Welcome to module 36 of object-oriented analysis and design. We have been talking about different uml diagrams. In the current module and continuing in the next 2, we will talk specifically about state machine diagrams.

(Refer Slide Time: 00:37)



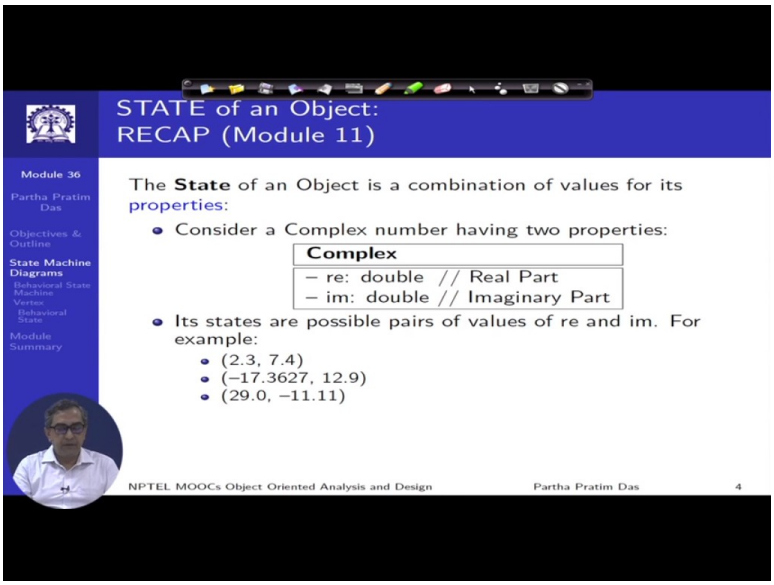
So the objective in all these 3 modules would be understand the state machine diagrams and accordingly this is the oultline that we will follow.

(Refer Slide Time: 00:51)



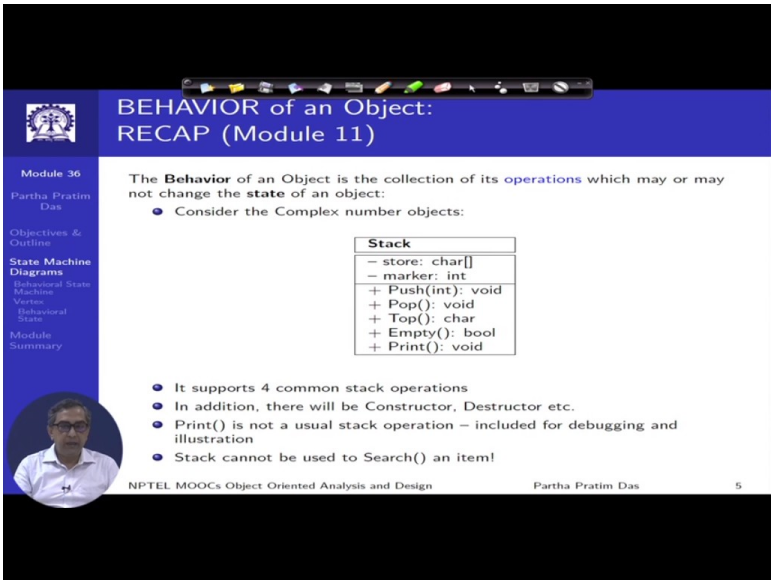
And as we have been following the blue part on this outline is actually what we will do in this current module. Remaining parts will be done in the subsequent modules.

(Refer Slide Time: 01:07)



So to start with we will make a quick recap in terms of some of the basic concepts which will be referred more frequently here in this module. First naturally the concept of the state of an object. So we have talked in terms of an object being defined in terms of its states, its operations and its identity, particularly if you talk about state then depending on the different properties that it has then the different combination of values that the properties can take define the state object. And we have seen that it could be static or it could be dynamic.

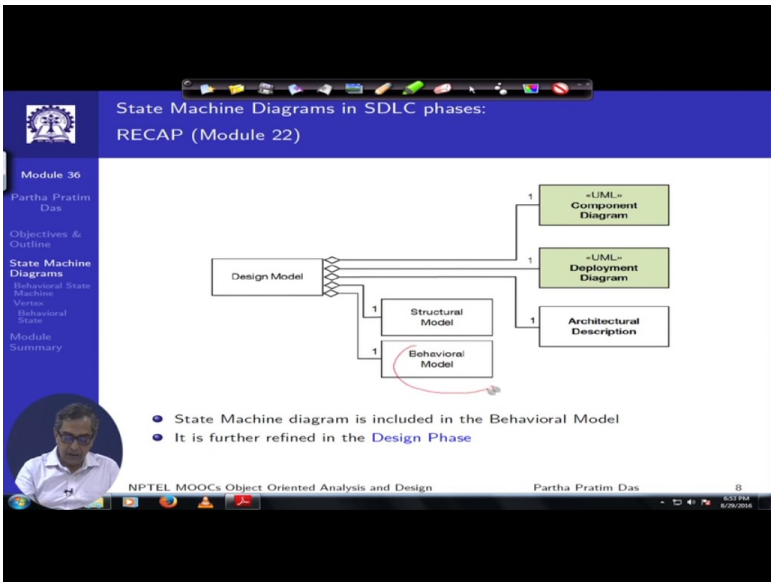
(Refer Slide Time: 01:52)



We have also seen that based on this there is a behavior of the object which is based on the combination of operations which certainly are defined based on the state of the object. We continue to refer to the client-server model am sure by now you have understood the client server model. In terms of the sdlc phases, software development lifecycle phases, a state machine diagram is first constructed in analysis phase.

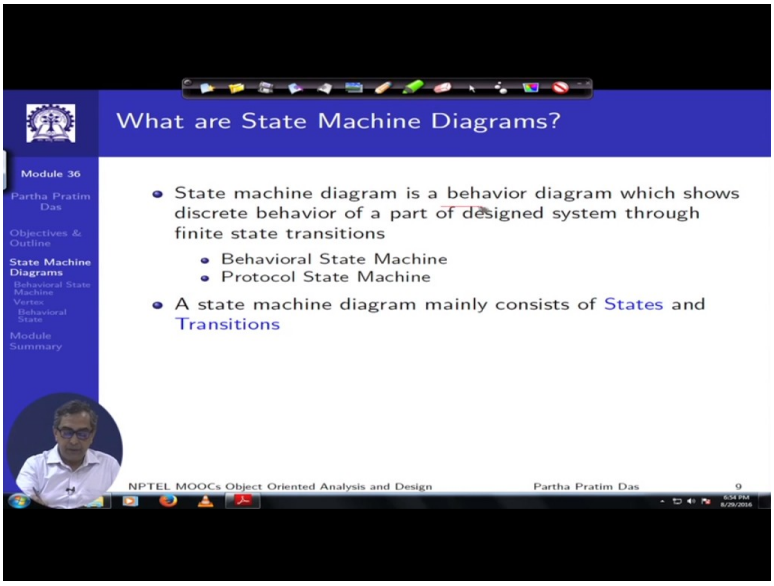
It is also called the state chart diagram; actually the earlier name was state chart diagram, in the recent versions of uml this is more frequently called as a state machine diagram. So in the analysis phase as we analyze different sequence and collaboration information and we talk about activity, we also extract the state based information in the state machine diagram.

(Refer Slide Time: 03:03)



And as is true for most of the other models we refine on this state machine diagram which is a behavioral model and had get a more accurate state machine diagram for the problem at state.

(Refer Slide Time: 03:20)

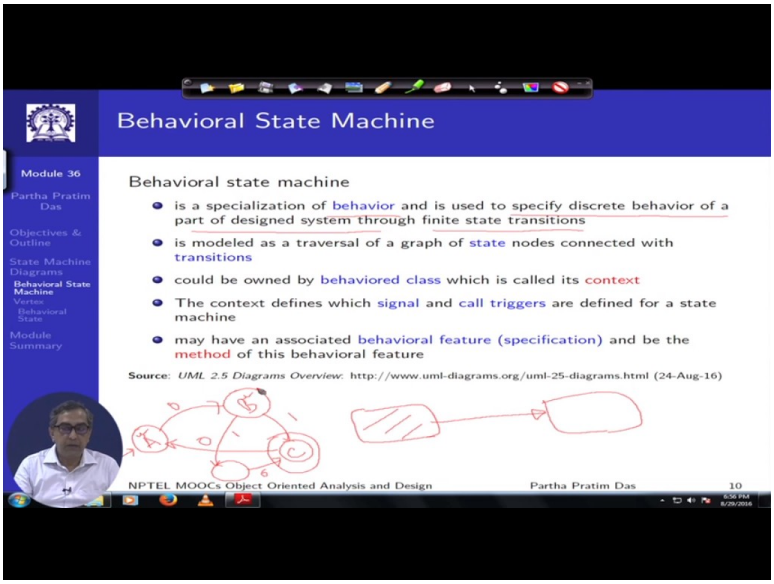


So this is the basic background and based on this we would like to talk about the state machine diagrams. The state machine diagram is a behavioral diagram; it shows the discrete behavior of part of the design system through finite state transitions. So quite a lot is being said here, one is let us look at these in in parts. one is this this is a behavior diagram, we understand some kind of a behavior and it shows a discrete behavior, discrete behavior in the sense that a behavior could be continuous as well.

But here you have discrete behavior that is it is defined in terms of certain set of discrete values and therefore the combination of this is values define a finite combination which we can think as a finite set of states and their transitions. And there are primarily 2 kinds of state machines that the state machine diagram would be involved with. 1 that is called the behavioral state machine and the other is a protocol state machine. Naturally a state machine will consist primarily of states and transitions.

So in that sense they are pretty much like though they are substantially endowed further but they are pretty much like the finite state machines or the regular automata that you have studied in other courses. But here they will take much more complex shape.

(Refer Slide Time: 04:52)

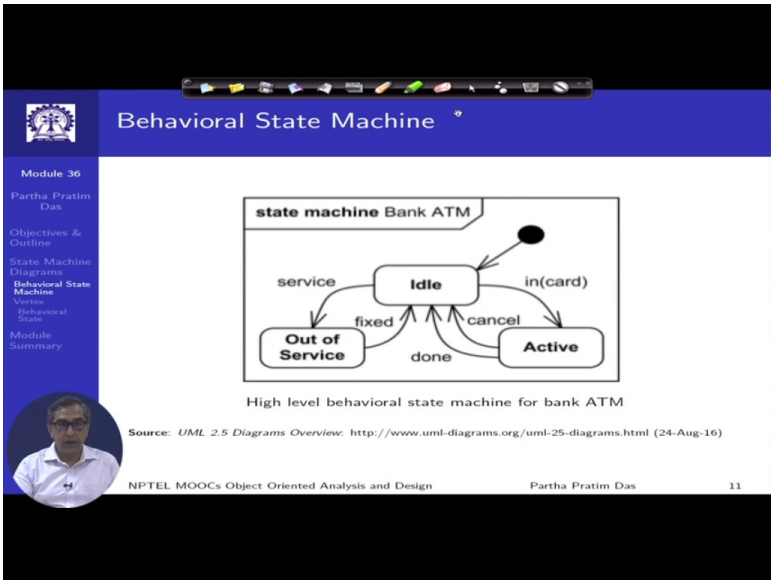


So first let us talk about the behavioral state machine in this module. This is a specialization of the behavior and is used to specify the discrete behavior of a part of the design system through finite state transitions. So just think about what you are saying, there are there will be states, say these are states and there are transitions between these states. And every state is a has a certain behavior. So unlike say the traditional fsm if you think about you would have done traditional fsms like this with 01 input you have state a, state b, you say on 0 go to this state on 1 come to this state, on 0 go to this state and so on.

You you must have seen such kind of state machines like this and some states are defined as final states, some as an initial state. Here in so states in in the traditional fsm sense are basically place called that markers which remember the kind of discrete combination of information that the particular state represents. But here we will see that in place of a behavioral state diagram, the states are more empowered. They have a behavioral feature or specification, in that a state will may be have certain conditions or event for entry, and every state may have some entry.

Every state may have certain conditions or actions on exit and typically they will have a behavior that happens in this state. So it is this is not a state at a very micro level of the system description. They could be at a much higher level of abstraction where a whole lot of things might happen within a behavioral state based on a certain entry and certain exit and that behavior is what will be captured in the action for this state. So the states in the in this context will often be owned by the behavioral class which is called the context of the state and which defines the signal and call triggers that will make an entry into the behavioral state of a behavioral state machine.

(Refer Slide Time: 07:30)



So let us slowly start taking a look into some of the possibilities. So we start with the simple one. Where this is the this is a state machine of a bank atm possibly. So this is where you start and when you start, the machine is in an idle state. Now this idle state is a behavioral state in the sense that while it is idle it is not a passive state it is in, it is still active to receive inputs, appropriate inputs if we keep on checking if somebody is pressing some buttons, if somebody is inserting a card or anything like that.

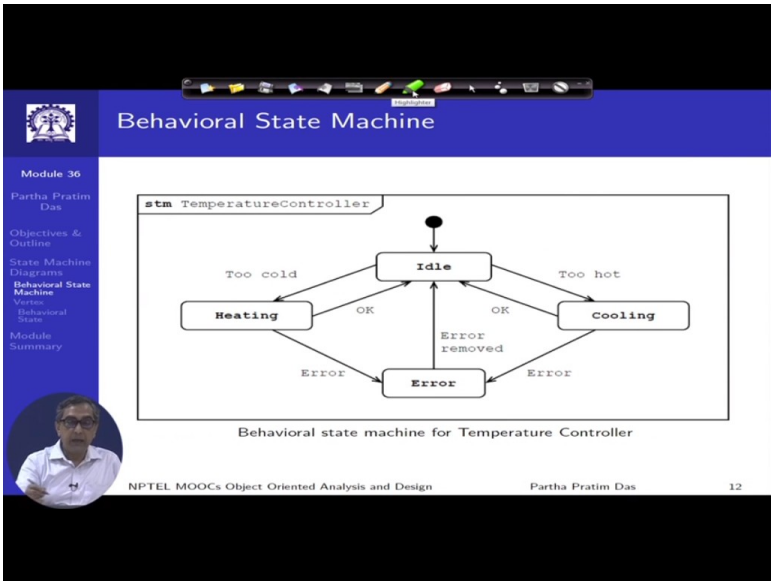
So at a macroscopic level, this is an idle state, but internally it may be doing quite a few things. but then if you have a specific event like in card which means a card is entered, then it gets into another state which is active state where this is a state where it is supposed to do a whole lot of things basically having an active will actually interact with the user and possibly allow the user to withdraw money or check balance or things like that and when all that is done, then it will come back to the idle state.

Or it could be that in the half way either the card is not accepted or the pin was wrong or the user decides to abort the transaction that the user was doing, then it could be cancel and it will again come back to the idle state. An alternate path could be that the machine is in idle state and it needs to be, serviced needs to be maintained certain preventive maintenance would be done, so it is put into the service mode. And when it is put into the service mode; it gets into a state which is out of service state.

And only when the maintenance or servicing is over, then it will again have a transition takes which will bring it back to the idle state. So we could in terms of very simple three states we can attain abstract high level, we can describe a atm application and atm process and this is what is the purpose of a behavioral state machine. There could certainly be lot more details than this, we will get to more

details but let us work through couple of other examples.

(Refer Slide Time: 10:04)



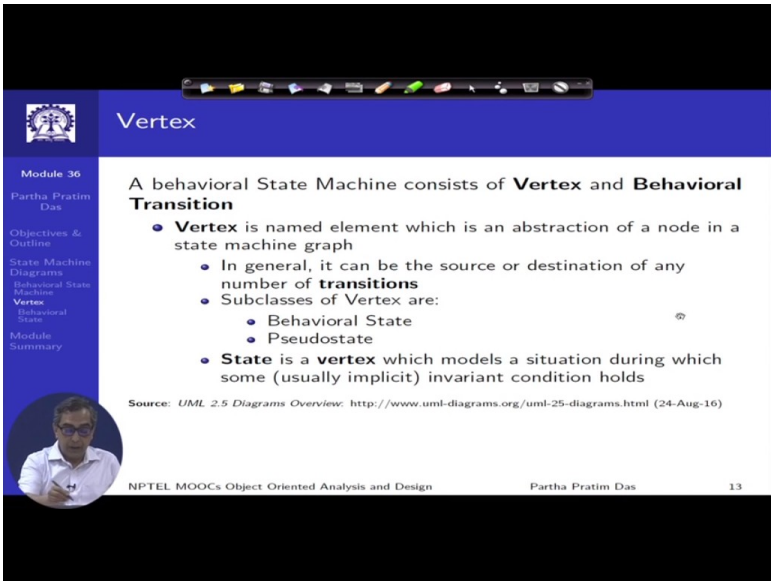
This is another example of a temperature controller which again starts am sorry this which again starts in the idle state and again what I am trying to repeatedly point out, this is not like just the finite state machine where if it is in idle state it will it will basically continue to be in the state unless some specific input comes in here, in the idle state itself it is doing a lot of actions, it is possibly trying to sense this is the temperature controller it is regularly collecting the current temperature value through the temperature sensors and based on the value of the temperature if it is too cold.

It changes state and puts on the heater. So when it state is heating again lot of things are happening in this heating state, the heater is on and again the temperature is being monitored and so when it is co become adequately hot or less cold then it says ok, heating is done and it comes back to the idle state. It has achieved the temperature that has to be managed. similarly, it can happen on the other side there is the temperature goes higher than what is specified then it will get into this cooling state, put the cooler on and so on.

And on completion, on achieving the target temperature it will come back to the idle state again or it is possible that from the either the heating state or the cooling state, there might be some error, the heater might malfunction, the cooler might malfunction and it might go into the error state and it will come back to the idle again when the error is removed. So each one of these again at a high level describes the overall behavior of the system.

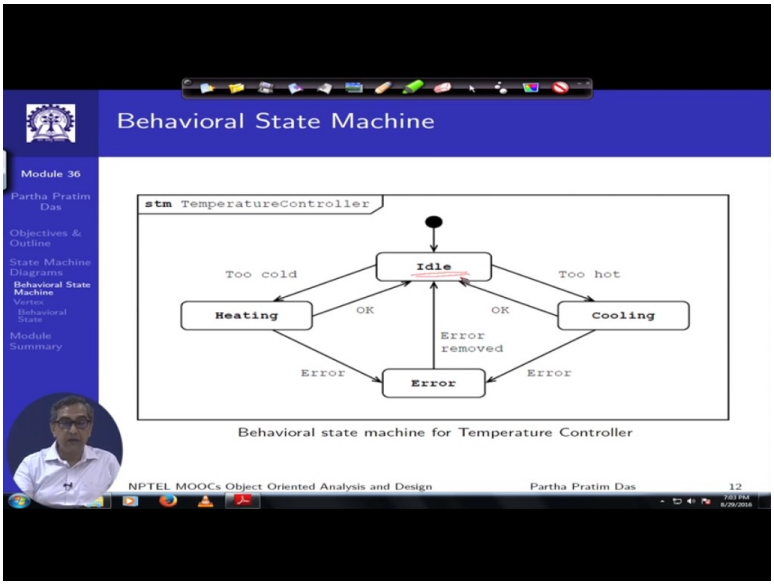
So when we capture the system requirements and start analyzing, we try to see the try to find out situations where we can identify discrete states, behavior states of the system which can in a crisp way describe the overall discrete behavior of the system that we are trying to describe.

(Refer Slide Time: 12:04)



So the behavioral state machine in in structural terms consists of vertices and behavioral transition. A vertex is a named element and which is an abstraction node in the state machine graph, we will see there are other kinds of states as well. And there are primarily 2 sub classes of vertices; one is the behavioral state vertex and the pseudostate vertex. So state is a vertex that models the situation during which usually some invariant conditions hold.

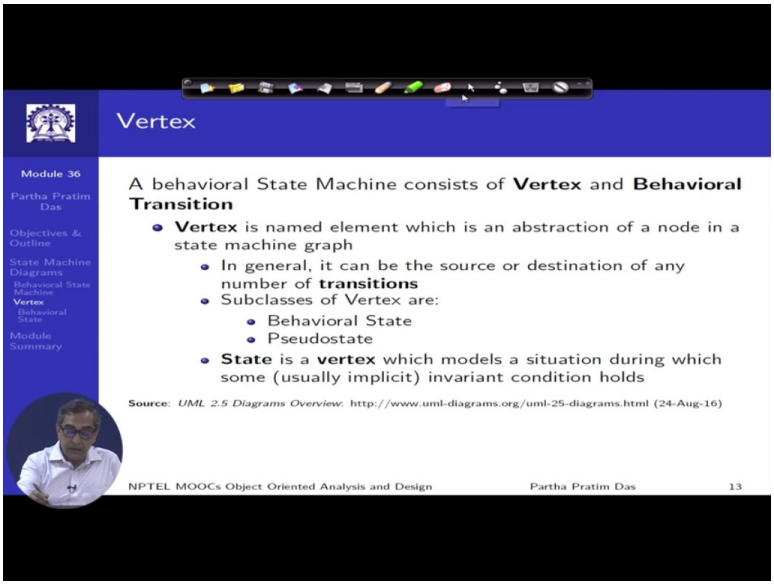
(Refer Slide Time: 12:51)



So you if you just refer back to let me just take you back to the sorry let me just take you back to here.

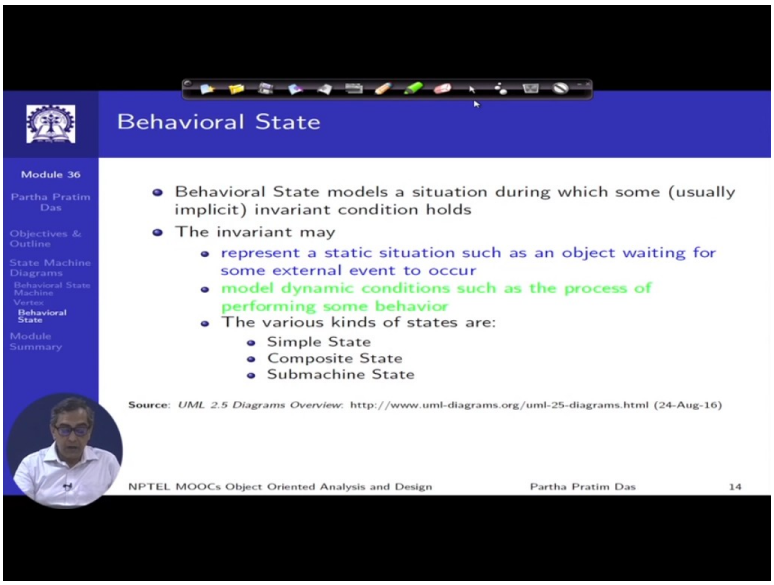
So this is a state because here certain invariant condition hold, the invariant condition is temperature is within the desired limits. This is where some other invariant conditions hold, this temperature continues to be remained lower than where it should be and the heater is actually operating and so on.

(Refer Slide Time: 13:17)



So that is how the state will be a vertex and so what you saw there are the different vertices of a behavioral state machine diagram.

(Refer Slide Time: 13:25)

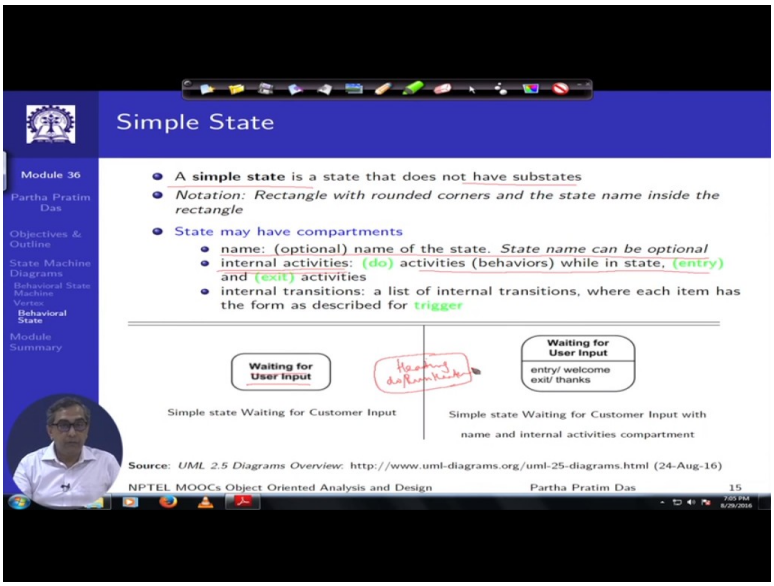


So the behavioral state ss the first of the 2 kinds of vertices that you have, the behavioral states has invariants as we say they represent a static situation such an object is waiting for some external event to occur or certain or particular condition is being maintained and it modeled dynamic conditions as that the process of performing some behavior like it is in terms of the temperature controller it is the

Process of heating which is regularly performing the heating operation to try to achieve the target temperature.

The behavioral state in turn will be of again 3 kinds, they are called simple states, composite states and submachine state. Let us look into the definition and then we will explain what they mean.

(Refer Slide Time: 14:15)



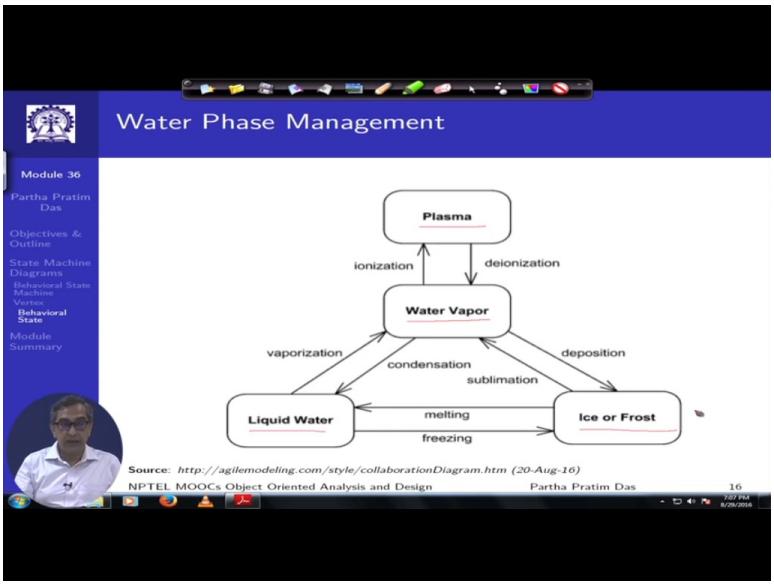
A simple state is one that does not have sub state. So we need to look at what are sub states. Will come to that when we go to the composite state. So a simple state is just a state, so it has a name that is a minimum so which describe what that simple state is doing. But in addition, so the name is is expected but it may be optional, it may not be there. but what we are the 2 other things is what is a internal activities that might happen in this state, that is the behavior has to where an object is in this state what does it do.

So if if the object is in a state whose name is heating then they do action could be run heater. So what is which is activity that happens during while the system is in that state, so that is the do part or the activity part of a simple behavioral state. besides we will have an entry may optionally may have entry and exit activities. For example you are waiting for input user input is one simple state and your entry is welcome. So the activity when you enter the state is you put on a welcome message and the activity when you exit the state is you put on a thanks message and so on.

So what is important to understand and that is true for the simple behavioral state and it actually in turn is true for any behavioral state is unlike the state of an fsm you just remember your internal condition

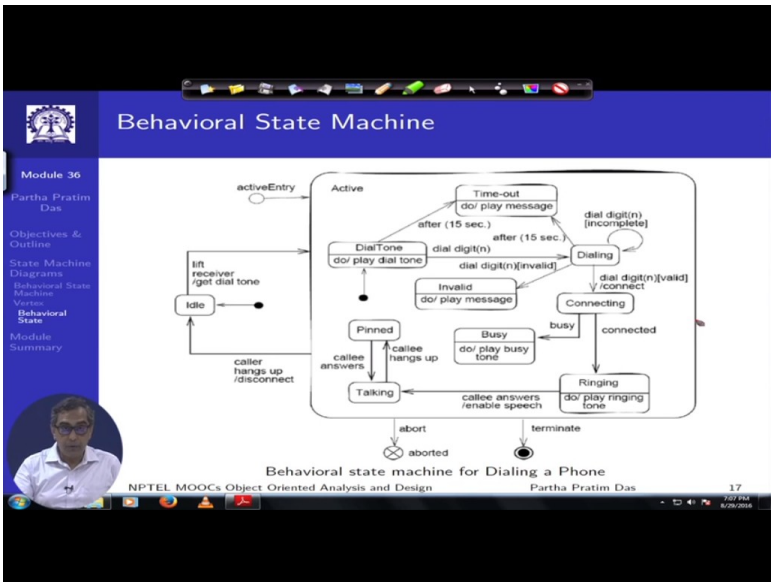
as to where you are in addition to that it has an entry activity, it has an exit activity and it has a performance activity, internal performance, internal transitions that may happen. So this is the behavioral state will always carry with it a certain behavior which is defined in terms of these internal activity and internal transitions that may happen in in interns of within that state machine.

(Refer Slide Time: 16:46)



Now so this is a just another example, so here you have multiple states, these are the 4 4 states, these are simple states and these are the different transitions. So if you have water vapor if you condense, it will become liquid water, if you freeze it will become ice, if you sublimate it becomes water vapor directly and so on. You can just follow and form a simple knowledge of physics you can understand that this kind of system can also be described in terms of a state machine diagram.

(Refer Slide Time: 17:18)



Let us look into something which is little bit more closer to the definition of behavioral state machine. This is basically for dialing a phone, so we can see these are the different states that a phone could be in. So in every state if you if you notice there is a name of the state, these are the name of the state. So this is the state dial tone, this is the state invalid, this is the busy, this is the state dialing and so on. So these are the states that a phone is expected to be in.

So if it is in a particular state then it has a do or action activity of that state. So if the phone is in dial tone state, then you are the activity is it will play the dial tone. So that is what we expect right, if we pick up the phone and that is in the dial tone state and if we do that and in which you can, in this state you can dial the digits and once you start dialing the digits you get into different state which is a dialing state where it could dial further digits and keep on remaining in that state.

It could be that you dial a dial a wrong digit or you dial a number which is not valid, then it will take you to a different state which is invalid state where the action is to play the message saying that you have dialed a wrong number or something. Otherwise if you have dialed a valid number then your next transition happens to connect and that takes you to connecting state for the phone where the number is being connected. And there are 2 possibilities from this connecting stage.

you might get connected where you might go to the ringing state where your action is to play ringing tone or it could be the number could be busy then your action is get then you get into a transition into a busy state and your action is to play the busy tone. Again in the ringing state if you are in then if the callee answers then you get into the talking state and then you continue in that talking state till the callee hangs up or get when the callee answers.

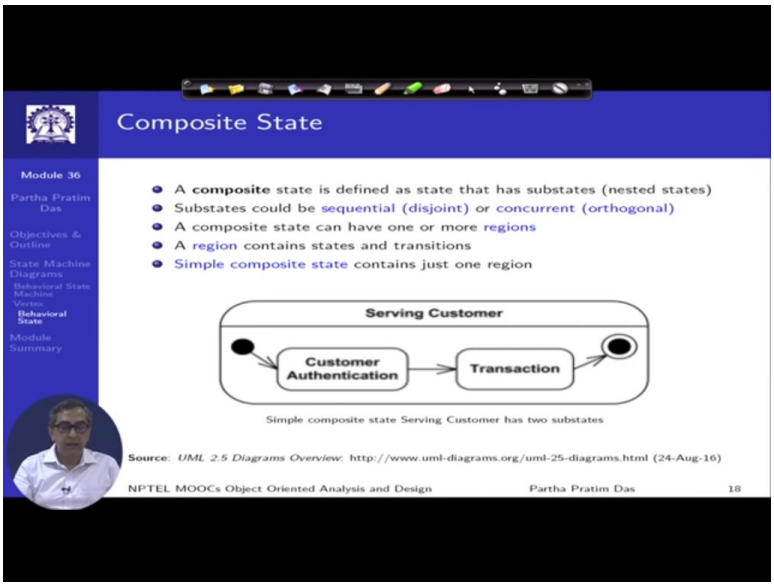
And this is how this whole state machine will define the different distinct you can you can understand that if you think of the phone in terms of a digital system then there will be several other states involved which will define how to play the dial tone, how to actually do the dialing, every key press could involve kinds of different states and so on. But we are always interested to model the system at an appropriate level and at this level if you are trying to model the dialing of a phone or making a call.

Then we are most interested to identify only those distinct states that we will the system will take and the actions that will happen and what are the transitions that will take between these states so that you get a finite behavior on this. Besides that you have some you may have an idle state when you are not

even trying to dial so in a state somewhere you actually enter the dialing dial tone state when you lift up the receiver and a state where you basically get back into when you hang up.

So this is a a em simple diagram showing how you can use the simple behavioral states to define a more complete state machine diagram state machine behavior of a system.

(Refer Slide Time: 21:20)



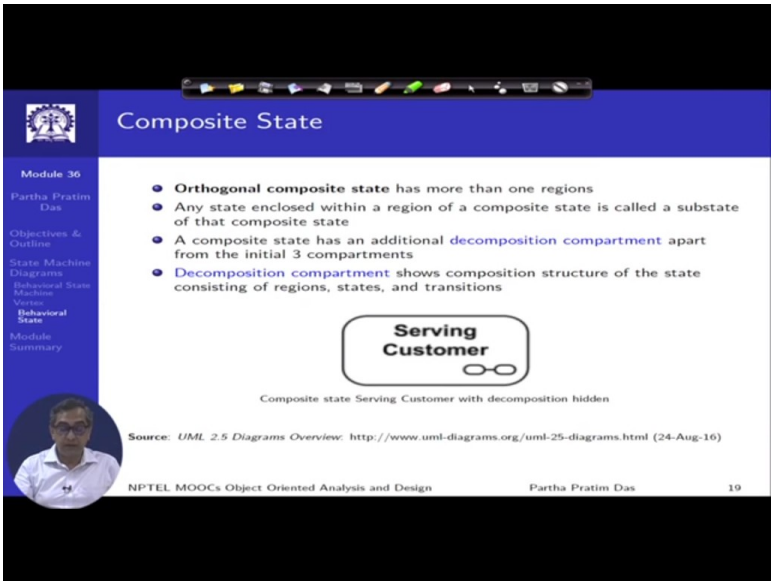
Now often we will find that the if we just want to describe everything in terms of simple state machine then we might find that the whole thing turns out to be quite complex or quite elaborate and we might just want to combine couple of different states and their transitions together into a bigger state which is called the composite state and when you are in a composite state, then all the internal states that comprise the composite state or that build up the composite state is known as the sub state or the nested states.

So here we are showing that this is the composite state which again has a number, has a name the servicing serving customer and serving customer has a customer authentication state and a transaction state and within that again lot of things could be happening but this as a whole is now a state where you say that this is where I enter and this is where I finally exit and I have sub states of customer authentication and customer and doing the transaction.

And this can happen these sub states can happen in a sequential manner in a disjointed manner like it is happening here or it could happen in a concurrent or orthogonal manner. it could happen in terms of what is known as different regions, a region is a in a composite state is a collection of states and

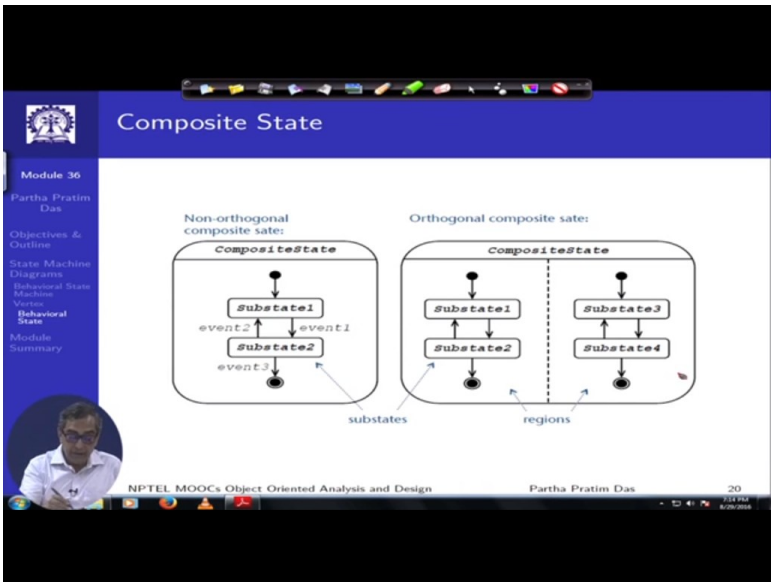
transitions that define one state machine in that part which is giving you that composite state and in terms of these composites and composites of composites and so on, you can very compactly represent a state machine diagrams.

(Refer Slide Time: 23:14)



Now composite states could be orthogonal, they are called orthogonal if there are there has more than one regions that is more than one independent traits of execution that happen in the composite state and when we have that we can actually show the sub states in terms of just a symbol instead of showing the details. in the earlier slides we show they showed the details of serving customer here we just showit as a icon, meaning that there are sub states in this which describes the details of the serving customer.

(Refer Slide Time: 23:48)

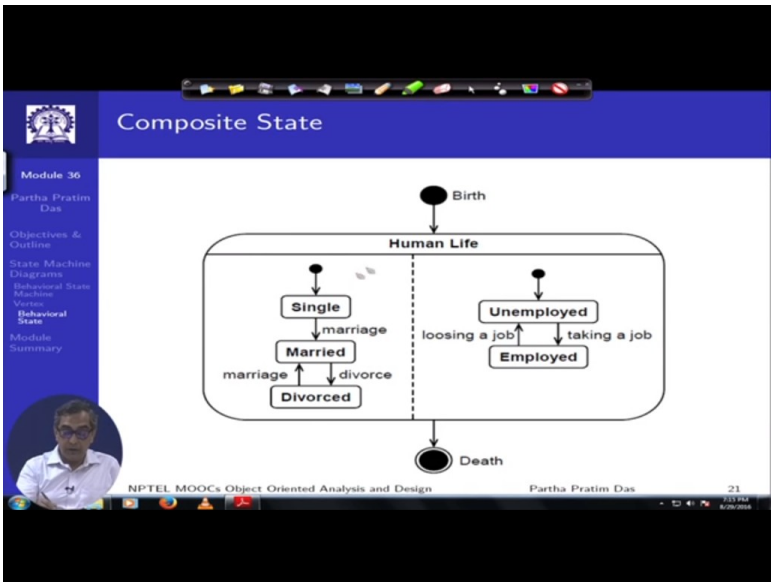


So this is kind of the composite state you might have. This is a non-orthogonal composite state which

means that this has only one region. So this is only one thread of execution, this is where you start and these are your states, on certain event 1 you change from sub state 1 to sub state 2, on event 2 you change back and on event 3 you terminate. Whereas in an in another composite state which is called orthogonal you will have more than one region.

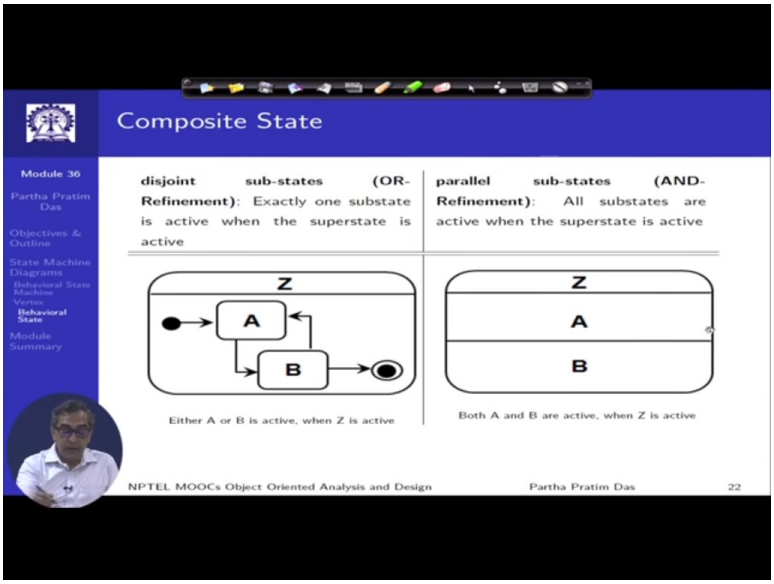
So this is one region and this is another region, these are the 2 different regions and they are typically shown by separate dotted lines and you have sub states in region 1, you have sub states in region 2 and you have events transitions happening between them but these are independent, these are kind of this what is happening here is in concurrence with what is happening here. So in time domain their happening is has concurrent events and when that happens then we say that is an orthogonal composite state.

(Refer Slide Time: 24:51)



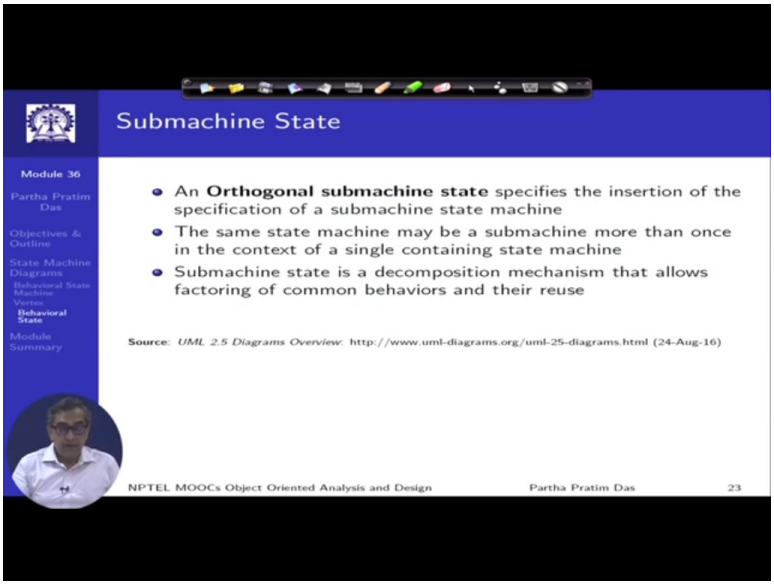
So this is one example of a human life, so these are these are 2 orthogonal 2 regions, 1 region of personal life and 1 is region of professional life and you can have multiple different transition states happening between the sub states of the composite state of the human life single, married, divorced, whereas on the other one unemployed, employed. So these every region will have certain different state of which will act independently on the overall composite object.

(Refer Slide Time: 25:23)



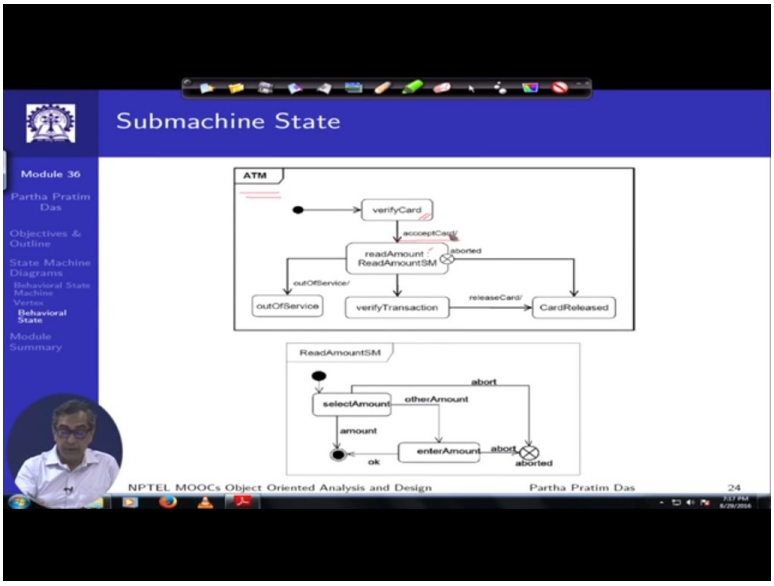
So you could show these in terms of disjointed sub states as in here or you could show them as parallel sub states depending on if they are shown like this they are disjointed, if they are shown as parallel, so they are like different regions and they happened concurrently. So these are the different notations that you need there.

(Refer Slide Time: 25:43)



Finally a third kind of so in terms of a behavioral state, we have simple state, we have seen composite state; we have seen simple composite state which has only one region, a non-orthogonal composite state and we have seen orthogonal composite state. The another that is commonly used in the representation is called submachine states. So you can define sub machines of a machine and actually define properties based on that. So submachines so let us try to look at an example that is easier to understand.

(Refer Slide Time: 26:22)



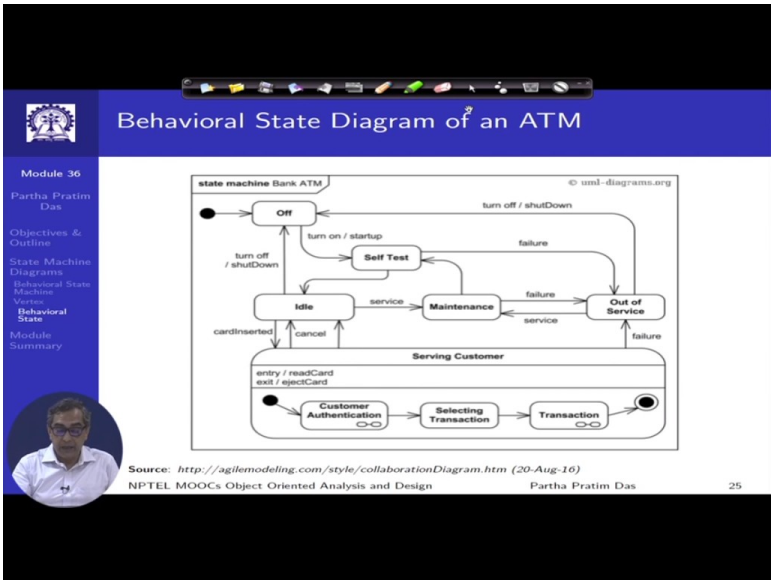
Say we look at the atm itself, so the states are you verify card and once that is accepted, the card is verified then you read the amount and do the transaction and the outcome could be that the transaction is aborted where you moved move to card released there may something goes wrong we will go to out of service or you actually succeed so if the transaction is verified and then the card is released for card release. So this reading amount and actually you know doing the all those balance check.

And all that is shown as one state but is actually another state machine itself. So the we are we mark this as read amount sm am just writing sm to indicate that sm is submachine, so read amount sm we will expect that there will be another state machine diagram which gives the details which says how to select the so when you are accepting accepted the card then you are here and you have select the amount then you have checked the amount is okay, you enter amount and all that.

And at the end you could either remain reach the final state which is success state which is here or you could you reach the abort state which is here. So sub machines are very frequently used so that a so if they are like the sub machines are like function calls, they are like subroutine calls, so we are saying that here I have a submachine and this same submachine could be used in multiple other finite state machine diagrams as well behavioral state machine diagrams as well.

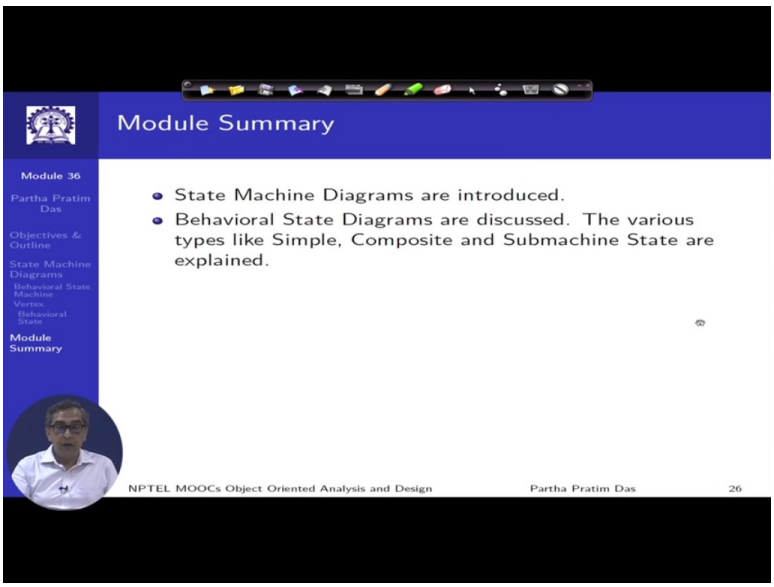
And they all will by the name refer to the same submachine where you are. So the submachine state or states which basically have complete submachine diagram incorporated in terms of it.

(Refer Slide Time: 28:10)



So given this is behavioral state diagram of an atm. I will not go through the details, we have just briefly discus this and I will request that you please go through each of these state transitions and get comfortable in terms of describing a atm system to this kind of an fsm.

(Refer Slide Time: 28:31)



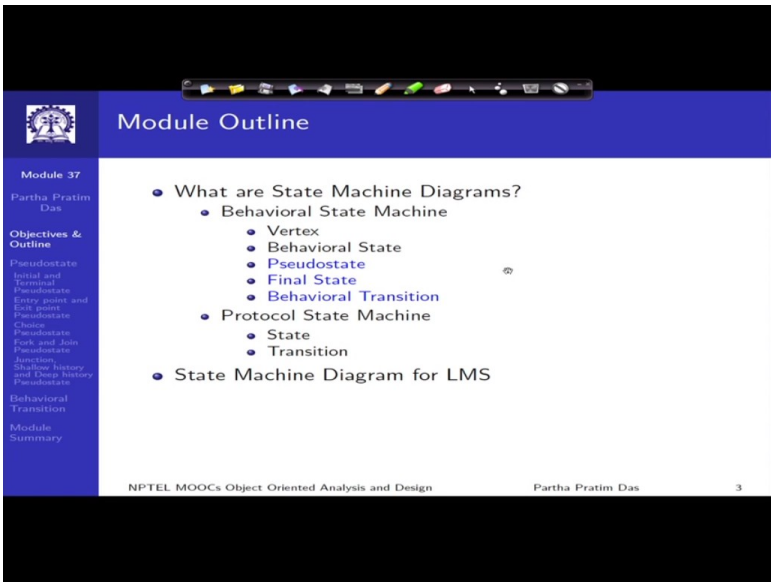
So to summarize state machine diagrams had been introduced, we have talked about what a state machine diagrams are and we have given an overview in terms of a behavioral state machine diagrams particularly talking about the vertex and the different behavioral states, the simple states, the composite states and the submachine state. In the next module we will continue discussing about other kinds of states in a state machine diagram.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 49
State Machine Diagrams: Part II

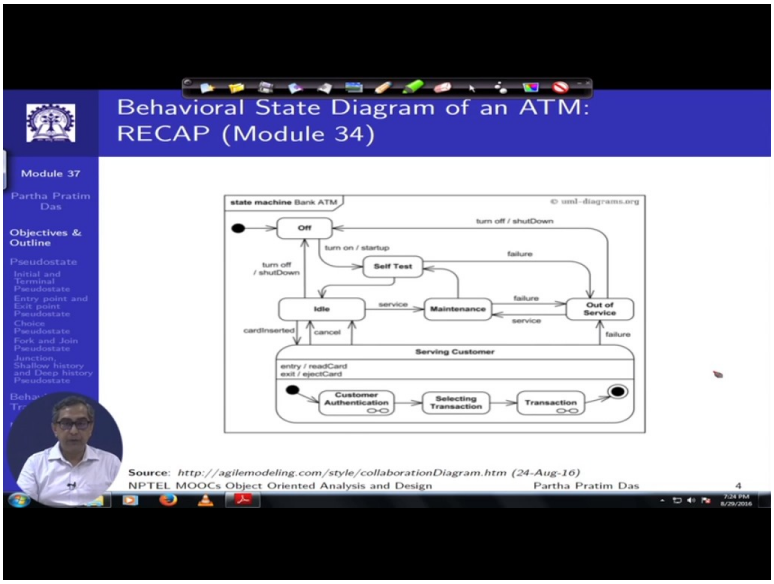
Welcome to module 37 of object-oriented analysis and design. From the last module we have been discussing about state machine diagrams. These are behavioral diagrams used to capture the discrete behavior of a complex system at an appropriate level.

(Refer Slide Time: 00:49)



And we have in this outline, we have already covered the vertex and behavioral state of a state machine diagram and where in we have seen that a state machine diagram could be of behavioral state nature or it could be of a protocol state machine nature. Protocol state machine we have not discussed as yet. In terms of the behavioral state machine we have seen that there could be different behavioral states: simple, composite which comprise lot of sub states in one region or in multiple regions, orthogonally where which happen concurrently or it could have sub machine states.

(Refer Slide Time: 01:44)

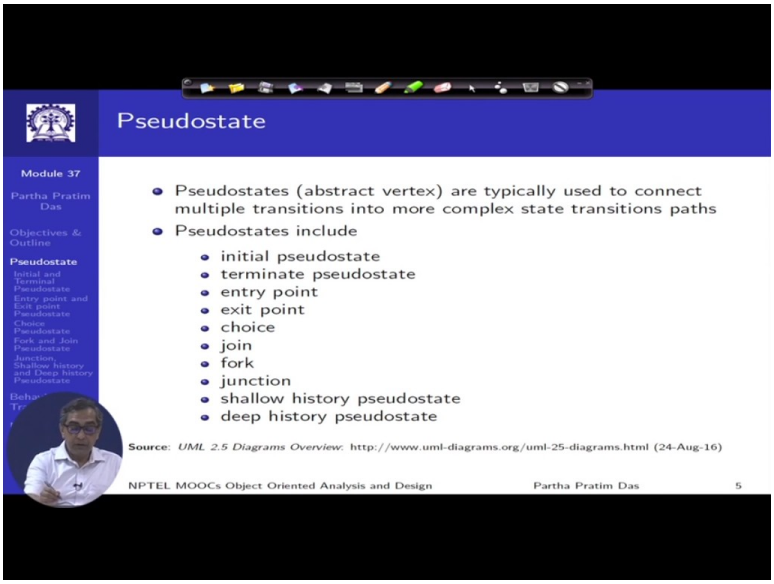


In this context we will next introduce the concept of pseudo states and talk about the behavioral transitions in this model. So umm this is the state machine diagram of an ATM that we closed the earlier module which and I expect that you have already tried to understand and work through this. So in this, what we can see that this is the start and this is the final end termination of the state machine diagram.

So this is when it is in a state off and you can turn it you can turn it on and get into a that gets into a self test mode and then if it is if its tested ok then it gets into the idle state where it will remain there till somebody inserts the card, where it gets into this serving customer state which is as you can see it is a composite state but it is a non orthogonal composite state where simple composite state because it has only one region and the operations here in are sequential in nature and here in you have other composite states.

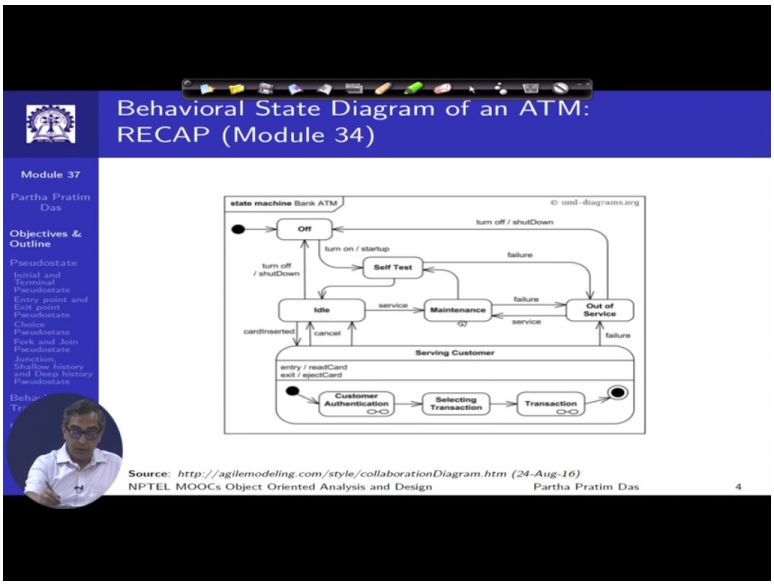
These symbols show you there are other composite states which will have you have further detailing of the customer authentication and the transaction kind of state behavior that is being talked of. This composite state has entry and exit as readCard and ejectCard, that is on entry the action is readCard, on exit the action is ejectCard and the overall action of this activity of this composite state is what is shown through this. And there are several exits if you cancel this comes back to idle, if it fails it goes to out of service, there would be different transitions between maintenance, out of service, self test and so on. So this mode comprehensively trys to define the state transitions of a of an ATM system at a behavioral level.

(Refer Slide Time: 03:49)



So moving on we are talking about pseudostates or abstract vertex. The idea is very simple; we are talking about state transition or state machine diagram. So these are these are typically how the states are drawn and you have transitions between them. The transitions may have some event, some action with the transition to happen. So these we have seen as they are called the different types of behavioral type and so on. But if you look at the whole diagram as a graph then you will find that just by using this state symbols you will not be able to define the whole diagram.

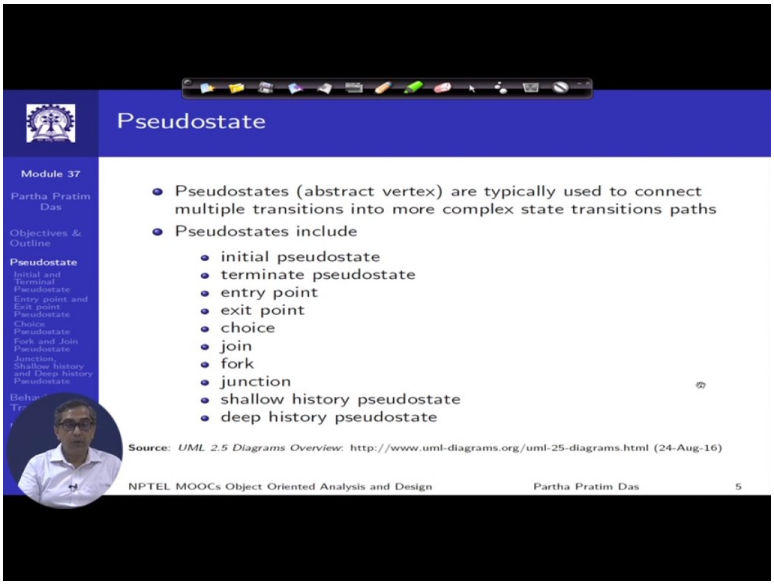
(Refer Slide Time: 04:42)



So for example if you just go back and take a look into this state machine diagram, these are these are all different behavioural states. But with their transition alone we cannot define the whole system because we need for example disc mode as well because whatever be whatever your beginning there have to be something where there is no state then your whole system will start. There will have to be

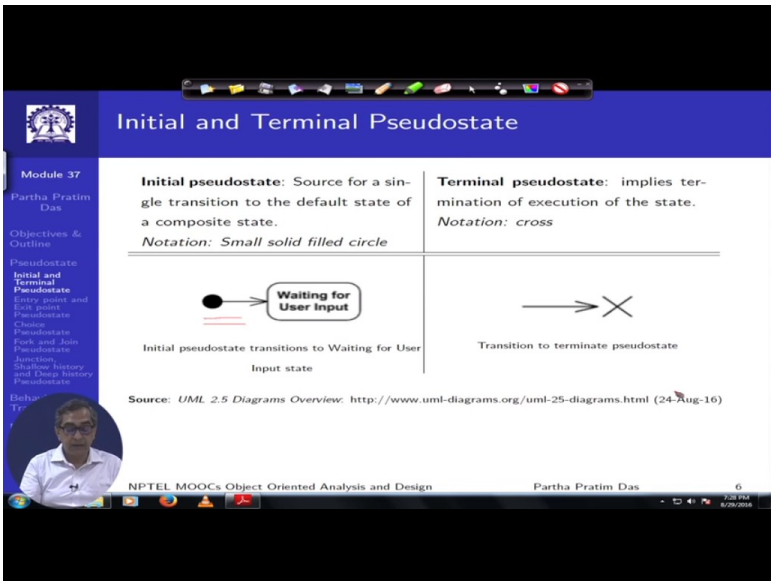
something where beyond which there is no further state and so on. So these kinds of states which need to be there as vertices in the state machine diagram but do not represent the behavioral state are typically called the psuedostates.

(Refer Slide Time: 05:30)



And we will say that psuedostates also carry a lot of information and there are as you can see the list, there are whole lots of pseudostates which not only define certain resting place for the transitions but they also define certain different ways that the transitions can actually get composed.

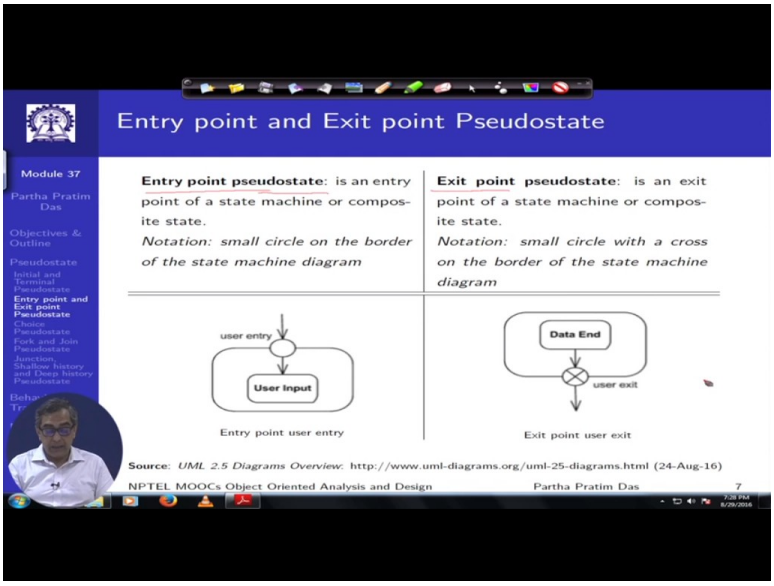
(Refer Slide Time: 05:48)



So initial pseudostate is what we have we have seen this is where you get started and this simply shown as a blog and a terminal pseudostate is beyond which you cannot proceed further. So implies the termination of the execution of these states and this is not seen as a final state of the state machine that

is the different thing. This is just termination of the execution so which means that in terms of the transition there is no further transition possibly in that, but the state machine is still continued to execute and so on. So these are the first simplest kind of pseudostates.

(Refer Slide Time: 06:30)

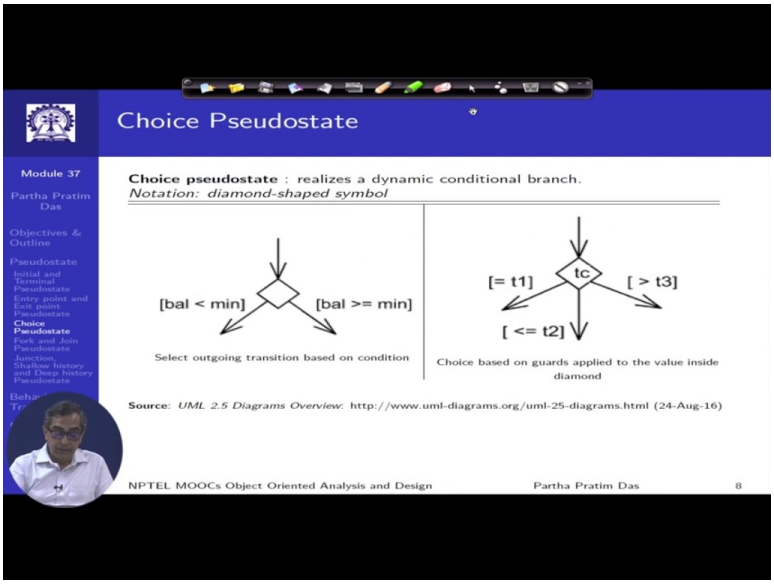


Then you have entry point and exit point pseudostates. So the entry point and the exit point pseudostates are this is a state and you say that this is a user entry, we talk about entry and exit of states. So this is the user entry that is happening here. So with this symbol you show that at this point for the state to actually make transition it needs to take some input, some entry from the user. Similarly there are these are this is another state and this symbol show that a user exit.

And this cross shows is an exit point of a state machine of the composite state and this is the point through which it will go out. So these are basically entry point and exit point pseudostates are basically more meaningful if you have a submission state or a composite state where you show by this entry point where your entry composite state and or the submission state or where you leave go out of the submission state or the composite state. Mind you these are not same as the initial.

And final state but the state machine which are basically true for the whole machine here you are talking about entry and exit from a composite state which is a part of this state.

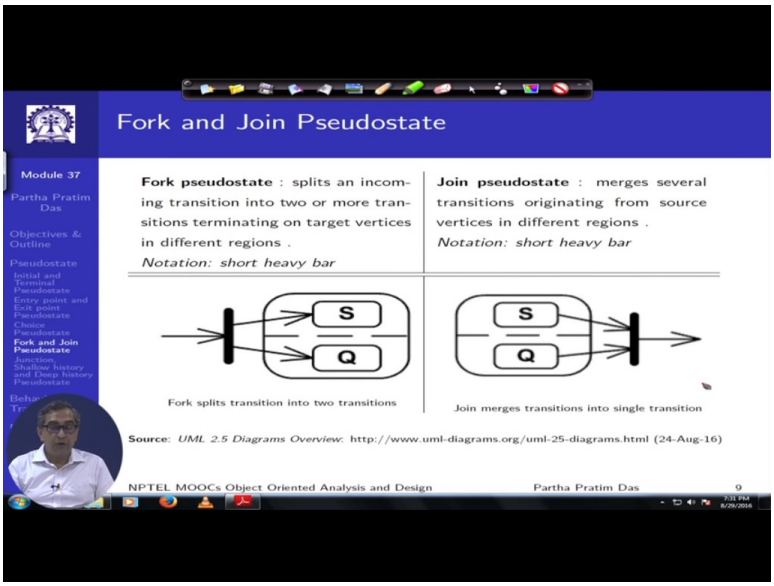
(Refer Slide Time: 07:53)



There are choice pseudo states which are basically like decision parts, so in terms of a transition it could happen that based on certain condition which are the guard condition and this will look very similar to the kind of decision nodes and kind of merge nodes and all that we did for the activity diagram but there they mean some I mean they are semantically similar but they mean different things.

Here you are saying that these are pseudo states so that if a transition is suppose to happen on this part depending on this condition will either happen on this part or happen on this part. So that is the choice that you are making. And the choice need not be binary; this could be multiple choices as well. And such vertices that happen in the state machine graph is known as choice pseudostates.

(Refer Slide Time: 08:57)



Then as we know that there could be concurrent access, there could be concurrent execution. So so like

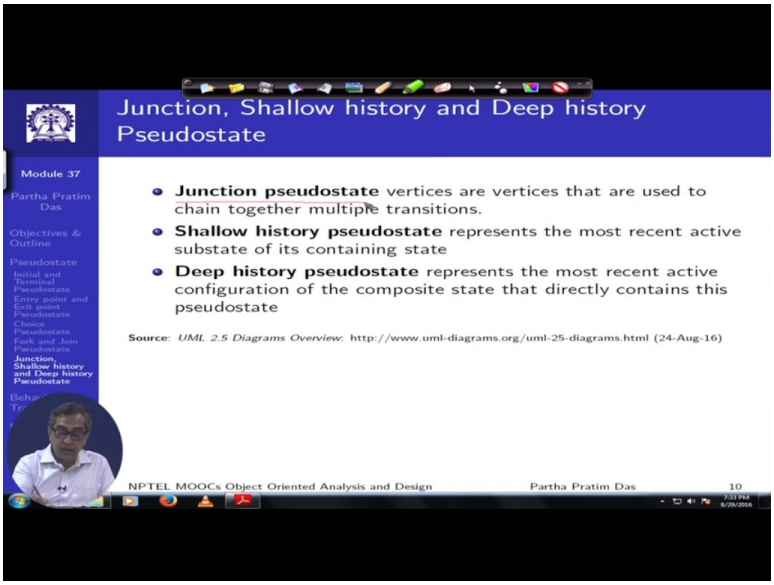
the fork node and the join node we had in the activity diagram, we have fork and join pseudostates in the state machine diagram. So this fork in so this what this say that if this is a fork pseudostate so if splits a transition incoming transition into 2 or more transitions terminating on thea target vertices in a different regions.

So what did we specify? we said that if we have a composite state with more than one region say as we see here this is a this whole thing is a composite state, this is one region, this is another region shown by separated by this dotted separator and this is the whole composite state, these are the sub states, S and Q are the sub states of the composite state and they are into 2 different regions. So if we want to show that we in transitions happen into both of these regions together.

So it is like the transition is happening on this and it happens on the sss sub states of region 1 and sub state of region 2 together concurrently, then we say we have a forking transition or we have a fork pseudostate. Similarly if we have multiple regions we need a merge the different regions the transitions, common transitions. So this is this was region 1, this is region 2 and they are merged together, this is again pretty much like the merge node of activity diagram. So they are merged together in the transitions into a single transition, so beyond this point it becomes a single transition.

So these are also nodes states in a state machine diagram but these are pseudostates required for expressing the concurrency in terms of the transition.

(Refer Slide Time: 11:05)

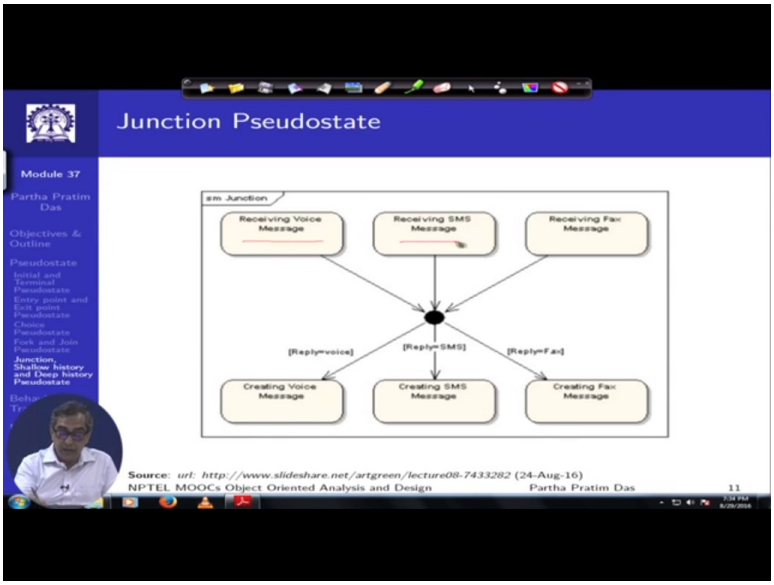


We have a couple of other pseudostates as well, one is called the junction pseudostate. A junction

pseudo state is where you create a pseudo state to do a confluence of multiple different transitions at one point and so they merge and they diverge together and so on. And you have shallow history pseduostate and deep history pseudostate is basically give some composite state is put on hold in terms of its execution.

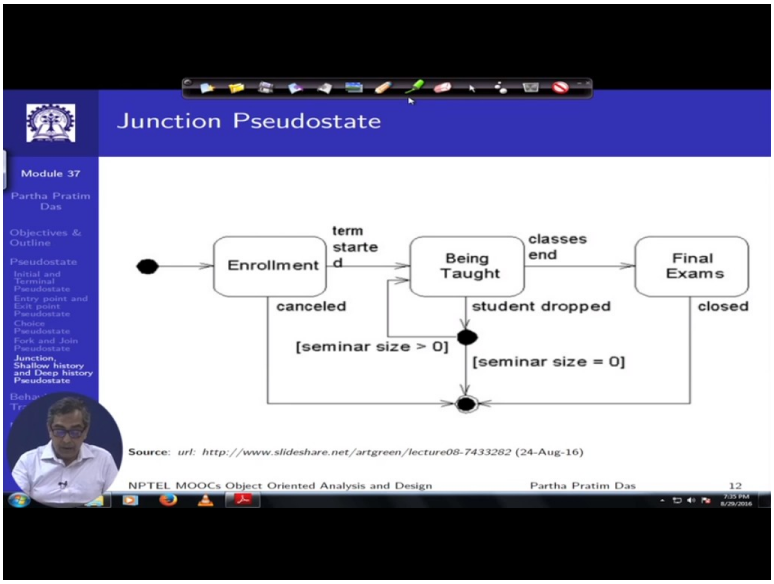
And then the execution is resumed at a later point of time it depends on how what whether it will resume from the most recent activity or it will resume from the most recent configuration depending on that you have 2 different kinds of this pseudostates and those are also part of state machine diagram.

(Refer Slide Time: 12:07)



So here they have a join junction pseudostate where this is receiving a voice message, receiving sms message, this is receiving a text message. So on this on junction the transition is on junction pseudostate up to which you will basically get into different transitions of depending on whether they are replying in voice, they replying on sms or replying as fax. So whenever you have such confluence message you say you have a junction pseudostate.

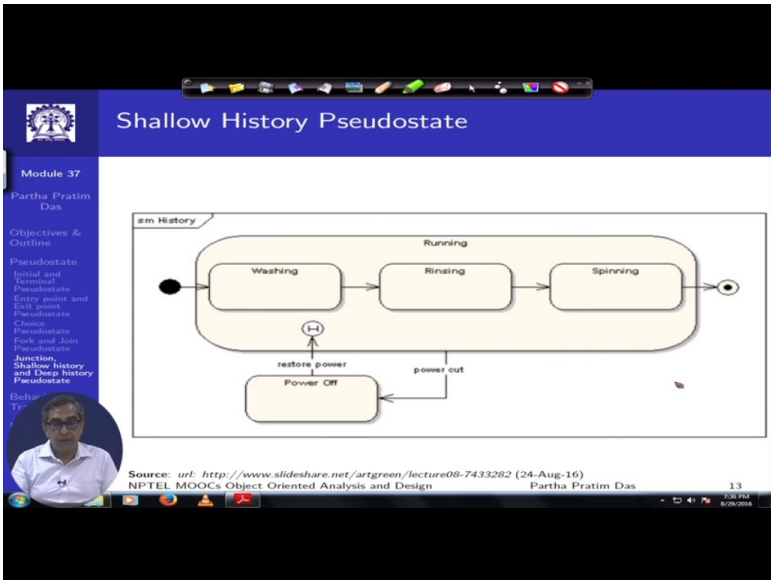
(Refer Slide Time: 12:45)



So this is another example. So this is here we are showing process of enrolment into a semester. So this is the start, initial node and this is the sub state enrollment here, the term started. So then you make transition to being taught and if the enrollment is cancelled then you reach here and here this this classes end you go for the final exam otherwise the student might drop out and then you you can if if for that the seminar has a size zero that is the class size becomes zero then reach a termination otherwise this greater than zero.

There are more students even after this student has dropped, there are more students when you come back and continue teaching. So at this node, you can you can easily see that the junction is taking place, junction is taking place here which also it merges to the final state final pseudostate of the state machine diagram. So junction is a nice representation for the pseudostates.

(Refer Slide Time: 14:00)



Here is we are showing some shallow history pseudostate. I will not go through, there are couple of examples that i have put in. I am not going to detailing on the discussion of them I would expect you to really read through and understand them more. So here we are just showing that there is a history, so there is a washing machine state transition state machine diagram for a washing machine. So it is in a washing state, then it is a rinsing state where it puts in more

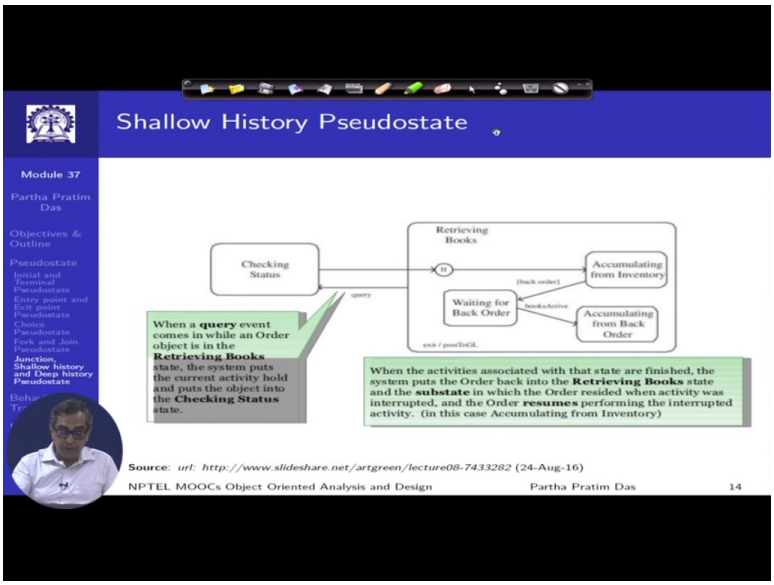
And more water and cleans your clothes and then finally spinning state finally take away water from the clothes and so this is a initial state, this is the final pseudo state. And now now suppose if if there is a power cut there is a power is put off then it goes to the power off state and then when you restore power it what you are showing that you need restore power you restore power to get back to a shallow history pseudostate which means that just think about the practical situation, if the power could have got disconnected when you are in washing state.

If that happens then when the power comes back you want the machine to restart in the washing state itself. You do not want the machine to be spinning because the rinsing has not happened. Whereas when the power went off if you are in the rinsing state you would like the machine to restart in the rinsing state otherwise if you are in the spinning then you would like to be started in spinning state. So this is the significance of history. So that is what you mean by history. So you remember try to remember the shallow history.

The shallow history is about the last active sub state that you were in and so when you had to be suspended because the power has cut off and when you resume after the power has been restored, you

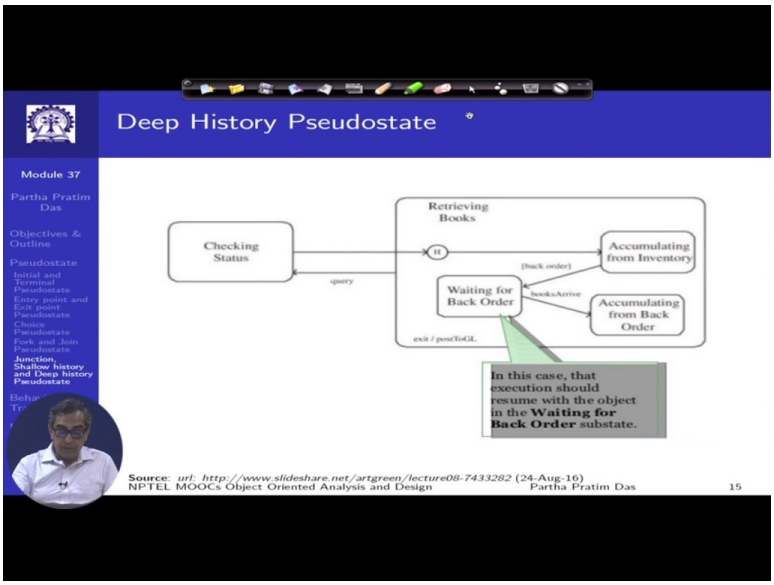
restart in the last activity that you had been in, that is the shallow history that you have. In contrast you could have a deep history pseduostate which will not only depend on the last sub state but it might depend on the last sequence of sub states that you have transited. So I have put in some examples I will please go through them.

(Refer Slide Time: 16:32)



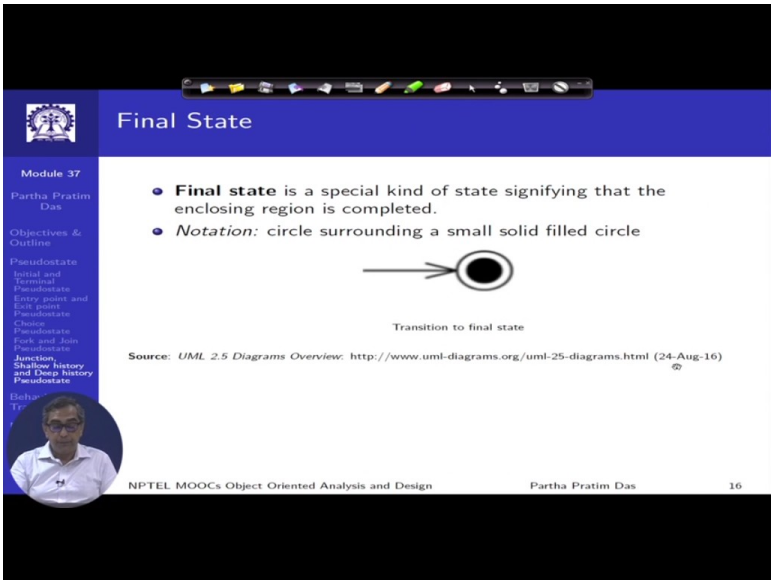
This is another example of shallow history pseudostate.

(Refer Slide Time: 16:38)



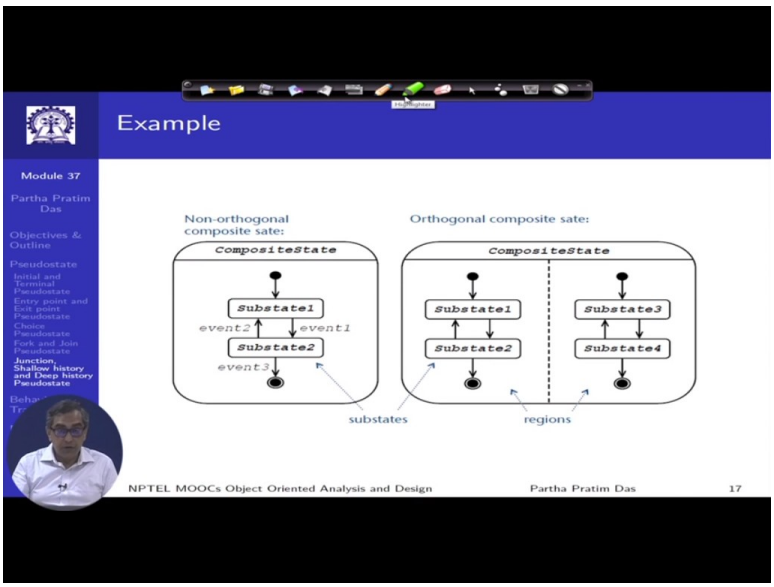
This is an example of deep history pseudostate. So from the presentation please go through them and try to understand them better.

(Refer Slide Time: 16:45)



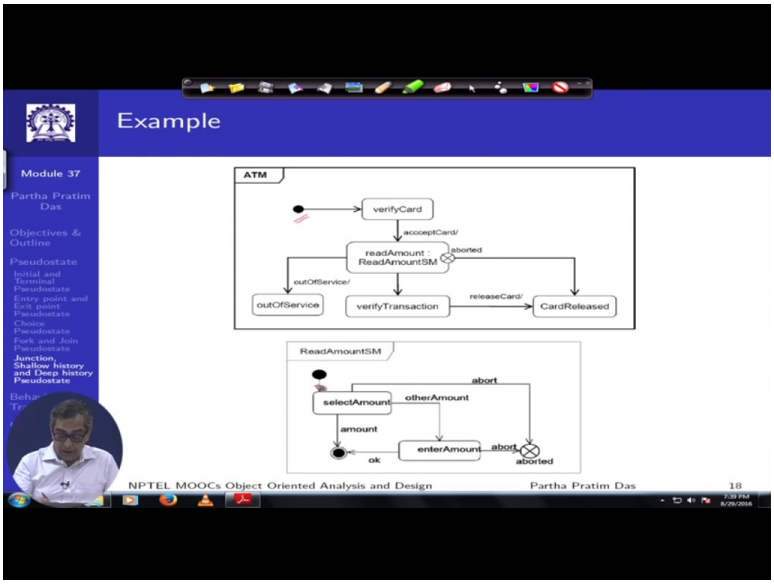
Now finally the there is a pseduostate called the final state which you have already seen in a number of places and once you reach this state then you know that the state machine diagram has actually come to a completion and there is no further state transitions to happen for this state machine diagram.

(Refer Slide Time: 17:08)



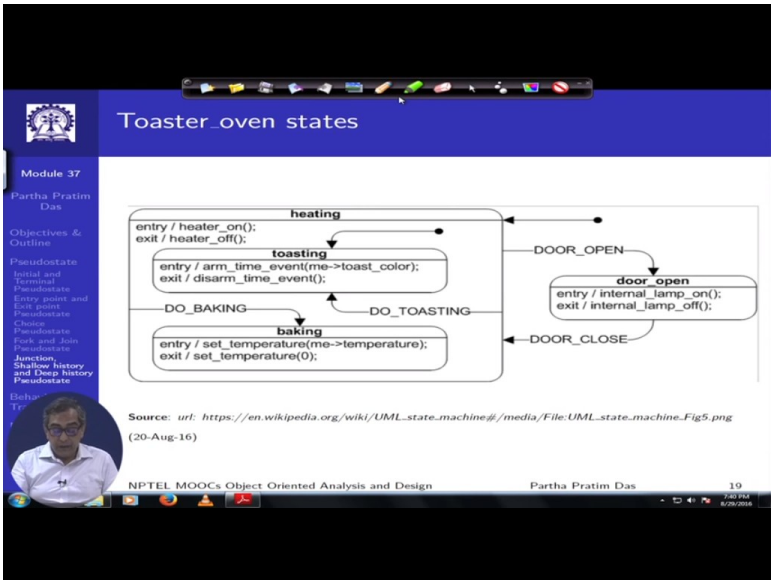
So these are some examples showing you different pseudostate. So this is this is the initial pseudostate, this is the final pseudo state which we have been seen quiet regularly.

(Refer Slide Time: 17:23)



Then if we see here this is again the initial, the initial, the final. This is this you can say is an exit pseudostate. So this is the exit pseudostate which takes you out of the submission diagram.

(Refer Slide Time: 17:44)



This is using this this is another toaster oven example. So you can see that, so what are the, if we identify first the simple behavioral states, then this is the simple behavioral state, this is the simple behavioral state, this is the simple behavioral state. These states do not have any other do not need other states to work. Each one of them some have some entry and exit like toasting is entry is arm time event that you said that this is the kind of toast color that I want.

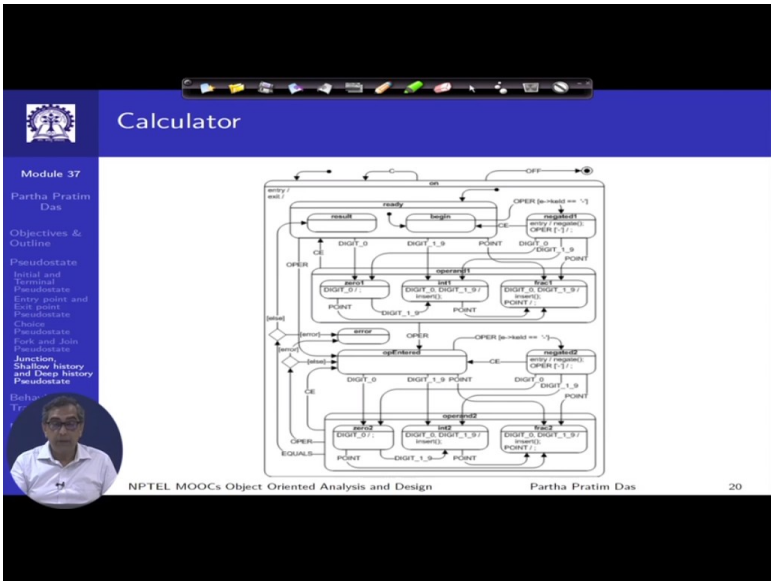
And exit is some kind of a, that is the toast has happened for the specified period of time. So the timing timer is disengaged then the toasting actually stops. Then you have the entry for door open which is

internal lamp being on and the exit is internal lamp being off. That is if you o open the toaster then the internal lamp will get on so that you can look inside and when you put the door closed then this will become off. So these are the entry exit points.

For the baking part, you will have a set temperature for entry and you will have set temperature to default possibly you non baking temperature on exit. So these are your simple states and then you can see that these simple states have combined into a composite state of heating which has the entry heating on and heating off and for this composite state this is the start, of the of the whole state transition process which starts with toasting starts am sorry this is the start which brings it here.

So in this if you have the instruction to perform toasting then this is where you do toasting do and start on this. If you have instructed to do baking then you start with this simple state machine and but when you get a door open event you go to go outside of this composite state and do the door open and once the door is closed you come back to the composite state. So this shows using the composite states and the pseudostates I will describe different state machines involves here.

(Refer Slide Time: 20:21)



This is the little bit more you know detailed example but it is a very simple example I find this to be an interesting one; this is basically trying to describe the calculator that we use. So this is like the calculator so if you look, so what is a calculator is, where we can put in so I want to do 35 plus 63 equal to 98. So what we do, I first enter these digits which is basically operand 1,then I enter the operator, then I enter these digits which is operand 2,then I enter equal to, then I get result.

Then I can again enter another operator, enter an operand, get the result and so on. So this behavior we are trying to describe in terms of a finite machine diagram. So if you if you look into that then certainly you need the sub machines or composite states which allow you to enter operands. So you will find operand 1 composite state here. So which can either start and that has different simple states, 0, int and fraction, 0 is just when you enter a 0.

And int is when you enter with starting with something like 1 to 9 and then you enter any of 0 or 1 to 9 and fraction is a decimal number. So if you enter a point you get it. So if you get the 0, you are in this simple state, if you get 1 to 9, you are in this simple state, if you get a point you are in this simple state. In between if you get you are in this simple state you get a digit 1 to 9, you come to this and you continue to remain in that and if you get a point you get to this state.

So this if if you look into this operand 1 composite state that basically models how you enter the different digits of an operand. So you will have an operand 2 also which models how you enter different digits of the second operand. So if we so having said that if we rewrite this again, so the overall action needs that I will enter the operand 1 which is basically doing this and then I will enter the operator.

So this composite state has a transition on the operator which is basically another simple state which is operator entered and then from that if this is entered and from that you have again transitions which take you to the second composite state of taking in the next second operand. And once you press equal to key which is basically saying that you evaluate and on equal to goes and it checks we are if there is an error, it gives you an error.

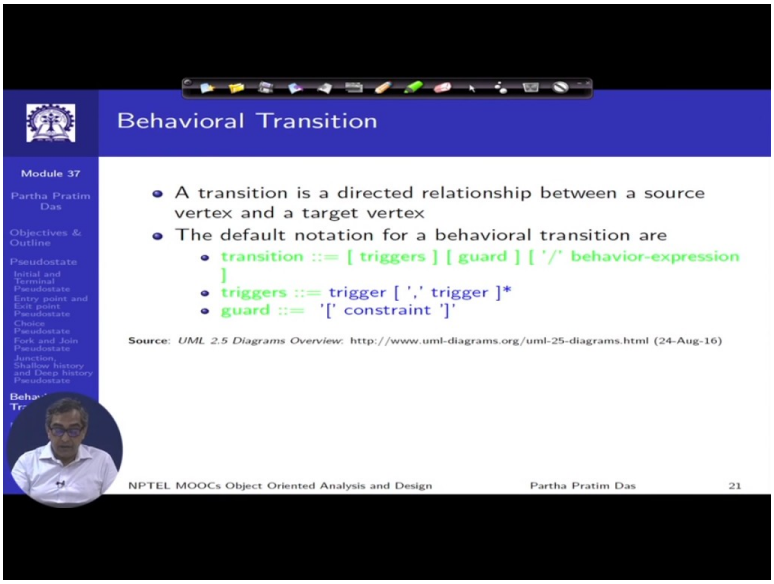
For example if you have given say suppose you have given this then this gives error,so it will go to an error and go to this simple error state. Otherwise it goes to the result state where it shows the result. and so shows you 98 here as a result and on this again you could you could actually start all over and enter a new expression or you could give an operator you could say plus 2, if you give an operator it comes to the operator simple state and then it enters, then you can enter the second operand again, the first operand by default with the results itself.

And then this way you can, there are there are further details and there are details in terms of what happens if you do see, clear entry naturally it will go back to the ready state of the calculator if you do

clear the whole thing you reset the whole operations back here and you can see that a operations of a calculator can be very nicely described in terms of a state machine diagram in this manner using simple states and composite states and transitions and different pseudostates you have this pseudostates, you have these pseudostates, here you have basically the choice states which tell you which will go and so on.

So this is just , am just am just trying to give you different flavors of examples to understand that the state machine diagrams are really powerful to capture the dynamic behavior, discrete behavior of a system in a very simple manner.

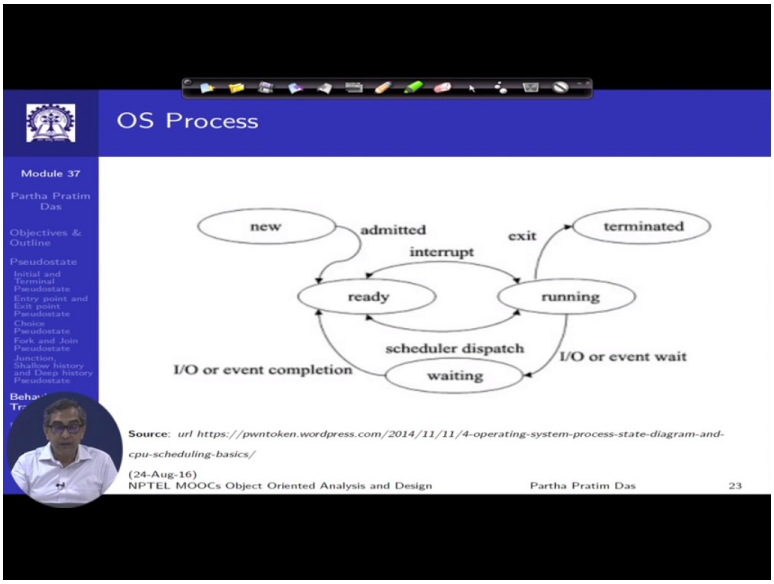
(Refer Slide Time: 25:37)



Besides the pseudostates you have the behavioral transition. The transition is a directed relationship between a source vertex and a target vertex which we have already seen. So these are the different transitions. So formally these are called behavioral transitions where this is the source vertex and this is the target vertex, for this transition this is the source vertex, this is the target vertex and the transitions may have certain conditions, start conditions based on which the transition takes place.

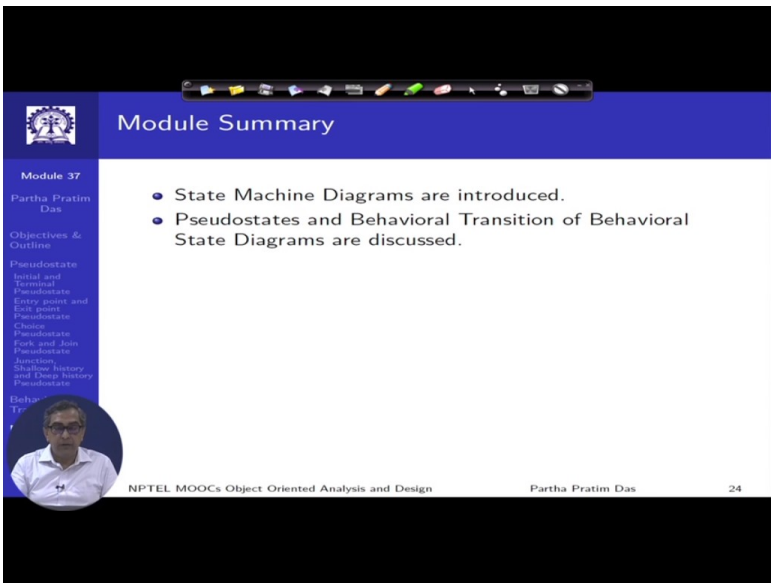
And accordingly say that if I mean am in state maintenance and a failure has happen, this event has happen then I take this behavioral transition into an other behavioral state, out of service. So by this we can complete the state machine diagram as on whole.

(Refer Slide Time: 26:33)



Just as an example I just put up a very different state machine which is which otherwise should be very known to all of you having computer science background, this is the typical process state machine of an operating system. So the same representation will also work for the here.

(Refer Slide Time: 26:53)



So to summarize we have been talking continuing to discuss about the state machine diagrams and then in this module beyond the behavioral states, we have talked about several kinds of pseduostates and explained different examples and also talked about the behavioral transitions and we will continue in the next module talk about the protocol state machines.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

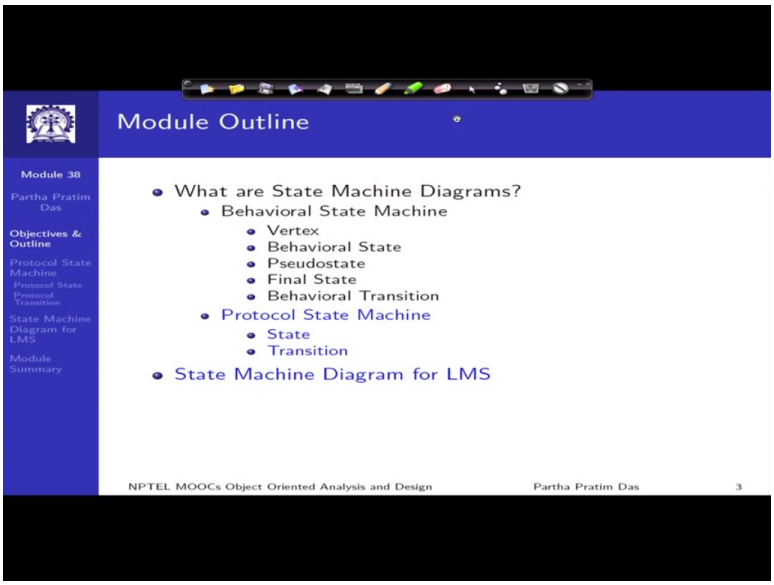
Lecture – 50
State Machine Diagrams: Part III

Welcome to module 38 of object-oriented analysis and design. From the last two modules we have been discussing about state machine diagrams. These are uml behavioral diagrams which are very important in refining and capturing the discrete behavior of a system in terms of a set of finite states and a state here is a at a higher level of abstraction than the state as we refer to them in terms of a finite state machine or a regular automata.

Here the state behave are often called behavioral in the sense that the states may have certain entry activity, certain exit activity and internal activities and transitions and we have seen that there are variety of behavioral states starting from the simple state to composite state and submission state and composite state states in turn could be orthogonal or non-orthogonal, that is they could be just sequential having one region or they could have multiple regions.

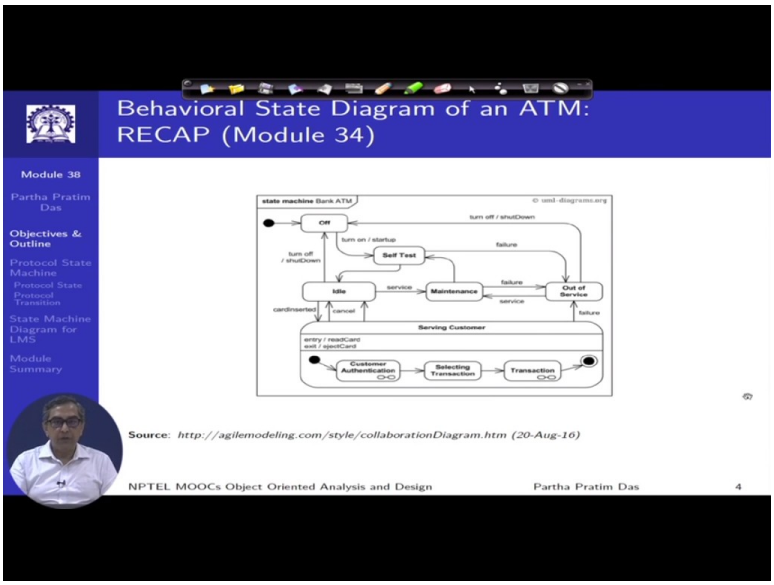
And we have also seen that the state machine diagrams so described need variety of other kinds of nodes for its complete description and these kinds of node known as the pseudo state nodes which include initial final kind of node and in addition it has for, join this kind of node as well which or choice nodes which defines different kinds of control transitions in the whole state machine process.

(Refer Slide Time: 02:07)



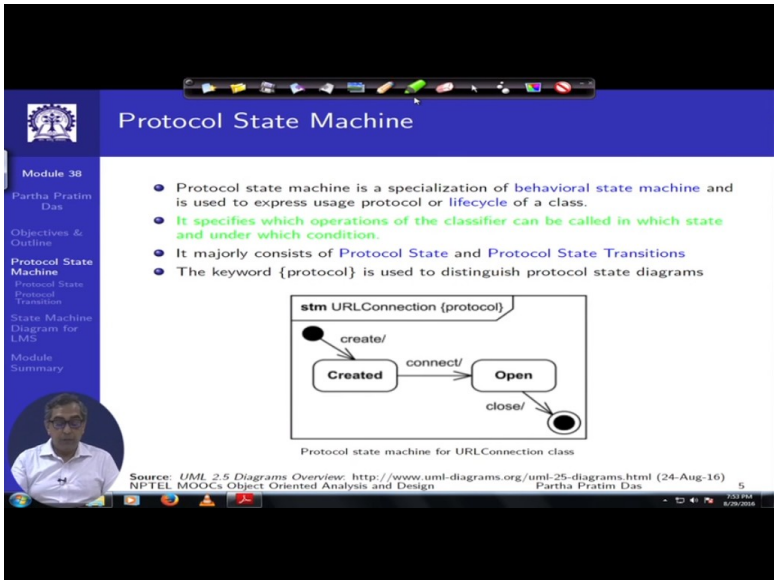
So will continue on that. This will be the last in the series of discussions on the state machine diagram. So will talk about the protocol state machine at the end we will briefly take a look into the leave management system example we have been discussing often and discover, try to discover the state machine in depth.

(Refer Slide Time: 02:30)



So this is just to recap that this is one of the behavioral state diagrams of an atm that we have done and am sure you have understood this well and if you not please go through that again.

(Refer Slide Time: 02:44)



Now first about protocol state machine. Protocol state machines are a specializations of the behavioral state and there they are used to express the usage protocol or lifecycle of a class. so let us understand what was a behavioral state machine. Behavioral state machine if we just take in the very simple form I

have a state s1 and I have a another state s2, these are behavioral in nature. So what we do? We have for this I have an entry, for this I also have an exit present here and I have some internal activity.

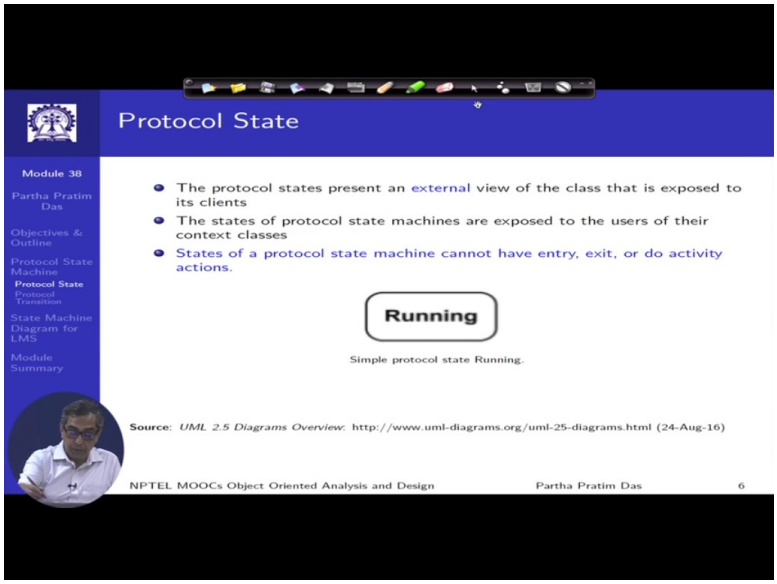
So they are behavioral states are kind of at a higher level of abstraction defining not only defining the overall conditions in which the system exits. But in terms of our modeling the system at a micro level may be going through several changes because of the do activities that is happening in that state but overall we see that the system is in single state. So we have seen several examples. But here in we in terms of the protocol state, we are primarily interested to express the usage protocol in terms of the lifecycle of a class that is in protocol state machine.

We will not have the entry, exit or do activity of a state. a state is more a designated of what is the condition on the lifecycle of an object of a class that the whole process is currently behaving. So and we will see that actually when we talk about the variety of different state machine diagrams, quite a significant part of the state machine is actually protocol state machine. It just talks about the lifecycle transitions without having further activity details amongst it.

When we represent a protocol state machine we usually use engage a keyword protocol written in the curly braces which designates that this is a protocol state machine. So this is like before this symbols are same this is a initial pseudo state, this is the final pseudo state final state and these are the protocol states. So the when you once you say these are the protocol state what you mean is these will not have entry, exit or activities within them. It just designates that this has happened. so I could actually model starting with a protocol state diagram.

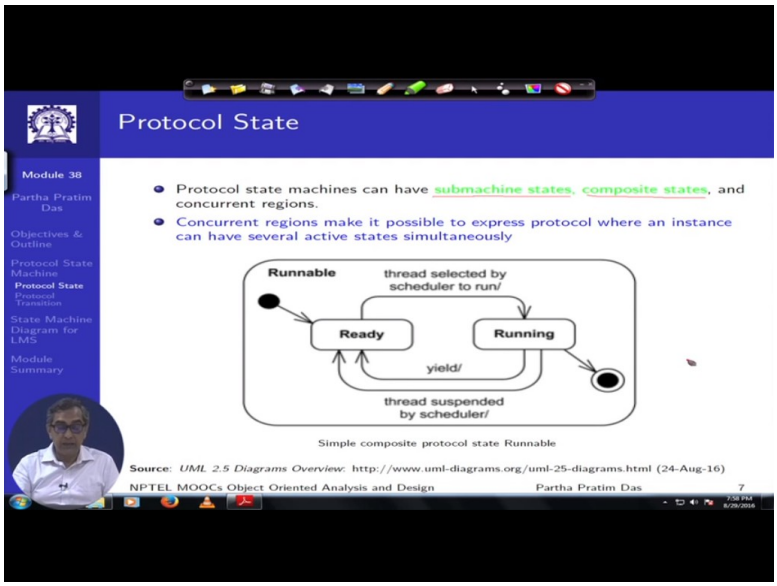
And then when we get more information we could elevate some of this protocol state actually behavioral state by adding all those behaviors. But first that we need to identify is the distinct configurations into which a system should exist so far as our modeling is concerned. So we saying in terms of url connection there is a created that is you are just beginning the process and then if the connect event happens, you have been able to connect with that url and open you are in open state, you are checking that url and when you close that browser, you have a close event it takes you to the final state.

(Refer Slide Time: 06:46)



So protocol states represent an external view of the class that is exposed to its clients that is whatever the client is doing with that class, you will be able to see that in terms of the protocol state. The states of a protocol state machine do not have this we have already mentioned. So a simple this is a protocol state a simple protocol state running, so protocol states will only have the names represent.

(Refer Slide Time: 07:17)



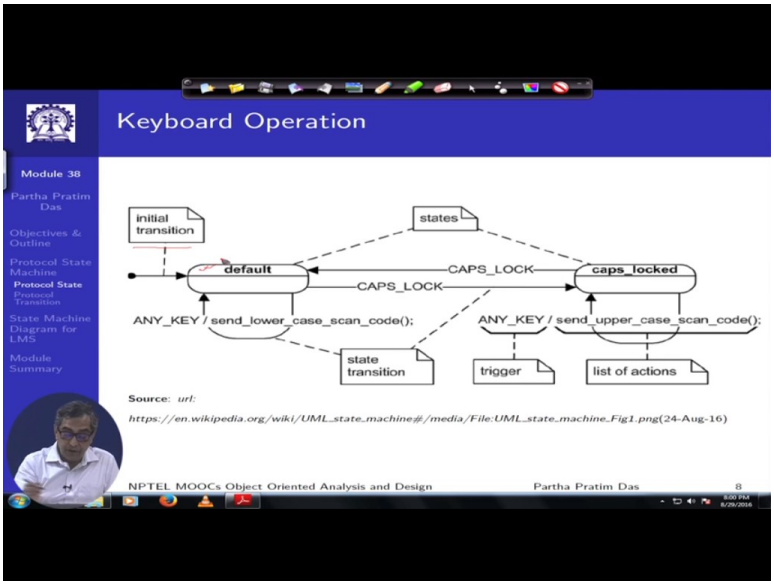
So in terms of that the protocol states also can have submission states or composite state and concurrent regions. The concurrent regions make it possible to express protocols where an instance can have several active states simultaneously. So the difference with the specializations with respect to the behavioral state diagram is in the fact that protocol states are just place holder of configuration but they do not have a behavior often.

But otherwise you can have a composite state at this for example it is been shown here is this is composite state runnable and within that we have 2 protocol states sub states, ready and running. so this is part of as you can understand, this is part of the process state transition diagram that we typically draw as a finite state machine here we are drawing it as a state machine uml. So you say that these are the sub states ready and running sub protocol states.

And so starting from the initial it could be in ready and if a from the ready if a particular thread is picked up by the scheduler to run, you get the run event and your state changes to running. And then when the thread is suspended because you have possibly run out of your time slot then this is suspended and you get back to the state ready from the state running or the process might yield because a higher priority process has come in so the process will yield and come back here.

So we can see that, likely we have seen in terms of the behavioral state machine here too we have composite state and here too we have sub states and so on. Similarly, we can have submission states and we can have concurrent region where the protocols run in in parallel in in concurrent so that we have multiple states being assumed at multiple different points in the dynamic system.

(Refer Slide Time: 09:32)

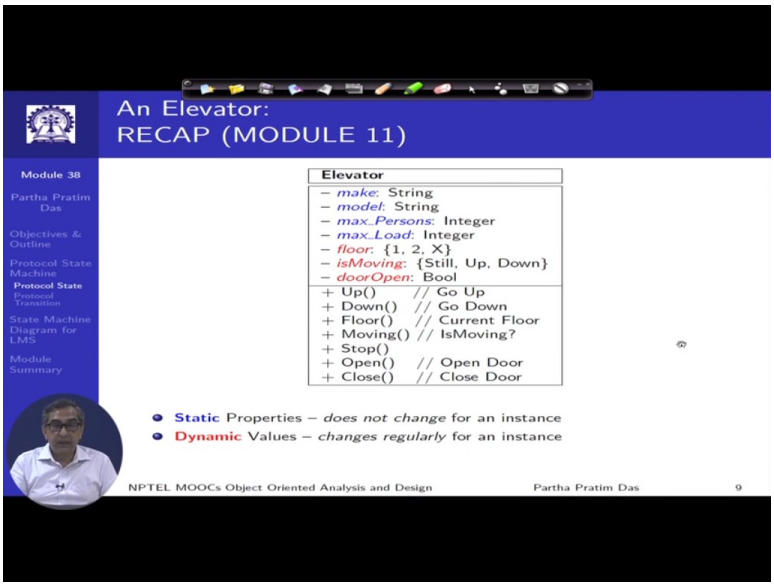


This is just a another example show you showing you a keyboard operation. So this is the initial transition, these are the different protocol states and so by default so these are the states that we can see and by default you are in the lower cases character mode, so if you press any key then you send the lower case scan code because you want you know that you are going to find out the lower case character from the keyboard and you continue back in the same state machine.

But the moment you press the caps lock key you have got the caps lock event then you make a transition to the caps lock state which is the sub state of the whole composite state of key board operation and varying if you press any key then you have the upper case scan mode switched on and each one of them will have the list of actions, this is the key press is the trigger that is happening for a state transition to take place.

And in this state if you have caps lock pressed again then you will again comeback to default, you will basically keep on toggling between these and these are the different transitions that you can see. So this is a typical way of representing or sample showing how the protocol state machine will actually look like or operate.

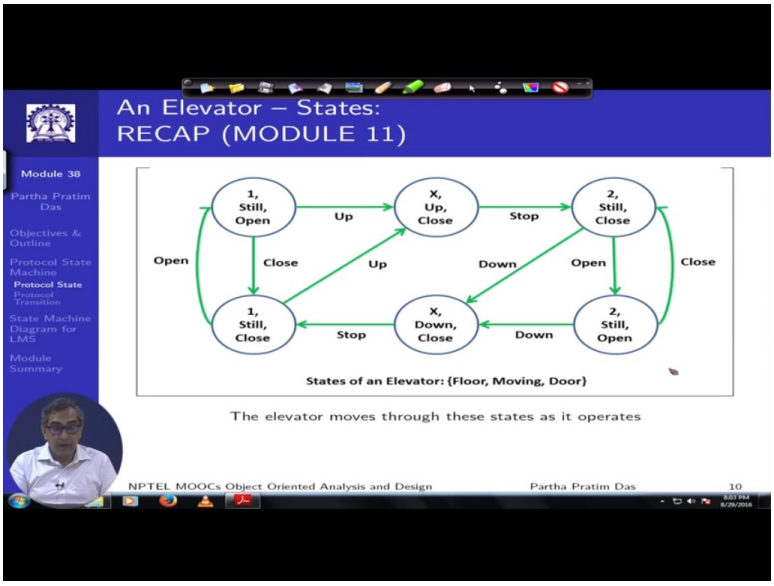
(Refer Slide Time: 11:09)



In this context I would like to also remind you about an example that we had talked about in a quite a few modules earlier talked about an elevator which that we said that elevator can work between 2 floors and we tried to model that elevator in terms of certain static properties which describe the elevator instance but more importantly we looked at the dynamic properties of it they move between one floor to the other.

So we said that the floor could have 3 possible values, so 1 and 2 the first floor and second floor and x which is indeterminate floor or the moving state and we said there is a is moving condition which defines weather the elevator is still or it is moving up or moving down and the door is represented to be open or close to a Boolean.

(Refer Slide Time: 12:24)



And so based on this we could work out with a state machine diagram for this whole scenario and if you now with all your knowledge of the state machine diagram look back into this simple state transition that we had done. Then we realize that this is nothing other than a basically a protocol state machine because you have here all these different descriptions, the triplets defining what is what is unique about that state and these events of trying to go up, trying to come down or trying to open the door and so on, you find the different transitions.

Naturally we can also think about this so this could be the first model of an elevator in terms of the state machine diagram which is the protocol state machine but we could also make this subsequently into a behavioral diagram saying that well if it is say if it is one still open, one still open means it is on the floor 1, it is not moving, it is still and the door is open. so at this there what what will be the entry exit for this?

The entry exit if you if you see the entry for this is is open right so if somebody has pressed the open key open button on the elevator and only then you will enter this. And on exit what you will have, you will have close. You may have close and up but you will at least have close because if you take this you will have close, you will have go this. So you can see that the close is common between these both of these transitions we can say exit is close and what is the do action that this state might have is is scan keys.

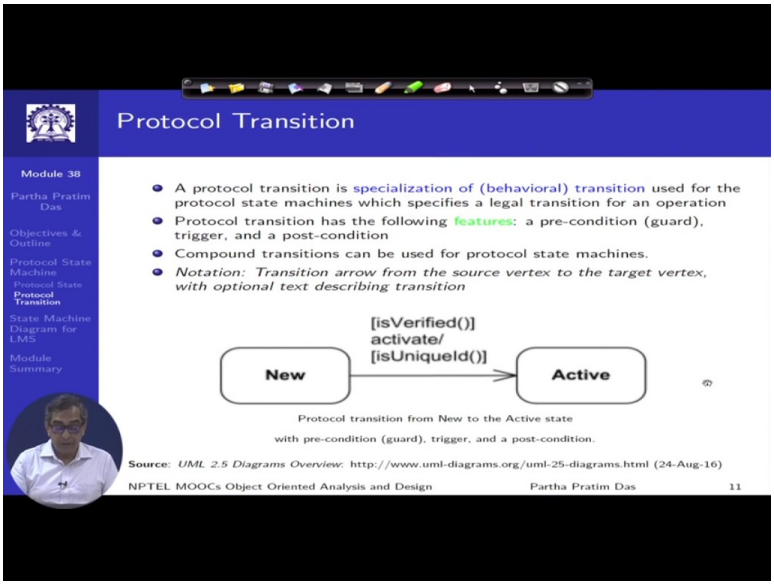
So these are different possibilities that will exist in that state and having identified that after we have

protocol state diagram which is just looking into the different 6 different configurations in which the um elevator can be we could then endeavor all these entries exit and activities and try to bring out the real behavioral state machine out of this. For example another if you if you take so if you consider this state, this state is x that means the floor is indeterminate.

Because the moving is up that is it is moving up and the door is closed. So which means that certainly the entry is clear, entry is up, exit is clear, what will happen in exit, it will stop and what actions have been taken, it is moving right so this is going up, so this activity is continues to happen in this state and whatever else need to happen because of that because the motor has to run, the pulley has to turn in certain direction and so on.

So all these if you describe along with the state description here, then you will elevate this protocol state machine into a behavioral state machine. So that is why we started by saying that protocol state machines are just specializations of the behavioral state machine which give you a compact information in terms of the lifecycle of the objects of the class. So you can you can really see what these these different lifecycles are happening and they will keep on continue to move around in between this protocol state states and will give you information in terms of how the system grows.

(Refer Slide Time: 16:21)

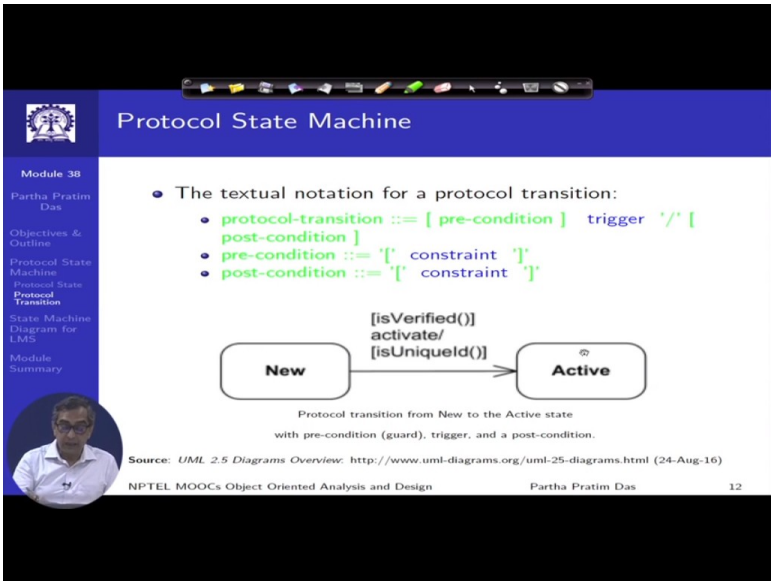


Protocol states do have naturally have transitions and often the transitions are guarded. So a protocol transition is again a specialization of a behavioral transition and used for the protocol state machine and has a following feature which is a pre condition or trigger and a post condition. Pre condition, trigger and the post condition. So precondition is what must satisfy before this transition can take place, trigger

is what makes the transition happen and post condition is what must be satisfied once the transition is taken place.

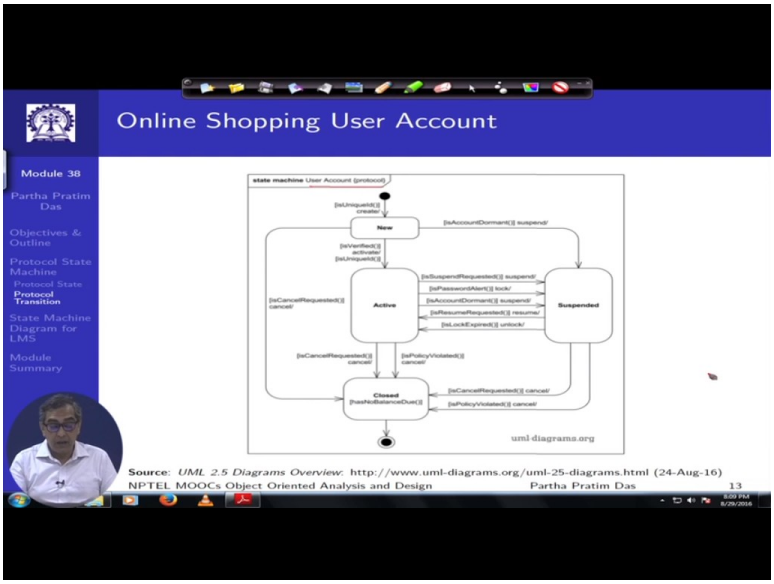
So this concept we have been saying repeatedly in different parts of uml we have seen this similar concepts in activity diagram, you have seen this concept in the behavioral transition as well. So here if we see if it is, if we are moving from a new to active state then the precondition is it verified that this is a this is a valid id. If it is then the trigger is activate that id and certainly the post condition is what will result from this is a unique id.

(Refer Slide Time: 17:45)



So this are the typical ways you will write protocol transitions and so this is the syntax for the protocol transition that you can get satisfied yourself certainly in terms of the protocol transition, the pre-condition and the post conditions are usually constraints. That is basically they are Boolean conditions which are true or false telling you weather the pre-condition is satisfied or the pre-condition is not satisfied, weather the post condition is satisfied or it fails.

(Refer Slide Time: 18:16)



So you could look at a protocol state machine diagram for online shopping and here I have picked up an example where you can see really the different kinds of protocol transitions happening here. So let us look at one two of this, this is unique you start here. So this is the initial pseudo state and then you have the new sub state or sub protocol sub state so the pre-condition is is unique id because and then you create that naturally only if it is a unique id it can be created.

And then from this state you make a transition to the suspend state, what is the pre-condition, is the account dormant. So if the account has been dormant has not been operated for a certain period of time, then you should a trigger suspend and put it to that makes the tran the system go into a suspend state. Similarly, in the in the new id if it is verified we have just seen then the trigger and action activate and get the unique id and you reach the active state of the system.

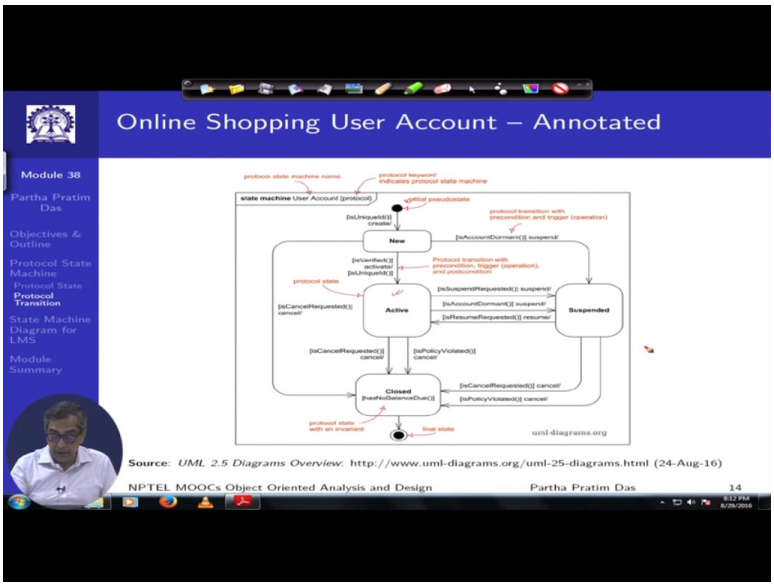
In the active state of the system again you could explicitly ask to suspend your term. to do that we reach the system. If the password alert, for example if you have hidden the wrong password for a number of times then to lock and you get in turn suspended. So all these you are expressing all those semantic you can express in terms of the protocol transitions using the pre condition constrain and the post condition constraint as applicable.

For example here you can see that you have a transition from suspend to active which is based on the pre-condition that is lockexpired. For example when the password is locked, typically it will not be locked eternally it will be locked for couple of hours, for a day or so and beyond the expiry of that this condition gets satisfied and need to get trigger to unlock so you come back to the active state. So in this

way you could follow this whole diagram very carefully to see different situations.

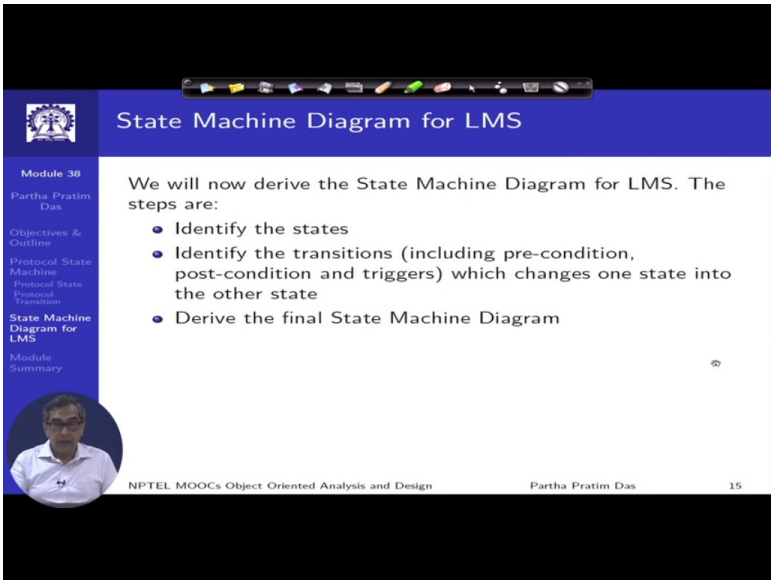
And how they are modeled in terms of the protocol state transition using the pre-condition, post condition and trigger models and learn more about how these state diagram modeling happens. These are same example where there is a little bit of annotation done so that you have ease in preparing for understanding of what these different icons are.

(Refer Slide Time: 21:09)



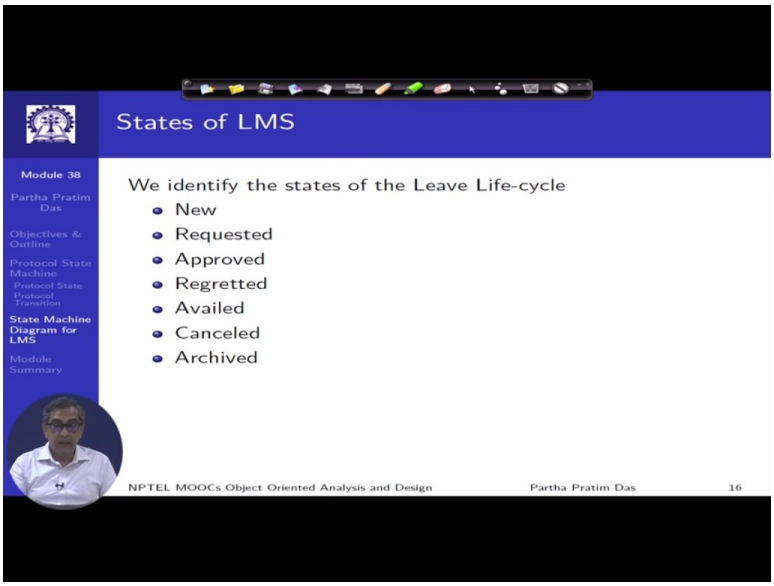
This is as you know the protocol state is a this we have seen several times. this we have just explained the protocol transition, this is another protocol state and this is a protocol state with the difference. It is a protocol state with an invariant it says the account is closed, has no balance due. So if that is the case, then only in that it is will get closed. So this is the invariant which is important for this state.

(Refer Slide Time: 21:43)



So we will had to, before we end we will take a quick look into the state machine that may be important in terms of the leave management system. We certainly need to identify the states, the transitions and the state machine diagram which we will start with by having a protocol state machine diagram.

(Refer Slide Time: 22:02)



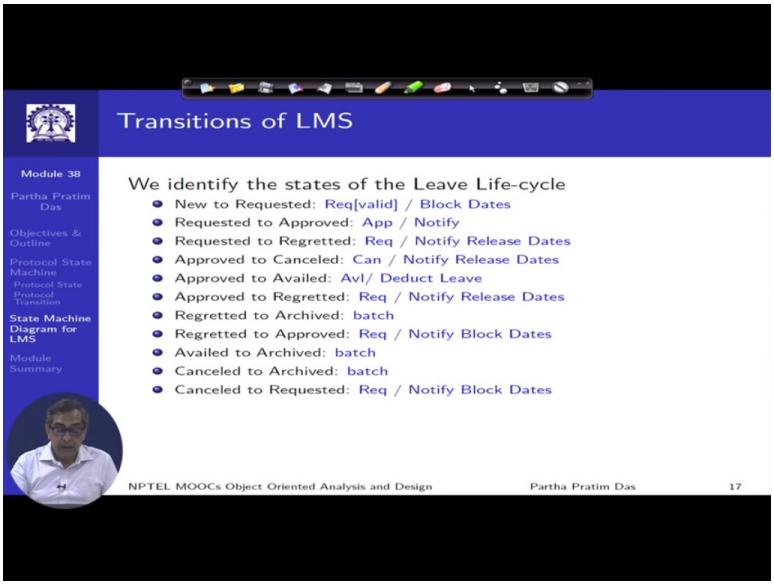
So what we identify is we have seen that a leave is a first created through a request and then it goes through an approval process and it may be approved, it may be regretted. If it is approved then it may be availed or it may be cancelled, if it is availed then it will be deducted and so on. So we can model this whole information in terms of a lifecycle of the leave. You can consider the leave as a lifecycle which has these different kind of states.

So we are just trying to in terms of building a protocol state machine we are just first trying to identify

what the different states could be. so when the leave is just first created, you you recalled that while any request is put in, we need to leave has to validate itself and make sure that the leave being requested is a valid leave. So when that happens if it is just a new leave and after the validation when the request has been run I mean when the request has been registered in the system not approved yet.

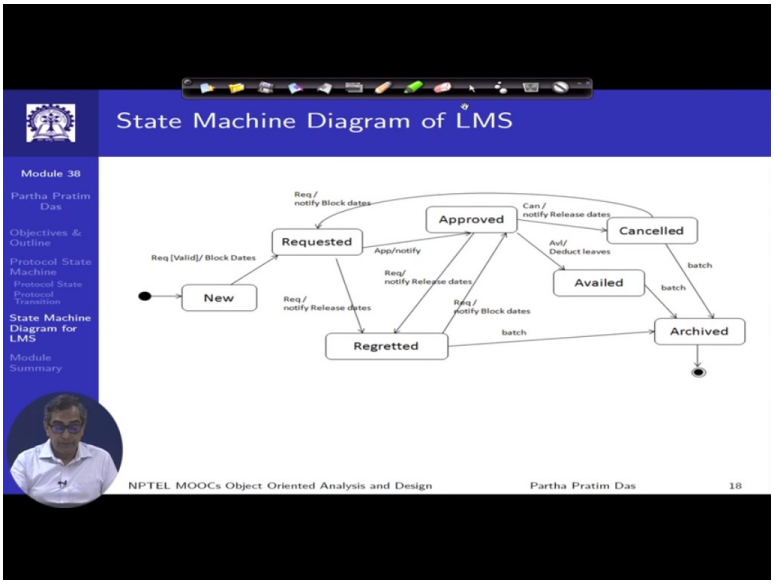
When it has been registered in the system as a valid leave, then it could be in the requested state and based on the request the lead or the manager may or may not approve and therefore it go to the approved state or regretted state and so on.

(Refer Slide Time: 23:39)



So if we just follow that then we will be able to identify that different transitions for the leave lifecycle like in terms of new to requested, requested to approved, approved to regretted and so on. So basically once you have identified the stage you look at, what are the transitions that are possible and what are the triggers and the pre-conditions for that if that can be defined and you try to through that.

(Refer Slide Time: 24:09)



So this is where I present the leave system in terms of the state machine, this is the initial state and certainly this is the state where the leave has been created. So new and then based on this you have a request so what is what is this transition, in this transition will be possible if the new leave is a valid one. right, if this is not valid then this is not right. Then we write in that notation then this is to be valid leave and a trigger is a request.

What is a post condition, a post condition is blocked, blocked dates. Why? Because if somebody has requested for a leave say from 1st June to 3rd June 2016 then once the request is registered that is what we are designating by moving the leave to the protocol state requested, then the those dates must be blocked so that no further leave request or other request can be made on those dates. So we can clearly see that as this transition happen from new to requested, this is the pre-condition that we need. Trigger is the request and this is the post-condition that must happen.

So in a similar manner we could continue on that if a leave has been requested then it may be approved, it could decide further this is just I have put just a cautionary you know transition information you could put more transition information, it could have multiple more states here depending on the different situations of approval, some approvals are conditional depending on the validity or availability of the supporting documents, some are direct approvals and so on.

So you could decide to have more protocols states describing that and then have more pre-conditions and post-conditions defining that or otherwise the requested leave could be regretted so any of these will have a post condition of notifications whether its approved or regretted, then the notification that

will be done and the leave will be in one of these states. An approved leave may be cancelled would go here, it may be revoked actually we could have this is regretted it should not actually happen here.

The better way to model could be i can have a revoked state protocol state. so I can a manager could revoke an approved leave and a revoked leave cannot be availed in further. An approved leave can be availed and then any of these if it is an achieved leave, it is a request regretted leave, a revoked leave or an availed leave could actually go into some kind of a you know we can say a am in search of a word, a dead leave should have no further changes can happen to that leave record, which is then archived in the system.

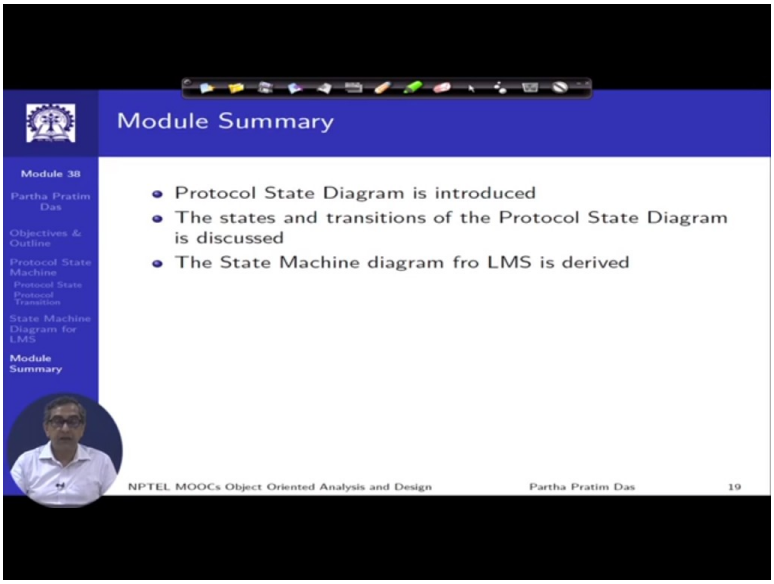
So this is just an outline showing you what are the different transitions that can happen. if you analyze further you will this is and this just gives you the protocol state diagram. So in each one of them for example the requested, you could now take make this protocol state into a behavioral state by defining the exact entry, exit and the 2 activities here because if in the requested state, before the transition happens you have the activity where the manager.

Or the reporting manager of the requesting employee has to look at this leave consider all factors including the documentation and another conditions and then take a decision on that. So those could be the activities in each one of these states that we identified and start encoding putting into the state machine diagram more and more you do that, this will slowly become a more complete behavioral state machine diagram of the leave management system.

So this is just to show you that even such a simple system as for leave management has a lot of scope there this does not immediately look like a problem, I mean is not an electronic problem or a or a physical problem like the temperature controller or the calculator where the states look to be more oblivious but here you can see that the dynamics of the system again depend on a set of states which is very typical for any of the complex systems that you would like to model uml.

The state machine diagram for that purpose or state charts for that purpose turn out to be really useful tools in that modeling state.

(Refer Slide Time: 29:27)



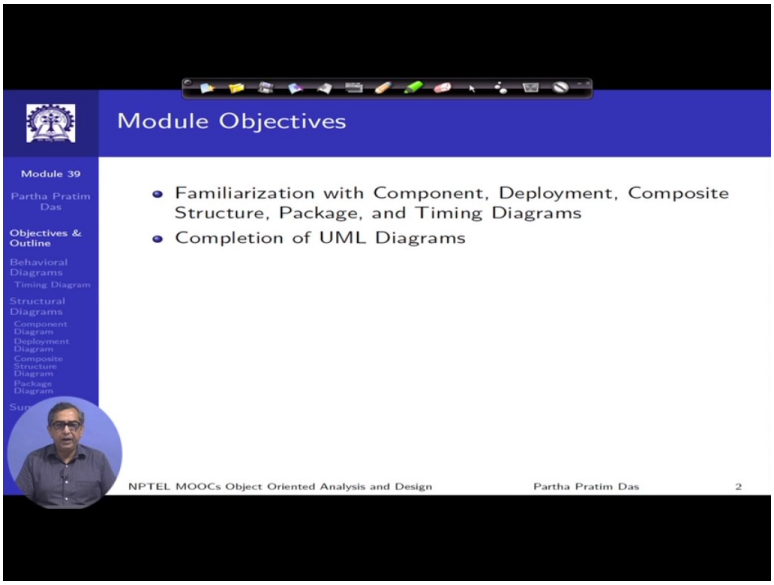
To summarize continuing on the discussion on the state machine diagram you have in this module discussed about protocol state machine diagrams, we have talked about sta protocol states and transitions along with that pre-condition and post-condition and you have taken a look into how LMS the state machine diagram for LMS can be constructed.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 51
Various UML Diagrams

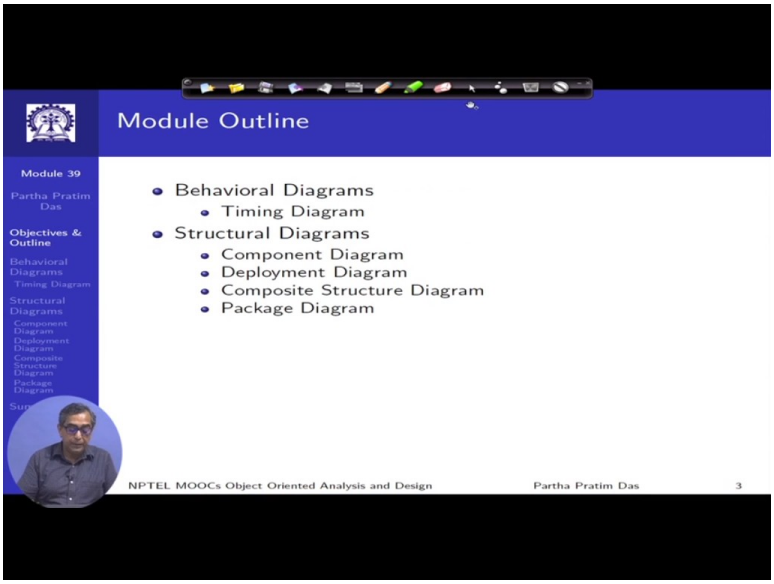
Welcome to module 39 of object-oriented analysis and design. Over the several modules, past modules, we have discussed at length about uml diagrams and we have discussed about variety of structures as well as behavioral diagrams.

(Refer Slide Time: 00:49)



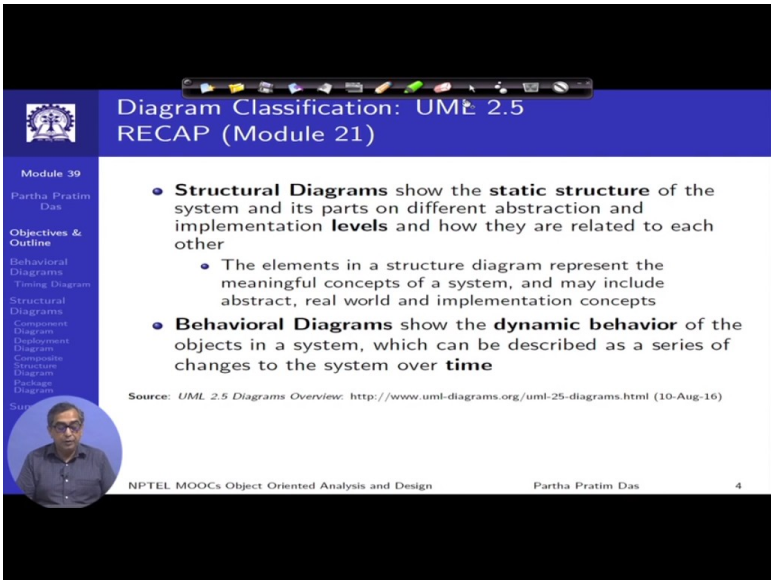
Now in this module we will quickly familiarize with you with a number of other uml diagrams which still left out, we do not have enough time to really go to each one of them at length. So I would just briefly familiarize you and leave it to you to read about them later and that should complete the discussion on uml diagrams.

(Refer Slide Time: 01:10)



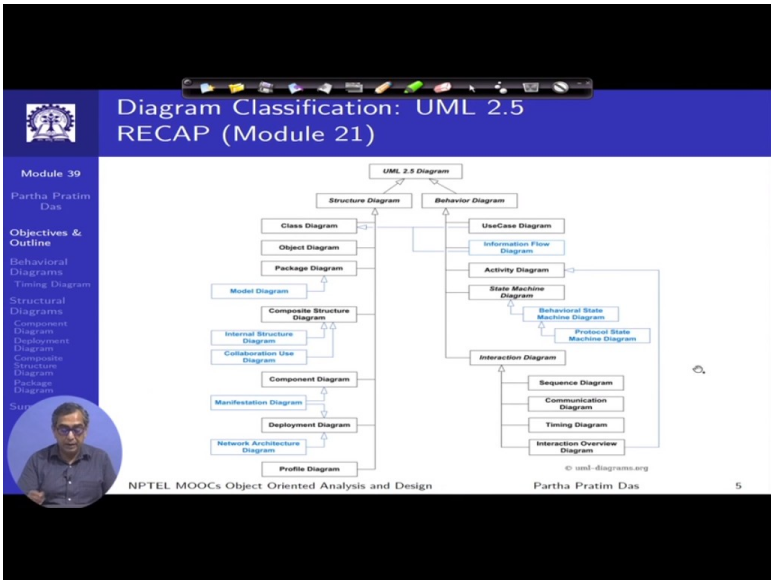
So the five diagrams that you would like to talk of, one belongs to the behavioral type and four belong to the structural type which will form the outline of this discussion.

(Refer Slide Time: 01:21)



So just to remind you the structural diagrams of the uml, so the static structure of the system and parts of different abstractions at a level and a behavioral diagrams show the dynamic behavior basically showing you how does a system behave over a time.

(Refer Slide Time: 01:45)

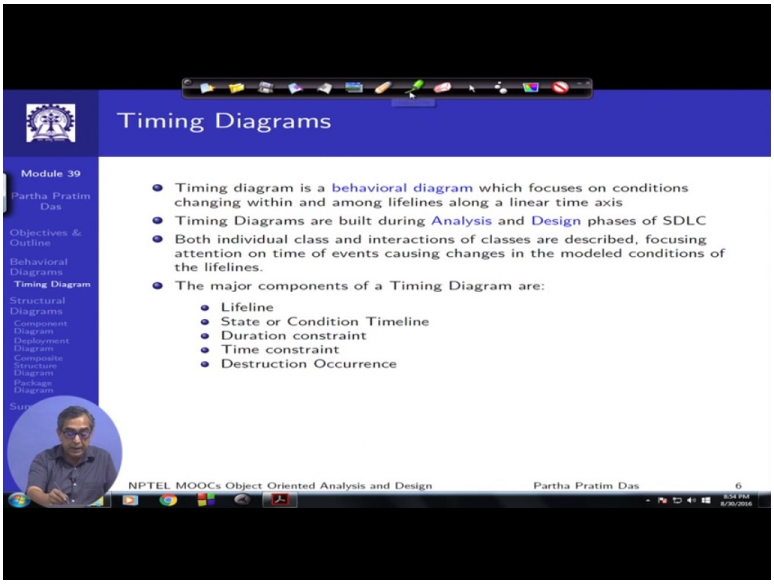


So these diagrams also in a varied way does that. So if you look into the overall diagram classification of uml 2.5 the standard that we have been discussing, these are the diagrams, we have done object almost not separately but then almost we have done. It is not pretty much in use anyway. we have talked about use case diagrams, activity diagrams, sequence diagram, communication diagrams, interaction overview diagram.

So these are the major diagram, the state machine diagram. So we have disused about all the major diagrams particularly the diagrams that are significantly involved in the analysis and design phase and that is where are focus is because we are talking about object-oriented analysis and design but there are few diagrams still left out certainly the timing diagram here in the behavioral part and the package diagram, composite structure diagram, the component diagram and deployment diagram.

These are diagrams which also come in handy but may have less connection in terms of the object oriented analysis and design. So these are just for your knowledge in terms of completion of the understating of uml. But these are diagrams which appear more during the implementation phase when you are actually organizing the code putting the components in that texture and so on. So They are very necessary for the system but beyond the basic concept of encapsulation and interface, they may not add real further value in terms of the object-oriented model. So we will quickly go through these diagrams one after the other.

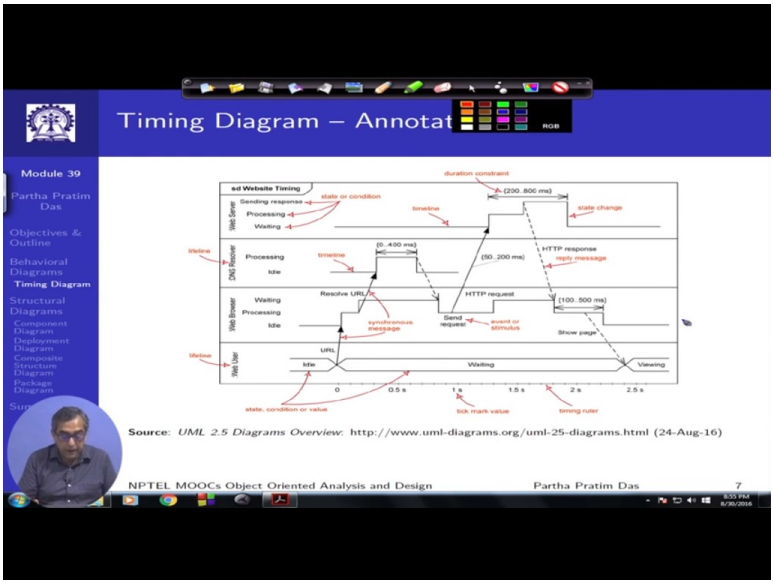
(Refer Slide Time: 03:30)



The timing diagram as I said is a is a behavioral diagram and actually the timing diagram is a again like you saw in the state machine diagram you saw a state machine diagram is a generalization of the finite state machine modules that we use in automata and switching theory and different other places. So timing diagrams are also generalizations of the typical timing displays that we have seen for different hardware descriptions?

The only difference is that the timing diagrams here are not necessarily meant only for the hardware systems or only for the electronic systems. They are also meant for different software descriptions. They primarily arise in the analysis phase and refine in the design phase and these are the major components naturally you can understand that since we are talking about time, lifeline would be major part of this diagram. Then you have different kinds of durations and time constraint and so on.

(Refer Slide Time: 04:32)



So let me just take a take an example and quickly walk you through that. So this is kind of this is very simple scenario being shown here which is in terms of trying to access a particular url. So you can see that this is a web user, web browser, there is a DNS resolver, there is a web server and this is this happens in 4 different streamlines. these are the concurrent timelines. so these are the lifelines this the lifeline these are all different lifelines.

So these are all concurrent lifelines happening so each one will go on having its own timing diagram and they interact between themselves. So for example to start here, the web user as if am sitting here in front of my browser and trying to access a url. So initially I am in idle and then I request for the url. So as I request for a url, a synchronous message, there is a synchronous message sent to the web browser and which was so these are the different states of the of the particular lifeline.

So this was in idle state and as soon as I send a synchronous message, this synchronous message in the sense that once I send this, I will be in a hold state, so am in a waiting state. I cannot proceed that that is what we all know that if you have requested for the url, you have to wait for that url to actually come. So the synchronous message then it makes a state change from idle to processing as you see here and then it sends so it does certain computation and sends another synchronous message to the DNS server to resolve the IP address.

So I my original request was to www.googele.co.in now I have to get the IP address for that, so that IP address will be requested here. So DNS server which was in idle will now change to processing state change and this is the duration constraint which specifies how much time will it take to actually find

out the IP address, given the url of the page that I am looking for. So once it has it has found that out so let us quickly so once this found that out it again reverse back to the idle state from the processing state.

And corresponding to this synchronous message it sends the response back. So here in the web browser I know I was waiting for that I state change my state, the web browser changes its state and goes into processing state. so now it has got the IP, now it is to actually want to get the page. So who will get page, the web server will get the page. So it now sends the request to the web server which was waiting and as soon as it gets its synchronous message, changes state and comes into the processing state.

So it is trying to find out where to find locate the page in the whole internet and then it does that and when it is ready to find, you have found out the page it will send the response which is the reply message. It is not it is not only responding but it will also actually send that page back and there is a duration constraint here which say that if it cannot respond to this message within this time then beyond this time it will possibly actually get back with a cannot respond kind of feature.

So the page could not be found kind of. So the reply has come back to the web browser then the web browser will show this page and this is the again the duration constant constraint to show the page and when you showing the page it sends back the response corresponding to this message and the I was waiting for the page to come I was on a hold, I was on a blocking call and now I get the page so I again change my state and get into viewing.

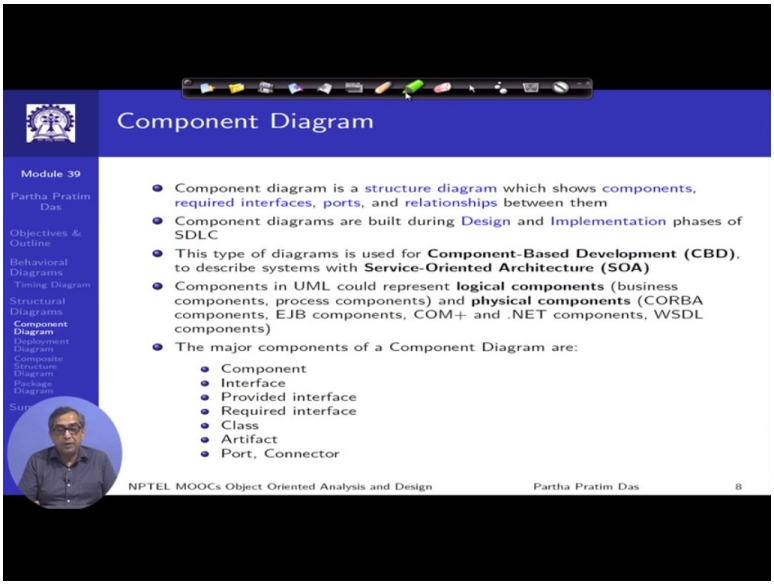
So this is so you can see that a timing diagram can very nicely show the not only the sequence of activities that happening which you would have seen you which you will see equivalently in the sequence diagram as well the corresponding sequence diagram but here it actually shows you in terms of the time period in terms of which this things will have happen and in terms of that time lifeline of each as to in exact clock time when what is expected and how much delay is expected and so on.

What is the time bound of certain things to happen. So you have the timing diagrams is very very critical for the behavior of, you can certainly make out that if you have a completely offline system it will not have any significant timing diagram for that. But timing diagram is significant for any real time system where there is an expectation of messages being responded to certain events being reacted on based on a fixed time boundary, there is a fixed time target within which tasks will have to be achieved

or the events will have to be responded.

and for that certainly you need to express that because sequence diagram as we said if you can recall we said that sequence diagram just gives you the sequence. It may not necessarily tell you the time lapse between one message to the other or between one message to its responds to another. Whereas timing diagram very frequently you would like to say that and also keep track of the internal states of the different lifelines so that the signal timing can be managed. So this is the last of the behavioral diagrams that we wanted to discuss.

(Refer Slide Time: 10:36)



And now I will move to quickly take a look into the remaining structural diagram. so first is the component which is structure diagram let me go back to the red pen, so this is the structure diagram which shows as the name suggest it shows the component diagrams. So it will primarily occur in the design and implementation phase because till you come to design there is no question of identifying components and so you identify components and the design phase.

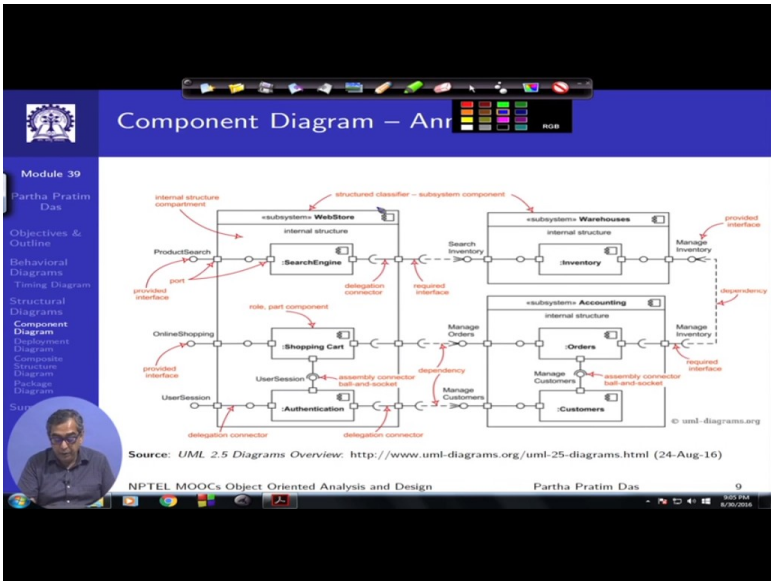
And in the implementation phase really either you acquire components or you create components and then you connect them together. So component diagrams have a major use in terms of component-based development this is this certain style of development where which is very widely used where you try to write very little amount of OOAD yourself but you try to identify different components from different third party sources and which satisfy the requirements of different subsystem that your system has.

Or system may be designed for and based on that you connect this components or maybe you write some proxy, wrappers around this components to achieve the actual functionality and am sure many of you are familiar with SOA which is service-oriented architecture where basically in state of actually just looking into functional calls or object invocation you can actually request for certain bundle of services to be provided and this is becoming very very common these days with very common infrastructure getting into the cloud and so on.

So whenever you have this kind of a scenario component diagram would be really useful. It is important to understand that the what is the basic notion of a component. a component is may be logical or it may be physical. It may be logical in the sense we are, am saying that this is a business component but this is not necessarily mean that this is you know physical hardware or there is a single library or a collection of library which is a business component.

A business component could have a logical boundary something that this is this is where the business is transacted. this is where the process is transacted process component or it could be physical components like these are the different I have listed primarily the software based components it could think of this as actually hardware components also in many phases for example if you go to the domain of automobile you will often find that you need a whole amount of component based development but if there in all the physical components may not just be software blocks they could also be hardware blocks the embedded systems are coming in.

(Refer Slide Time: 13:42)



So there are different components of a component diagram so I would just do a small illustration with

an annotated diagram. these annotated diagrams are really useful for you to understand this better. So we are talking about if you look into certainly by now you have understand the style of uml so this is one component, this is another component and this is another. And these are kind of little big component so you say by the keyword subsystem components these are subsystems that we are building.

And this in turn we has this subsystem component of web store which is to use certain web searching activity has 3 different sub components, the search engine, search the shopping cart for online shopping and the authentication for maintaining user sessions. Similarly the warehouse where you actually have the inventory has the inventory subcomponent the accounting has order as well as customer. So these are the basic component structure, you can see that the implicit hierarchy of containment in terms of the component.

And components really connect and talk to each other through what is known as ports. so basically how do you connect you have different ports and ports have the required interface. Because you cannot for example you cannot just I mean if you think of it this way that if you have a three pin male plug you cannot insert that into a 2 pin female connector so these are ports and they need to have the same interface. so for the 3 pin plug you need a 3 pin connection.

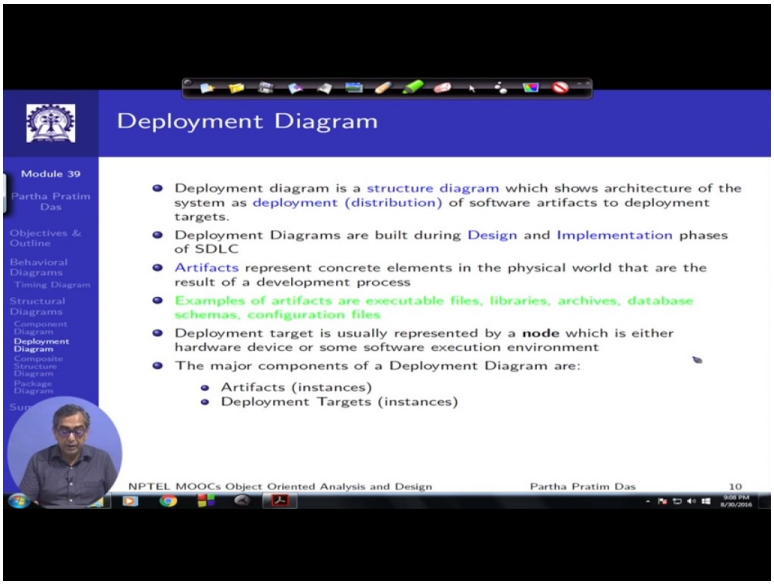
So in a similar manner, you need the required interface to supported on these ports and through that you can connect the components as you can see here, as you can see here, the detail notation you can you can read up then you actually need and then the you can provide the separate interface external interface also of online shopping, user session, the product search or you can or ports could just be internal to the system like this is connection from one subcomponent to another subcomponent within the web store component which is internal connection that we have.

So whenever you have this requirement you put some kind of special connectors which are called assembly connector ball and socket connector. So just to keep you know similarities with the connectors in the physical world. So in this way the component diagram gives you a good idea in terms of the different software components where how they are content hierarchically and who connects to which port or which component, through what interface and so on.

And there are connectors can be so you have ports, you have connectors you have those connectors can

be delegate that is specialized and connector for building up the whole system.

(Refer Slide Time: 16:59)



The next kind of diagram that you will see are again structural diagrams but these are structural diagrams for deployment. The deployment basically in terms of the architecture of the system. overall so far we have been just talking about software as soft so this is a this is a collection of code, library, binary and so on and we just put them all together and say this is this is my component, this is my software this is my activity and so on.

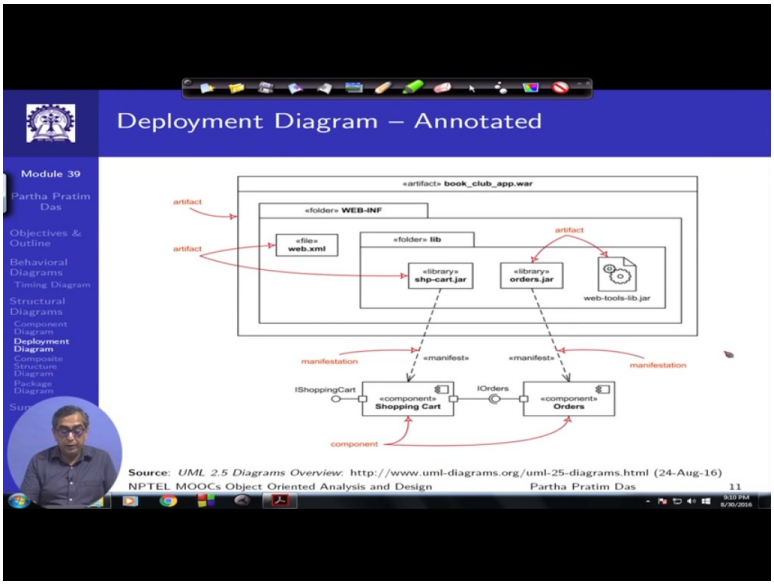
But finally it has to get to certain in terms of certain physical sense which we say are artifacts. Artifacts are like final physical realization of the different parts of the software. they execute the files, the libraries, the archives; the actual database and so on so forth and you will have targets or devices on which these archives work. So you will have different mobile devices software. So the deployment diagram will tell you which artifact is to be located in what target.

So that is the main purpose of the deployment diagram had it been I mean earlier we used to have a single system which had everything so the deployment was really not an issue. But now any system small or large will have different target systems, different processor boards, different mother boards, cores, different servers themselves, different cluster of servers themselves and so on. So even if you have a software component information you need to decide what is of your system works on which target.

And how do they communicate between them. so deployment diagram will basically tell you how you

organize all those into the final system architecture.

(Refer Slide Time: 19:05)

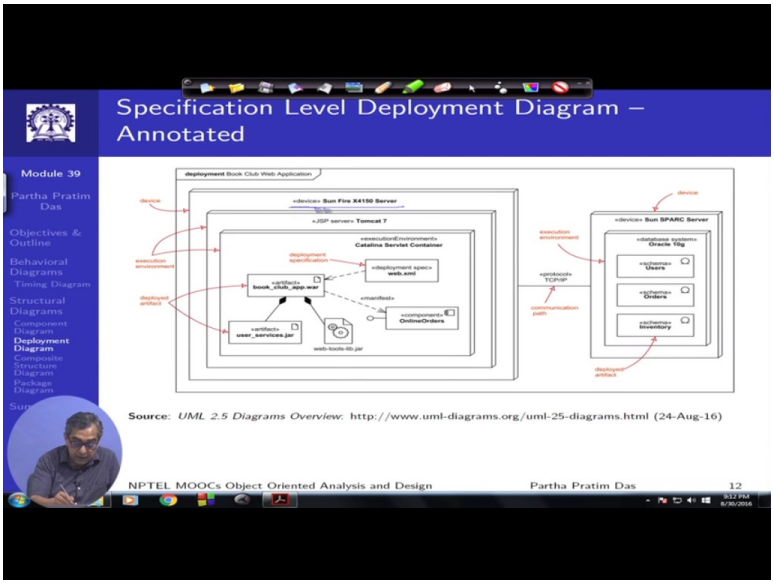


This is just we are showing you another annotated simple example. so these are the artifacts. So this is a web.xml this is again a kind of web access online shopping example. So this is a file, xml file which kind of configure the system, this is a jar file those of you who know java who identify a jar file, if you do not identify this file you can just assume this is a library which kind of provide the shopping cart functionality. This is another java library which provides the orders functionality and so on.

So these are the different artifacts that you have and you can see that how these are physically deployed. so this is physically deployed in terms of a folder then see this a folder lib in which you have these artifacts. So this is the container, this is the target in which you have these; you have another folder web-inf which contains this folder lib as well as it contains these artifacts. So look from this folder's point of view this lib folder turns out to be an artifact.

And then you can so the where does this particular artifact manifest. This is called the manifestation which will basically related to the component which uses or which is realized in terms of this library. This library is the physical entity physical binary code jar code of the functionality and this is the dynamic component which has the behavior when executes. So this basically is a logical component that you have which will so the jar code, the physical binary code when executed it manifest itself in terms of the component behavior. so that is the basic propose of the deployment diagram.

(Refer Slide Time: 21:22)



And these are some of the other examples of deployment which is this is called specification level deployment. You may not follow every detail here but just try to understand here we are actually talking of devices. This device is sun fire x4150 server. this is actually a physical server you have talking of. And then within that you have another which is called execution environment. the execution environment that is a tomcat. Tomcat is nothing but a web server it is regularly used.

And then you have execution environment which is Catalina servlet container. Now this is something of a execution environment that the user has created through certain other third parties support. So these artifices that we just saw they now get deployed in this execution environment or in this device. That device itself is deployed in the server. So it works in terms of that it works on that server. Similarly, there are other there could be another device which is another server which connects, communicates through certain TCP/IP protocol and these are different inventory, orders, users.

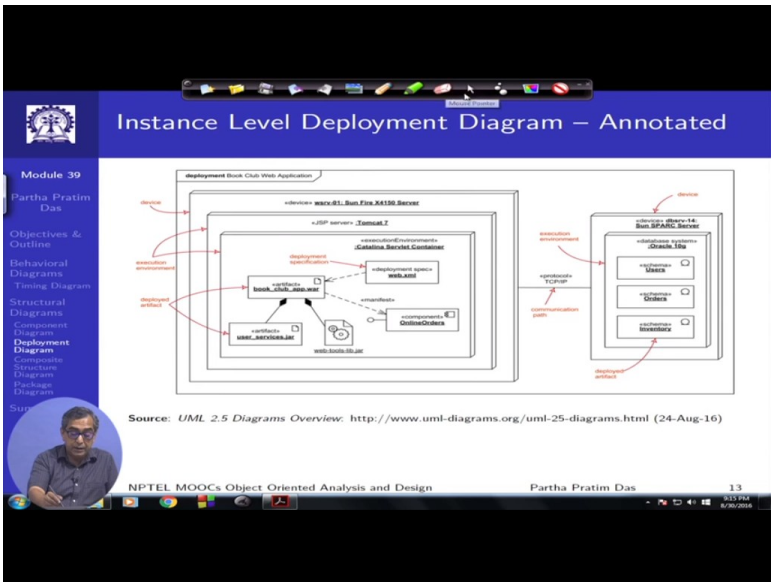
These are different artifacts which are deployed here using an execution environment which is oracle database 10g. So all these information as we can see is not available while we talked about the other diagram but here especially trying to look into these and get the details of system specification level, deployment. this is just another variant of this is called the instance level deployment. So earlier we just talked about the sun this particular server which is a generic class.

So there are hundreds of such server available in the world and here you talk about just the specific instance of that server. So this is you get to this level when you are ready with you system or you are about to ready with your system and you will really know want to know what are the servers on which

you will put these different components, different artifacts to actually execute. So that is why this is called the instance level diagram.

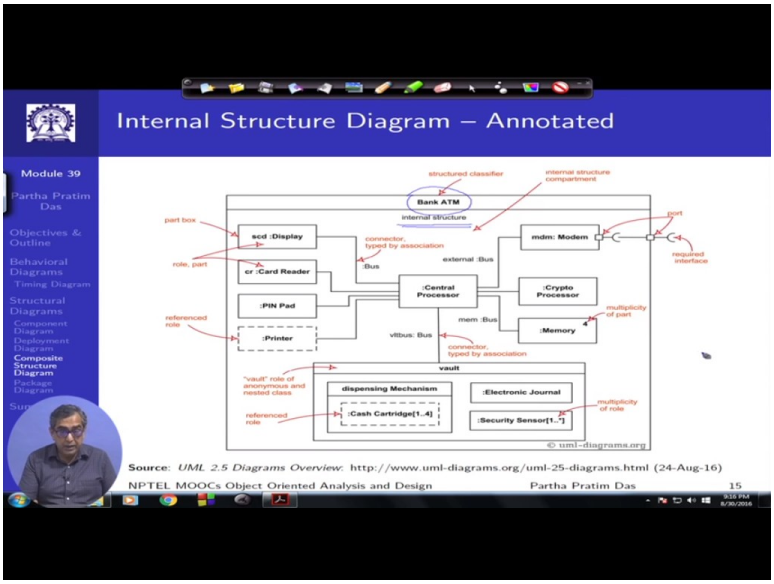
You now have the instances of these you have an unnamed instance of tomcat which is actually running here and so on.

(Refer Slide Time: 23:56)



So that is the job of deployment diagram. the next is the composite structure diagrams. The composite structure diagram is again a basically a structure diagram which show you the internal structure diagram and a collaboration use diagram. So it shows the internal structure of a classifier. Classifier is a kind of a system which can be decomposed into its properties and parts and we can also show collaboration.

(Refer Slide Time: 24:25)



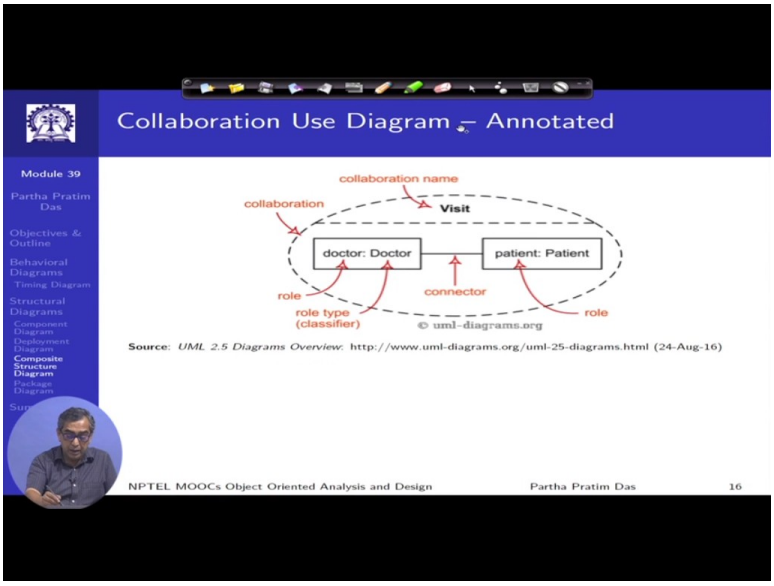
Let us quickly go to an example. so here we are talking about a bank ATM and this is we are trying to show the internal structure. So in terms of the internal structure now you can see what does a you are trying to describe now the internal structure of an automatic teller machine. So it must have a display obviously, it must have a card reader where you put the card, it must have a key board being paired so that it can key in the print.

It must have a processor, it will have a modem to connects to through the well link, it must have encryption processor, crypto processor, it must have a memory and certainly it must have a vault. The money has to results. So all these are significantly are not really all software components, the physical components, the hardware components. so the internal structure diagram the internal structure is showing that this is how the different parts of the system the composite structure that is the reason this diagram is called a composite structure diagram.

So it some of these might turn out to be whatever you put in here in here in the crypto processor could be a software component but you need to understand the physical structure, physical internal structure of the system to be able to realize really how different how the whole system will get into being. And the internal link you have the printer as an internal structure, you can see how it is shown in a dotted line because it is a referenced role in the sense that we are saying that bank ATM the referenced role means the bank ATM will have a printer.

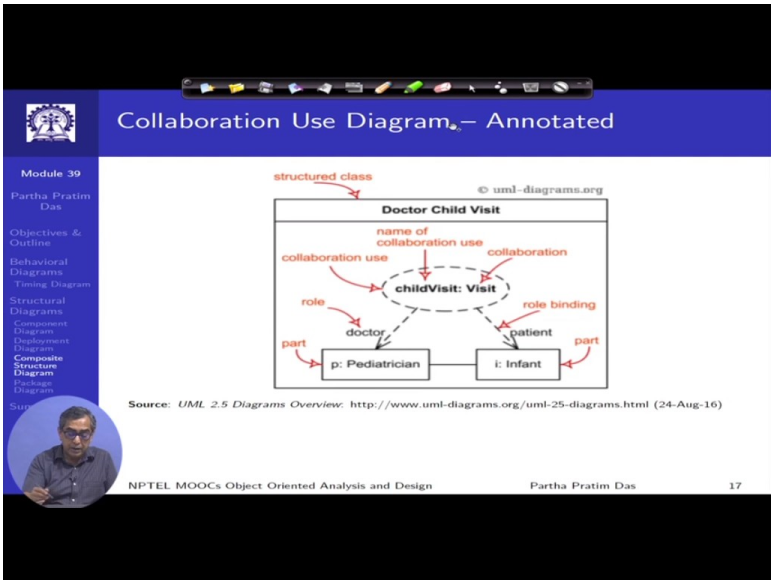
That is not a part of the ATM machine, but it will have a printer. so the printer is will be put in by somebody else and which the bank ATM would be able to use.

(Refer Slide Time: 26:32)



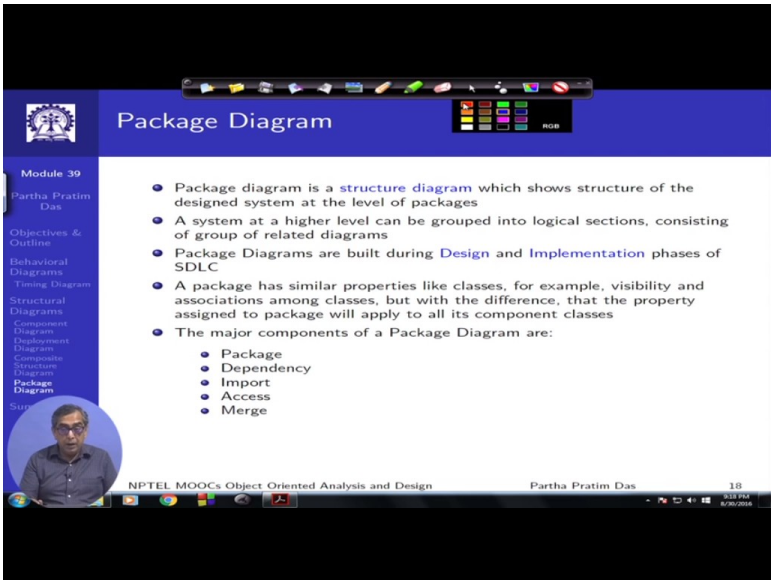
Besides that there are different collaboration use. I will skip these. you can study them.

(Refer Slide Time: 26:38)



These are some of the other ways you can show the composite structure.

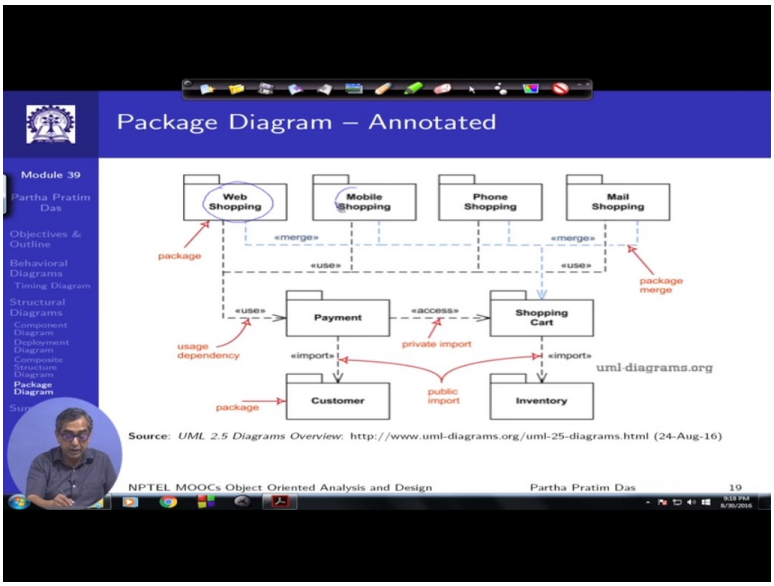
(Refer Slide Time: 26:40)



And I will rather move on to the package diagram which is another very important diagram. the package there are so far mostly if you see in terms of the component, in terms of the internal structure and all that, we were primarily trying to take the whole system and trying to break it down into different parts. in the package diagram we try to do the reverse. What you try to say that we have different components, we have different parts. how should we put them together into different envelopes into different packages so that they can be better managed, they can be better maintained.

So packages again have a number of components through which the package diagrams express themselves.

(Refer Slide Time: 27:30)

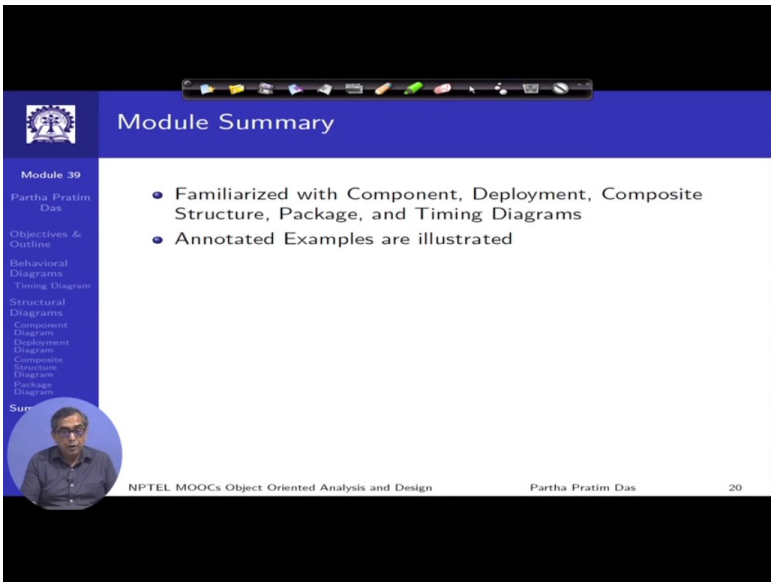


So this is again looking into example so here you are saying that there are different packages like web

shopping package, the mobile shopping package, the phone shopping package and the mall shopping package. So this package is basically contain all that you need to do in terms of the shopping on the web. All that you need to do in terms of shopping in your mobile phone and so on. and then it could as this dotted blue lines show when you can merge them together this is called a package merge and big may a bigger package.

And then you say that my shopping cart component actually is having this package to use. so this packaging really helps you to refer to a multiple independent parts in terms of a common generalization and this is these are the individual packages and then you could define dependencies between the different packages. this is the payment package, and this is a web shopping package. So you can say that they have a dependency in terms of in use of a payment using a web shopping and so on.

(Refer Slide Time: 28:43)



So these are these are different package diagrams. so to summarize we are trying to conclude on the uml diagrams. So have we have already done the different structural and behavioral diagram discussions and this module we tried to talk about the one more behavioral diagram as timing diagram and 4 other structural diagrams and this will complete our exposure to the uml diagram as a whole and we are now equipped to use the whole of uml to model and represent.

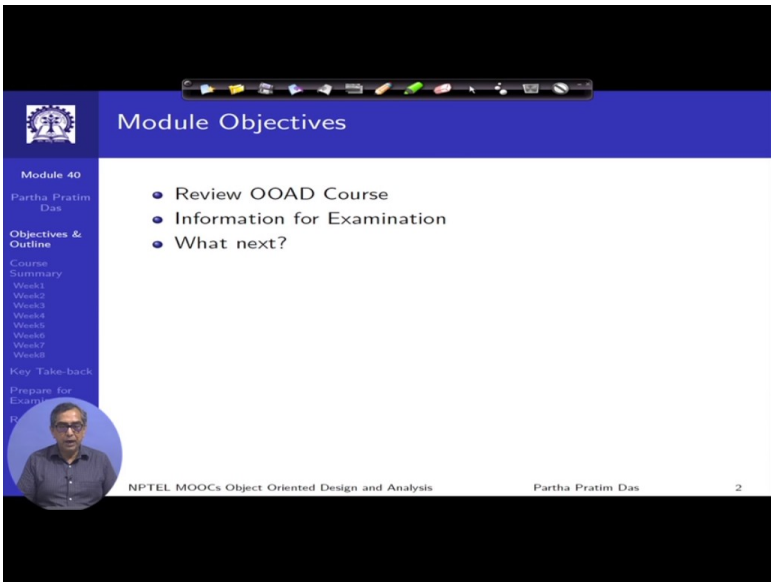
And refine any situation of analysis and design and in some cases might take them to the different stages of implementation as well.

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 52
Closing Comments

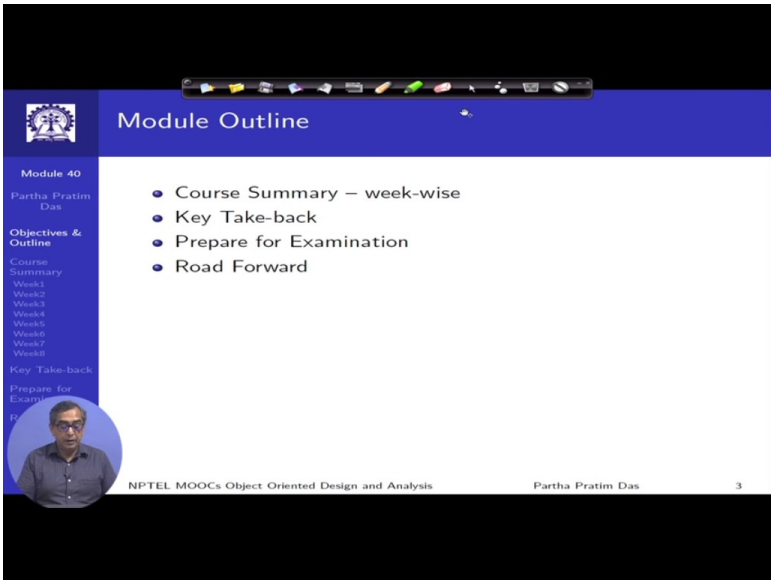
Welcome to module 40 of object oriented analysis and design. As you know this is going to be the last module of the course. So I would not discuss about any new topic, new aspect of object-oriented analysis and design.

(Refer Slide Time: 00:49)



Rather I will spend a little while putting some closing comments. So the objective is to review briefly what we have done in this course and mentioned to you why you should be preparing for your examination and beyond this course what you should be looking at.

(Refer Slide Time: 01:03)



So this forms the basic outline of the module.

(Refer Slide Time: 01:04)

So if we look back and I would take a quick review based on how the modules were organized in terms of different weeks and moving from one week to the next. In most cases we tried to achieve something foundational about the course, about the discussion. So we started off with what is the software complexity all about. So we started with a very basic question of is software at all an engineering and reasoned to come to the conclusion that we still develop software.

We are far away from actually constructing software, the way several engineers construct buildings. But in the process so we identified some of the fundamental reasons as to why the software projects fail and this lead us to identifying four elements of complexity that really plague the software. Complexity

of the problem domain, difficulty of managing the development process, flexibility possible through software, this is to appears to be an advantage and goes against the software and the problem of characterizing behavior of discrete systems and this is the reality that will have to work with.

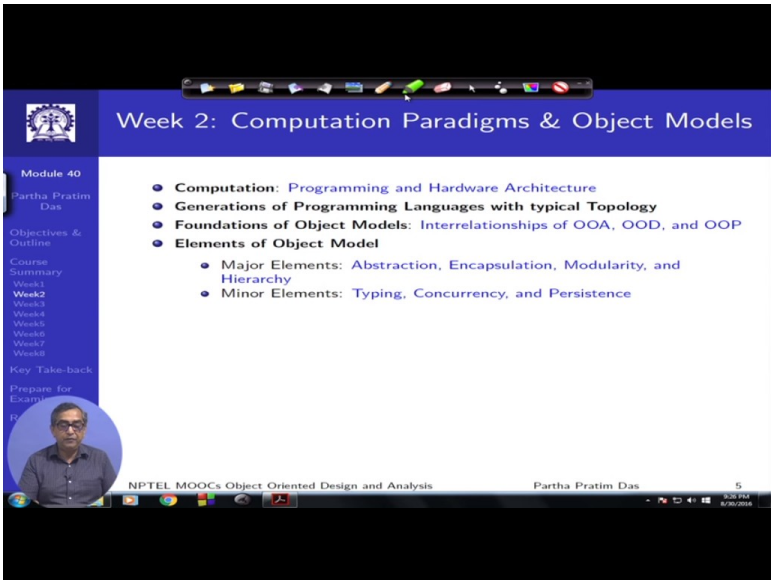
So we then try to look into how the complex systems are structured, what is their generic structure and we observed looking at natural, man-made, social different kinds of system that there is a whole lot of commonality between the structure of complex systems and that can be leveraged to really come up with designs or come up with the design approaches which can prove to be really good in terms of software development.

So we identified the attributes of complex systems and 5 attributes were identified hierarchic structure, relative primitive, separation of concerns, common patterns and stable intermediate form and this lead us to discuss about organized and disorganized complexity and as we have seen that these organized complexity became the source of main study in the object-oriented analysis and design that the organized complexity manifest itself in terms of decomposition and abstraction hierarchy, class and object structure and can lead to what is known as a canonical form of a system representation.

We also recognize that there are other forms of complexity which is disorganized which is primarily due to the limitation of the human brain and we can hardly do much about this other than doing the organized, managing the organized complexity better. Now finally to bring an order to the chaotic scenario of complex systems, we argued that we need to look at decomposition, abstraction and hierarchy. These are this will be the key tools by which we can design and model the system.

So this was the basic analyzing the foundational reality of software and the reality of real-world systems, the complex systems was a main take back from the study of software complexity and object-oriented solutions for them in weak one.

(Refer Slide Time: 04:47)

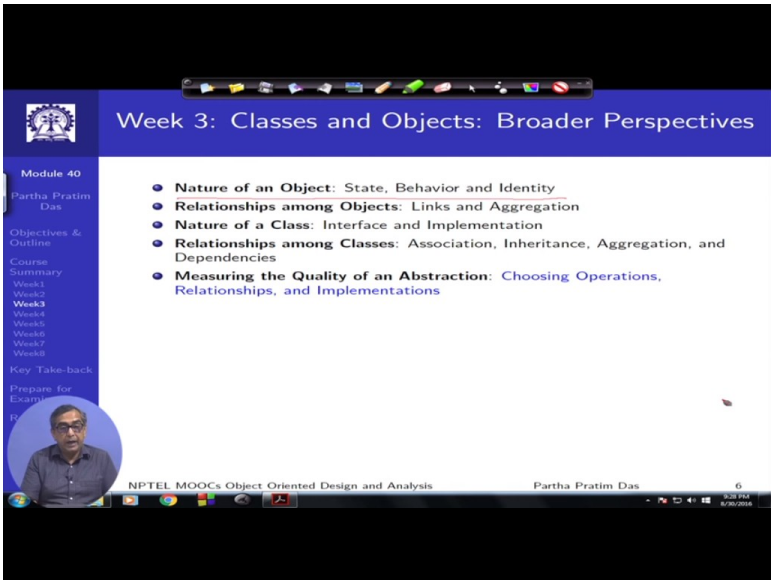


In week2, we took a look little bit differently in terms of what is am if the software is complex if the systems are complex then we wanted to also look at what is in terms of the computational paradigms and object models and we look we can deep look into different programming and hardware architecture, the generations of languages how languages have evolved, how programming paradigms have evolved, what has been the typical topology and so on.

And along with that we introduced the foundations of object model which we have did it, we have done it in three stages saying one part is object oriented analysis, then you have object oriented design and finally you culminate all that into object oriented programming which were your actual software system gets realized and in terms of doing that we we could identify that there are 4 major elements in an object modeled the abstraction, the encapsulation, the modularity and the hierarchy and also there are 3minor elements.

So these this point onward these elements have become the guiding factor of deciding on how we do the analysis, the object oriented analysis how you actually do it, going to doing a design which is efficient which is manageable for a complex system. So that was the primary primary aspect to discuss in week 2.

(Refer Slide Time: 06:25)



In week 3 we moved closer to the commonly known notions of object oriented terminology, we talked about the nature of objects and identified that an object is something that exists as a state, as behavior and as identity and we deliberated at length in terms of what and how is an object and we extended that for relationships among objects we showed that the objects just existing by themselves does not provide anything interesting but they become interesting when they have links when they get aggregated between themselves.

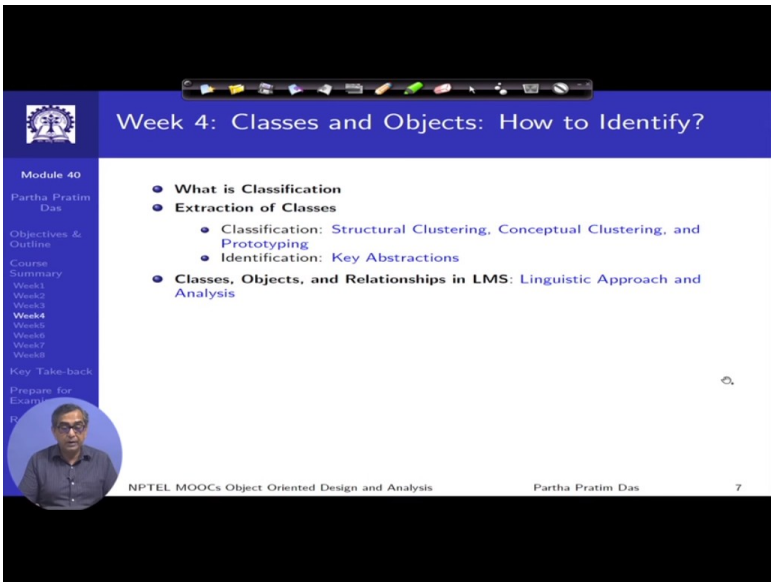
While objects are interesting in terms of actual programming we need a layout we need a template through which we need to express these all these object behavior. So next was the discussion of the nature of classes, a particularly in terms of the interface and implementation. We saw that the interface is how the objects instantiated from that class will interact with others and implementation is how actually the an object of that class come into being how actually the behavior, the state and behavior of the object will be managed.

And both of these are pretty important to be able to finally realize the objects of an object-oriented programming system. Relationships amongst classes give us a lot of direction in terms of how should the modeling be done what are the different modeling possibilities that exist in terms of the real world behavior in terms of the real world connectivity's and in this we also took a different measure of looking into that since all of these are objects, classes all are different forms of abstraction of the real world.

Because certainly the real world was not created to with the stamp of this is this object or this is the

class, this is the relationship and so on. So when we identify such objects and classes what should be our measure of quality of abstraction when we say that the obstruction is good when we say the abstraction is not so good and has scope for improvement so we talked about measures of quality of abstraction, in terms of choosing relations operations, choosing relationships and choosing implementation?

(Refer Slide Time: 09:10)



And we then made this a tool in terms of carrying out a detailed exercise on our part to be able to show you that how we can put all these discussion of object-oriented analysis into practice. So the basic question that we started to address in week 4 is good that we realize that object oriented approach will give us a much better alignment with the inherent structure of complex systems and they should be organized in terms of objects and classes and their behaviors, relationships and so on.

But that is still leaves one basic fundamental question unanswered as to given a real world situation how do I identify the classes, how do I identify the objects. So we talked about the basic classification process to help the extraction of classes and we learnt about 3 basic very fundamental foundational methodology for classification in terms of structural clusterintg, conceptual clustering, and prototype based classification or prototyping and we also learned about how we do key abstractions for identification.

And these become effective tools at our hand to look at a real world situation at length and actually try to extract the classes, objects relationships, hierarchies, refinements, dependencies and so on. So we spent a major part of week 4 in trying to apply all these in the context of a running system example

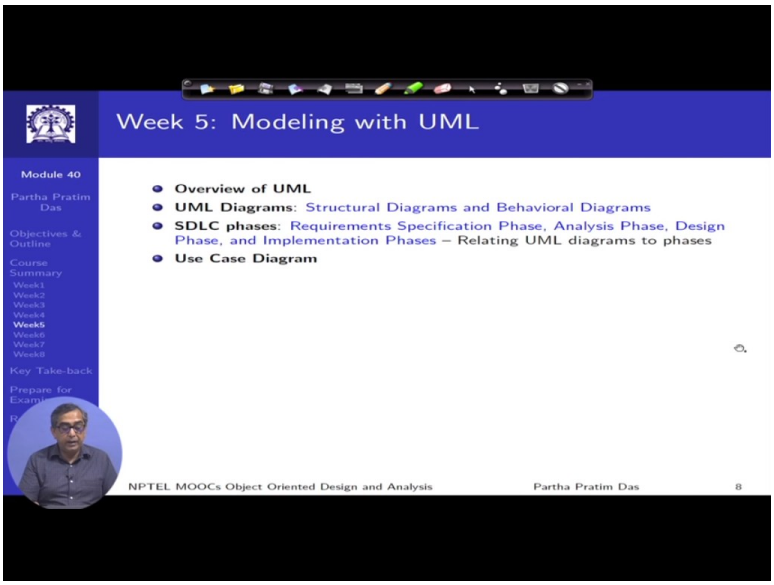
which we had introduced in I think in week 1 or week 2 early in the course a leave management system and we demonstrated with one special part of the approach which is a linguistic approach.

There could be other approaches as well primarily trying to analyze the different parts of speech of a given English description of a system and reference to certain video based interaction between the customer and the client we try to analyze and identify nouns, verbs, different parts of speech to lead us to the different classes their relationships their hierarchies and so on and every time we worked on on such a model identifying classes, objects, relationships we tried to get back to the quality parameters that we have defined for the abstraction.

And measure as to are we really improving from the earlier model and in that process we showed through a series of refinements that possibly went over 3 or 4 modules that starting from a 2 page English description we are able to come out with a significant number of classes, their hierarchies are major attributes, their relationships and put them and and those more or less everything is has certainly the dimensions of who is doing it, what is the expertise of the person analyzing.

But still we could identify several techniques by which more or less uniformly all these information can be extracted.

(Refer Slide Time: 12:37)



And that was the basic objective of the week 4 which culminated the first half of this course with the introduction of the object-oriented paradigm as a whole and culminating with certain practical approaches to identifying objects, classes and relationships within the system. The next half of the

course covering week 5 to the week 8 has been dedicated to the other aspect of the course which is once the information once the object-oriented components, classes, object, relationships hierarchies have been extracted the different processes have been identified.

Different use cases have become known, how do we represent them in terms of a language which is not biased by a particular programming style, a particular platform or a particular individual a particular process and so on and in that context we introduced what is known as a unified modeling language on uml and we discuss we provided started with overview showed that uml did not happen in a day, it went to a evolution of several years when different concepts.

Finally got unified in terms of a uml collection of uml diagrams which are classified as structural which talks about basically the levels in which different part of the system exists and the behavioral diagrams which say how the system will behave over period of time now to combine these 2, the first part where we identified the basic object-oriented features of a system, the classes behaviors object behaviors, relationships and so on and their expression through such a standardized language to be able to bridge this gap.

We also took a detour into the basic sdlc process which is a software development lifecycle process. So we wanted to say that well it is good that we have a language which is a platform agnostic, which is process agnostic, which is people agnostic and so on and we know how to extract different object system information from the real world through the methods that we did in the first part but when do you do what? If the uml has several diagrams which diagram should I use at which stage of software development? what am I?

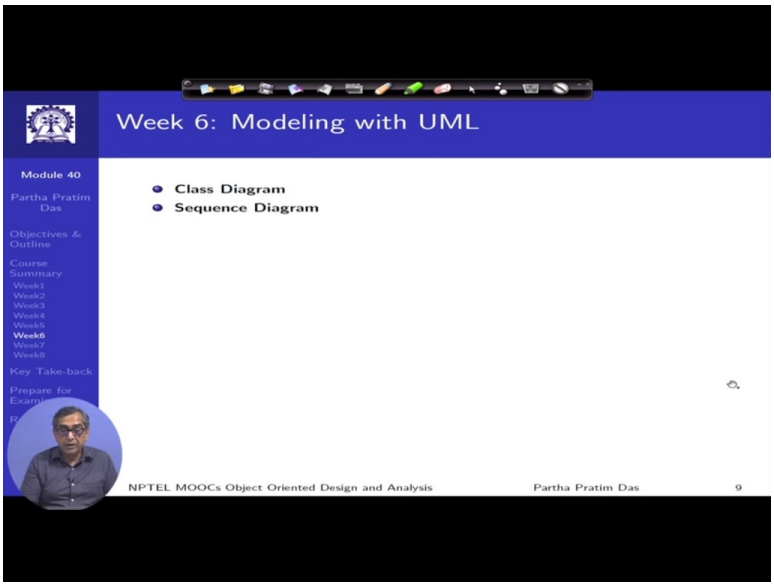
What information should I be interested or focused on extracting at different stages? So we took a tour in terms of the sdlc phases primarily the part of that requirements to analysis to design which is significantly these are the 3 where OOAD is significantly involved but it spills on to the high level low level design aspects the object-oriented programming aspects in the implementation phase, we also talked about this, we have not considered the other phases like the actual deployment of the system or the maintenance and the overall other lifecycle aspects or the testing phase and so on.

They also involve object oriented analysis and design but certainly in a less significant way than these phases and with this equipped with all these, we have dedicated a significant amount of I mean more

that I think40 percent of the time of this course in actually explaining understanding the uml diagrams and we have done that in the kind of perceived order of their importance and how widely you are expected to use them.

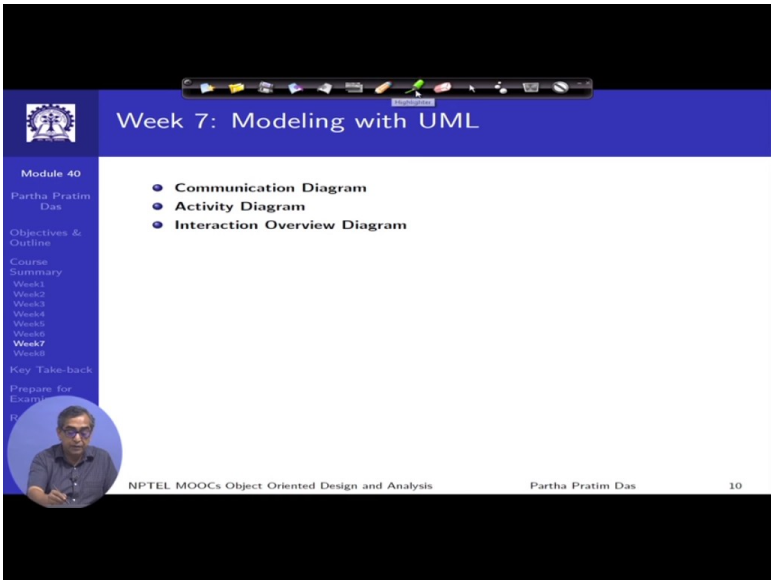
So we started with the use case diagram which is certainly the most foundational which you use in the requirements phase and you will need to use that every time you come across a system.

(Refer Slide Time: 16:54)



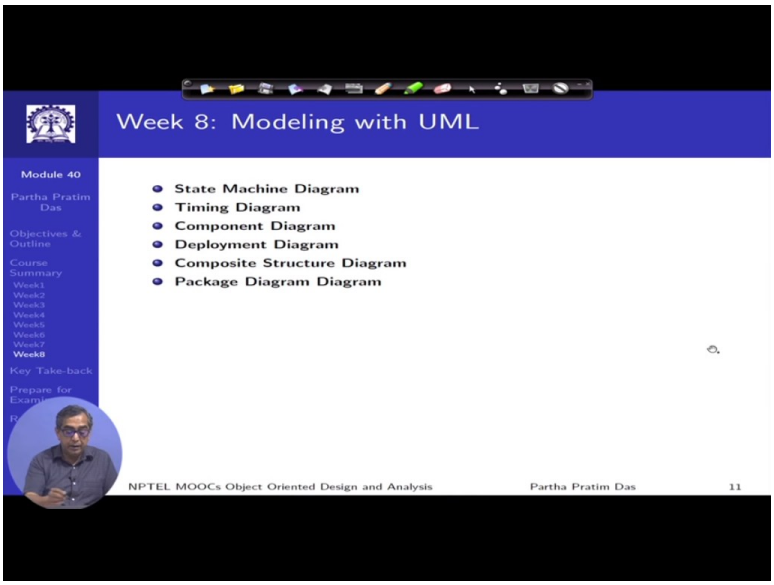
And then we moved on to a structural diagram the class diagram which is again possibly the most widely used uml diagram which also leads to the direct consequence in terms of object-oriented programming, then we moved on to sequence diagram which talks about the processes and algorithms particularly in the temporal domain.

(Refer Slide Time: 17:18)



And continued to discuss about different behavioral diagrams in terms of the communication diagram which is the communication diagram which talked about the special information of messages and all through we kept regular reference to what is happening in terms of the original computing model which in our assumption has been the client-server model. We talked about the activity diagram which is the sequencing of operations as identified from the different use cases the interaction view diagram which is another smaller diagram.

(Refer Slide Time: 17:56)



And then finally we in the current week we ended up with the state machine diagram which is again a very important diagram giving us lot of state related information in the system and its more powerful than the finite state machine, it basically gives a state chat information and subsequently just in the last module we took a quick look into some of the other uml diagrams which are also very important but in

the context of OOAD they may have less significance because they relate more closely to the implementation phases than the analysis and design phases.

(Refer Slide Time: 18:40)

So having looked at that later if we just look back turn back and in one slide I want to summarize as to what have we learned? I think this will be the core learning that we are looking at. Certainly the first message we have learnt is software is complex because if you leave a side everything and anything else it is complex because systems are complex that is an inherence of that. So there is no compromise on this to the first lesson we have learned.

Second is object-oriented analysis and design helps understanding and working with complex systems. Why? Primarily there are several reasons but primarily for 2 reasons, one is object-oriented analysis representation design is natural, it is not a synthetic approach, it is something that you see around happening in your real world and you just imbibing that in your process and that is the reason that there is such a great alignment between the structures of complex systems as we saw.

And the different primitives that the object-oriented analysis and design have started to offer. It is scalable that it works for a very small system a payroll system of 10 employees to a mass landing systems if I may talk of everywhere object-oriented analysis of design work in every scale so that gives a major reason as to why it should be used it is used in almost every kind of complex system. Second fundamental reason is OOAD is a process, people and platform agnostic.

The whole approach does not depend on individual, does not depend on what process you want to

follow or which platform you are trying to use by platform I mean software, hardware, physical systems everything. So OOAD it is a very very generic approach. The third take back I would say is a uml we have learnt uml as a very powerful modeling instrument and it is powerful for several reasons. One is it provides a language of expression that again is process, people and platform agnostic.

It does not say that you have to use C++ or java, it does not say you have to work on unix, it does not say that you will have to use agile or waterfall or rational rise process, none of that. You could be you could do anything in terms of all these different choices of design and exercise but you have a language which is standardized which in many cases is executable and which certainly across both the community will understand. Second reason that uml stands out is it aligns very perfectly its designed very perfectly in alignment with object-oriented paradigm.

So whatever we want to say or whatever you identify in terms of object-oriented and design that can be expressed very aptly, very suitably in terms of the uml primitives. So that is a great reason as to why uml is a powerful modeling instrument. Finally it it is powerful because it can work in multiple levels of details and multiple facets. What does that mean is there is uml when you are at the requirements phase where you almost know nothing about the system.

So you are at a very abstract, very vague level of details. Then you slowly capture certain diagrams, you capture use case some basic class diagram and then you start refining them. So as you keep on refining them, you are increasing on the level of details further and further and further and you continue to do that and uml will continue to support you and it is multifaceted because it is you talk about time, you have sequence diagram, timing diagram, you talk about space, you have collaboration diagram.

Component diagram, you talk about object hierarchy, you have object diagram, class diagram. So you may want to look at the system from multiple different angels and uml today tries to give you a ability to look at a system, represent a system, expresses a system from almost any angle that you feel comfortably. So the third take back in my view is your understanding and you know ability to use uml as a modeling instrument. Finally, I would like to say that object orientation is not a tool, language or platform. It depends actually on how one looks at the system.

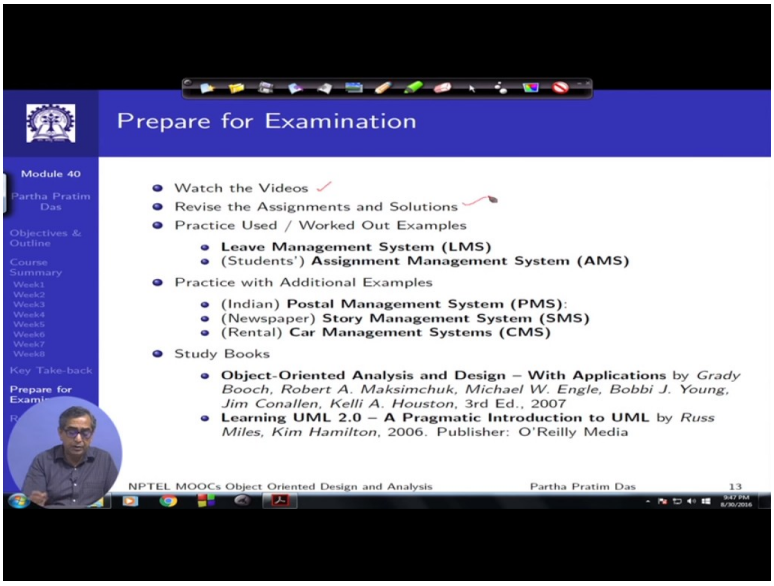
I am all often faced with this question is, is it necessary to use say C++ to be able to design and implement object-oriented system? Can I use c and do that? I often would say that yes, the question is

object-orientation is not significantly in that, will certainly the tool helps certainly the language helps but you can still do a very good object-oriented design in a language like c which has zero object-oriented feature provided you can look at the system in a object-oriented manner, you can look at the system with the proper object-oriented analysis and design approaches.

So to summarize in terms of the take back this is my one line recommendation to you all is object-orientation please keeping mind is in the mind of the developer. uml will help analysis approaches will help the language will help, certain tools, rational lows visual paradigm, they will help but whether how well you have actually analyzed and design a system in object-oriented terms depend on how as a developer you think object-oriented, how any concept that you come across, how quickly you can identify, see through and identify what is a key abstraction.

How quickly you can see that abstraction, how quickly you can see through that what is an appropriate state, what is the appropriate set of behavior for an object, what is should be a right hierarchy of specialization for this. The more of this you can reason out in your mind, you become a better programmer, you become a better developer for the object-oriented paradigm and

(Refer Slide Time: 25:54)



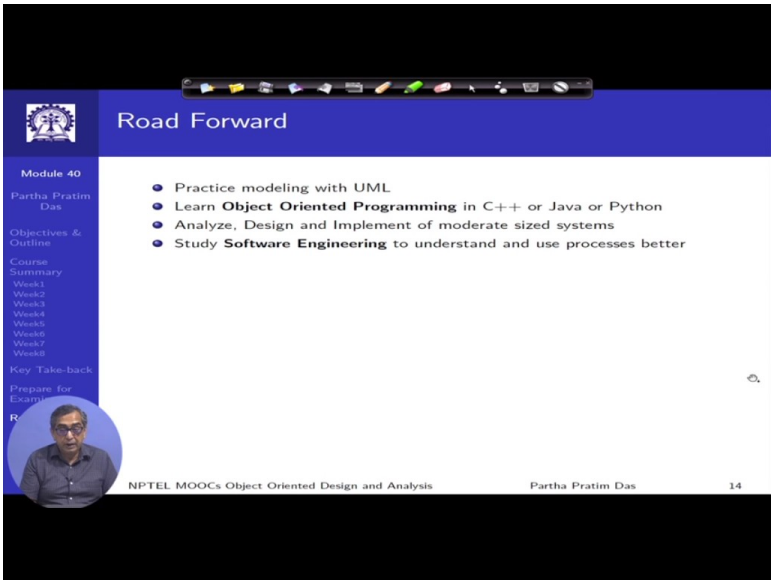
That I would emphasize is a key take back which you should focus on as you barge into the last couple of weeks of reading and practicing on this course to specifically prepare for your examination I will have a very classical advice that you certainly watch the videos, revise the assignments and solution. There is nothing new to that but certainly I would strongly advice that you practice that used and work out examples, this example I have used very extensively all through the presentations.

And this example has often been referred to in terms of the assignment. So I would strongly advise that you practice on these examples more and more, practice, try to do all kinds of diagram for them, try to do all kinds of analysis for them, try to look into the systems in multiple different ways that you would like to do in a multi-faceted manner, in multiple levels. Besides these are some of the systems for which we will we have prepared the small system descriptions, postal management system; story management system and a car management system.

So these are also like one and a half 2 page description. So we will soon upload this pdf files and I would suggest that you can use them as additional examples. if you have practiced with the leave management system watching the video, then it would be good to pick up one of the systems and try to do the similar analysis and try to build similar models, uml models based on these sample example systems also if you need to refer to books.

Then these are the 2 books certainly this is the most important book the Grady Booches book on OOAD which we have been following from the beginning particularly for the first half of the course where we discuss about different object-oriented concepts, fundamentally much of the material is actually from this book. So it will help if you refer to this book but in addition I would suggest that if you have questions on uml, you could also refer to the learning uml 2.0 I found this quite a nice book to follow.

But just to say along with that there are several sites which have very good information including the uml 2.5 diagram site which for which the URL you will find in several of our presentation slides. So you could also pick up for the uml you could also pick up and work from that.
(Refer Slide Time: 28:21)

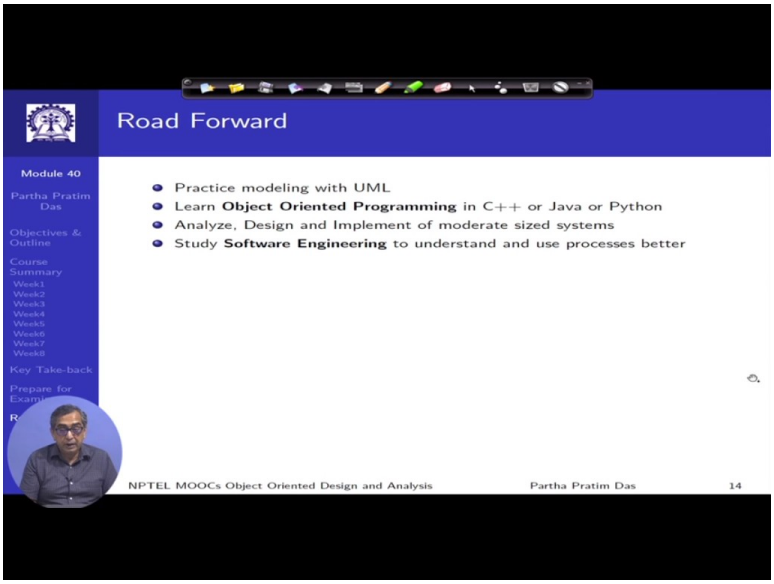


Finally going forward that is beyond this course if you would like to study further understand further then certainly kind of things that you can do is can practice modeling with uml. Uml is very very vast as we have seen; it has taken us I mean really about 15, 16 more modules to even discuss introduce you to all these diagrams. So it will take you much longer to practice and get familiar with the nuances of uml and how powerfully you can use that.

But certainly if you do that it will turn out to be something very good in terms of your you as a as a developer, I would say as a developer of software systems besides that it is good when you are looking at analysis and design it is also good to look at the possible ways to implement. So I would strongly advise that you take also take courses in object-oriented programming these are my 3 recommendations in the order of preference I would suggest that you do a course of OOP in C++ or in java or even parallely on python and practice them.

Certainly there is in terms of analysis and design is nothing better than actually being hands-on. so do analyze, design and implement moderate size systems so that you can know through your experiences to what works and what does not work and at a second level, I would also suggest that you can consider studying software engineering more formally which will have a significant part of object oriented analysis and design but you can study software engineering more formally to understand and use processes better in terms of exercising the object-oriented analysis and design practices.

(Refer Slide Time: 30:21)



So to summarize I would simply say that the course on object-oriented analysis and design is hereby concluded. I wish you all the very best for your forthcoming examination if you are taking it and certainly for your future endeavor with software systems analysis, design and the application of the uml diagram as well as the different approaches of OOAD that we have discussed here.

THIS BOOK IS NOT FOR SALE NOR COMMERCIAL USE

PH: (044) 2257 5905(08)

nptel.ac.in

swayam.gov.in