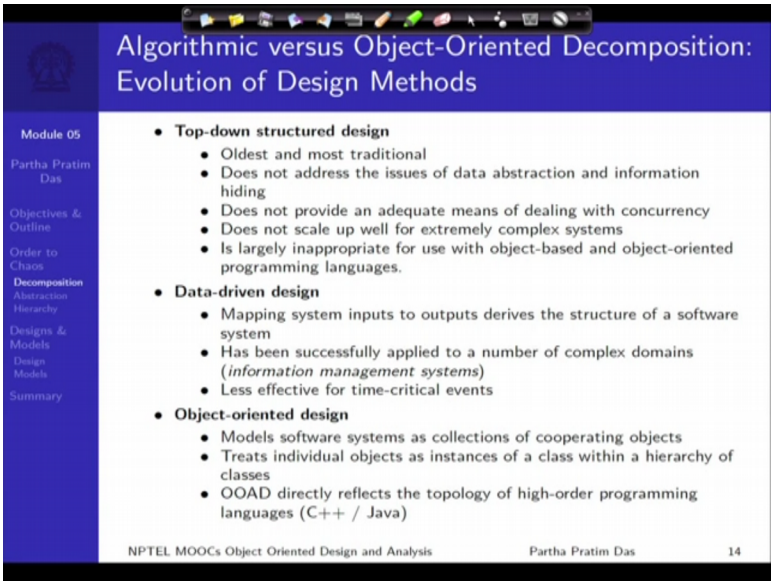
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 08
Bringing Order to Chaos

Welcome to module 5 of object-oriented analysis and design. We have been discussing about the rule 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 it 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 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 that 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 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 um 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.

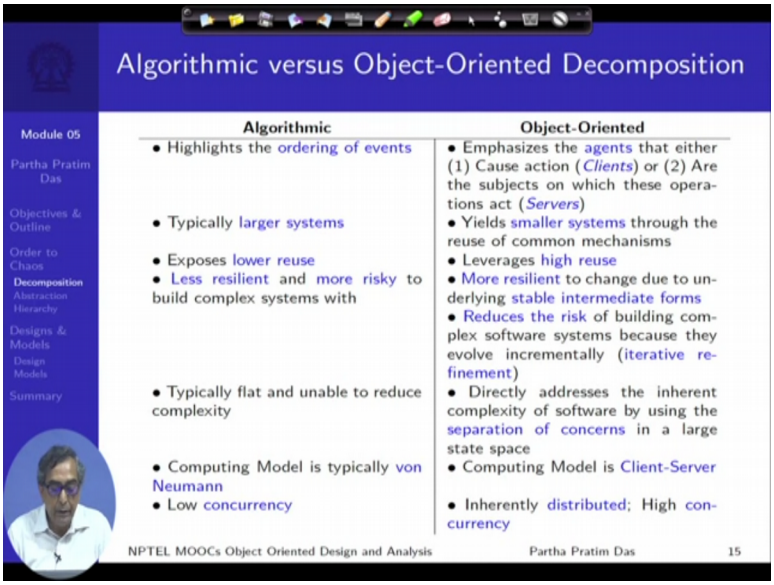
So, data-driven design significantly had a fall out in terms of variety of information management systems those which have been significantly based on variety of database systems and so on but naturally they are not very good in terms of f or effective in terms of time critical events time critical systems 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 design 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 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.

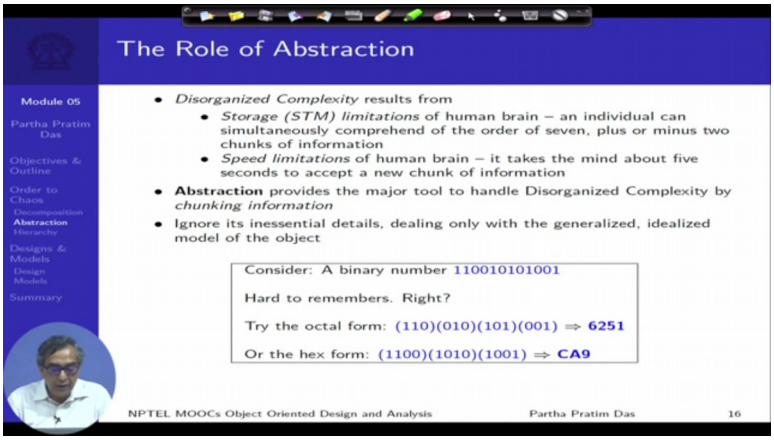
Which is inherently distributed as a high concurrency and therefore in large number of aspects object oriented decomposition has a really good benefit but before we move on just one word of caution please do not think that algorithmic decomposition is useless. After you have come to a certain level of 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.

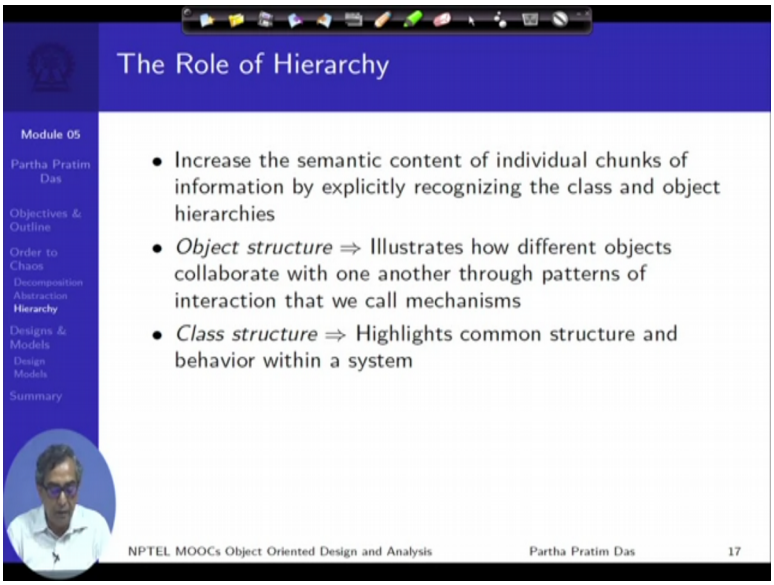
It takes a whole lot of time to process new chunk of information irrespective of what that chunk is. These are all psyhc psychologically proven data that you can you can use and therefore to address the disorganized complexity, only major tools that we have in hand is abstraction which basically what 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)

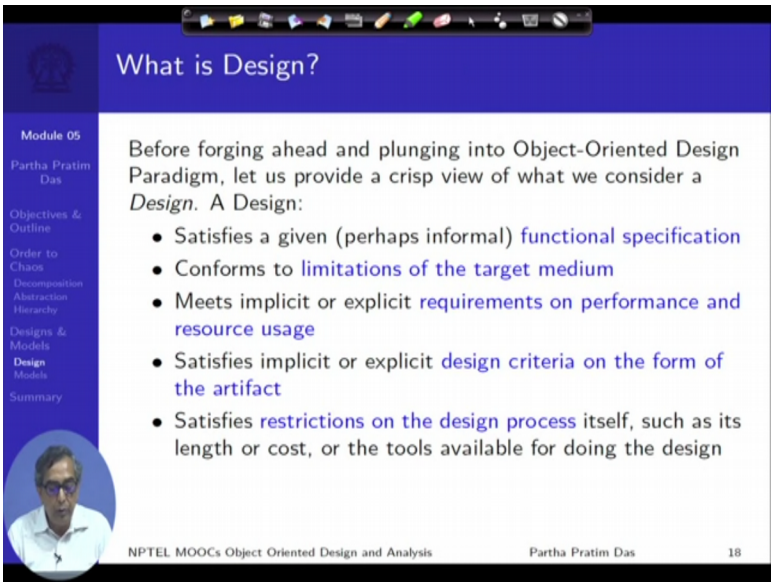


Finally the design process benefit a lot from the role of hierarchy we have already seen different hierarchies in the canonical form and again it can increase the semantic content of individual chunks by explicitly recognizing the hierarchy structures and we have already observed that we have already 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

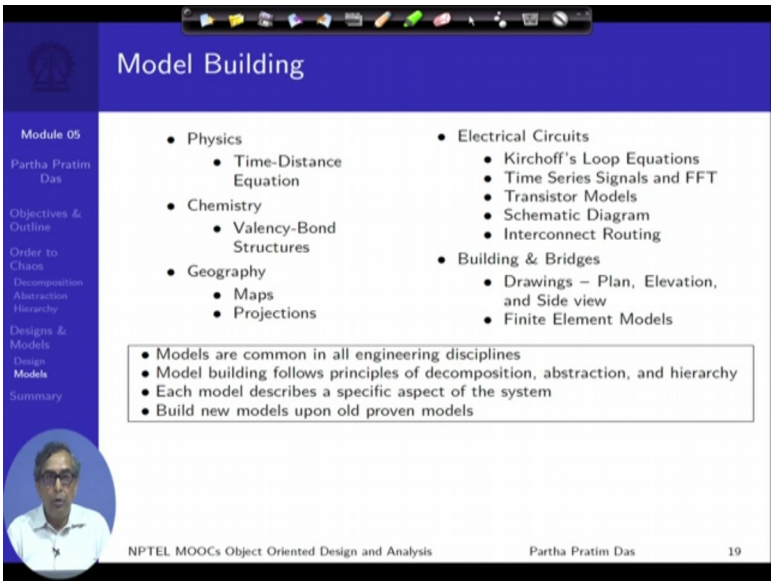
bandwidth and all these are available for realizing that system. So that is the question of target medium. The third it must meet implicit or explicit requirements of performance and resource usage.

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 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 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 done 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 model in different domains so I have just on the left.

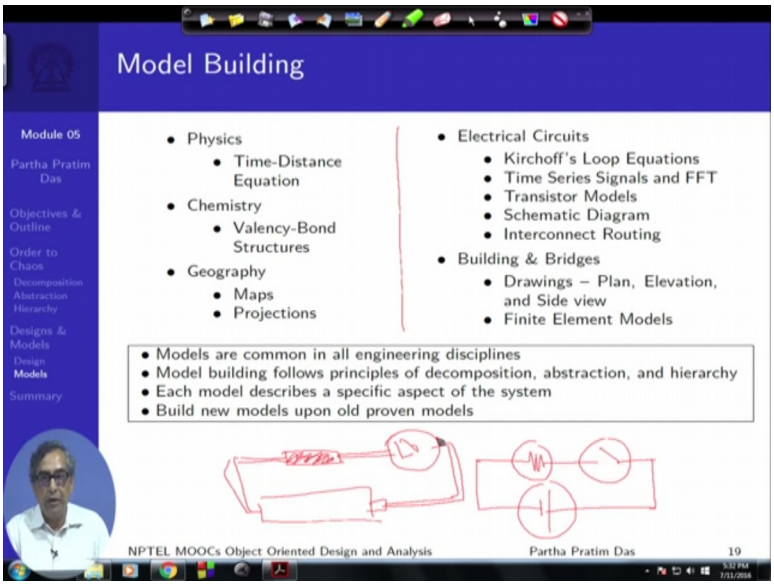
Specifically I have kept some school level generic domains like in physics we have talked about time distance equation that is if I know that a train has been moving at 10 kilometers per second, ab a rather 10 kilometers per per minute and then has been accelerating at a certain rate then our if after 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 valence 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.

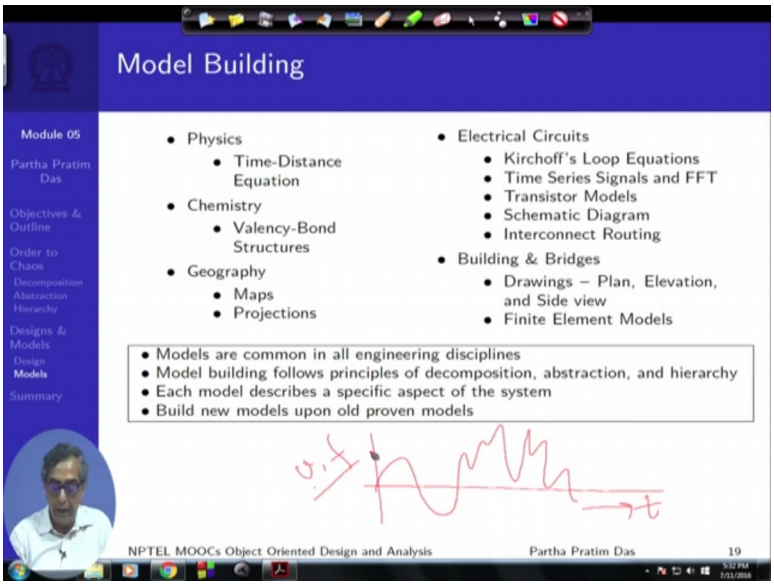
Let's talk about electrical circuits. Now if you want to describe the electrical circuit, you can do it in several different ways. For example you could just do a schematic diagram.

(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 sin 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 the terms of arri arriving at that abstraction we can decompose the models further.

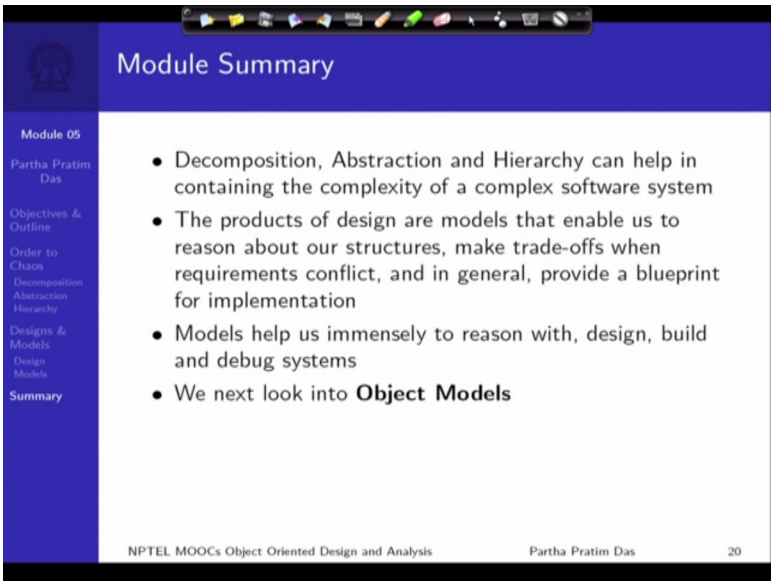
We can put them in hierarchal terms, we have often drawn electrical drawings where you will have different subsystems drawn at different levels of details and as you go deeper into a subsystem, you do more and more detailed models and at the top level you just have this is the input, output of different 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 is 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 is chemistry, certainly whole of mathematics is about that we have been doing this in geography, in electrical engineering, in mechanical engineering everywhere. So, an every model will describe a specific aspect of the system and this model based approach has a big advance 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 he 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.