

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)

The screenshot shows a presentation slide with a blue header and a white body. The header contains the text 'Module Objectives'. The body contains a list of three bullet points: 'Understand the major elements / essential characteristics of Object Models', 'Understand *Modularity*', and 'Understand *Hierarchy*'. On the left side, there is a vertical sidebar with a blue background and white text. The sidebar lists the following items: 'Module 09', 'Partha Pratim Das', 'Objectives & Outline', 'Elements of Object Model', 'Modularity', 'Objectives & Challenges', 'Intelligent Modularization', 'Hierarchy', 'Types of Inheritance', and 'Summary'. In the bottom right corner of the slide, there is a small circular video inset showing a man with glasses and a blue shirt. At the bottom of the slide, there is a footer with the text 'NPTEL MOOCs Object Oriented Design and Analysis' and 'Partha Pratim Das'.

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)



Module Outline

Module 09

Partha Pratim Das

Objectives & Outline

Elements of Object Model

Modularity

Objectives & Challenges


Intelligent Modularization

Hierarchy

Types of Inheritance

Summary

- Elements of Object Model – Recap
- Modularity
 - Objectives & Challenges
 - Intelligent Modularization
- Hierarchy
 - Types of Inheritance




NPTEL MOOCs Object Oriented Design and Analysis

Partha Pratim Das

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

(Refer Slide Time: 02:16)



Elements of Object Model – Recap

Module 09

Partha Pratim Das

Objectives & Outline

Elements of Object Model

Modularity

Objectives & Challenges

Intelligent Modularization

Hierarchy

Types of Inheritance


Summary

There are **four major** (*mandatory and essential*) elements of the object model:

- 1 Abstraction
- 2 Encapsulation
- 3 **Modularity**
- 4 **Hierarchy**

In addition, there are **three minor** (*useful but not essential*) elements of the object model:

- 1 Typing
- 2 Concurrency
- 3 Persistence



NPTEL MOOCs Object Oriented Design and Analysis

Partha Pratim Das

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)

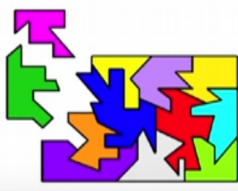
mod02lec14

Modularity

Module 09
Partha Pratim Das

Objectives & Outline
Elements of Object Model
Modularity
Objectives & Challenges
Intelligent Modularization
Hierarchy
Types of Inheritance
Summary

- **Modularity** is to *partition a program into individual components to manage complexity*
- Modules
 - Compile separately or together
 - Connected with the other module



NPTEL MOOCs Object Oriented Design and Analysis 2:31 / 26:14 Partha Pratim Das

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.

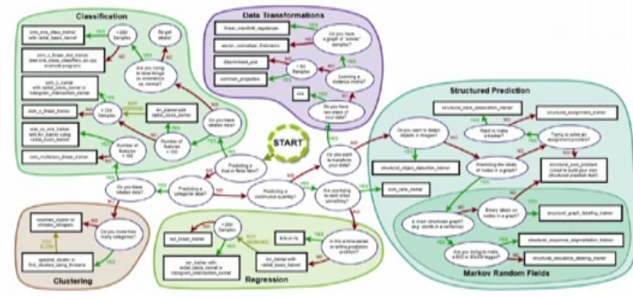
And the inherent inability of human beings to deal with more than a few semantic elements in any design or any implementation. A typical characteristic of a modular designs or module, modules themselves would be that they compile separately doing cases, they could compile together and obviously there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

(Refer Slide Time: 03:50)

Example of Modularity

Module 09
Partha Pratim Das

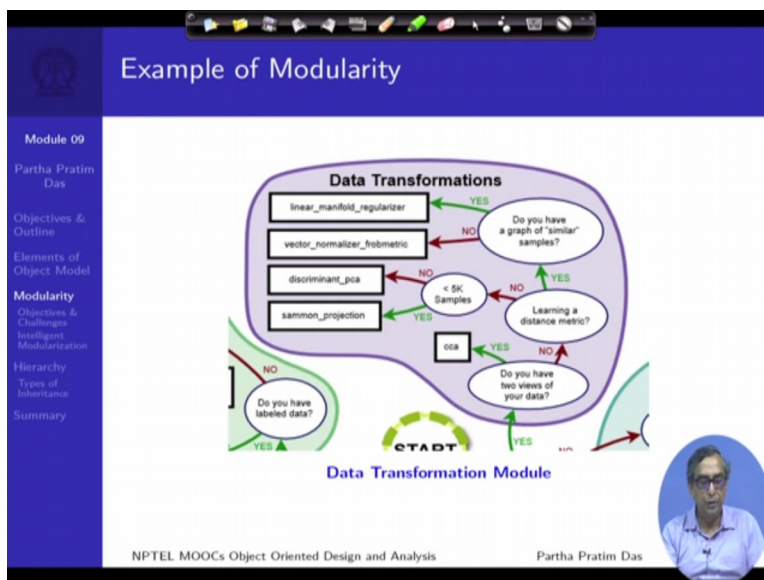
Objectives & Outline
Elements of Object Model
Modularity
Objectives & Challenges
Intelligent Modularization
Hierarchy
Types of Inheritance
Summary



Modular decomposition of Machine Learning System

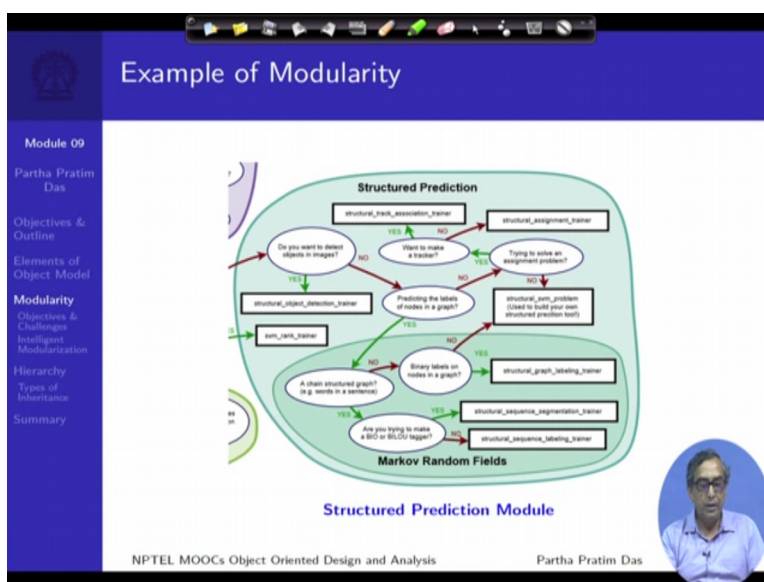
NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das 6

So, will start with an example so this is please do not get worried about the clutter about 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 a different components that you need to deal with and these different sets shown in different colors show the different models. So I will quickly take a walk around the modules. (Refer Slide Time: 04:25)



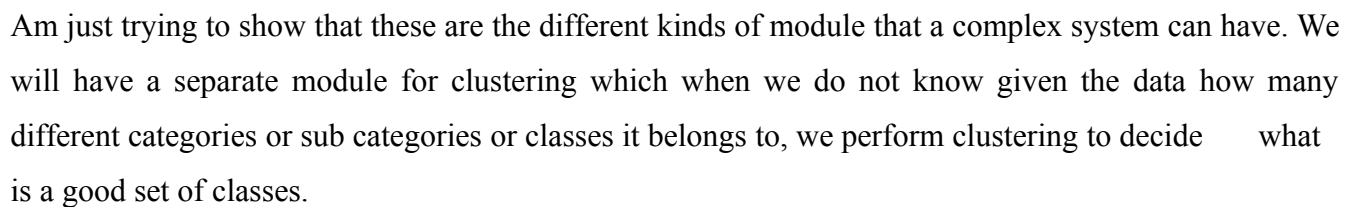
So, one module is data transformation where what we need to do is in whatever form we are getting the data for machine learning, we have to transform them, represent them into different standard forms of a linear manifolds or vector manifolds and so on so all components relating to this transformation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

(Refer Slide Time: 04:54)

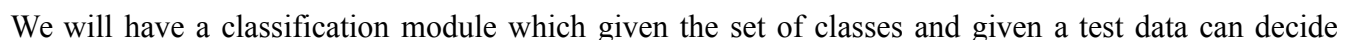


A second module or another module is structured prediction which talks about given a certain pattern of

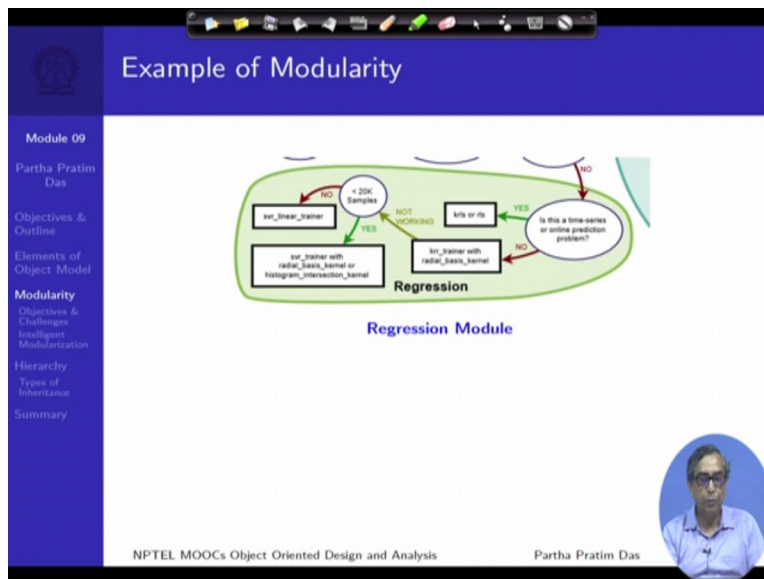
(Refer Slide Time: 05:25)



(Refer Slide Time: 05:46)

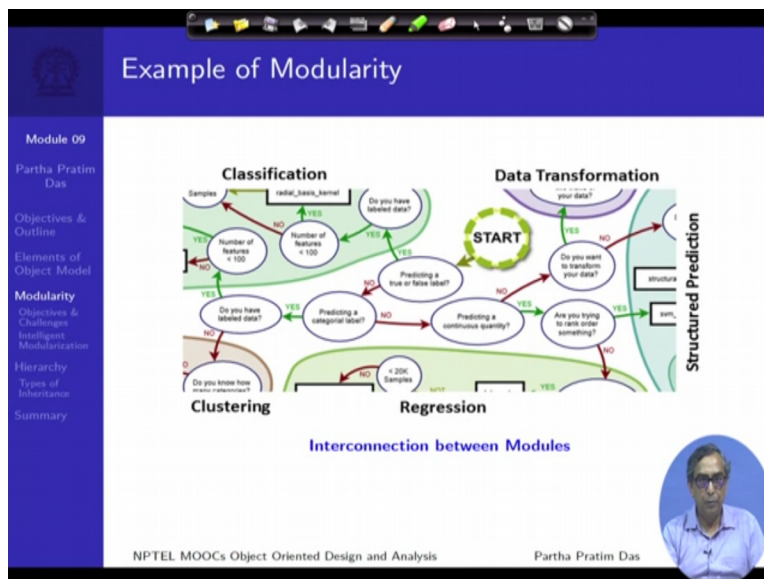


which of the classes it does belong to using different machine learning algorithms
(Refer Slide Time: 05:59)



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 the 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 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 is just to give you a glimpse of the glimpse of what kind of the complexity the 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)

The slide is titled "Objectives of Modularity" and is part of "Module 09" by Partha Pratim Das. It lists five main objectives of modularity, each with sub-points:

- **Modular Decomposition**
 - Systematic mechanism to divide problem into individual components
- **Modular Composability**
 - Enable reuse of existing components
- **Modular Understandability**
 - Understand module as a unit
 - Easy to change
- **Modular Continuity**
 - Small changes in system requirements result in changes in individual module rather the whole system
 - Reduces Side-effects
- **Modular Protection**
 - Errors are localized and do not spread to other m

The slide also includes a sidebar with navigation links: Module 09, Partha Pratim Das, Objectives & Outline, Elements of Object Model, Modularity, Objectives & Challenges, Designing Modularization, Hierarchy, Types of Inheritance, and Summary. At the bottom, it mentions "NPTEL MOOCs Object Oriented Design and Analysis" and "Partha Pratim Das". A small circular video inset of the speaker is visible in the bottom right corner.

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 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 composability that is as we divide the system into smaller modules, every modules 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 various 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 to include different standard library headers you say this is we have `stdio.h`, we have to include `math.h`, we have to 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 views of modularity, very organized use of modularity.

So what are we doing? Why 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 composability. 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 `atan` 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 through 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 `atan` 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 composability 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 `stdlib.h` and so on so forth. So, this is just a simple example relating to basic programming that shows that modularity is 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 once 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 had 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 a 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 we are given to deal with.

(Refer Slide Time: 15:46)

Challenges of Modularity

Module 09
Partha Pratim Das

- Decomposition into smaller module may be difficult
- Arbitrary modularization is sometimes worse than no modularization at all

Modularization should follow the semantic grouping of common and interrelated functionality of the system

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

Now certainly in life no lunch is free so when you engage in modularization try to meet this objective 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 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 happen.

And keep in mind that if the modularization is arbitrary then often it turns out to be actually more disastrous 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

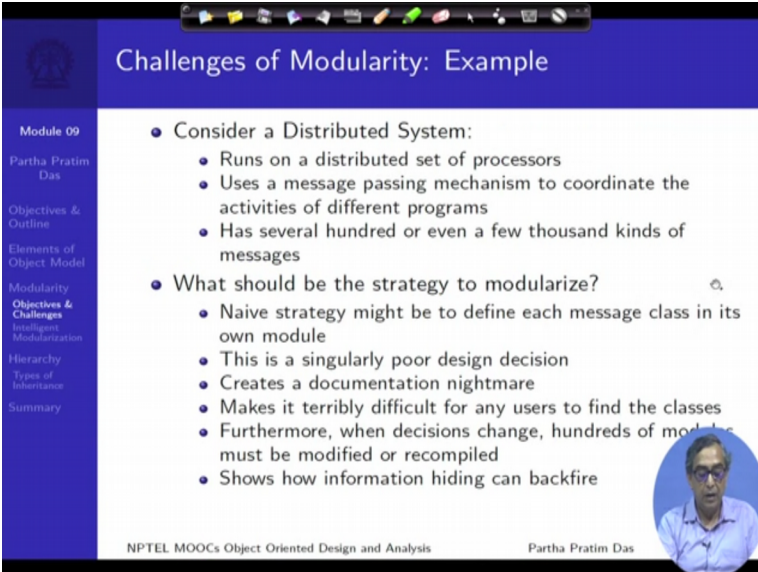
functionality.

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 `ma stdio.h`.

Now there are functions which some of you may know 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 grid 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, error 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)



The slide is titled "Challenges of Modularity: Example" and is part of Module 09. It lists two main challenges:

- Consider a Distributed System:
 - Runs on a distributed set of processors
 - Uses a message passing mechanism to coordinate the activities of different programs
 - Has several hundred or even a few thousand kinds of messages
- What should be the strategy to modularize?
 - Naive strategy might be to define each message class in its own module
 - This is a singularly poor design decision
 - Creates a documentation nightmare
 - Makes it terribly difficult for any users to find the classes
 - Furthermore, when decisions change, hundreds of modules must be modified or recompiled
 - Shows how information hiding can backfire

The slide also includes a sidebar with navigation links: Module 09, Partha Pratim Das, Objectives & Outline, Elements of Object Model, Modularity, Objectives & Challenges, Intelligent Modularization, Hierarchy, Types of Interactor, and Summary. At the bottom, it mentions NPTEL MOOCs Object Oriented Design and Analysis and Partha Pratim Das.

In terms of challenges I have just tried to describe the 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 the what should you

modularize. Should you modularize since every module must be as independent as possible you will put every modules 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 the 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)

Challenges of Modularity: Example

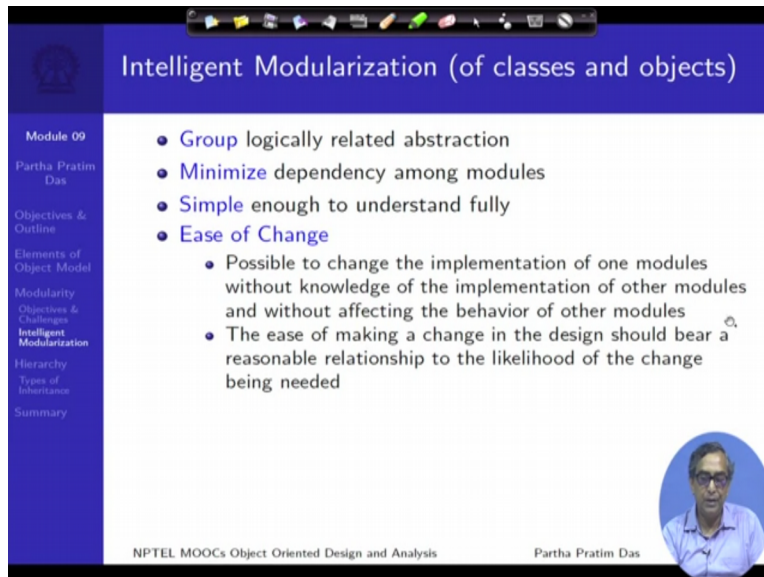
Module 09
Partha Pratim Das
Objectives & Outline
Elements of Object Model
Modularity
Objectives & Challenges
Modularity
Hierarchy
Types of Inheritance
Summary

- Consider a Distributed System:
 - Runs on a distributed set of processors
 - Uses a message passing mechanism to coordinate the activities of different programs
 - Has several hundred or even a few thousand kinds of messages
- What should be the strategy to modularize?
 - Naive strategy might be to define each message class in its own module
 - This is a singularly poor design decision
 - Creates a documentation nightmare
 - Makes it terribly difficult for any users to find the classes
 - Furthermore, when decisions change, hundreds of modules must be modified or recompiled
 - Shows how information hiding can backfire

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

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)



The slide is titled "Intelligent Modularization (of classes and objects)". It features a blue sidebar on the left with a table of contents including "Module 09", "Partha Pratim Das", "Objectives & Outline", "Elements of Object Model", "Modularity", "Objectives & Challenges", "Intelligent Modularization", "Hierarchy", "Types of Inheritance", and "Summary". The main content area lists four guidelines for intelligent modularization:

- Group logically related abstraction
- Minimize dependency among modules
- Simple enough to understand fully
- Ease of Change
 - Possible to change the implementation of one modules without knowledge of the implementation of other modules and without affecting the behavior of other modules
 - The ease of making a change in the design should bear a reasonable relationship to the likelihood of the change being needed

At the bottom right, there is a circular portrait of Partha Pratim Das. The footer text reads "NPTEL MOOCs Object Oriented Design and Analysis" and "Partha Pratim Das".

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 module 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 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 no 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 make every module simple.

And now 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 have to 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 whatever do a design when we start doing a implementations that organizing the code, we are confident that we are doing a correct job but experience shows that for different limitations of the human mind .of the system and of the business scenario change has an inevitable or 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)

Modularization:
Example – *Hydroponics Gardening System*

Module 09
Partha Pratim Das

Objectives & Outline
Elements of Object Model
Modularity
Objectives & Challenges
Intelligent Modularization
Hierarchy
Types of Abstraction
Summary

- Suppose we decide to use a commercially available workstation where the user can control the system's operation
- At this workstation, an operator could create new growing plans, modify old ones, and follow the progress of currently active ones
- Key Abstraction
 - Growing plan
 - User Interface

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

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 for 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 module is 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)

The slide is titled "Hierarchy" in a blue header. Below the header, on the left, is a vertical navigation menu with the following items: "Module 09", "Partha Pratim Das", "Objectives & Outline", "Elements of Object Model", "Modularity", "Objectives & Challenges", "Modularization", "Hierarchy", "Types of Inheritance", and "Summary". The main content area has a blue background and contains the text "Hierarchy is a *ranking or ordering of abstractions*". Below this text are two diagrams: (a) a 3D pyramid with five horizontal layers in red, yellow, green, blue, and purple from bottom to top; and (b) a black and white line drawing of a multi-tiered cake with a person standing next to it, holding a large spoon. At the bottom of the slide, there is a small circular video inset of a man with glasses and a blue shirt. The footer of the slide contains the text "NPTEL MOOCs Object Oriented Design and Analysis" on the left and "Partha Pratim Das" on the right.

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.