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

Lecture – 22
Classification of Coupling

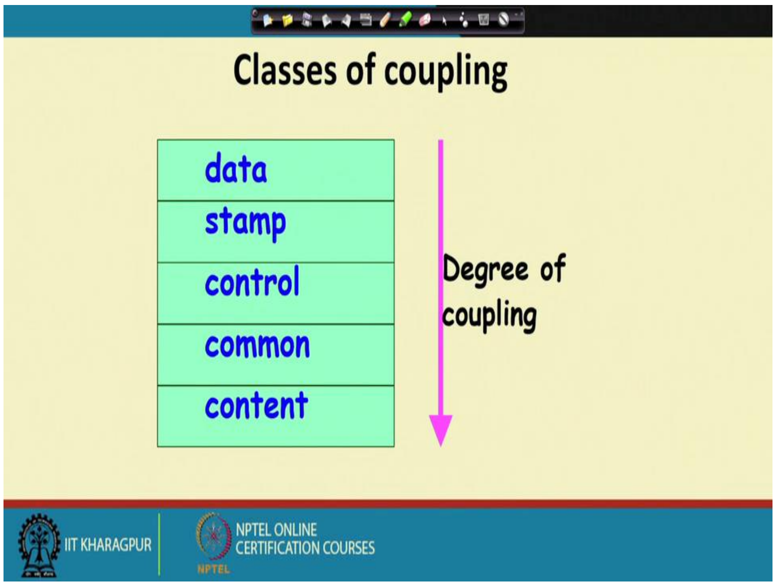
Welcome to this lecture. In the last lecture, we are trying to classify the cohesiveness and the coupling existing between different modules, because that will give us a hint is to how good is a design?

We saw how to check whether a design is functionally independent or not? If, it is a functionally independent design then we will have a case of very high cohesion and low coupling.

In the last lecture we had discussed that by looking at a module structure can we identify, what is the class of cohesion that exists in the module? For that we looked at seven classes of cohesion and we said that by examining the functions in module, we can say approximately which class of cohesion does the module belong or the module has which class of cohesion. And, then we are trying to do the same thing with respect to coupling between two modules. And, we said that coupling is the degree of dependence between two modules or the degree to which they interact or exchange data with each other. If there is no interaction between two modules then there is no coupling.

If, they interact only by exchanging some very simple data items like an integer or a character, then there is very low coupling called as a data coupling. But if they interact by exchanging complex data like; arrays, trees, lists, very complex structures and so on then we called it as a stamp coupling. Now, let’s proceed that can we by looking at the structure of two modules that are given to us, can we say that how good or bad is the coupling between these two modules? So, let’s proceed from that point.

(Refer Slide Time: 02:41)

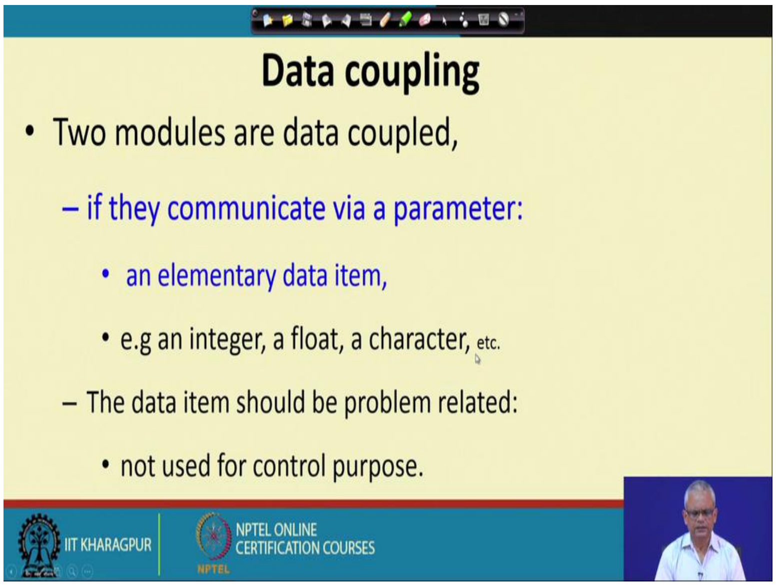


Classes of coupling

data
stamp
control
common
content

We said that the coupling existing between two modules can be classified into five classes: data coupling, stamp coupling, control coupling, common coupling and content coupling. Content coupling is the worst form of coupling and nowadays the modern programming languages, they don't allow programmers to write program, where module have content coupling. Now, let's look at these different forms of coupling and we even look at the case of content coupling just to be aware of the issues involved.

(Refer Slide Time: 03:24)



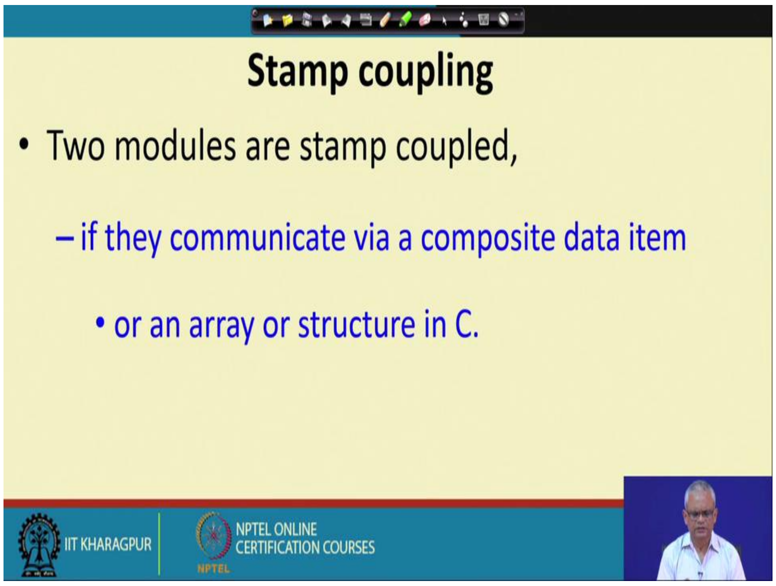
Data coupling

- Two modules are data coupled,
  - if they communicate via a parameter:
    - an elementary data item,
    - e.g an integer, a float, a character, etc.
  - The data item should be problem related:
    - not used for control purpose.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

In data coupling, two modules they interact, but then they exchange only very simple data items, for example, just an integer, a two integers or may be a floating point number, may be some character and so on. And of course, one thing we must be sure that these are used like data members in the other module to which it is invoked and this should not be used to set a control parameter in the other module. In that case, we call it as a data coupling.

(Refer Slide Time: 04:11)



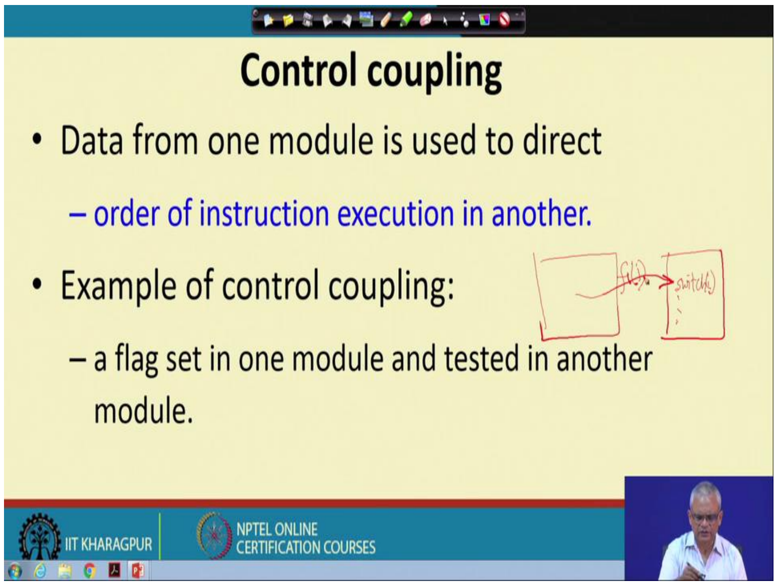
Stamp coupling

- Two modules are stamp coupled,
  - if they communicate via a composite data item
    - or an array or structure in C.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Slightly worse form of coupling is the stamp coupling. This also easy to identify a given a module structure whether it have stamp coupling or not. If, we find that one module invokes another module and interacts by passing or receiving a complex data structure like an array, linked list, structure et cetera. Then, we say that it is a case of stamp coupling.

(Refer Slide Time: 04:44)



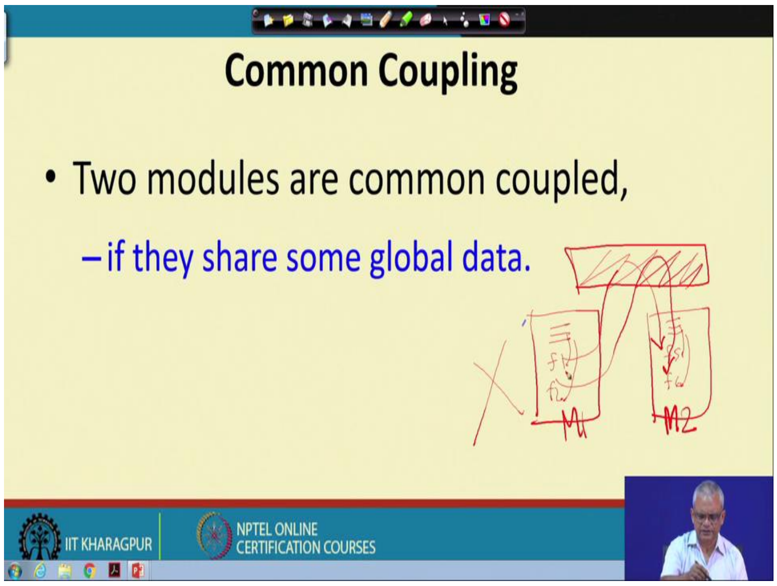
Control coupling

- Data from one module is used to direct
  - order of instruction execution in another.
- Example of control coupling:
  - a flag set in one module and tested in another module.

f(i)
switch(i)
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

A still worse form coupling is the control coupling. Here, the two modules pass data items between each other, but then the data item passed by one module is used as a control element in another module. For example, it may they send an integer which is used as a flag tested in another module. So, let say we have two modules here. And, let say this module invokes a function let say f1 and f1 passes an integer i and here this integer i is switch on integer; one of the functions switches on integer i and depending on the integer value i, it does different things. So, the value that is passed here is used as a control element in the other module. And, this also a bad case of coupling. It makes understanding the design very difficult and we call this as a case of control coupling.

(Refer Slide Time: 06:19)

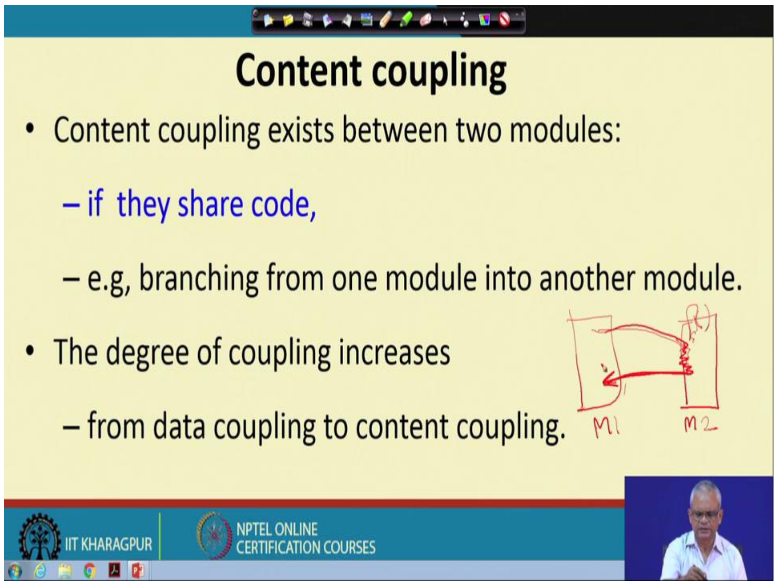


Common Coupling

- Two modules are common coupled,
  - if they share some global data.

Another bad form of coupling is the common coupling. Here two modules are coupled, but they are coupled through some global data. Normally, if we have a module here and another module here (on the above slide). These two module have their own data and the different functions. Let say one module is M1 and another module is M2 (on above slide) and these module has some data that is visible within the module. Let say there are functions f1, f2 in module M1 and let say, f5, f6 functions in M2. Now, let say there have some global and these two module us those global data. And, different functions within the module access the global data and communicate among by using the global data. So, this type of coupling is bad case of coupling, where, we have the modules communicating with the help of some global data. This type of coupling we call as the common coupling and it’s a bad case of coupling. If there is a common coupling, then we say that the modules are highly coupled.

(Refer Slide Time: 07:53)



Content coupling

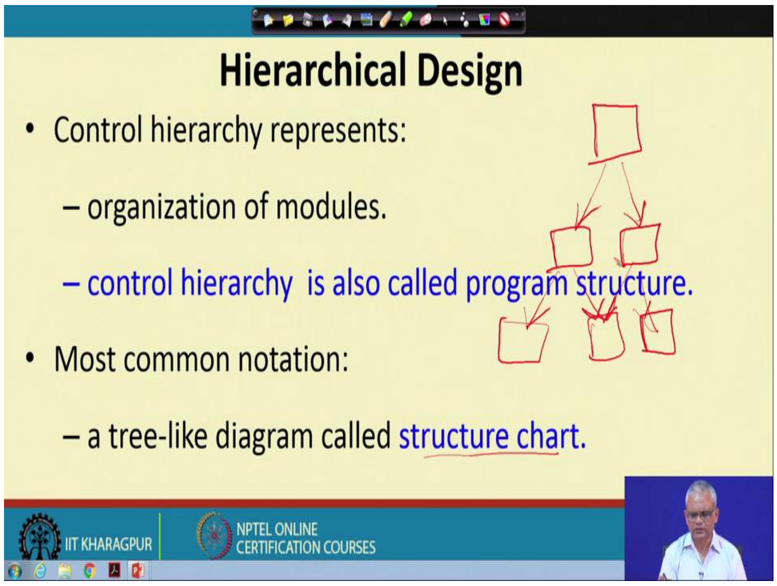
- Content coupling exists between two modules:
  - if they share code,
  - e.g, branching from one module into another module.
- The degree of coupling increases
  - from data coupling to content coupling.

f()
M1
M2
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

And, the worst form of coupling is the content coupling. Modern programming languages, does not allow this kind of coupling. So, you cannot even write a code using modern programming languages, where they do have content coupling. But earlier in machine language and assembly language program in old computers, it was possible to have two modules which have content coupling.

For example, we have two module M1 and M2 (on the above slide) and these modules share code. It is possible for M1 to execute some piece of code of M2 and come back to its own code. So, just because we don’t want to write a certain amount of code in M1, M1 just had a jump here to M2 module and use the code of M2 and, then again jump back. So, this a very bad form of coupling, it makes understanding this module extremely difficult. And, it’s a bad programming which is not allowed in the modern languages. Using modern programming language, you cannot just jump into another module and executes some parts of another module. Let say, we have a function f1 in M2 and we jump into the middle of this function and run few statements of f1 and come back in M1. So, that form of thing is not permissible. Either you call the full function or don’t use anything here. You can not just jump into one module, run part of the function and come back. So, this is the case of a content coupling and largely disallowed in modern programming languages.

(Refer Slide Time: 09:46)



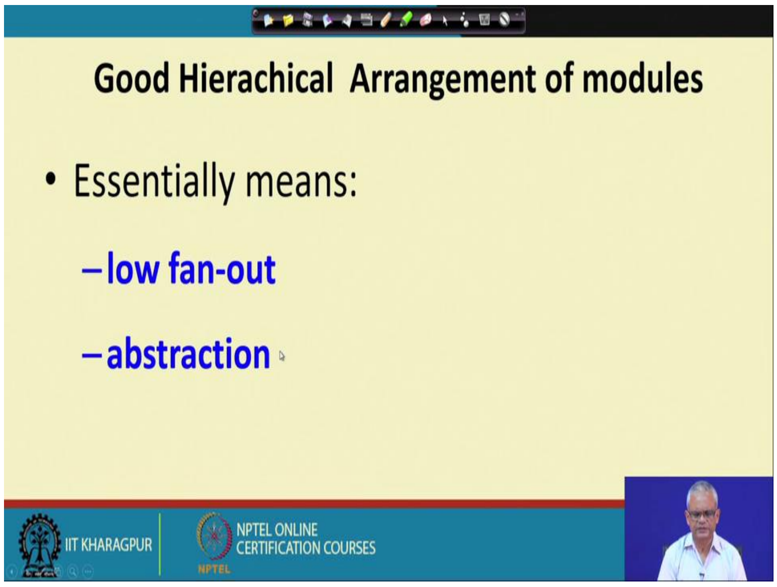
Hierarchical Design

- Control hierarchy represents:
  - organization of modules.
  - control hierarchy is also called program structure.
- Most common notation:
  - a tree-like diagram called structure chart.

Now, let’s look at other aspects of a good design: one is the hierarchal design. Here, we look at how the modules call each other and represent that as a control hierarchy (Diagram on the above slide). In the above diagram we can see, first top module calls below two modules and these two modules in turn they call other modules. And, this we call as the control hierarchy and this is also called as the program structure and typically represented by a structure chart.

We look at the structure chart in next couple of lectures. And, this program structure is a tree like structure and it’s hierarchical. Using a hierarchical structure a good design is organized. So, we say that a good design as a hierarchical module structure and it has a well-defined control hierarchy.

(Refer Slide Time: 11:05)

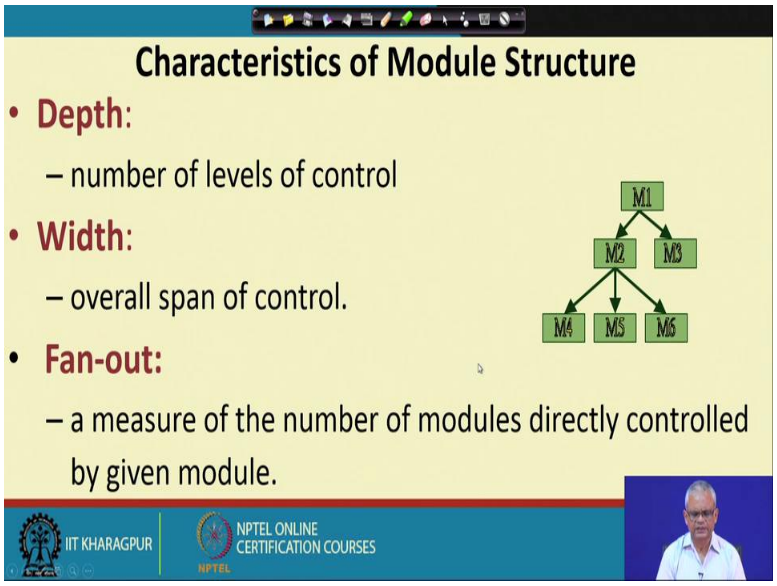

Good Hierachical Arrangement of modules

- Essentially means:
  - low fan-out
  - abstraction

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

But then given a control structure, how do we say that whether the control structure is good or bad? For that we need to define two terms: fan-out abstraction, fan-in abstraction. A good design should have low fan out, it should have abstraction, what we mean by that?

(Refer Slide Time: 11:33)


Characteristics of Module Structure

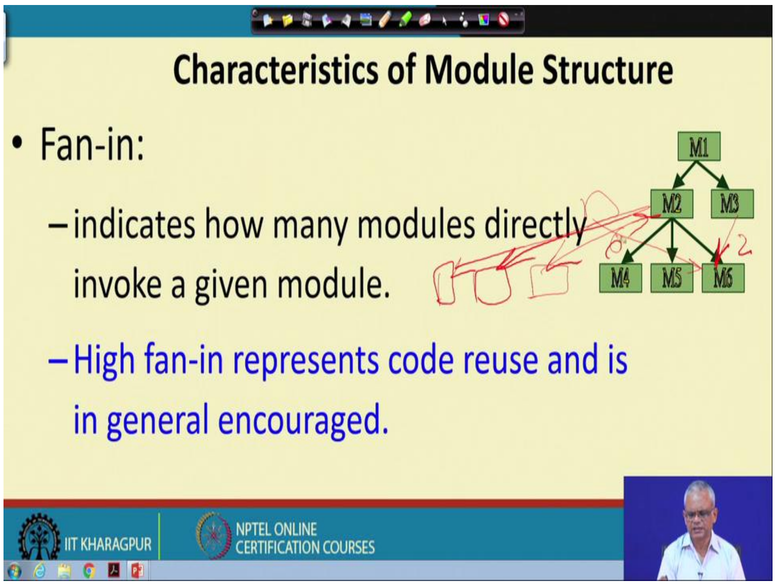
- Depth:
  - number of levels of control
- Width:
  - overall span of control.
- Fan-out:
  - a measure of the number of modules directly controlled by given module.

$$\begin{tikzcd}[column sep=small] & & M1 \arrow[dl] \arrow[dr] & \\ & M2 \arrow[dl] \arrow[d] \arrow[dr] & & M3 \\ M4 & M5 & M6 & \end{tikzcd}$$

Let say this is a module structure of an application (on the above slide). How they call each other? And, into how many layers they are organized or levels of controlled?

The fan-out of every module is a how many other modules it calls. For example, the module M1 fan-out is 2 here. For module M2 the fan-out is 3 here, for module M3 it is 0. So, the term fan-out says how many other module it invokes in a hierarchical structure.

(Refer Slide Time: 12:39)

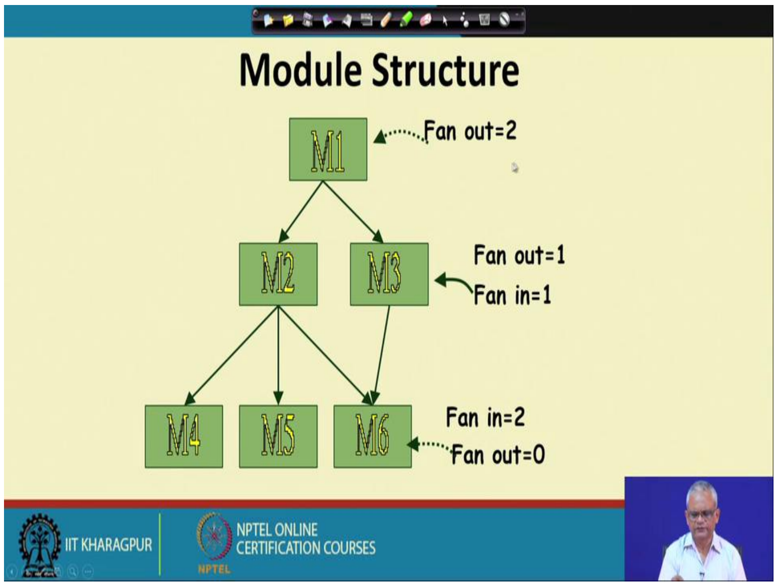


Characteristics of Module Structure

- Fan-in:
  - indicates how many modules directly invoke a given module.
  - High fan-in represents code reuse and is in general encouraged.

Now, what about fan-in? Fan-in for a module is how many different modules call this module. For example, here the fan-in of every module is 1. But let say we also had M3 calling module M6. Then the fan in of module M6 will be 2. Now, one thing to remember is that a very high fan-out is not good. Let say, we can see from the diagram that M2 fan-out is 3. If M2 calls let say another three modules, then we say the fan-out of module M2 is 6. A high fan-out is bad because M2 is depending on too many other modules. And, it’s a hint or it’s likely that M2 has a very bad cohesion. It just doing too many things and that’s why it is needing many modules. A high fan-out is bad, but a high fan-in is a good thing, because there are many modules which will depend on a high fan-in module. Let say there are other modules also, which are depending on M5. So, a high fan-in is good because it means that we are able to achieve reuse of code and that’s a good design. High fan-out is bad, but a high fan-in is good.

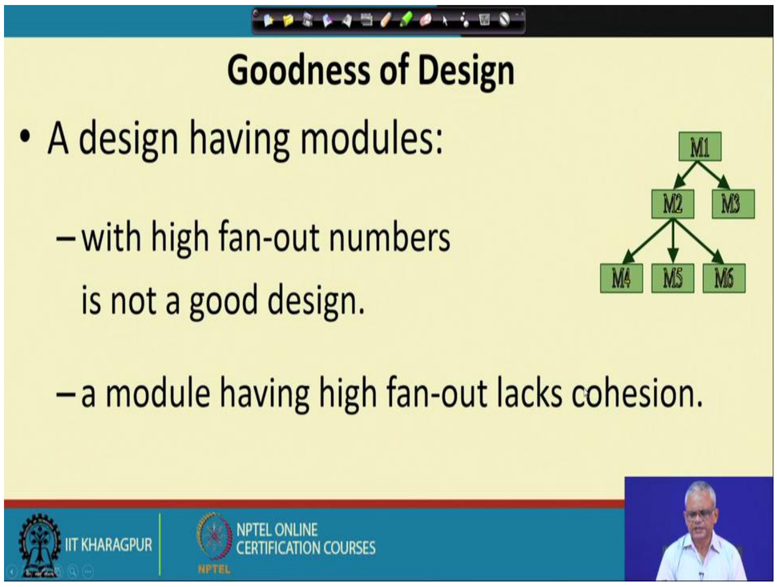
(Refer Slide Time: 14:35)



Module Structure

This is an example where the fan-out of this M1 is 2, for this M3 fan-out is 1 and fan-in is also 1, for M6 fan-out is 0 and fan-in is 2.

(Refer Slide Time: 14:53)



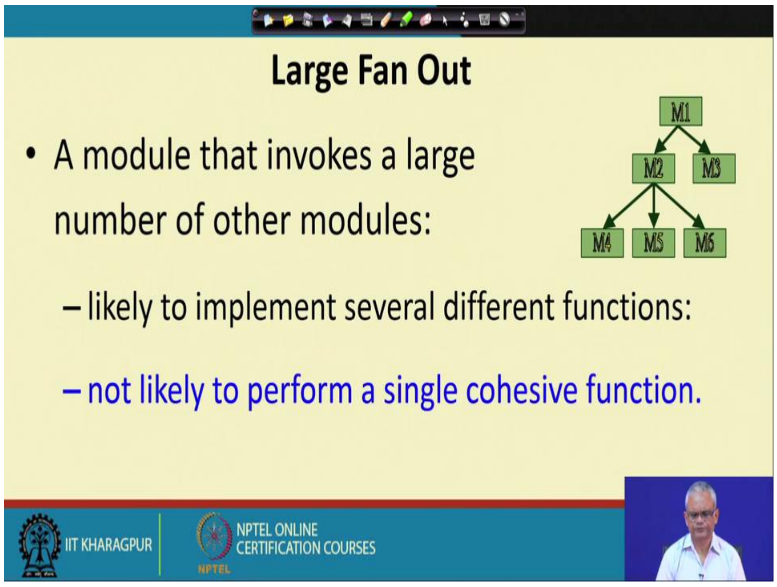
Goodness of Design

- A design having modules:
  - with high fan-out numbers is not a good design.
  - a module having high fan-out lacks cohesion.

M1
M2
M3
M4
M5
M6
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

We can compute the fan-in and fan-out and if we have a very high fan-out for some modules, we will say that it indicates that the design lacks cohesion. But if there is high fan-in then that’s not bad, it’s actually good design. If there is a high fan-in then good reuse can be observed in that application.

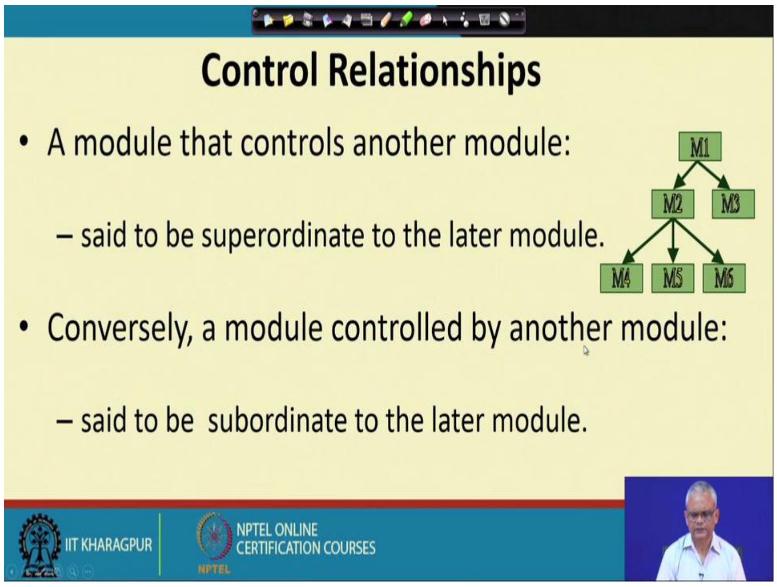
(Refer Slide Time: 15:31)


Large Fan Out

- A module that invokes a large number of other modules:
  - likely to implement several different functions:
  - not likely to perform a single cohesive function.

A large fan-out as we are mentioning is that it is a indication, that a module is doing very different things and it’s cohesion will be poor. Because, it’s needing too many other functions, it’s invoking them different modules.

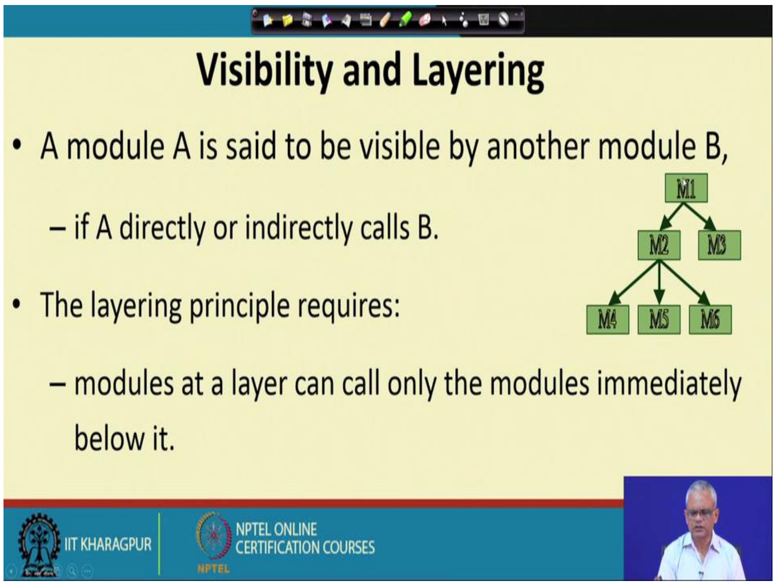
(Refer Slide Time: 15:54)


Control Relationships

- A module that controls another module:
  - said to be superordinate to the later module.
- Conversely, a module controlled by another module:
  - said to be subordinate to the later module.

There are few other terminology with respect to the control relationship. From the above diagram on slide, we can say that M1 is superordinate of M2 and M3 or we can say that M1 controls M2 and M3. In other words, we can say that M2 and M3 are subordinate of M1 and M4, M5 and M6 are subordinate of M 2.

(Refer Slide Time: 16:26)



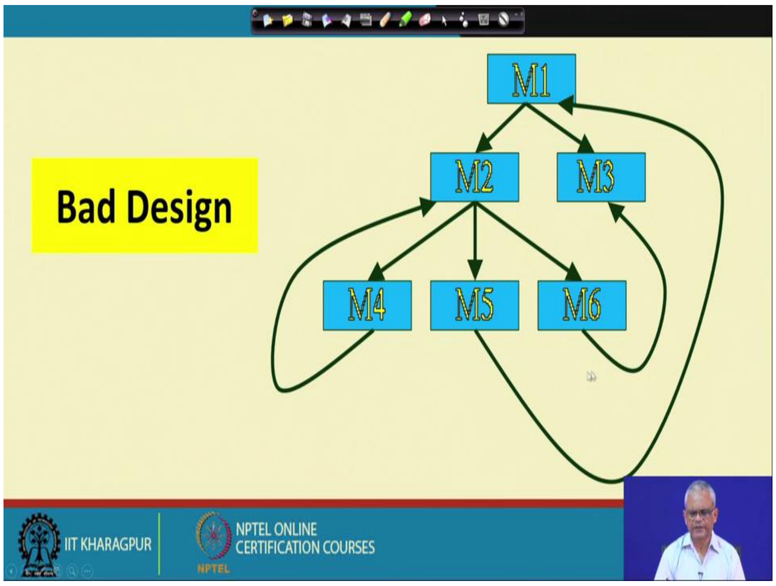
Visibility and Layering

- A module A is said to be visible by another module B,
  - if A directly or indirectly calls B.
- The layering principle requires:
  - modules at a layer can call only the modules immediately below it.

M1
M2
M3
M4
M5
M6
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

M1 calls M2 and M3 and therefore, M2 and M3 are visible to M1, but M1 is not visible to M2 and M3. Similarly, from M2 the visibility is M4, M5, and M6. So, we say that superordinate modules are actually abstractions to subordinate modules and they are not visible to subordinate modules. In our above diagram, M1 is the most abstract module and M4 is the most concrete module and this is a layered structure. First of all, to have a good design we must have a layered structure. That is the control hierarchy, we should be able to organize into distinct layers and then we can identify the fan-in, fan-out and so on.

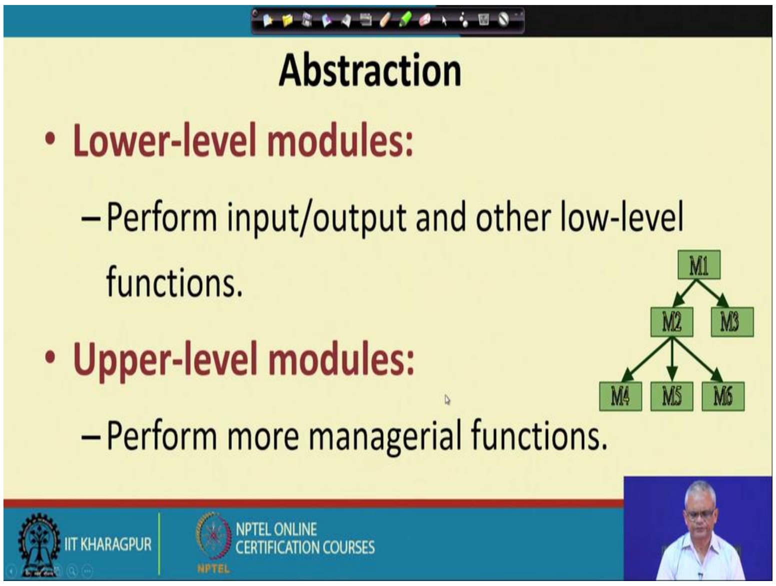
(Refer Slide Time: 17:38)



In the above slide, we can see a bad case of the design. First of all, there is no layering and also it's not a simple tree like structure and it violet the principle of abstraction. For example, in our previous design the most concrete module was M4, M5, and M6. And, they should not be able to see the higher-level modules M1 et cetera. They should not be visible here, but in this diagram, M5 is invoking M1, so M1 is visible to M5. So, this will be a very bad case of design where it will be very difficult to debug, because we can see from here that it has some cyclical relation. For cyclical relation, it may happen, for a given bug, we may just go round and round and still not be able to trace the bug.

But, if it was a strict tree like structure with layering. If we observe a bug in any module then that is either on that module or with it is subordinate modules. So, understandability, debugging, reuse et cetera are facilitated by a layered design. And, if we have a design like cyclic relation which violates the principle of abstraction, layering and it's a case of bad design.

(Refer Slide Time: 19:16)



Abstraction

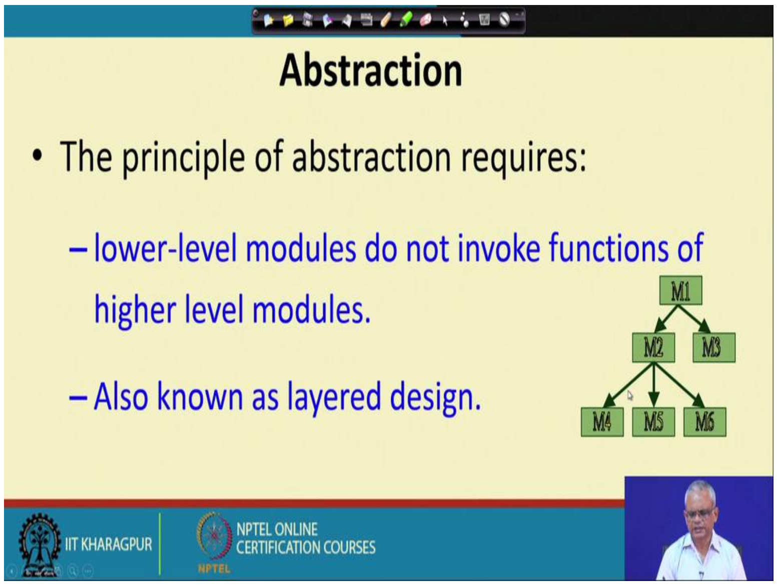
- Lower-level modules:
  - Perform input/output and other low-level functions.
- Upper-level modules:
  - Perform more managerial functions.

$$\begin{tikzcd} & M1 \arrow[dl] \arrow[dr] & \\ M2 \arrow[d] \arrow[dl] \arrow[dr] & & M3 \\ M4 \quad M5 & M6 & \end{tikzcd}$$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

In a good design, the modules are layered and the top level modules are the abstract modules. Top level modules do more managerial function, they just invoke the lower level modules to do some work. Top level modules invoke some other modules which we called as the middle managers. Middle managers again invoke other modules which are the lowest level modules. Lowest level modules do input output. So, lowest level modules take some input and after processing the input, give some output.

(Refer Slide Time: 20:01)

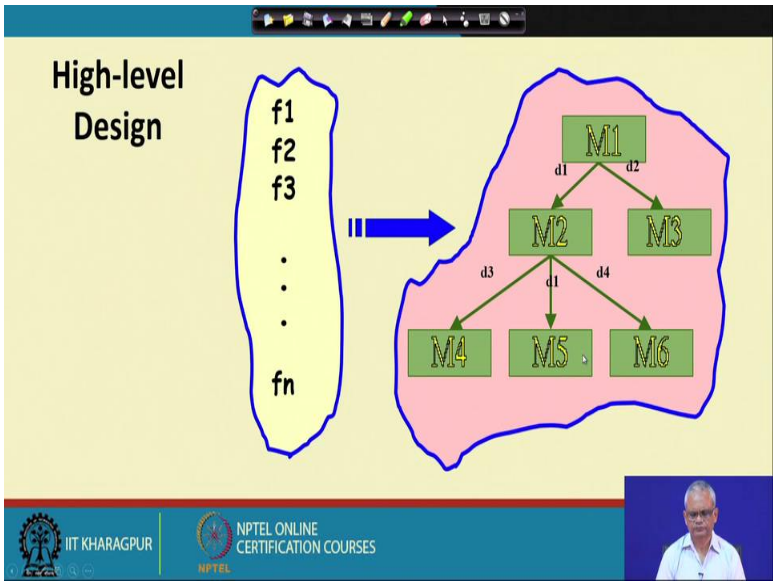


Abstraction

- The principle of abstraction requires:
  - lower-level modules do not invoke functions of higher level modules.
  - Also known as layered design.

So, M1 calls M2, M3 and M2, M3 do some processing, again get data from M4, M5, M6 and do some processing and pass it onto a M1 and so on. In a good layered design, a lower layered design should not be able to invoke an upper level design and that is invisible, that should be invisible to the lower level modules.

(Refer Slide Time: 20:23)

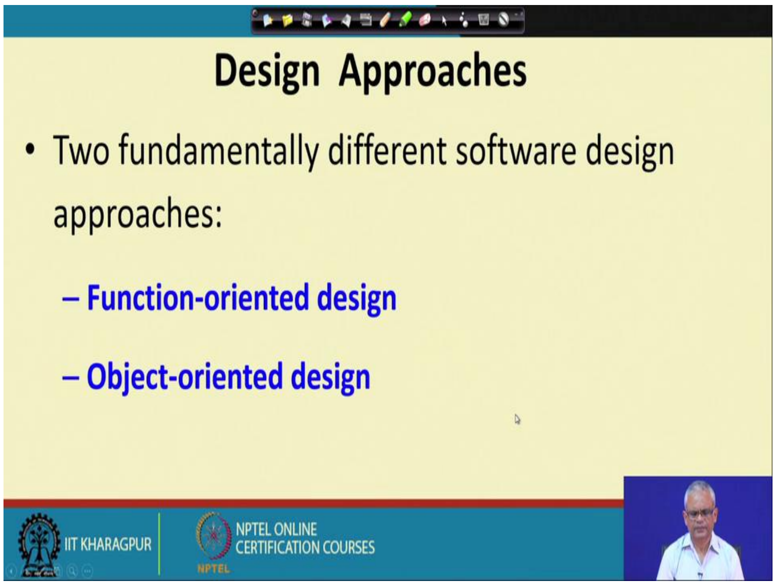


High-level Design

We can therefore say that if write a program and identify that a set of functions f1 to fn are to be written, and then the task of high level design is to organize those functions into a module structure. We should be able to assign these functions to different module such that each module has high cohesion and there is low coupling among modules and also they are organized in a tree like expression. And, this is the crux of the procedural design approach, as well as for the object-oriented design approach. This principle is used for both design approach. So, here observe that we identify what are the functions in our applications to be written, and then these functions are organized into different modules such that each module has high cohesion and low coupling and also, these are organized in a nice tree like expression.

In next few lecture, we will do some procedural design. There we will identify the functions by using a data flow diagram, structured analysis principle and, then by using a structure chart, structure design, we will be able to come up with this module structure and a tree like diagram.

(Refer Slide Time: 22:03)

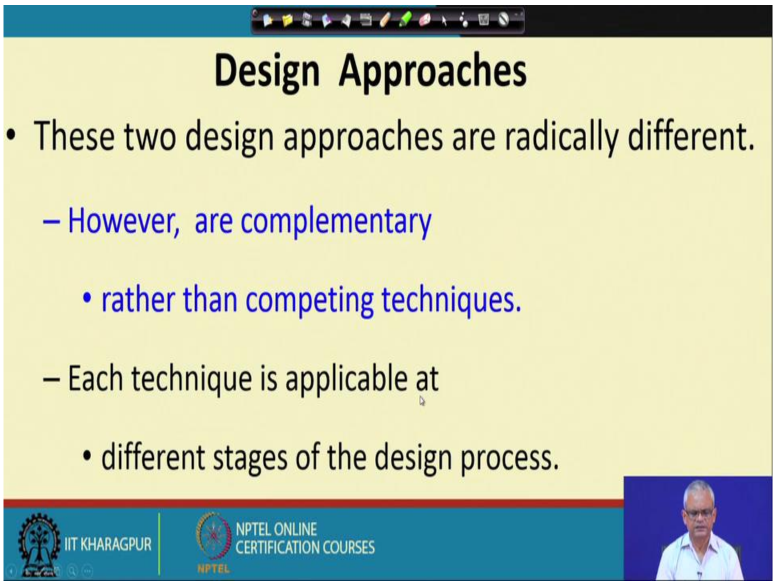


Design Approaches

- Two fundamentally different software design approaches:
  - Function-oriented design
  - Object-oriented design

Two design approaches are popular one is the function-oriented design and other is the object-oriented design. Given a design, what are desirable of a design? So, we can say that this design should be functionally independent, tree like structure, layered and so on. We will look at two approaches: the function-oriented design and object-oriented design, to arrive at a good design structure. But before we look at the nitty gritty of these two design techniques, let’s have some very overall understanding of what is involved in functionally-oriented design and object-oriented design.

(Refer Slide Time: 22:50)



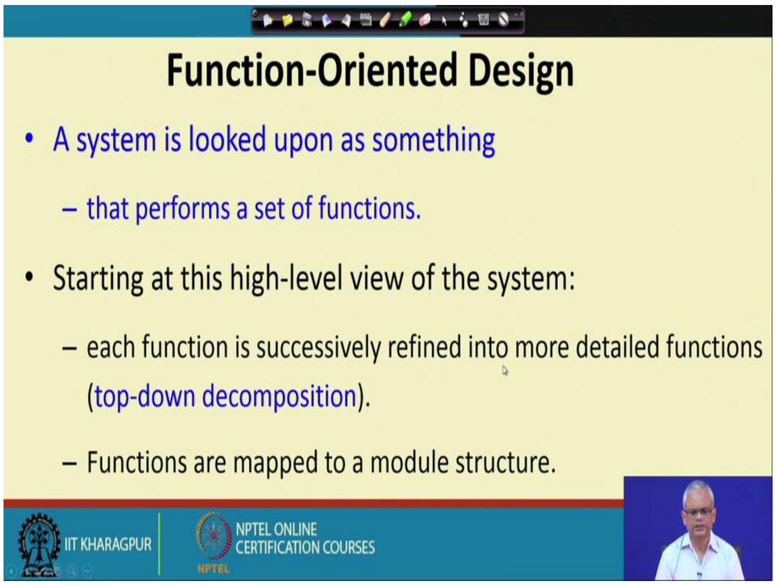
Design Approaches

- These two design approaches are radically different.
  - However, are complementary
    - rather than competing techniques.
  - Each technique is applicable at
    - different stages of the design process.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

These two design techniques even though popularly they are believed that they are computing approaches, but not really as we proceed, we will see that they are complimentary approaches, because object-oriented design also uses the principles of procedural design. So, the procedural design techniques are also used as part of the object-oriented design, we will examine those results.

(Refer Slide Time: 23:29)



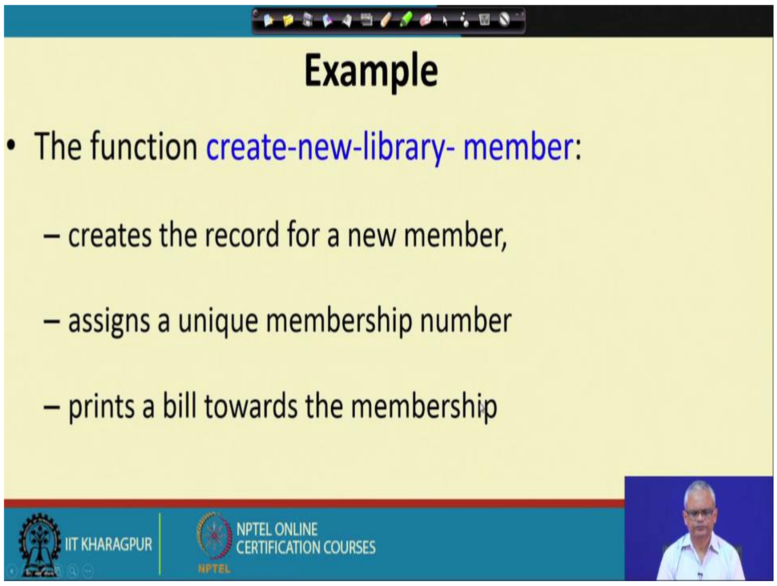
Function-Oriented Design

- A system is looked upon as something
  - that performs a set of functions.
- Starting at this high-level view of the system:
  - each function is successively refined into more detailed functions (top-down decomposition).
  - Functions are mapped to a module structure.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

First let’s looks at the function-oriented design. Here, by looking at the description of the problem we will identify the set of functions which needs to perform. Once we identify the set of functions, then we will map this into a module structure. Identifying the set of functions to be performed is called as the structure analysis. And, then these identified set of functions, will map into a module structure. This module structure is also called as structure design. We will also call it as the top down principle, because initially we consider the system as performing various large granularity functions. And, then we will split this into small functions. Will successively refine into more detailed functions or smaller functions. And, then there will be a large number of functions and each of those we will map into the module structure. And, that's the reason why it is called as a top down decomposition principle. As, we will study this function-oriented design techniques in more detail, it will become clear why it is called a top down decomposition. Once, we achieve the top down decomposition we map the function structure into a module structure.

(Refer Slide Time: 25:20)



Example

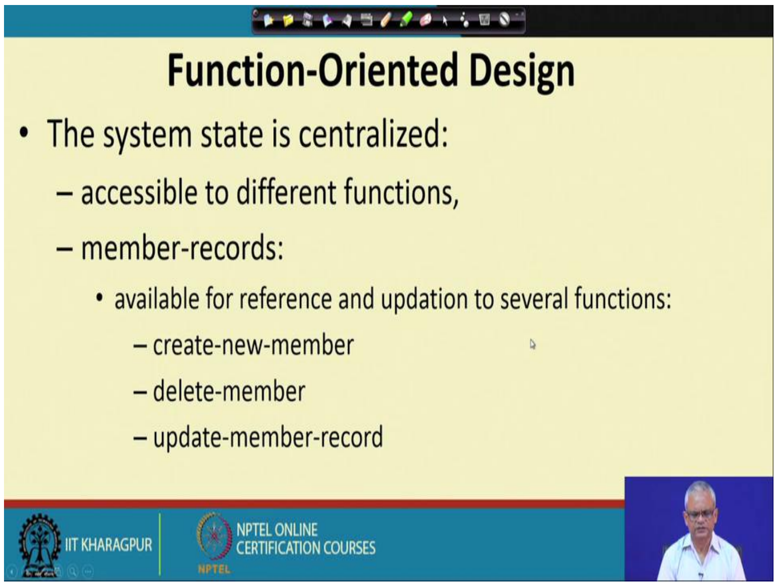
- The function create-new-library- member:
  - creates the record for a new member,
  - assigns a unique membership number
  - prints a bill towards the membership

IIT KHARAGPUR

NPTEL ONLINE CERTIFICATION COURSES

Just to give an example of a top down decomposition, let say in a library software we have a function named as create new library member. Now, we can see that the create new library member requires some activities to be performed. For example, we need to first create a record for the new member, we will have to assign a membership number and also we will have to print a bill towards the membership. So, we can consider that the function create new library member to be consisting of smaller functions or sub functions, that is create record, assign a unique membership id and then print bill. Similarly, how we say, for every function, what are the activities to be performed under that? For that we can split the function into simpler set of functions. And, we can do that recursively in the sense that we can again look at the sub functions and find if this can be split into still simpler functions.

(Refer Slide Time: 26:40)



Function-Oriented Design

- The system state is centralized:
  - accessible to different functions,
  - member-records:
    - available for reference and updation to several functions:
      - create-new-member
      - delete-member
      - update-member-record

This is one of the important principle of function-oriented design that we look at what are the functions that the system performs and then split those into simpler functions. And, the other thing that to notice about function-oriented design is that all these functions, they operated some centralized data structure. For example, in a library system, all the functions may be needing to operate on the set of books or the set of member records and so on.

So, these are two distinguish characteristic of a function-oriented design: First one is we need to identify the functions that the systems performs, break those function into simpler functions and second one is identify the data that are centralized and all these functions operated on that. We will take one example and see how we go about doing the function-oriented design. Later we will look at this technique in more detail, we will look at the data flow diagram in technique to do the structured analysis and, then we take the structure chart and see how to do the structure design.

We will stop here and continue from this point in the next lecture.

Thank you.