

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

**Lecture – 20**  
**Modular Design**

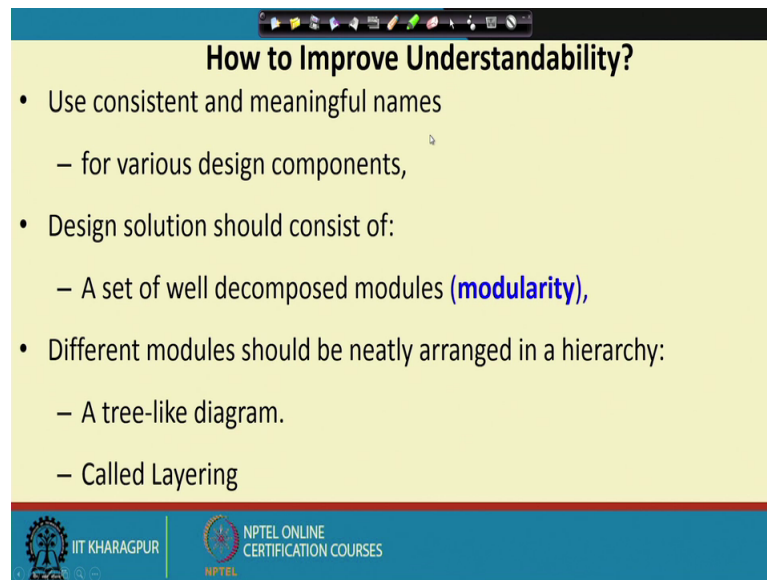
Welcome to this lecture. In the last lecture, we had looked at a some very basic issues in software design. And, then it said that it is important to distinguish a good design and from a bad design. And, then we are trying to characterize, what is a good design? And, then we said that there are many factors that determine, whether design is good or bad, but correct implementation of the functionality that possibly the important requirement of a good design, because unless it a correct design it is not a good design.

But, there are several design alternatives which can be we can come up with different alternate designs, which are all correct, but then how do we decide which is a better design. And, then we had said that understandability of a design is a major issue, we can rate a design solution to be good or bad based on it is understandability.

But, there are 2 questions that are arise now. That how do we know? That, which design is more understandable, because that a debatable question right, because somebody may say that this design is more understandable other another person will say that no that is not a very understandable design, we should be able to tell more formally what do you mean by a understandable design?

And, also we have to address this question that how do we really come up with a understandable design? So, let us proceed from that first let us look at the characteristics of a design, that enhance it is understandability. So, the first characteristic is the use of consistent and meaningful names.

(Refer Slide Time: 02:44)



**How to Improve Understandability?**

- Use consistent and meaningful names
  - for various design components,
- Design solution should consist of:
  - A set of well decomposed modules (**modularity**),
- Different modules should be neatly arranged in a hierarchy:
  - A tree-like diagram.
  - Called Layering

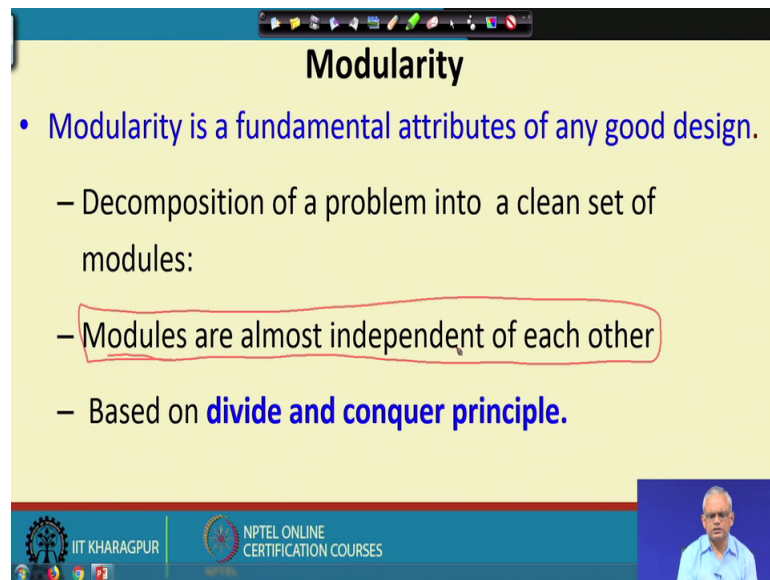
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Because, unless the names in the design are meaningful. In the problem context and also they have been used consistently. Anybody, trying to understand the design will get confused. All design components should have names, that corresponds to the problem domain as far as possible and they should be meaningful somebody trying to understand. And, the second characteristic of a design which is understandable is that it should have in decomposed well into modules. This we call it as the modularity of the design, we need to look at this question that what do we mean by modularity?

What is a well decomposed set of modules that we will do next, but then this is a important characteristic that the design should be modular. And, the modules should have been such that they have been well designed. Just arbitrary set of modules is not a good design; the module should have been well decomposed.

And, also the call relationship should like a tree diagram; this is also called as layering. Because, each layer of the tree is a implementation of the previous layer and a specification further lower layer. This is a desirable characteristic we will see why little later, that is not only that we should have a well decomposed set of modules, but also their call relation should be represent able in the form of a tree like diagram.

(Refer Slide Time: 05:05)



The slide is titled "Modularity" and contains the following text:

- Modularity is a fundamental attributes of any good design.
  - Decomposition of a problem into a clean set of modules:
  - Modules are almost independent of each other
  - Based on **divide and conquer principle**.

The slide also features a video inset of a speaker in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

Now, let us try to understand what we mean by modularity or well decomposed set of modules. We say that a design is modular, if the modules are almost independent of each.

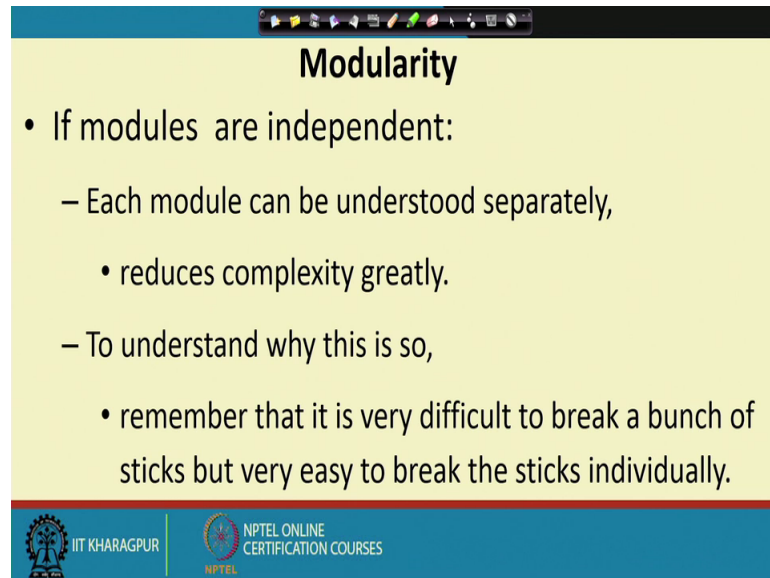
Other a set of modules are almost independent of each other, if they either do not call any other module or they call very very less. If, a module calls another modules and it requires many parts to be completed by other modules so, on then it becomes a dependent module.

In a good decomposed well decomposed set of module the module should be as independent as possible. It is not possible to have a completely independent set of modules that no module interacts with other, but the interaction among the module should be kept to a minimum. To understand why it is so, we will argue that this is actually the divide and conquer principle.

A modular design uses the divide and conquer principle in the sense that to be able to understand the design, we just take up each module and then we understand. And, if after we have understood all the modules we have understood the design.

But, if the modules have complex relations that is each module calls several other modules then while understanding a single module. We need to also see what it gets done by the other modules and so on. So, as long as the modules do not interact much we can understand them well, because based on the divide and conquer principle.

(Refer Slide Time: 07:34)



**Modularity**

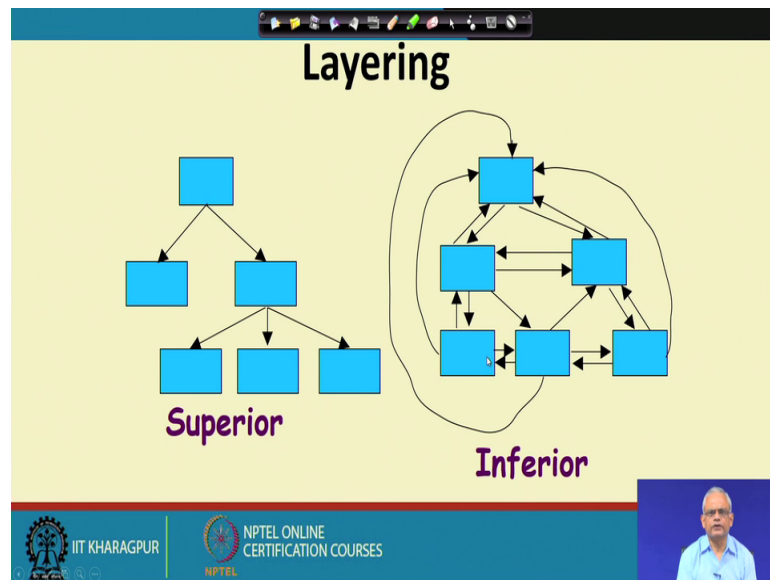
- If modules are independent:
  - Each module can be understood separately,
    - reduces complexity greatly.
  - To understand why this is so,
    - remember that it is very difficult to break a bunch of sticks but very easy to break the sticks individually.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if the modules are independent, we can one by one you just look up the modules and understand this, and this is based on the design and divide and conquer principle.

If a bunch of sticks are tied together and we tried to break it, then becomes very difficult, but if we take one stick at a time and break it, then becomes very easy. It is thus the similar thing here is that the modules one by one, we tried to understand we can easily understand, but trying to understand all modules interconnected complex dependence among modules, then if we start with one module we see that we need to understand another module in the third module and so on.

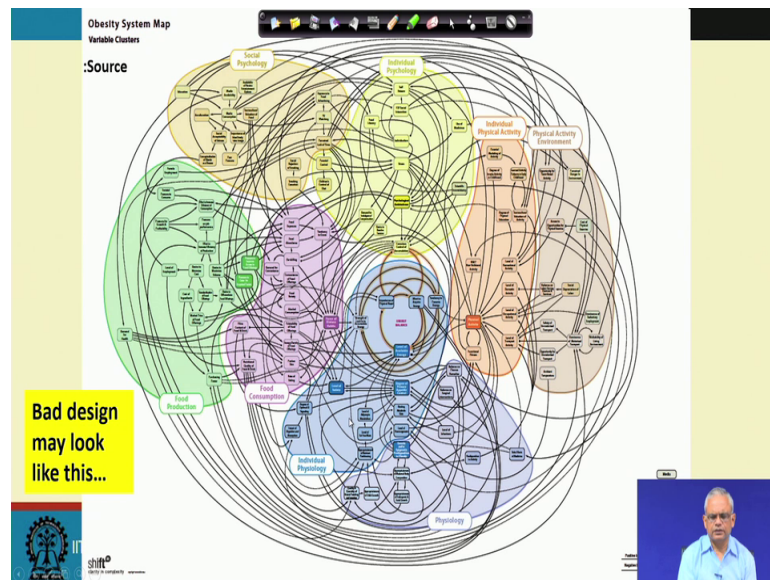
(Refer Slide Time: 08:44)



Another requirement is a layering a tree like call relationship among modules is a superior, whereas an arbitrary call relation is an inferior design. We will see why this is a good idea the layering is a good idea; one is bug localization if there is a bug it does not, effect the other modules here only the once are below here.

Whereas here any bug can have effect on any other module. And, when you see the bug system that is a failure we do not know what is the module, that is having the bug, but here if we observe a bug here? We know that either the bug is here or in the previous one it cannot be here we do not have to even look at this.

(Refer Slide Time: 10:05)



A bad design can come up with a very complex set of module interactions, it is not really a set of modules and interactions it is best diagram for some other purpose, but I just showing it for illustration, that it can look extremely odd and for one module to understand it you might have to understand this module and then this module and then this module and so on. And, we will end up to for understanding one we need to address all the modules and it becomes very very difficult to understand we need a nice tree like diagram.

(Refer Slide Time: 10:55)

## Modularity

- In technical terms, modules should display:
  - **high cohesion**
  - **low coupling.**
- We next discuss:
  - cohesion and coupling.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, far we said that modularity is a set of independent modules and the we said that independent modules are actually not practical let all modules are independent. There will be some interactions between modules, but can we give a more meaningful characterization of modularity. We, had intuitive understanding of what is modular that modules are almost independent of each other, do not depend too much represented in a tree like diagram and so on.

Let us try to give a better characteristic or more precise characteristic of what is a modular design?. And, we will do that in terms of 2 concepts; one is called as cohesion and the other is called a coupling. A modular design should display high cohesion and low coupling.

And, if we understand this terms cohesion and coupling and we can measure them, then given a design solution, we can check whether it has high cohesion and low coupling compared to another design which is given to us.

(Refer Slide Time: 12:25)

The slide is titled "Modularity" and contains the following text and diagrams:

- Arrangement of modules in a hierarchy ensures:
  - **Low fan-out**
  - **Abstraction**

There are two hand-drawn diagrams in blue ink. The first diagram on the left shows a hierarchical tree structure with a root box at the top, two child boxes below it, and two more boxes at the bottom level, one of which is circled. The second diagram on the right shows a circular arrangement of boxes with arrows indicating interactions between them, representing high coupling.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man in a blue shirt.

So, let us try to understand how to measure define cohesion and coupling? And of course, the module should be layered a good layering a tree like diagram, we will see that it has the low fan out abstraction; we will look at these concept as we proceed.

But, then let me just mention briefly, that in a tree like diagram. If, we want to let us say understand this module, then we see that it calls only the lower modules and maybe we

will have to look at this 2. If, we want to understand this one then we do not even have to look at this. We know that this will call only it is lower layer modules. And, there are no there are no layer modules, then we can understand this we can understand these and then based one that we can understand this and so on.

But, if it is a module structure like this, then we can do that to able to understand one we need to just go across all other modules and so on, becomes very very difficult. So, a tree like hierarchy, we have low fan out that is it depends on less number of modules. And, also there is a abstraction, abstraction in the sense that these are you can consider these modules or implementation of this module. And therefore, you start from the leaf layer modules and then look at more concrete module and so on.

(Refer Slide Time: 14:54)

**Coupling: Degree of dependence among components**

No dependencies

Loosely coupled-some dependencies

Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

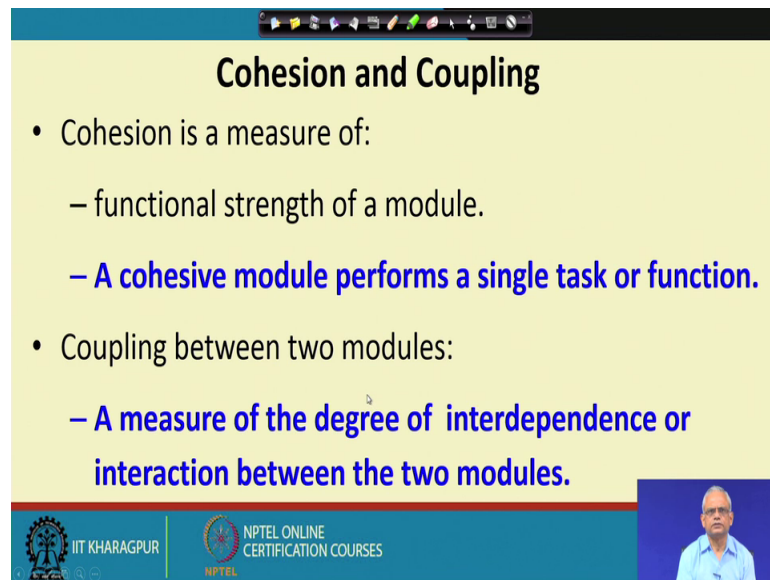
Source:  
Pfleeger, S., *Software Engineering Theory and Practice*, Prentice Hall, 2001.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

This is an illustration of how bad it can become if the modules are having lot of dependencies. So, this there are no dependencies, this there are very less number of dependency, but look at here that there are very high dependencies across modules and it becomes very difficult to understand maintain this kind of design you would rather go for this if not go for this.



(Refer Slide Time: 15:32)



**Cohesion and Coupling**

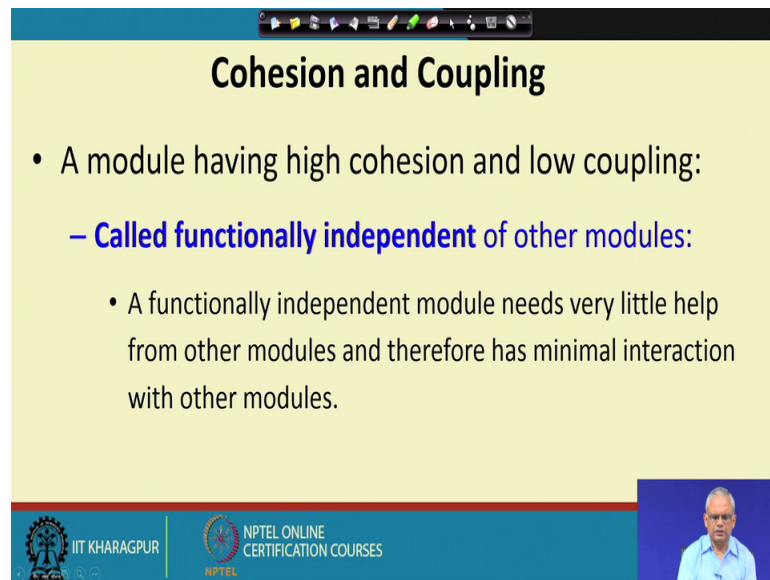
- Cohesion is a measure of:
  - functional strength of a module.
  - **A cohesive module performs a single task or function.**
- Coupling between two modules:
  - **A measure of the degree of interdependence or interaction between the two modules.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see the concepts of cohesion and coupling. If, we can determine the cohesion and coupling of a given design solution, we can tell that whether that is a good design or bad design. We say that a cohesive a module is cohesive, if it performs a single task or function. If, it tries to do too many things then it is not cohesive on the other hand the coupling is defined between 2 modules, 2 modulus are coupled if they dependent on each other.

So, that they exchange some data items call each other and then, we say that these are coupled modules, this is 2 important concepts cohesion it defined it for a single module.

(Refer Slide Time: 16:43)



The slide is titled "Cohesion and Coupling" and contains the following text:

- A module having high cohesion and low coupling:
  - **Called functionally independent** of other modules:
    - A functionally independent module needs very little help from other modules and therefore has minimal interaction with other modules.

The slide also features logos for IIT Kharagpur and NPTEL Online Certification Courses, and a small video inset of a speaker in the bottom right corner.

And then we say that a single module is cohesive, if it performs a single task it does many things then that is not really cohesive. Coupling is that how much dependent one module is on another module. If, they exchange too many data items call each other frequently exchanging data items and so on. Then, they are highly coupled, but if they do not call each other there is no data exchange between 2 modules we say that these 2 modules are independent.

And, if that is a characteristic of almost every module, we will say that there is a independence. And, similarly they are cohesive there is a high cohesion and low coupling then we say that this is a good design.

Now, let us look at how do we determine the cohesion and coupling? Before, that if a design solution we find that it as high cohesion and low coupling, then we say that is a functionally independent set of modules. So, they are each module is doing some function independently, it is not dependent on others and the term that we use is functionally independent.

So, the goal of any good design technique is to come up with a functionally independent set of modules.

(Refer Slide Time: 18:32)

**Advantages of Functional Independence**

- Better understandability
- Complexity of design is reduced,
- Different modules easily understood in isolation:
  - Modules are independent

The slide contains two diagrams. The first diagram, labeled 'No dependencies', shows four blue rectangular boxes arranged in a 2x2 grid with no arrows between them. The second diagram, labeled 'Highly coupled-many dependencies', shows four blue rectangular boxes arranged in a 2x2 grid with multiple arrows between them, indicating a complex web of dependencies.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, we should be clear why we want a functionally independent set of modules. as, you said that this results in better understandability this functionally independent set of modules, compared to a highly dependent set of modulus.

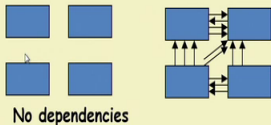
If, we have a functionally independent set of modules the complexity in the design is reduced and we can take up one module at a time quickly able to understand. Similarly, we can if there is a bug we know that which module it contains a bug. Otherwise we might see that whether it has come from some module to which it has called.

So, if there is a bug while executing this design module we observe it, then it might have come from here or it can have come from here etcetera.

(Refer Slide Time: 19:43)

**Why Functional Independence is Advantageous?**

- Functional independence reduces error propagation.
  - degree of interaction between modules is low.
  - an error existing in one module does not directly affect other modules.
- Reuse of modules is possible.



The diagram illustrates two scenarios of module interaction. On the left, four blue rectangular boxes are arranged in a 2x2 grid, with no lines or arrows between them, representing 'No dependencies'. On the right, four blue rectangular boxes are arranged in a 2x2 grid, but they are interconnected with a dense network of black arrows pointing in all directions, representing a highly coupled system where a change in one module affects all others.

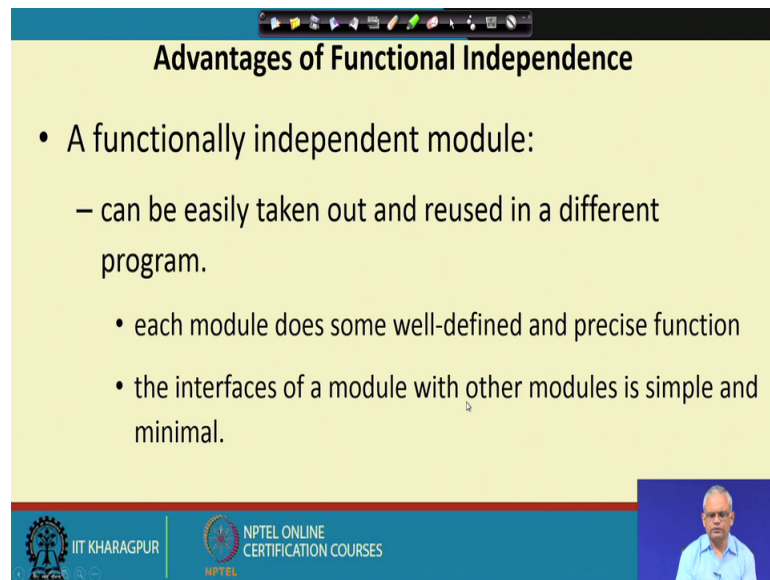
No dependencies

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, in a error debugging becomes easy and also understandability is easy and not only that, we can re use module. Let us say this is a module and we want to use it in another project, we can just take this and use it there, but here observe that it we want to use this module in another project. We will see that this needs help from here and this in turn needs help from here and similarly this needs help from here and so on.

So, they are all coupled we cannot take just one module and use it we have to take across all this modules that entire project you have to take and that becomes in feasible. So, if you have a functionally independent set of modules, any module that we need in another project we can easily re use that.

(Refer Slide Time: 20:56)



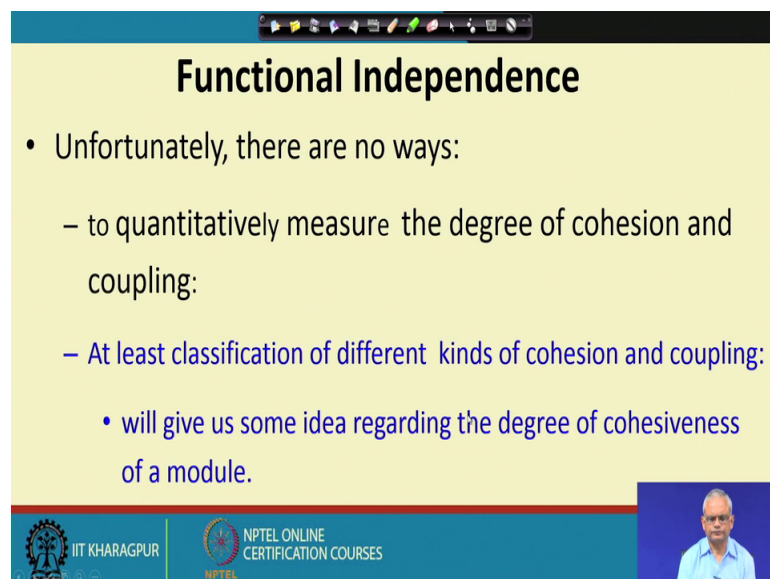
**Advantages of Functional Independence**

- A functionally independent module:
  - can be easily taken out and reused in a different program.
  - each module does some well-defined and precise function
  - the interfaces of a module with other modules is simple and minimal.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Each module in a functionally independent set of module has well defined precise functions. And, it does not interact with other modules and or you fetal it interfaces with other modules simple and minimal. And therefore, becomes easy to re use modules.

(Refer Slide Time: 21:20)



**Functional Independence**

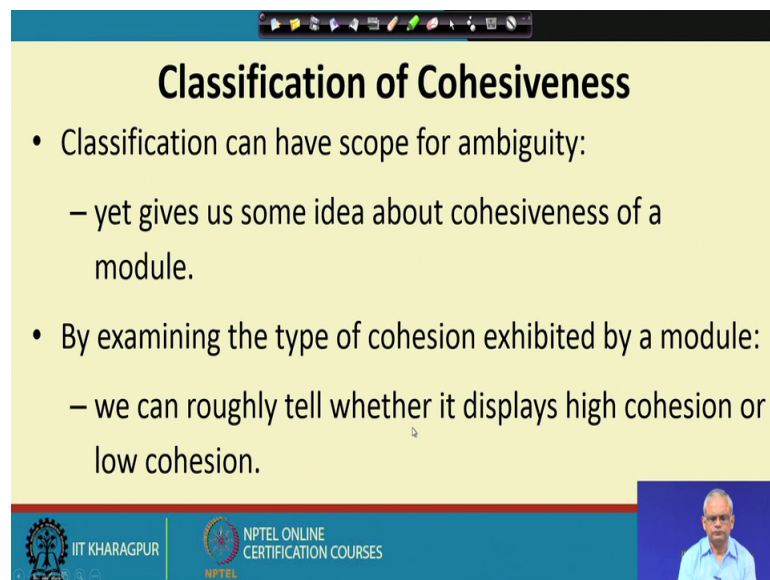
- Unfortunately, there are no ways:
  - to quantitatively measure the degree of cohesion and coupling:
  - At least classification of different kinds of cohesion and coupling:
    - will give us some idea regarding the degree of cohesiveness of a module.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But, how do we measure functional independence? Even though we know that a functional independent set of module has high cohesion and low coupling. Can, we look at a design and say that whether there is a low coupling.

Let us try to explore this further; we will classify the different types of cohesion and coupling. So, that given a design we will try to see, which class of cohesion and coupling it has and based on that we will say that, whether it is a cohesive highly cohesive and low coupling etcetera.

(Refer Slide Time: 22:09)



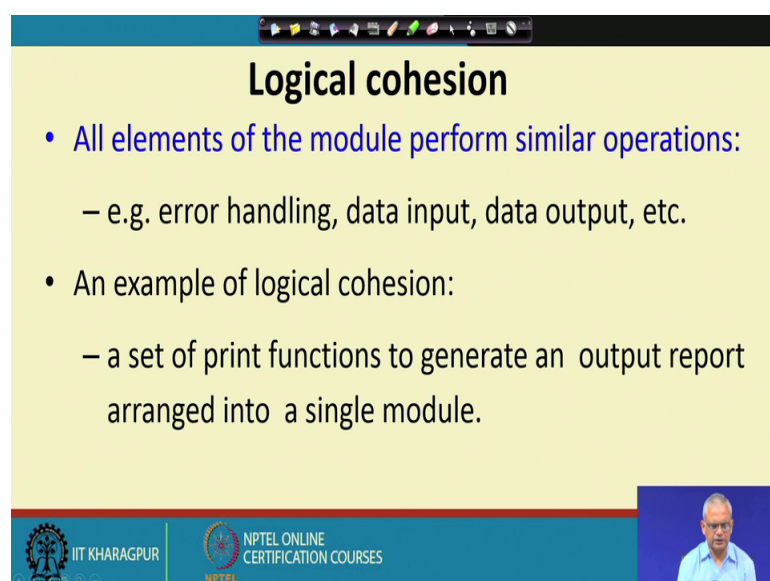
**Classification of Cohesiveness**

- Classification can have scope for ambiguity:
  - yet gives us some idea about cohesiveness of a module.
- By examining the type of cohesion exhibited by a module:
  - we can roughly tell whether it displays high cohesion or low cohesion.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

First let us look at the cohesiveness and let us try to classify what are the type of cohesion? So, that if we look at a design, we will see that what set of cohesion it has?

(Refer Slide Time: 22:26)



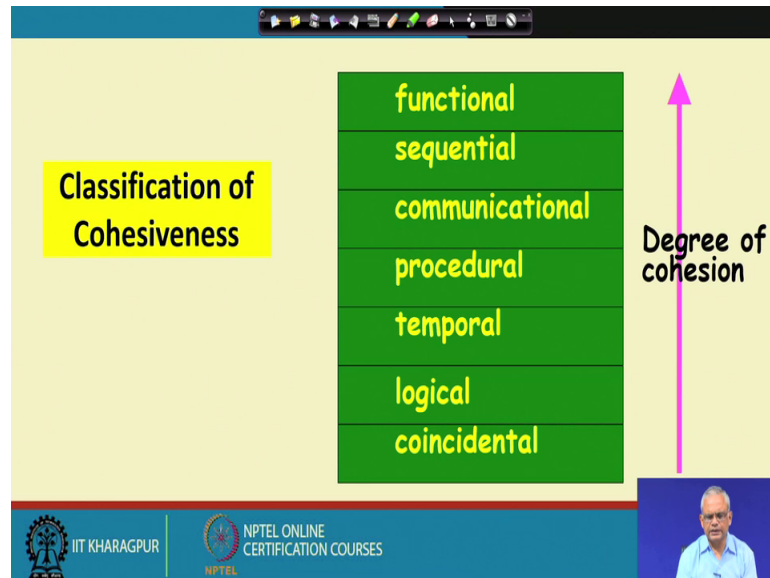
**Logical cohesion**

- All elements of the module perform similar operations:
  - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
  - a set of print functions to generate an output report arranged into a single module.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us see the different types of cohesion, we can look at a design and we can say that it has any one of these 7 types of cohesion.

(Refer Slide Time: 22:32)



We can look at a module and you can say that this module has either this co incidental or logical temporal, procedural, communicational, sequential or functional cohesion.

If, it has functional cohesion then the, it has high cohesion it is a good design, but if it is has coincidental cohesion on the other hand, it is not a good design. And, something in between is moderate cohesion and we want to look at all modules. And, see what type of cohesion are there in most of them are high cohesion or a slightly moderate cohesion we will say that the design is good or ok, but if they have low cohesion, we will see that we will say conclude that it is not a good design.

(Refer Slide Time: 23:41)

**Coincidental cohesion**

- The module performs a set of tasks:
  - which relate to each other very loosely, if at all.
    - That is, the module contains a random collection of functions.
    - functions have been put in the module out of pure coincidence without any thought or design.

First let us look at the worst form of cohesion it is called as the co incidental cohesion.

Here, if we look at the module look through what functionality it has we will see that the functions are rather random collection of the functions. There is no thinking or plan behind putting some functions in the module, we just taken out random functions and made them into a module.

(Refer Slide Time: 24:15)

**Coincidental Cohesion - example**

```
Module AAA{  
  
    Print-inventory();  
  
    Register-Student();  
  
    Issue-Book();  
};
```

Just to give an example, let us say we look at one module some name A A A and then it contains the functions print-inventory, register-student, issue-book.

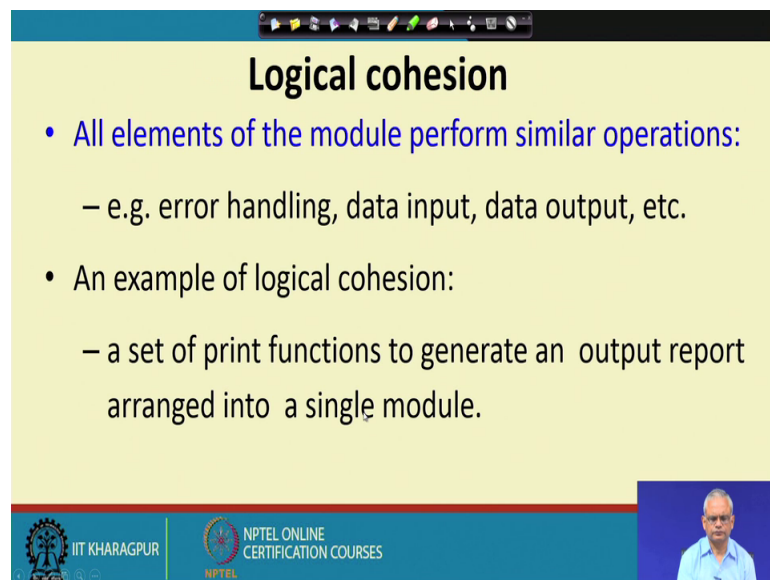


So, this actually correspond to the functionalities required by various other various requirements, print inventories part of some inventory functionality register student is some registration issue book. So, these are random set of functions, they do not neither belong to the same requirement nor there are any relationship between these functions.

And, if we ask that what does the module A achieve? We cannot really answer we will say that it has some part of the inventory does some register in student, it also does issue book etcetera. So, there is no single thing single function that this module achieves it.

Does various things here and there somebody has just put these modules randomly into this module A A A and this you call as co incidental cohesion.

(Refer Slide Time: 25:45)



The slide is titled "Logical cohesion" and contains the following text:

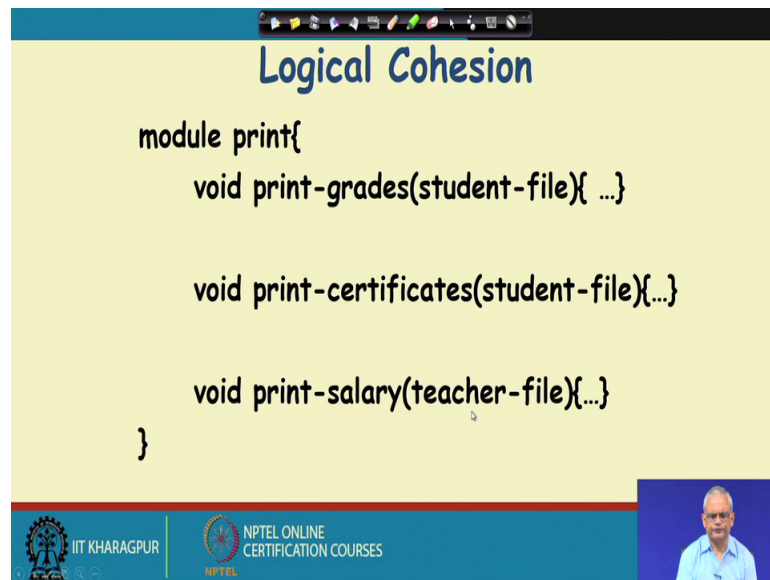
- All elements of the module perform similar operations:
  - e.g. error handling, data input, data output, etc.
- An example of logical cohesion:
  - a set of print functions to generate an output report arranged into a single module.

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom, and a small video inset of a speaker in the bottom right corner.

We, consider logical cohesion module has logical cohesion, if all elements, all functions in the module performs similar operation. For example, all functions in a modular doing error handling or all functions in a modular reading the data input or all functions in a modular just print f kind of thing data output.

Then, we say that this module has a logical cohesion that is the functions are logically related that they are doing similar things and that is why designer has put them there. And, this is also not a very good form of cohesion slightly better than co incidental cohesion, but still it is not good.

(Refer Slide Time: 26:43)



## Logical Cohesion

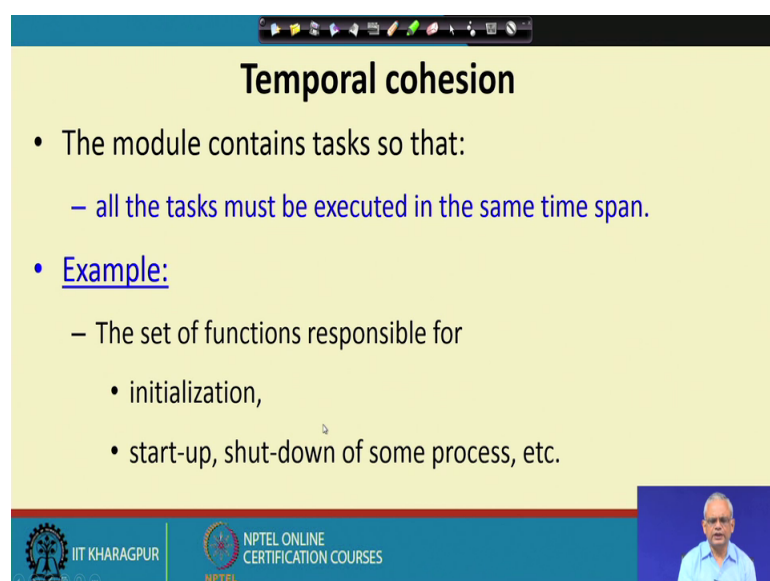
```
module print{  
    void print-grades(student-file){ ...}  
  
    void print-certificates(student-file){...}  
  
    void print-salary(teacher-file){...}  
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us look at an example of logical cohesion, let us say we look at a module and find that its name is print and then it does various types of printing, across various functionalities printing, grade, certificates, salary etcetera. These are part of various other functionalities or requirements.

And, then we say that this module has logical cohesion, it is not the bottom most cohesion, but still it is not a good cohesion.

(Refer Slide Time: 27:23)



## Temporal cohesion

- The module contains tasks so that:
  - all the tasks must be executed in the same time span.
- Example:
  - The set of functions responsible for
    - initialization,
    - start-up, shut-down of some process, etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we look at the temporal cohesion, we will stop here continue in the next lecture.