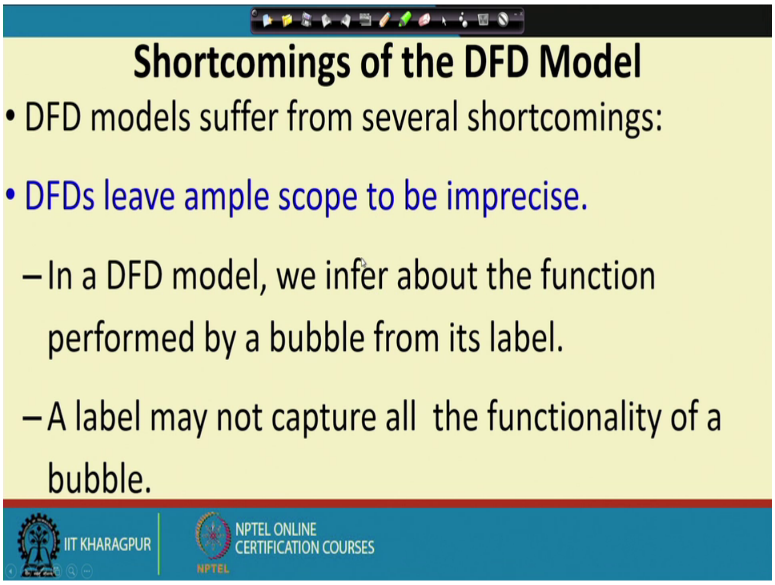
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
DFD Model- More Examples

Welcome to this lecture over the last couple of lectures we were discussing about the data flow diagrams. The data flow diagrams are popularly used to do the structured analysis that is capturing all the functionalities of the system and then doing a top down decomposition of the functionalities. We found that DFD is a elegant mechanism very simple and it produces data flow model of a software, identifies various functions at various levels and the interactions terms of the data exchange among those functions.

And that forms the background or the design document based on which further design can be carried out. Structured analysis is as you were mentioning so popular not restricted only to the design aspect, but also for many other problems including in requirements analysis DFD's are used. But let us find out are there any shortcomings of the DFD and that will give us some idea about for what applications DFD's are not suitable and what needs to be done to the DFD's to extend those to be applicable to those areas. Now, let us look at the shortcomings of the DFD model.

(Refer Slide Time: 02:16)

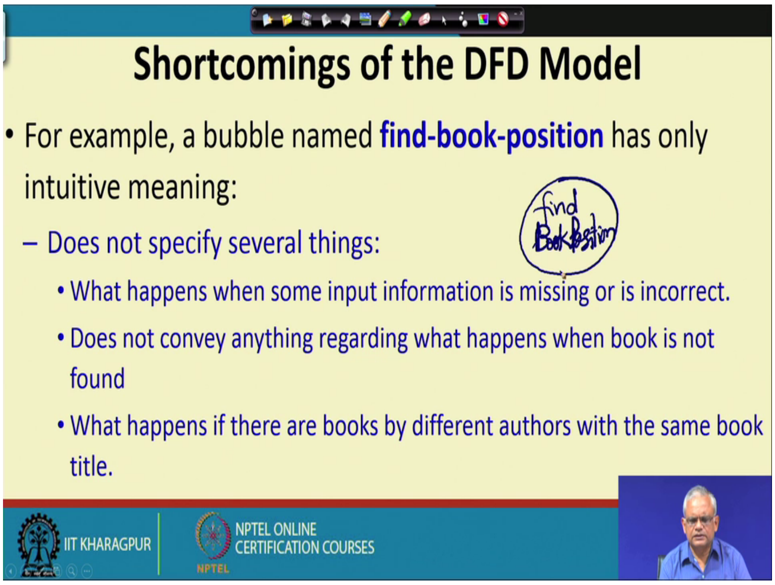


Shortcomings of the DFD Model

- DFD models suffer from several shortcomings:
- DFDs leave ample scope to be imprecise.
  - In a DFD model, we infer about the function performed by a bubble from its label.
  - A label may not capture all the functionality of a bubble.

The first difficulty is that here the functions are only identified by the names written on the bubble. And therefore, can be very imprecise, just by looking at the label we have to in firm infer about what all is involved in the function and therefore, many things will remain ambiguous incomplete.

(Refer Slide Time: 02:50)



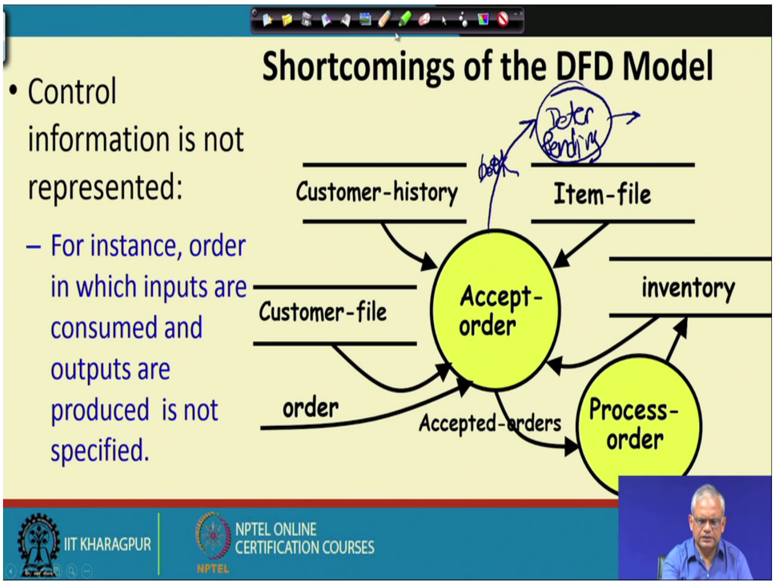
Shortcomings of the DFD Model

- For example, a bubble named **find-book-position** has only intuitive meaning:
  - Does not specify several things:
    - What happens when some input information is missing or is incorrect.
    - Does not convey anything regarding what happens when book is not found
    - What happens if there are books by different authors with the same book title.

Just to give an example, let us say we have a bubble and the name of the bubble is find-book-position. So, we have find the book position as a bubble, but then we can imagine that the find-book-position given a books description it outputs where it is located in a library, but that is a very rough idea a rough description of the problem. For example, it does not know, it does not tell us that how do we describe the book position, how do we give the input do we give the name of the book, do we give the author of the book and so on. And also what would happen if the book is not there, what would this functionality do.

We are given a book which is not present in the library and also what would happen if given a book name and multiple entries in the library match same book title, but different authors or maybe same author and in multiple books does it display all of them or does it display one. So, those things are not clear and therefore, the DFD model the first glance is imprecise.

(Refer Slide Time: 04:53)



Shortcomings of the DFD Model

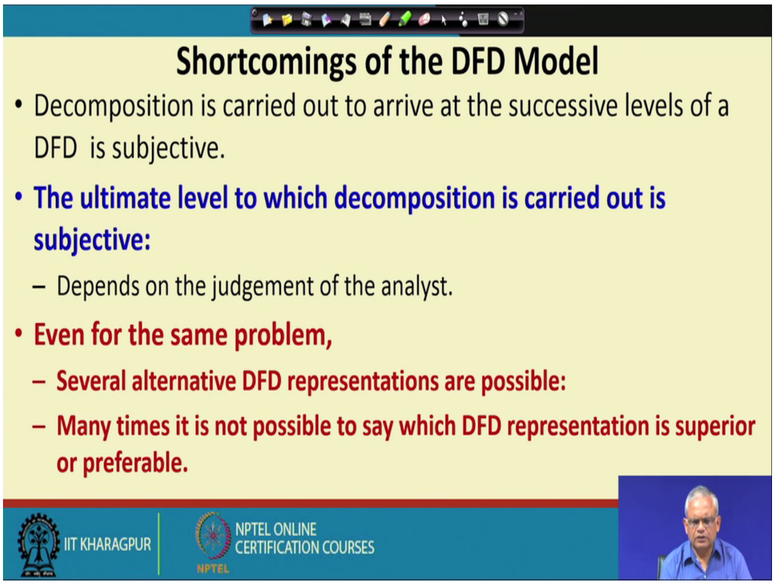
- Control information is not represented:
  - For instance, order in which inputs are consumed and outputs are produced is not specified.

Another major problem with the DFD representation is that, control for flow aspects are not represented. For example, we have this DFD model now in what order does the accept order take the data; does it take first customer history, then the item file, then the customer file and then the order or first it gets the order and then the customer file and so on. This is especially important in case of real time systems where the timing issues need to be analyzed. For example, do these 2 bubbles they execute parallely or they executed sequentially; if there are multiple bubbles that are connected this bubble.

Let us say we have another bubble connected here, let us say pending determined pending order. So, would this execute first or would this execute first. So these aspects these are the control aspects are not key not really represented here, we just write what data flows here may be some data book name etcetera. So all are just data exchanges among the different entities and the bubbles are captured.

And that is the reason why timing analysis using a data flow diagram is very difficult and this diagram has been extended with control flow notations to indicate in what order the processing takes place and that is for doing the timing analysis. But we will not discuss about those aspects in this lecture series.

(Refer Slide Time: 07:17)



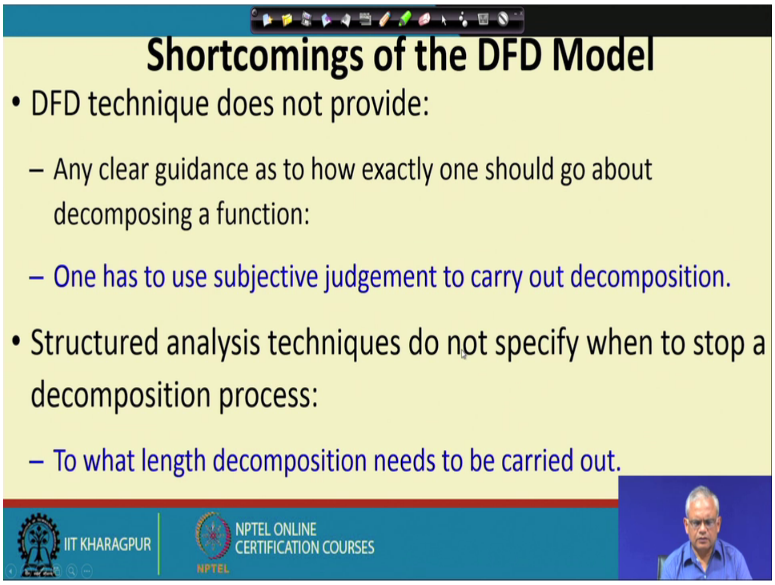
Shortcomings of the DFD Model

- Decomposition is carried out to arrive at the successive levels of a DFD is subjective.
- The ultimate level to which decomposition is carried out is subjective:
  - Depends on the judgement of the analyst.
- Even for the same problem,
  - Several alternative DFD representations are possible:
  - Many times it is not possible to say which DFD representation is superior or preferable.

There are other problems also one of the problems is that we achieve a functional decomposition at different levels of the DFD, but then there is no guideline about to what depth the decomposition should be carried out. We have only mentioned through a very load statement that the decomposition should be carried out until we reach a function which is very simple. But then that is a vague statement, what is function is simple to one person may not be the same for another person and therefore, different designers can carry out decomposition to different levels.

Another problem is that for different designers they can come up with very different DFD representation for the same problem and also many times we do not have any way to tell which DFD representation is superior and therefore, we have to explore multiple DFD representation ok.

(Refer Slide Time: 08:46)



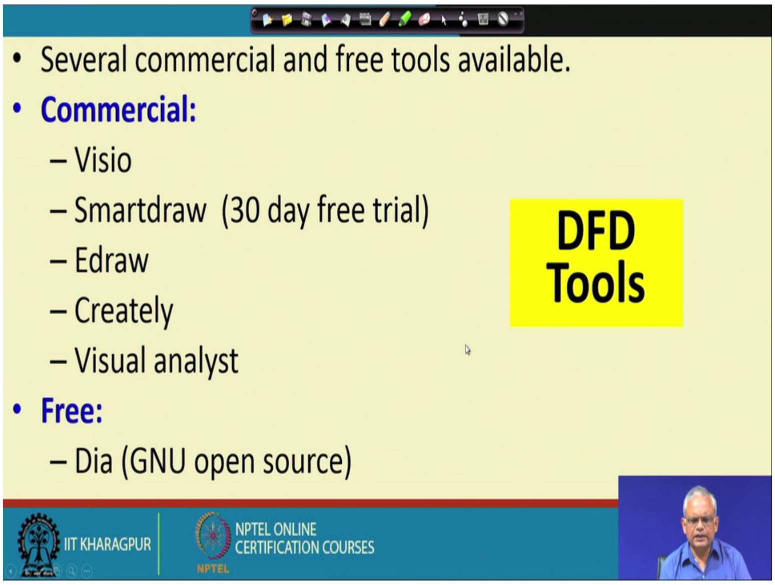
Shortcomings of the DFD Model

- DFD technique does not provide:
  - Any clear guidance as to how exactly one should go about decomposing a function:
  - One has to use subjective judgement to carry out decomposition.
- Structured analysis techniques do not specify when to stop a decomposition process:
  - To what length decomposition needs to be carried out.

We already seen that how to decompose, that is not clear we just say that see bubble should be decomposed in the next level. Take 1 bubble from a layer level and then decompose it.

But then we never said how to decompose it, we just said that look through the activities that are to be carried out by that process or that bubble and then identify the sub activities and those sub activities become the processes or bubbles in the next level. But then that subjective because different designers may come up with different sub functions and they will come up with different DFD representations.

(Refer Slide Time: 09:55)



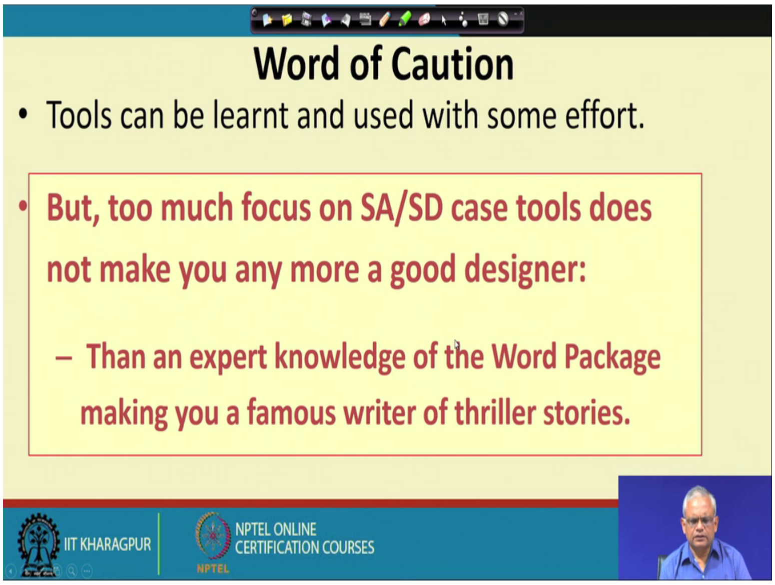
- Several commercial and free tools available.
- **Commercial:**
  - Visio
  - Smartdraw (30 day free trial)
  - Edraw
  - Creately
  - Visual analyst
- **Free:**
  - Dia (GNU open source)

DFD Tools

But in any case DFD is a very useful technique very easily learnt and once you learn it you will find, it comes to use not only in software design, but in many other areas. In software development for example, in testing and so on and also in non software development areas, there are many tools that are available using which you can develop the DFD model it simplifies the task of developing the model and produces good diagrams. There are several commercial tools which are popular for example video, smartdraw which includes a 30 day free trial, edraw, creately, visual analyst and so on and there are some free tools like dia which is the GNU open source tool.

Please use some of the tools we find that the tools are extremely easy to learn and based on the concepts we discussed can just start using the tool. The download is very fast for example, dia and typically we have pick and paste the different DFD symbols would be there as many menu items. You can just pick them and paste wherever necessary and draw the dataflow arrows between bubbles so and almost every tool they develop the data dictionary as you develop the functional model.

(Refer Slide Time: 12:12)



Word of Caution

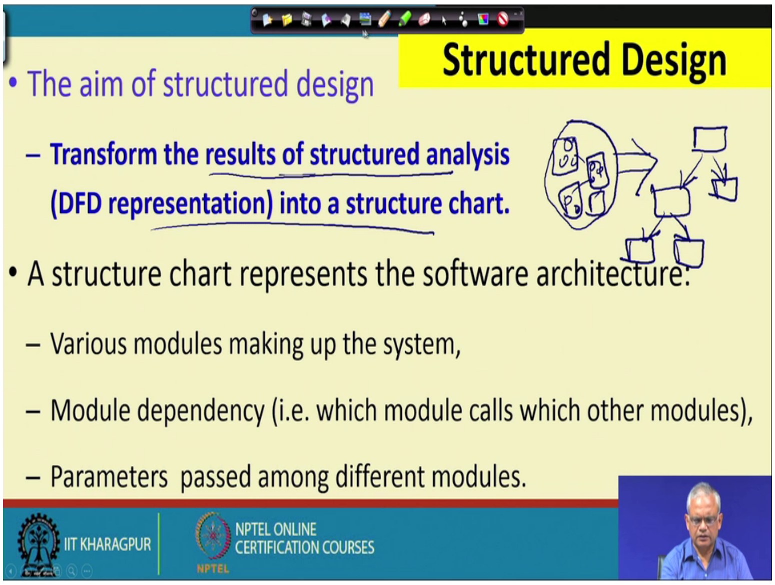
- Tools can be learnt and used with some effort.
- But, too much focus on SA/SD case tools does not make you any more a good designer:
  - Than an expert knowledge of the Word Package making you a famous writer of thriller stories.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

But then like to give a simple caution that, the concepts are more important understanding what does DFD model, how does it model, the hierarchy balancing and so on. But if you ignore the basic concepts, but just spend time on the tool just trying to master the tool without knowing the basic concepts it may be counterproductive. If you are hoping that just by learning the tools well you will become a good designer then your hope will be misplaced, because just to give an analogy.

That let us say you want to become a famous writer of a thriller stories and then let us say you learn the word processing package very well that will not make you the writer of thriller stories, a good writer of thriller stories what you need is how to write thriller stories the storyline and so on. Just learning the word processing package is not does not lead you or does not automatically make you a good story writer. In the same way just by learning the structure analysis and structure design tool you do not become a good designer you need to understand the concepts well and of course, you need to practice with several problems and that will make you a good designer.

(Refer Slide Time: 14:09)



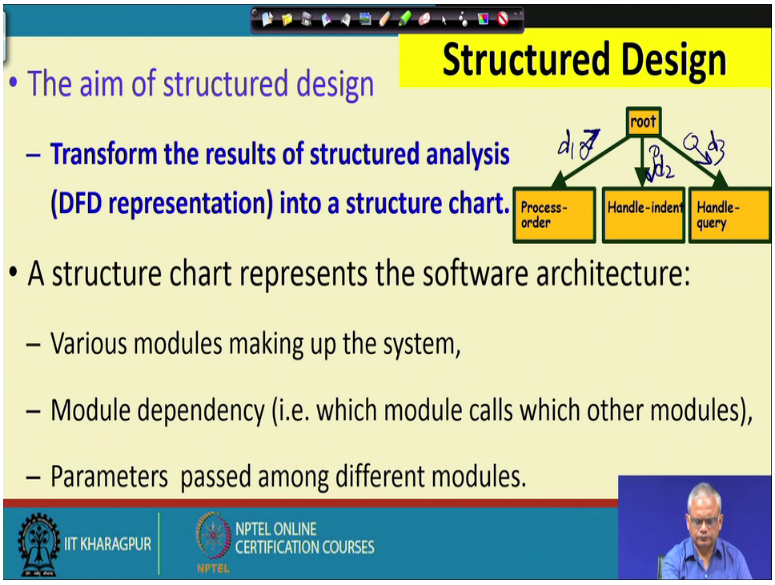
Structured Design

- The aim of structured design
  - Transform the results of structured analysis (DFD representation) into a structure chart.
- A structure chart represents the software architecture:
  - Various modules making up the system,
  - Module dependency (i.e. which module calls which other modules),
  - Parameters passed among different modules.

Now, once the data flow diagram model is complete, based on the model we develop the design the high level design and we call the process as the structure design. The structure design it takes the results of the structured analysis and it transforms this structured analysis into a structure chart. So if this is our set of DFD's the DFD model as you are saying is consists of a set of data flow diagrams the root level, level 1, level 2 diagrams and so on.

Through a structure design we transform the DFD model into a structure chart. So we will see the methodology by which by analyzing the DFD model we will be able to come up with a structure chart we will give some methodologies; that is the structure design methodology using which you can transfer the data flow diagram into a structure chart. And in the structure chart representation we will have the different modules of the system, the module dependency which will be represented in the form of arrows and also we will represent the different data items that are passed between the modules.

(Refer Slide Time: 16:16)

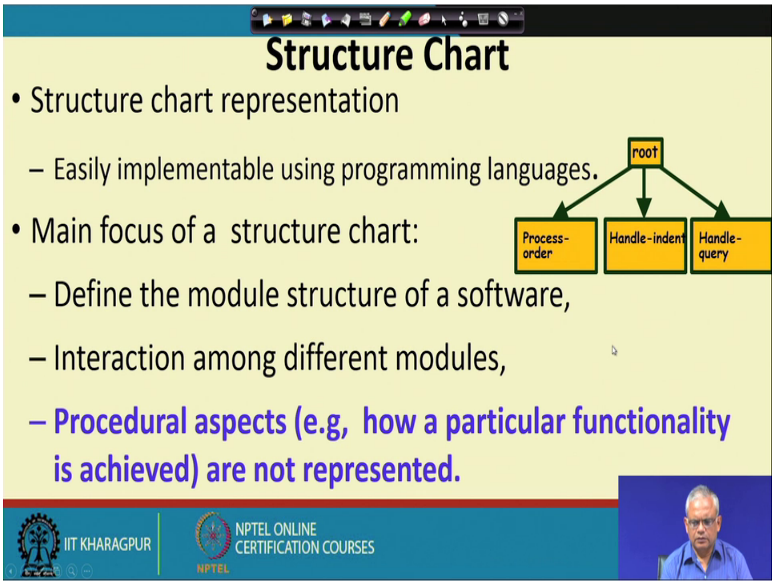


Structured Design

- The aim of structured design
  - Transform the results of structured analysis (DFD representation) into a structure chart.
- A structure chart represents the software architecture:
  - Various modules making up the system,
  - Module dependency (i.e. which module calls which other modules),
  - Parameters passed among different modules.

So, this is an example of a structure chart of a problem and we will have these are the modules of our design and for each we will write a module in the code. It represents the call relationship between 2 modules, so the root calls all these 3. We will also represent the data flow that occurs between the different modules through a some notation we will see, we will write the name of that at a d 1, d 2 and so on.

(Refer Slide Time: 17:10)



Structure Chart

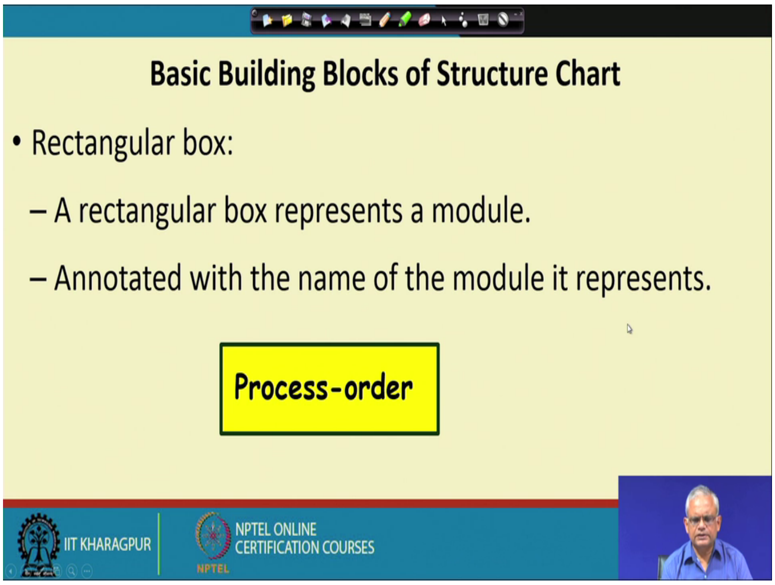
- Structure chart representation
  - Easily implementable using programming languages.
- Main focus of a structure chart:
  - Define the module structure of a software,
  - Interaction among different modules,
  - Procedural aspects (e.g, how a particular functionality is achieved) are not represented.

Now, let us see what are the elements of the structure chart, the structure chart identifies the module structure and we can easily program the structure chart. It is actually the high level design and in the next step based on the structure chart we do a detailed design where for each module we identify the algorithms data structures and so on and that is directly transferred to code.

So, here in the structure chart we only represent the module structure what are the modules, what is the call relationship among the modules. So these are the modules the different modules M 1, M 2, M 3 etcetera and the call relationship among modules and the data exchanged among modules, but we do not represent in the structure chart. What exactly happens in M 2 what is the procedure or the algorithms used in M 2 or M 3 that we do in the detailed design, where we identify the algorithms the data structures that are used and we develop a module specification for each module.

So, let me just repeat that part that the structure chart representation is called as the high level representation, the high level design and based on the high level design we carry out the detailed design, where for each module we develop a module specification which contains the data structures that are to be used in the module and also the algorithms that would be used in this module.

(Refer Slide Time: 19:36)

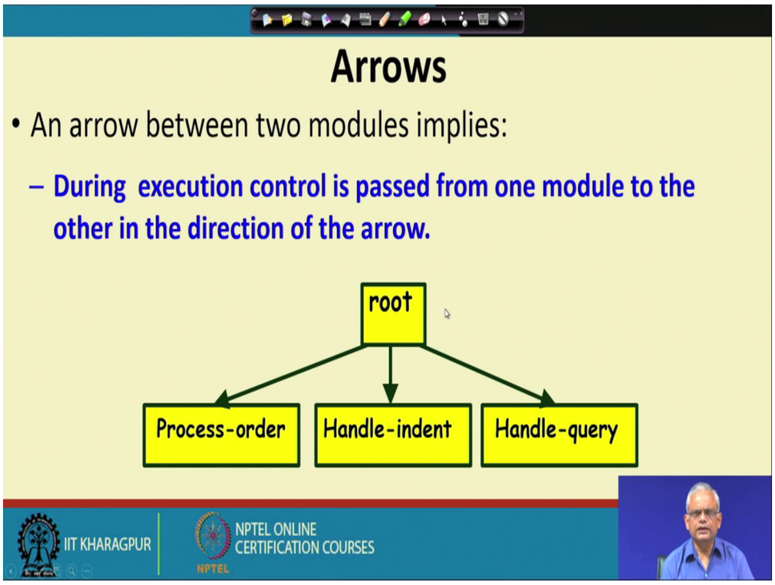


Basic Building Blocks of Structure Chart

- Rectangular box:
  - A rectangular box represents a module.
  - Annotated with the name of the module it represents.

Now, let us see what are the basic building blocks of the structure chart of course, the central part of a structure chart at the modules. The modules are rectangles and the name of the module is written very simple that we draw a rectangle for each module and we just write the name of the module on the rectangle.

(Refer Slide Time: 20:07)



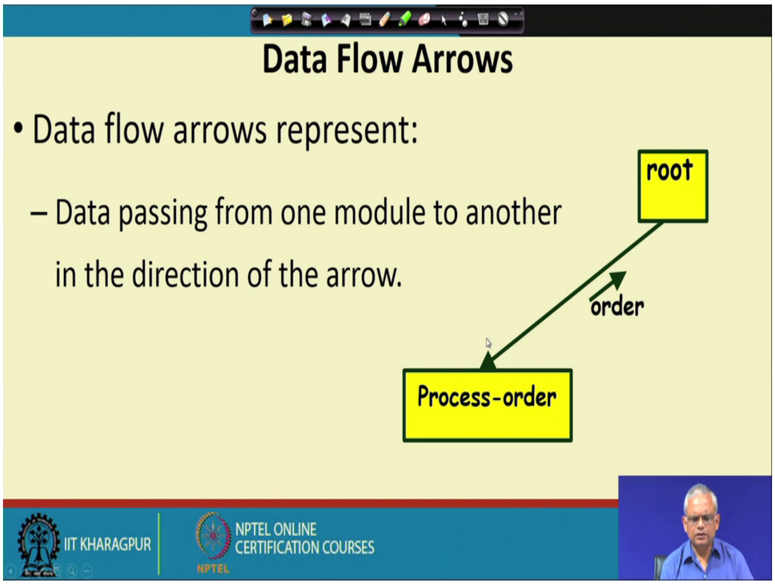
Arrows

- An arrow between two modules implies:
  - During execution control is passed from one module to the other in the direction of the arrow.

The second element of a structure chart is the arrows the arrow represents the invocation relation that is which module call switch module. And when there is a call from one module to another module, then the control is passed from one module to the other. So if root calls process order then the control is passed from root to the process order and then once the process order completes the control is back with root and then it may call handle indent and then the handle query.

But then this sequential aspect whether it will call first this one, next this one, third is the handle query it is not really represented here. So this just says that they are called, but the order is not indicated here, it may first call handle indent and then make all process order and so on.

(Refer Slide Time: 21:22)

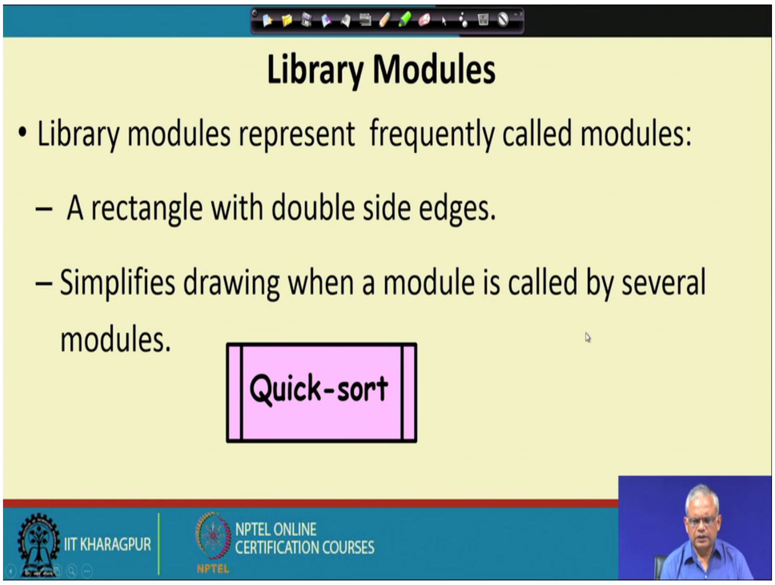


Data Flow Arrows

- Data flow arrows represent:
  - Data passing from one module to another in the direction of the arrow.

The third item or the third element is the data flow. So represented here the data flow aspect we draw a arrow here on this invocation showing the direction of the data flow. So what this represents is that the root calls the process order and the process order gives back the order to the root, very simple notations.

(Refer Slide Time: 22:09)

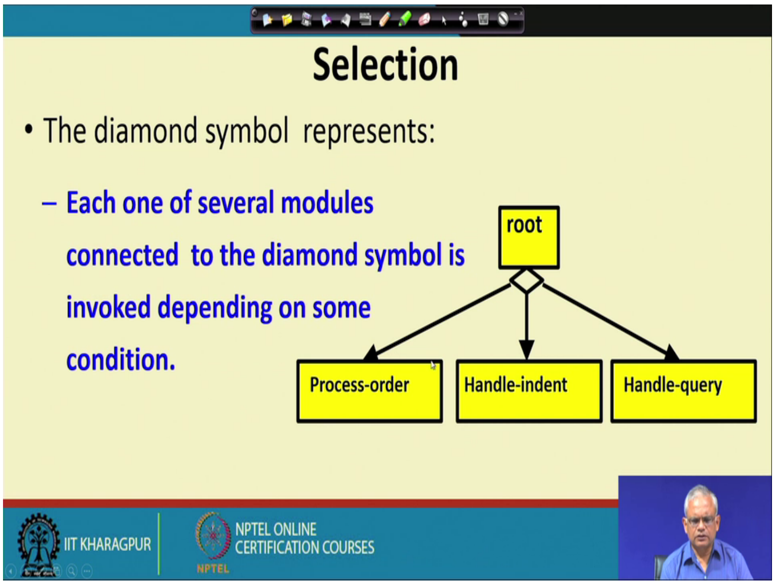


Library Modules

- Library modules represent frequently called modules:
  - A rectangle with double side edges.
  - Simplifies drawing when a module is called by several modules.

Now, there is one more element in the drawing of the structure chart which is the library module. It is a module actually so it is a rectangle, but then we just draw these 2 parallel lines here and this indicates the ones that are made into a library some of the functions are called frequently by other modules and to represent that these are the library modules, we just draw them in this notation.

(Refer Slide Time: 22:53)

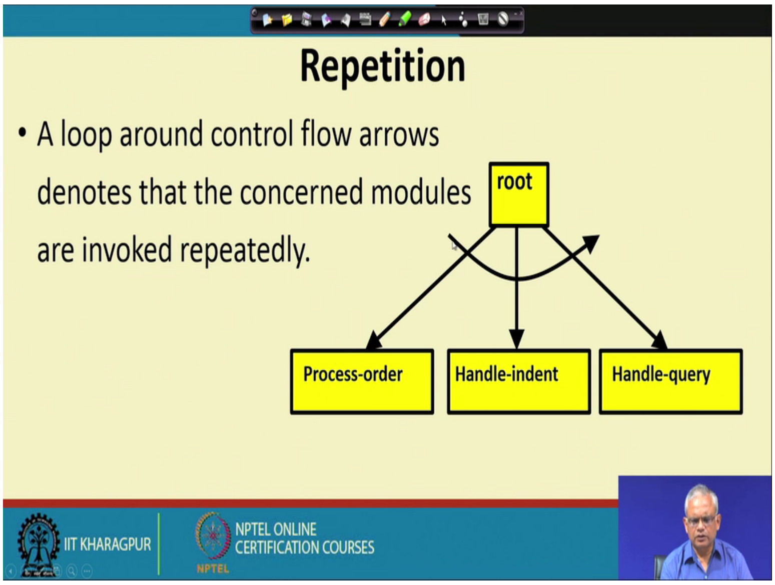


Selection

- The diamond symbol represents:
  - Each one of several modules connected to the diamond symbol is invoked depending on some condition.

There are other few other notations for example selection, so you draw a diamond here, we draw a diamond here and it indicates selection. So it will call one of the module depending on some decision here.

(Refer Slide Time: 23:28)

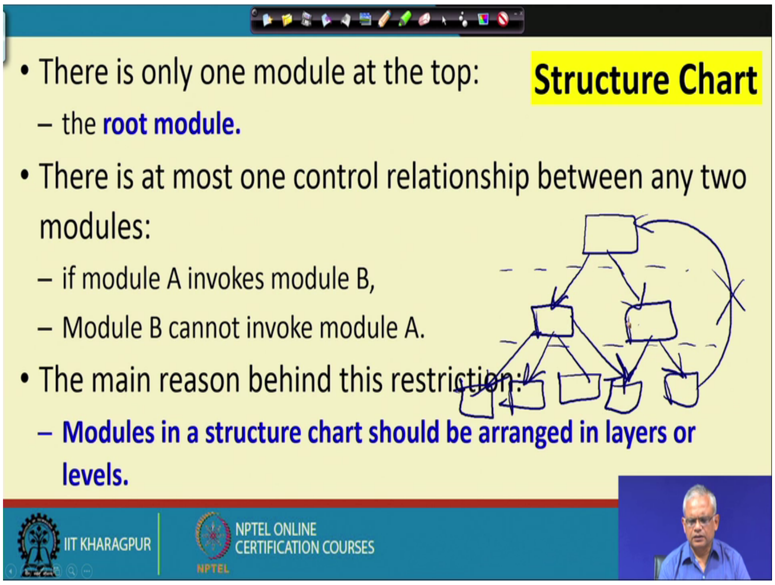


Repetition

- A loop around control flow arrows denotes that the concerned modules are invoked repeatedly.

One more element here is the repetition on the invocation arrow if we draw an arc shaped arrow here that indicates that these modules are called many times; in a loop.

(Refer Slide Time: 23:52)



Structure Chart

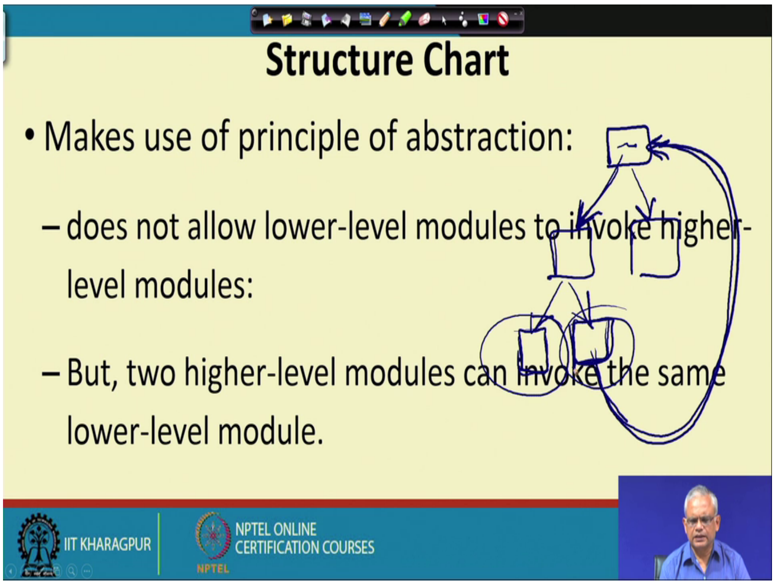
- There is only one module at the top:
  - the **root module.**
- There is at most one control relationship between any two modules:
  - if module A invokes module B,
  - Module B cannot invoke module A.
- The main reason behind this restriction:
  - **Modules in a structure chart should be arranged in layers or levels.**

Now, the structure chart there will be only one module as the root and then there are control relationship between two modules. If the module invokes the other module and based on the methodology that we give and also any structure design methodology we always come up with a structure like this. It is a tree like structure these are arranged in levels, but then we do not have lower level module calling a higher level module we do not have this kind of arrow here.

So, there is the control relationship from one module to the next level of module, but we cannot have a module calling a higher level module this is an example of a bad design. We had discussed while discussing about the good characteristics of design said that this as the design has to be layered and higher layer module can call a lower layer module not vice versa.

We will see how the design methodology that we discuss it will give us it will automatically result in a layered representation or a layered design and there is no back arrows in that. If you are by chance while applying any methodology you come up with arrows which invoke the higher level modules mostly that is not a good design unless there are some exceptional situation, it does not it indicates that it is a bad design and you have to be careful.

(Refer Slide Time: 26:25)



Structure Chart

- Makes use of principle of abstraction:
  - does not allow lower-level modules to invoke higher-level modules:
  - But, two higher-level modules can invoke the same lower-level module.

The main reason why the lower level design modules should not call the higher level modules is, because of the principle of abstraction. This we had also discussed earlier that due to the principle of abstraction at any time we are concerned about only some modules you do not have to get concerned about all the modules together and if our module structure is a tree like diagram. If we look at these modules, bottom level modules then we do not really are concerned about the upper layer modules, because the results are produced solely by these modules based on some data that is passed here.

But just imagine that if we have a back arrow here, so this will violate the principle of abstraction and lower level module is calling a higher level module. And therefore, understanding these design will be difficult, because we when we try to understand this one, we see that we need to understand the one that it calls and then we need to understand the one that it calls and we will end up just going round and round and same thing happens while debugging.

An error if there is a failure we are trying to debug and find where which module has the error we see that we are looking at all modules round and round and we will have difficulty in debugging and understanding and so on. And therefore, the layering of the module it is an instance of the principle of abstraction and helps us to develop a design which is easily understandable and we can easily debug that. We are almost end of this lecture we will stop here and we will continue in the next lecture.

Thank you.