

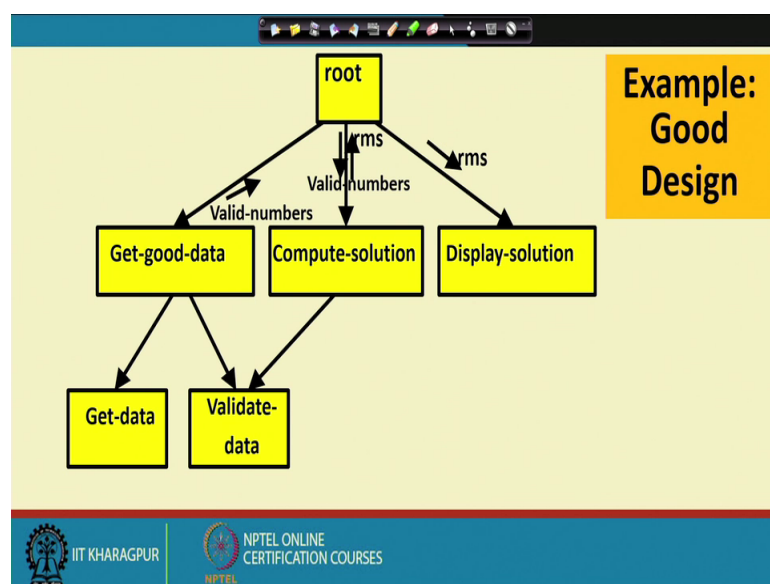
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 28
Essentials of Structure Chart

Welcome to this lecture, in the last lecture we were discussing about structure design some very fundamental aspects of structure design. We said that the structure design we take the result of the structured analysis that is the DFD model and through methodology we come up with the structure chart representation of the design. Therefore, the output of the structure design is a structure chart. The structure chart is also called as the high level design or the software architecture and this high level design is next transformed into detailed design, well for each module we write the module specification that is the algorithms to be used in the data structure.

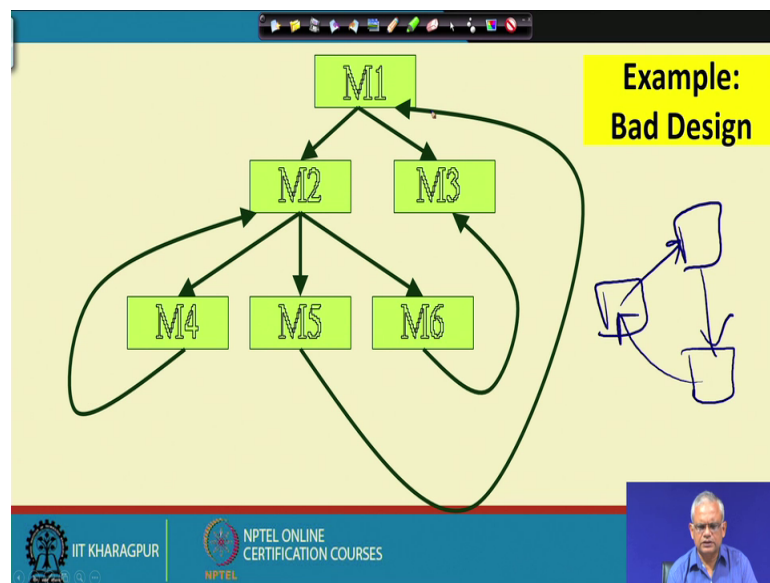
But towards the end of the last lecture we were discussing about what is a good structure design and we said that a good structure design should have a layered structure and lower layer module should not call a higher layer module and that will violate the principle of abstraction. It will make it difficult to understand the design and also to debug. Now, let us look at how a good structure design should look like.

(Refer Slide Time: 02:02)



So, if we apply structure design methodology we should typically come up with this kind of a structure, where we have nice layering the different modules are arranged in layers and also there is no back arrow here. That is not violating the principle of abstraction of course, a module at a layer different modules can come called the same module and if many modules are using a module then that is a library module, need to make it into a library module and that can be called at different layers.

(Refer Slide Time: 02:55)

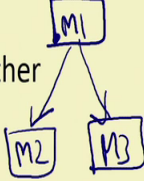


This is an example of a bad design, now see here even though we have some layering here, but then there are back arrows here. So, a bad design is one where we either do not have layering, so we just have arbitrary module structure and also there are back arrows here. So, either layering is not present or back arrows are present this will be called or this will be considered as a bad design.

(Refer Slide Time: 03:44)

Shortcomings of Structure Chart

- By examining a structure chart:
 - we can not say whether a module calls another module just once or many times.
- Also, by looking at a structure chart:
 - we can not tell the order in which the different modules are invoked.



The diagram shows a structure chart with a root module 'M1' in a box. Two arrows originate from the bottom of 'M1', pointing to two child modules 'M2' and 'M3', also in boxes. This represents a hierarchical relationship where M1 calls M2 and M3.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

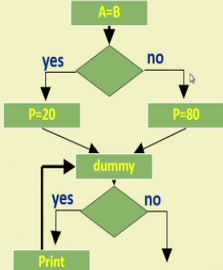
Before we see how to come up with a structure chart representation, let us be aware of what are the shortcomings of a structure chart. If we have a call relation represented between two modules we do not know here it just says that this module M 1 calls M 2 and M 3. But it does not tell us that how many times does it call M 2 does it call 5 times, does it call 10 times and M 3 20 times etcetera. These information is not represented it just says that it calls one or more times and also by looking at this representation we do not know whether M 1 calls M 3 faster M 2 first. So, that information is not represented here.

(Refer Slide Time: 04:52)

Flow Chart (Aside)

- We are all familiar with the flow chart representations:
 - Flow chart is a convenient technique to represent the flow of control in a system.

- A=B
- if(c == 100)
- P=20
- else p= 80
- while(p>20)
- print(student mark)



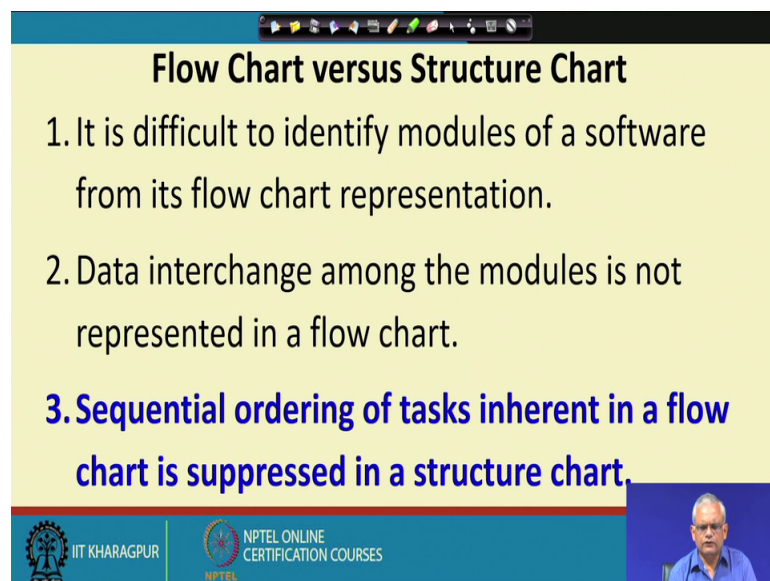
```
graph TD
    A{A=B} -- yes --> B[P=20]
    A -- no --> C[P=80]
    B --> D[dummy]
    C --> D
    D --> E{ }
    E -- yes --> D
    E -- no --> F[Print]
```

The flowchart starts with a decision diamond labeled 'A=B'. If the answer is 'yes', it goes to a process box 'P=20'. If 'no', it goes to 'P=80'. Both paths merge into a 'dummy' process box. From 'dummy', it goes to another decision diamond. If 'yes', it loops back to the 'dummy' box. If 'no', it goes to a 'Print' process box.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Just to complete our discussion we will discuss about; how does it compare with a flowchart, even though this is a bit of digression, but for sake of completeness we will see that how does a module structure represented in a structure chart represent it differs from a flowchart. We know flowchart is a very popular technique represents the control flow as different statements are executed we draw them. It is a very popular notation I am sure that everybody knows about this, that we have decision loops and so on. These are sequential flow decision and back arrows would be loops and so on.


(Refer Slide Time: 05:59)



Flow Chart versus Structure Chart

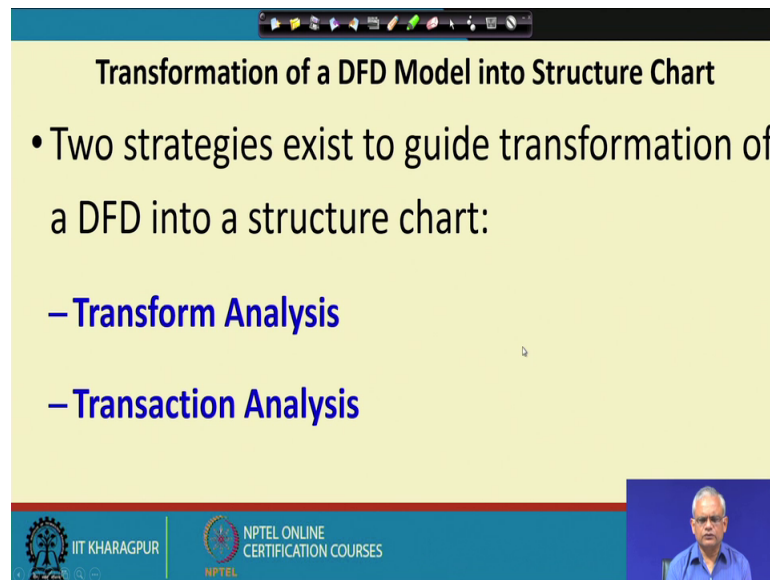
1. It is difficult to identify modules of a software from its flow chart representation.
2. Data interchange among the modules is not represented in a flow chart.
- 3. Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Given a flowchart representation the first thing we would notice is that the flowchart does not tell about what are the modules, the module structure of the software is not represented in a flowchart. The second difference is that the flow chart represents only the control flow among the statements, but what data is exchanged between these is not represented. And also in looking at a flow chart we can trace through how the execution occurs. So, there is a sequential nature that is we can identify after which statement which statement etcetera get executed, but in a structure chart there is no such order and this is suppressed in a structure chart.

(Refer Slide Time: 07:10)



The slide is titled "Transformation of a DFD Model into Structure Chart". It lists two strategies to guide the transformation of a DFD into a structure chart:

- Two strategies exist to guide transformation of a DFD into a structure chart:
 - **Transform Analysis**
 - **Transaction Analysis**

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.

Now, let us look at the methodologies to transform DFD representation into a structure chart representation. Given a DFD model depending on the context or depending on the specific DFD we will either apply the transform analysis or the transaction analysis. These two are two different methodologies and therefore, by looking at the DFD we have to first understand which methodology is applicable and then we must apply that methodology which is the systematic procedure to come up with a structure chart representation of a DFD.

So, let us now look at what exactly is involved in transform analysis, that is what is this methodology and also in which situation or by looking at the given DFD model can we make out whether to apply transform analysis and similarly we will look at what are the steps in the transaction analysis. And also by looking at a given DFD can we identify whether to apply transaction analysis to that or transform analysis to that. Over the next few minutes we will be addressing that issue.

(Refer Slide Time: 09:09)

Transform Analysis

- The first step in transform analysis:
 - Divide the DFD into 3 parts:
 - input,
 - logical processing,
 - output.

The diagram shows a DFD with three processes: P1, P2, and P3. P1 is labeled 'input', P2 is labeled 'logical processing', and P3 is labeled 'output'. Arrows indicate data flow from P1 to P2, and from P2 to P3. The processes are enclosed in hand-drawn ovals, and the entire diagram is annotated with blue ink.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

First let us look at the transform analysis this is much simpler than the transaction analysis. Here given a DFD model we divide it into 3 parts input, logical processing and the output. So, given some DFD model let us say this is a DFD model this is a DFD model and of course, we will have the data annotations and so on and the annotations on the processes name of the processes. We will divide this DFD into 3 parts based on some observation which we will discuss we will label them as the input part and as the name implies it takes the input from some processor some from user and it transforms it from physical input to some logical form. For example, it may take input as a text and come up with a tree structure or it may take numbers and develop a table structure of the number.

And then the processing part that represents the processing that is done on the input data and also from other data elements. And then finally, the output part it transforms data from a logical representation like a table or a tree or something and it produces the output which is nicely formatted output. So, from our discussion can consider that the input part is responsible for reading the input and carrying out on that input some validation and structuring into some logical form. The output part is does just the reverse of this it has some modules, which convert the logical data into physical part and it takes care of formatting and displaying it nicely for the user, the rest of the modules sorry rest of the processes at the central processing.

(Refer Slide Time: 12:19)

Transform Analysis

- Input portion in the DFD:
 - processes which convert input data from physical to logical form.
 - e.g. read characters from the terminal and store in internal tables or lists.
- Each input portion:
 - called an **afferent branch**.
 - Possible to have more than one afferent branch in a DFD.

```
graph TD; Move((move)) --> Validate((Validate-move)); Validate -- board --> Display((Display-board)); Display -- game --> Check((Check-winner)); Check -- result --> Display; Play((Play-move)) <--> |board| Display; subgraph Afferent; Validate; end; subgraph CentralProcessing; Play; end;
```

So, the input portion of the DFD it converts input data so it reads data from another process or from the user and converts from physical form to logical form; for example, it may read characters from a terminal and store in a list. Technically we call its input part as a afferent branch so this is the technical term the input part of a DFD is called as afferent branch so far we are not really discussed that. Given a DFD model what do you observe so that we decide to apply the transform analysis that we will discuss next. Right now we are just looking at the methodologies of the transform analysis and we said that given a DFD model we need to identify the input part which we call as the afferent part or the afferent branch the efferent branch at the output part in the central processing ok.

So, this is the DFD model which is given to us and we see that this takes the input here you see others are producing output or does not take anything just take the data here. This takes the data from another process or maybe from the user and we can easily identify that this is the input part at the afferent part.

(Refer Slide Time: 14:24)

Transform Analysis

- Output portion of a DFD:
 - transforms output data from logical form to physical form.
 - e.g., from list or array into output characters.
 - Each output portion:
 - called an **effluent branch**.
- The remaining portions of a DFD
 - called **central transform**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

These is the effluent part and just see here that this is the one that is producing output and this we will call as the afferent part, the afferent part takes the input and it stores in some logical form we call the logical form as a board here. And the output part takes the logical form and it produces a nice output properly formatted and the rest we call as the processing part, central processing part

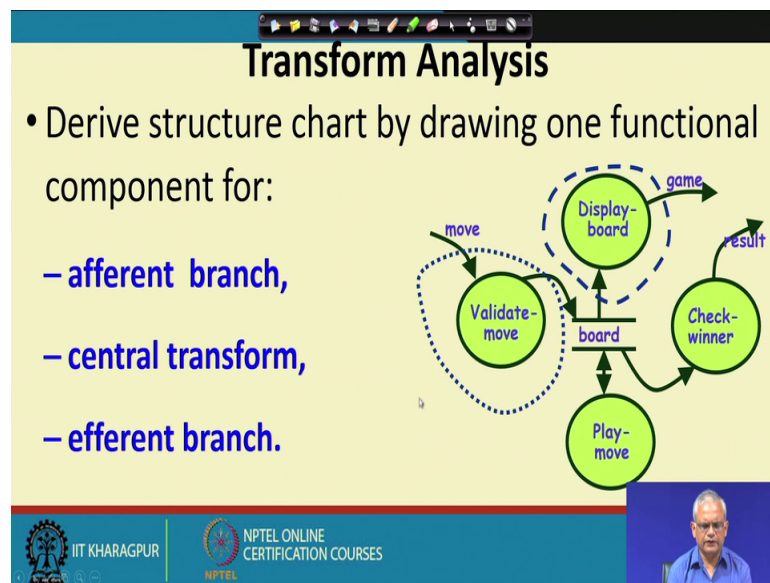
But then you might ask the question that; so this is also producing some output why not make it a output or afferent why not this part of the afferent branch. Here comes the design alternative you can make it a afferent part, but then that will not become a good design because the main tasks done by the check winner is a processing. Even though it produces an output, but still we will consider as a processing part this is the discretion that we have to use based on the experience. Just by observing that it takes input data produces output data many times is not sufficient to label them as afferent or effluent branch. We need to see what really they do the check winner even though it produces some output, but the main work it does is processing and not really converting data from logical form and just nicely displaying.

So, given a DFD we need to if we find that the transform analysis is applicable we divide it into 3 parts the input part, the processing part and the output part which technically called as the afferent branch, effluent branch and the central transform. And once we have done that we just need to represent that in the form of a structure chart. So, if this is

the input part this is the output part and this is the central processing part the ones that are not part of either input or output they become the central processing part or the central transform. And from this coming up with the module structure it is straight forward we just draw root which we will call these 3 the validate move, display board and let us say check winner or we will call it as process move. Process move we will call it we need not use the same name we will say that get move process move and output move output game.

So, the transform analysis is really very simple once you determine that it is applicable, transform analysis is applicable to a DFD. We identify the input part. output part and the central processing and in the structure chart we draw a root level and from there we call these 3 modules.

(Refer Slide Time: 19:22)



So, the output of transform analysis is structure chart, where we just draw the 3 modules corresponding to the afferent branch, central transform and the efferent branch.

(Refer Slide Time: 19:38)

The slide is titled "Transform Analysis" in a yellow box. It contains the following text:

- Identifying input and output transforms:
 - requires experience and skill.
- Some guidelines for identifying central transforms:
 - Trace inputs until a bubble is found whose output cannot be deduced from the inputs alone.
 - Processes which validate input are not central transforms.
 - Processes which sort input or filter data from it are.

A diagram shows two bubbles, P1 and P2, connected by arrows. An arrow labeled 'd1' points into P1, and an arrow labeled 'd2' points into P2. An arrow labeled 'd3' points from P1 to P2, and another arrow labeled 'd4' points from P2 to the right.

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.

As we have seen that it is not really totally mechanical it requires some experience and skill even though input data to a process indicates that it should be part of the input, but then many times it may not be. And similarly just because this produces output may not be an output part, it maybe central transform or something.

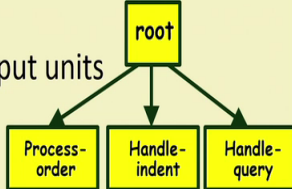
But there are some guidelines how to identify the central transform, the guideline essentially says that it should not be involved in its main activity should not be to just read data from user and convert it to logical form. And similarly it should not just transform logical data into physical part here we trace the input, where the output cannot be deduced from the input alone so that means, that it is not a input branch. In the input branch we just read the input in any input branch the process reads the input and it does some transformation and that.

So, the output is determined based on the input to this, but for a central processing unit a central transform, it will use some previous data and so on and just by looking this data will not be able to identify will not be able to that there will be no simple function to convert this to this. And of course, the processes which just validate input etcetera these are not central transform their part of the input part, but then if it does some processing on the input like filtering, sorting, validating sorting and filtering these can be central transforms.

(Refer Slide Time: 22:15)

Transform Analysis

- First level of structure chart:
 - Draw a box for each input and output units
 - A box for the central transform.
- Next, refine the structure chart:
 - Add subfunctions required by each high-level module.
 - Many levels of modules may required to be added.



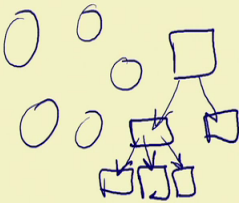
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, once we identify the input part, output part and central transform we just represent them three different modules and have a called relationship. And once we have the module structure like this, next we refine it by drawing any module that may be necessary to carry out the work of this for example, we might see that we need to let us say validate we need to let us say format it nicely and so on.

(Refer Slide Time: 23:16)

Factoring

- The process of breaking functional components into subcomponents.
- Factoring includes adding:
 - **Read and write modules,**
 - **Error-handling modules,**
 - **Initialization and termination modules, etc.**
- Finally check:
 - Whether all bubbles have been mapped to modules.



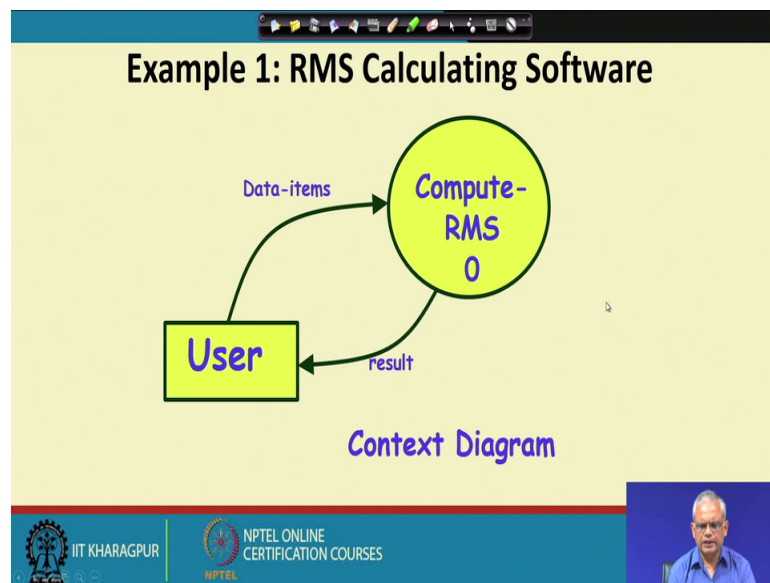
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Once you have the basic structure of a structure chart, we factor the lower level modules we might I add read write modules, error handling modules, initialization modules and so

on. And the final check whether we have correctly transformed a DFD into a structure chart is to observe and see that each of the process belongs to either input part or the output part are the central transform and they are represented on the structure chart. It should not be the case that we have not included a process neither input, output or the processing part.

So, once by the factoring once we have the basic module structure we add modules at the lowest level to add read write modules read from terminal, read from file, read from network and so on, error handling initialization etcetera. So, we can add a couple of layers here if necessary and these are the lower level modules which will be called by this module to do some activities and this is called as factoring, where the basic skeleton of the structure chart we might need to enhance it by factoring.

(Refer Slide Time: 25:05)





Let us look at an example this is the simplest example we had discussed it earlier for developing the DFD representation computing the root mean square. And the context diagram if you remember we had a single user who would input data items and this will return the result which is the root mean square.

(Refer Slide Time: 25:36)

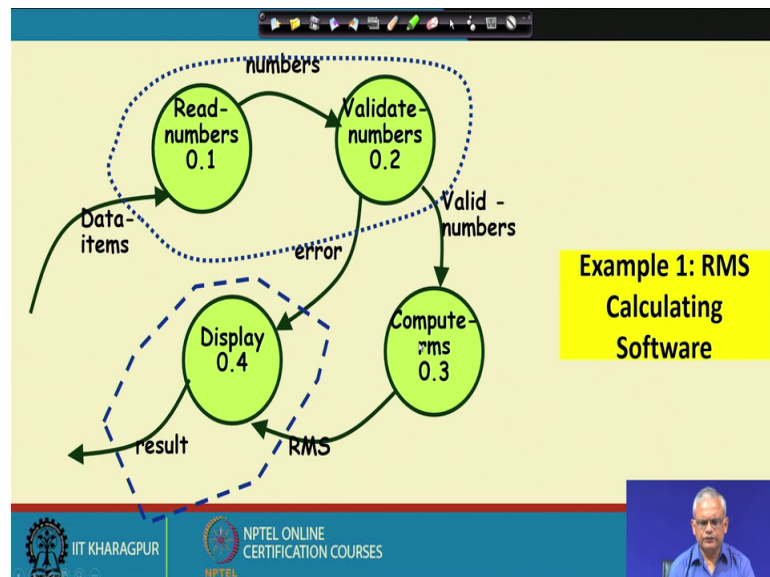
Example 1: RMS Calculating Software

- From a cursory analysis of the problem description,
 - easy to see that the system needs to perform:
 - accept the input numbers from the user,
 - validate the numbers,
 - calculate the root mean square of the input numbers,
 - display the result.



And if you remember we had developed the level 1 diagram where we identified the activities of the RMS software, that is accept input validate the number; whether they lie between minus 1000 and plus 1000 and calculate the root mean square and display the result.

(Refer Slide Time: 26:01)



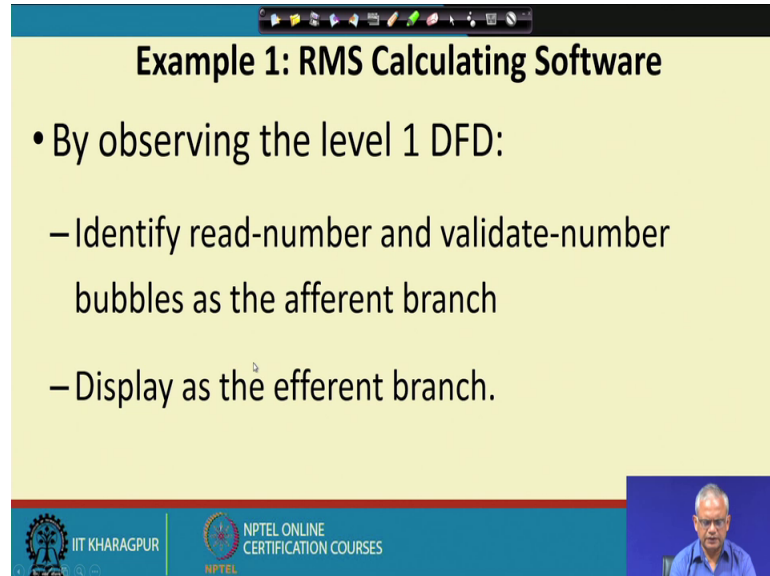
And based on that we had come up with such a DFD representation, read data items and these numbers are passed on to validate number either produces valid numbers and these are passed on to the compute RMS, which produces the result and this is passed on to the

display. And the validate numbers may also produce some error messages which are passed onto the display.

(Refer Slide Time: 26:38)

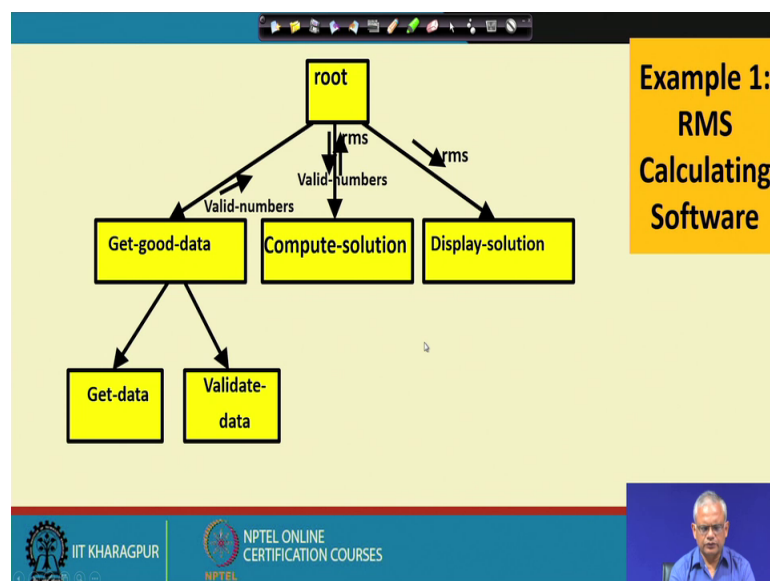
Example 1: RMS Calculating Software

- By observing the level 1 DFD:
 - Identify read-number and validate-number bubbles as the afferent branch
 - Display as the efferent branch.



And now if we apply the transform analysis we will make this as the input part, because validate read etcetera are part of the input, display is part of the output and the rest are the processing part. And for this we will draw 3 modules for the input part, output part and the central processing part.

(Refer Slide Time: 27:14)



So, this we will get here get good data, compute solution and display solution and we might factor get good data into get data and validate data. Observe that these two were two different bubbles on the DFD representation. We have made them into two modules here, but that is not really done all the time, it is just a guideline that if you have some bubble many bubbles match mapped on to a module, then we need to factor that and this will become the bubbles.

We are almost at the end of this lecture we looked at the transform analysis methodology, to transfer a DFD into structure chart representation. Next we look at the transaction analysis and how to come up with the module structure for a DFD based on the transaction analysis. And we will also discuss when to apply transform analysis and when to apply the transaction analysis.

We will stop now and continue in the next lecture.

Thank you.