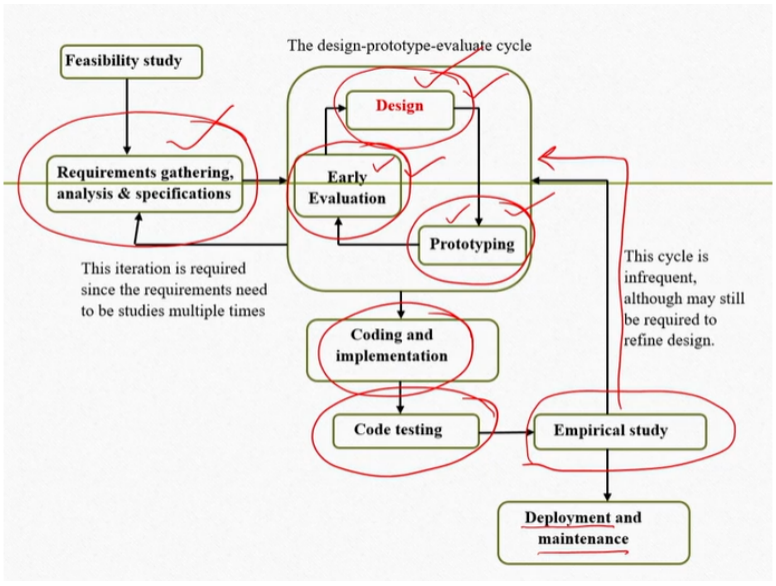
Design and Implementation of Human – Computer Interfaces

Prof. Dr. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Module No # 05
Lecture No # 20
Basics of System Design

Hello and welcome to the NTPEL MOOC's course on design and implementation of human computer interfaces lecture number18 where we will start our discussion on system design. So before we start we will quickly recap the interactive system development life cycle and then we will understand where we what we discussed and where we are at present. Then from there we will forward the discussion.

(Refer Slide Time: 01:17)


Feasibility study
The design-prototype-evaluate cycle
Design
Requirements gathering, analysis & specifications
Early Evaluation
Prototyping
This iteration is required since the requirements need to be studies multiple times
This cycle is infrequent, although may still be required to refine design.
Coding and implementation
Code testing
Empirical study
Deployment and maintenance

So if you recollect in the interactive system development life cycle there are several stages some of the stages from some cycles. So; essentially the lifecycle contains iterative components as well as some non-iterative components. So we started our discussion with the requirement gathering analysis and specification stage. Where; we learned about how to create an SRS document or software requirement specification document.

In the document we can include both functional requirement as well as non-functional requirements. After the requirement gathering stage we have a cycle which is design prototype and evaluation cycle. So there are 3 soft stages involved here one is the design stage one is the prototyping stage and the third one is the evaluation of the prototyping that stage.

Now if you may recollect here what we mentioned is that design actually refers to 2 thing one is interface design and system design or code design. So we have so far discussed interface design and this cycle refers to design of the interface. So once we have some design ideas then we create a prototype get it evaluated and refine our design based on the evaluation results and so on the cycle continuing still we arrive at some stable interface design.

Also design during the discussion we have learned about how to go for designing an interface particularly we learned about design guidelines. In prototyping we learned how to make different prototypes ranging from low cost and affordable prototypes to high cost system oriented prototypes. So there are broadly 3 types low fidelity, medium fidelity and high fidelity.

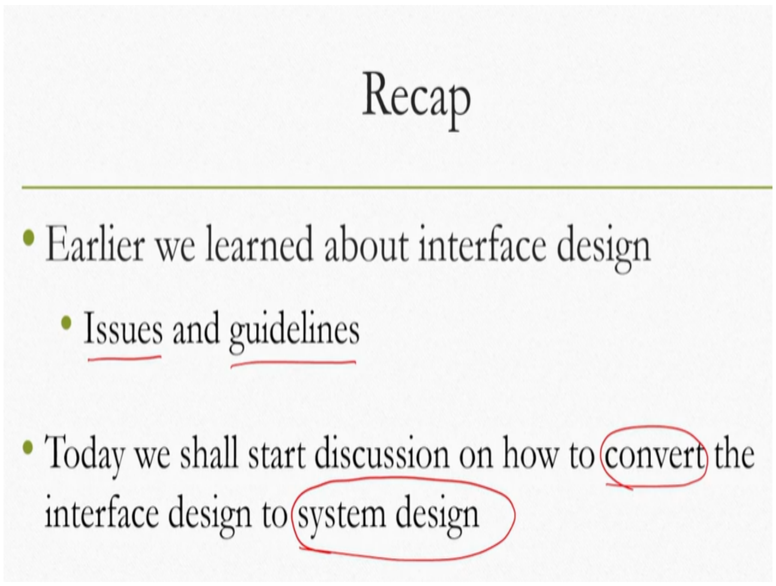
So learned about these different prototypes and how to use them also we learned in details about different prototype evaluation techniques. We learned about to broad techniques one is evaluation with experts and evaluation with users with the objective of getting the evaluation done rapidly and without much cost. So once we arrive at some stable design then we go for system design where we design the components of the system in modules and some modules.

Now then that is followed by implementation of the design in the coding and implementation stage then comes testing of the code that is another stage. A very important stage so after testing we get an executable system this is followed a special stage called empirical study where we test the system or usability. So getting only an executable system is not the goal our overall goal is to get a usable system.

So we have to conduct empirical study stage now occasionally it may happen that in empirical study we found some usability problems. So we need to refine our interface designs and the subsequence stages we need to carry out. So there is a loop from here but it is very infrequent unlike the design prototype evaluate cycle. So after we arrive at an executable and usable system we go for deployment followed by maintenance stage that is our overall interactive system development life cycle.

Now in this life cycle we have so far covered requirement gathering stage and interface design stage. In this lecture we are going to take the discussion forward. So we have covered under interface design stage how to design the interface create prototype as well as evaluate the prototype. So entire cycle we have covered for interface design and then in this lecture onwards we will take it forward.

(Refer Slide Time: 05:54)

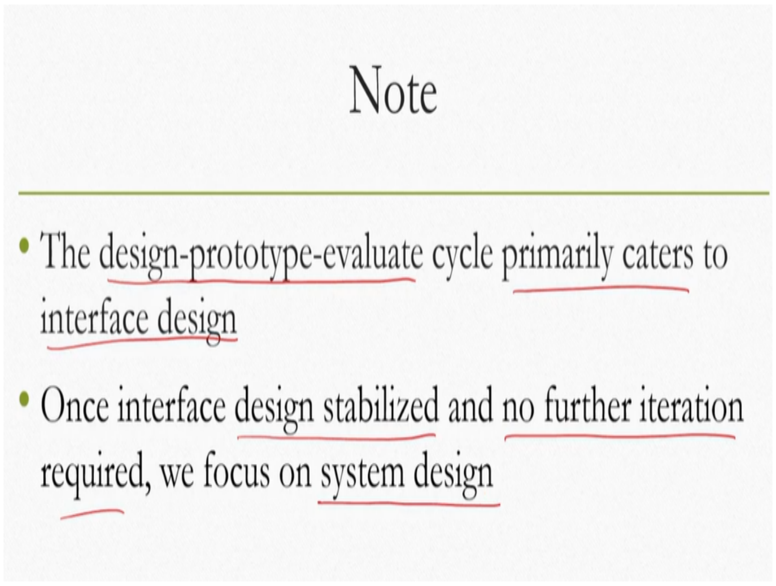


Recap

- Earlier we learned about interface design
  - Issues and guidelines
- Today we shall start discussion on how to convert the interface design to system design

So we have learned about interface design issues and guidelines in the earlier lectures. And now we are going to discuss once we design an interface how to convert the interface design to the design of a system specifically software. Note that here in this entire course wherever we use the term system we primarily refer to the software. So we are going to talk about design of the system which is based on the design of the interface that we arrive at executing the design prototype evaluate cycle.

(Refer Slide Time: 06:46)

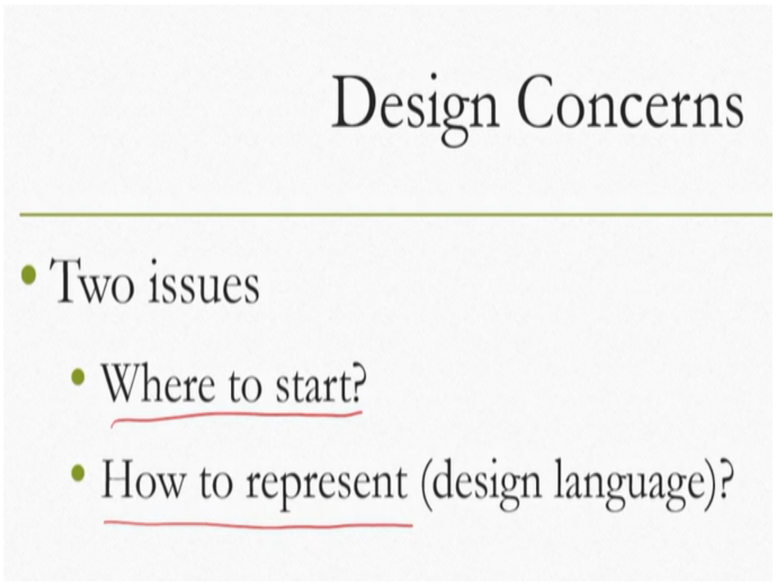


Note

- The design-prototype-evaluate cycle primarily caters to interface design
- Once interface design stabilized and no further iteration required, we focus on system design

Now the design prototype evaluate cycle as I mentioned primarily caters to the interface design aspect of the lifecycle. Once we get a stable design so the design is stabilized that means no further usability issues are discovered or in significant number those issues are discovered after the evaluation stage. So no further iteration of the cycle is done and we reach at a stable design and then we focus on converting that stable interface design to the design of a system.

(Refer Slide Time: 07:34)



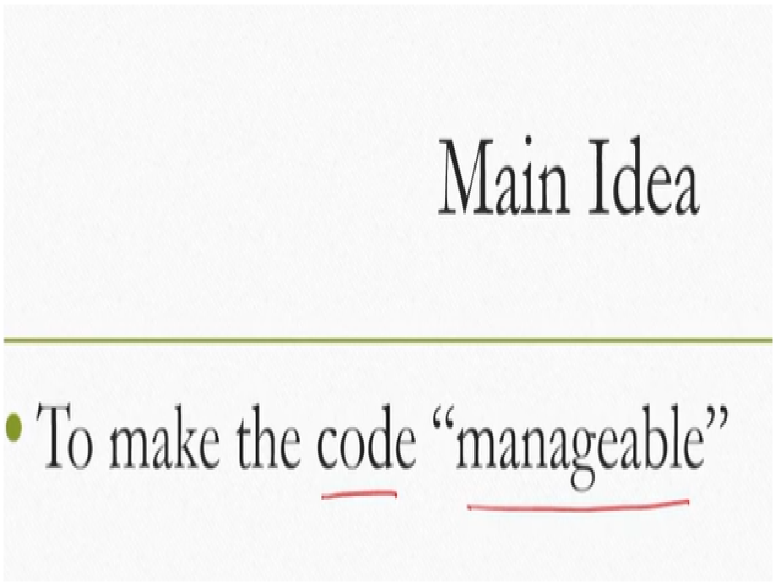
Design Concerns

- Two issues
  - Where to start?
  - How to represent (design language)?

Now when we are taking of design of a system the same 2 issues come back what are those issues? One is where to start? And the second is how to represent? In case of interface design we encounter the same issues and if you recollect we discuss that to answer the question where to start the interface design process we can start with design guidelines. Also our intuition experience this can be a starting point and we say how to represent interface design we talked about prototypes to express the design.

So in case of interface design prototypes can be used to represent the design or essentially that is the design language. So in case of system design this same issues are there where to start and how to represent?

(Refer Slide Time: 08:35)



Main Idea

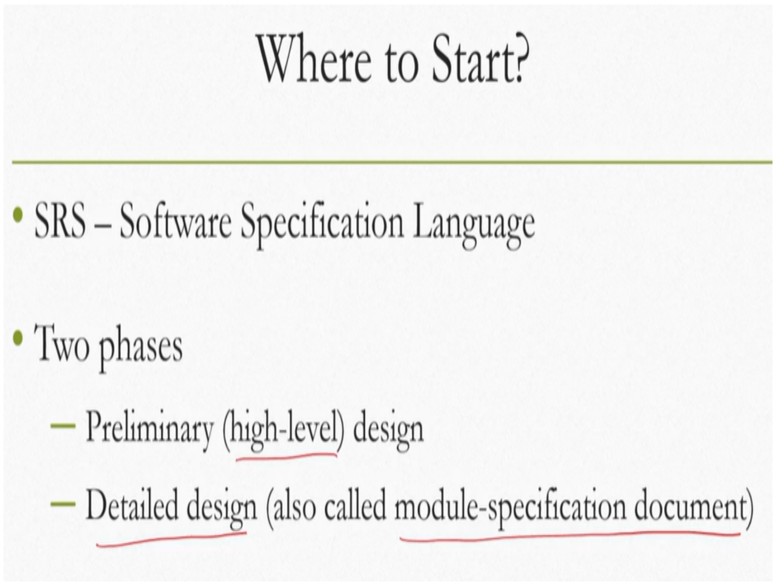
- To make the code "manageable"

Now the core objective of a system design is to make the final code manageable what this means? So when we are saying that we need to go for design of an interface we are primarily concerned about what the user perceives what the user gets to see a interact with or gets to perceive an interact with. So there our primary concern is our user now when we are talking of system design our primary concern is implementation that is we need to built a software for that we need to write a code and that code how to write it in a way such that it is manageable.

Why that issue is important? Generally codes return in a team for complex system, typically complex systems require writing of hundred or thousands of lines of code and if single persons are interested with this task then it becomes very difficult. And there is possibility of huge number of errors. To address this problems what happens is that? The code is divided into units and different teams are entrusted to develop the units separately and then finally combine them together to create a whole system.

So that is typically what is called a modular design to manage the overall development process smoothly. So when we are going to designing the system we should keep that in mind and try to design it in a modular way such that so that when we convert it to code the modular development is done or is possible. So our main objective in system design is to help in managing the code development well and help in making the code manageable. So let us begin with the first question where to start the system design process?

(Refer Slide Time: 10:52)


Where to Start?

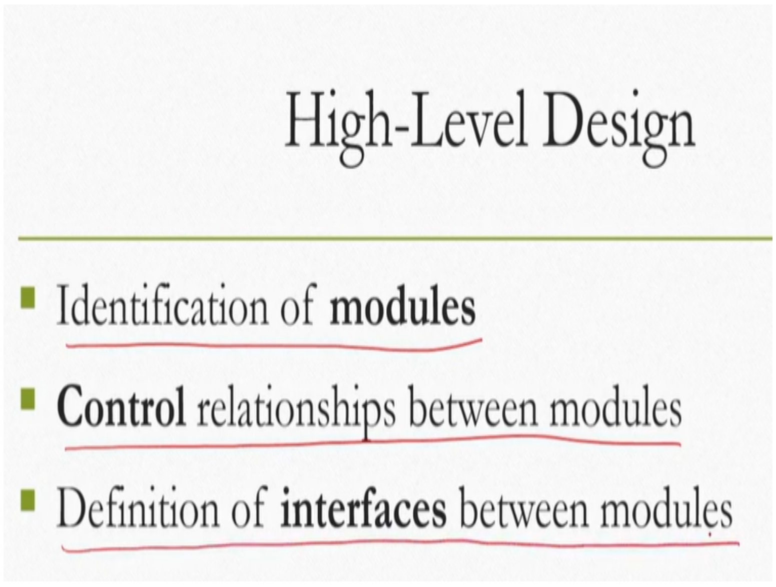
- SRS – Software Specification Language
- Two phases
  - Preliminary (high-level) design
  - Detailed design (also called module-specification document)

Now earlier in the requirement gathering stage we saw the output is SRS or software requirement specification document. Let us begin with the question where to start so we start with the SRS document or software requirement specification document. If you recollect this is the outcome of the requirement gathering analysis and specification and we start our system design with this document.

Now when we are going for designing our system there are broadly 2 phases what are those 2 phases? One is the preliminary design where we go for designing a system at a very high level. So this is essentially a high level design of the whole system and then there comes the detailed design phase. Now in this phase what happens is that? We go to the minute detail of the system now this is also known as modules specification document.

So when we are creating a detailed specification document this is also called module specification design document. So in the preliminary stage we design at a high level at the modular level and in detail design stage we go to minute details of the modules.

(Refer Slide Time: 12:36)



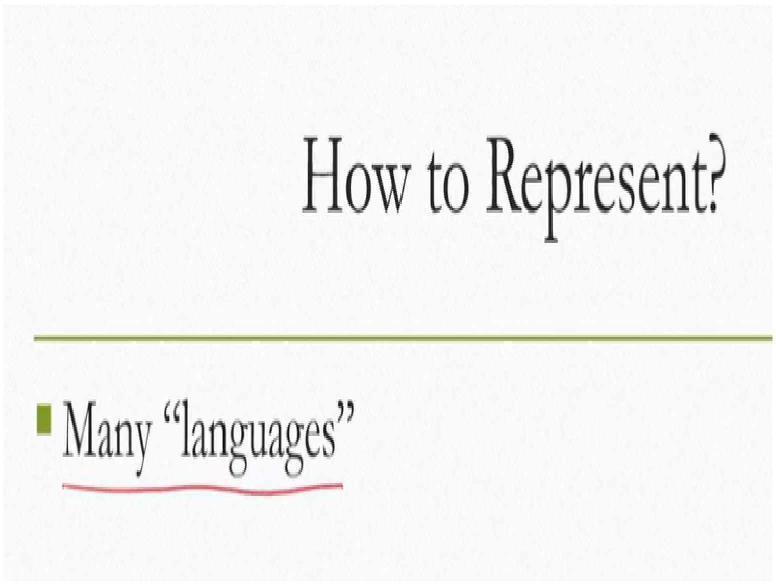
High-Level Design

- Identification of **modules**
- **Control** relationships between modules
- Definition of **interfaces** between modules

Let us begin with the high level design phase let us try to understand what we do at this phase. So at this level as we earlier mentioned that our whole objective is to make the code manageable. For that we need to go for a modular design approach where we can divide the big problem into smaller modules which are easier to manage rather than trying to manage the whole problem at a time.

So at the high level design we start with identification of modules also at this level what we do is? We try to find out control relationships between the modules and at this level we provide definition of interfaces between modules. So modules need to interact with each other how they interact what kind of interfaces should be there that definition also you provide at this phase of system design that is high level design phase.

(Refer Slide Time: 13:49)

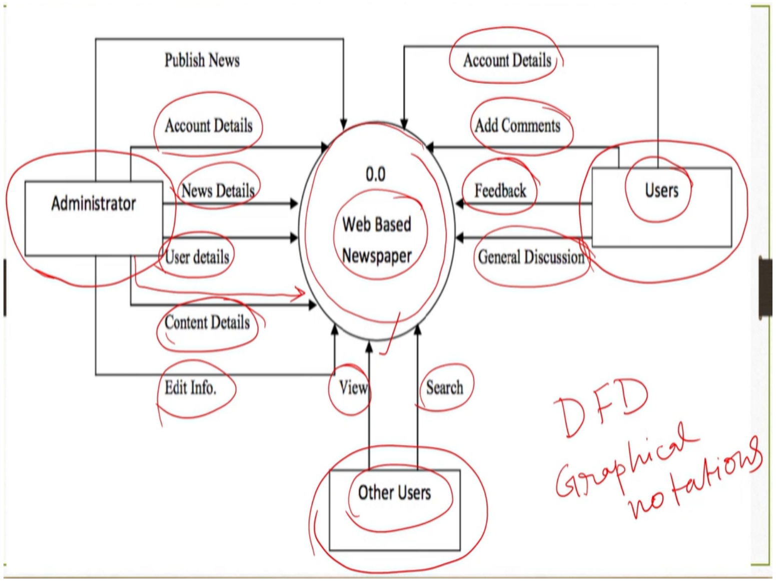


How to Represent?

- Many “languages”

The next question then is once we manage to come with a design how should; we represent it. So that it becomes easier to understand by others who are going to implement the design. So what language to use there are actually many languages which we can take request to implement the system design idea.

(Refer Slide Time: 14:23)



Let us see one example later on we will see another example so this example actually shows one particular language used to present a high level system design. This is called DFT or data flow diagram as you can see here in this example so essentially what is happening is that? We are using graphical notations to express the design idea so it is a graphical language to represent design.

There are certain conventions and standards used like any language that we will learn later but here I want to just give you some idea about the idea about DFT or the idea of the graphical languages and how it can be used to express a design. Suppose we are asked to design and web based newspaper application. So essentially we are looking for designing software which is a newspaper application which is based on an web.

So a user can get the newspaper through a browser so what happens is that somebody as designed this system and this is a top level view of the deign where this entire circle which contains this text web based newspaper is the representation of the system. Whole system is represented with this circle here then there are some rectangles used which are representing who all can interact with the systems who all are the stake holders for this system.

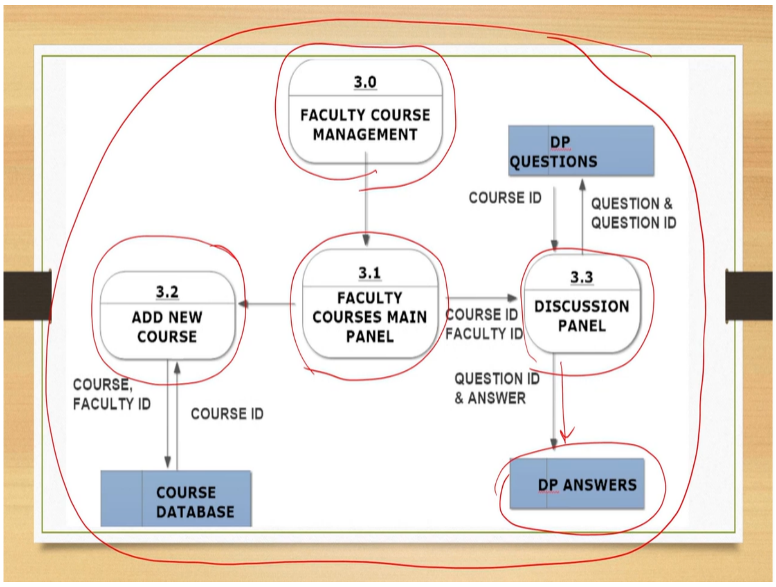
For example there can be administrator there can be a special type of user or there can be some other type of users. Also it shows how these different stakeholders who are users of the system can interact with the system in the form of labeled arrows like the one shown here. So this arrow is there with a level which shows what data flows and in which deduction it flows. For example administrator can interact with the system in different ways by providing news details user details, content details several information account details etc.,

A special type of user marked as user share can interact with the system with general discussion providing feedback addition of comments and account details. That is how the design is (())(18:00) and then this particular notation are used to represent the design. Similarly other user can view or search so the other user can interact only with these 2 ways other interaction mechanisms are not available to the other users.

In that way we can indicate who are the user how they interact and what they interact so that is the graphical language. Now of course when we are representing the whole system with a single circle that does not convey anything at a very high level it says that there is the system and there are these users and those users can interact with the system in so and so manner. But then what is inside the system?

What is the design of the system? Same language DFD can be used to represent the lower levels of the design as well although still at a very high level.

(Refer Slide Time: 19:08)



For example that circle which represented the whole software can further be expanded in the second level will learn about this level idea of levels and all these things in a later lecture exclusively on DFD. But right now just to give an idea so that software contents several modules as represented with these symbols one is faculty login module, one is faculty main menu, one is conduct examination menu.

So this is of course not the design of the earlier system but this is design of some other system which is related class room teaching. There is another module on faculty course management so these particular symbols are used to represent modules or processes and then there are symbols to represent some databases. Like the symbols used here rectangles this represents several databases that are used by the system.

So essentially this is a second level representation top level will represent the whole system in the form of a single circle as shown in the previous slide previous part of the lecture. And if we want to express more details of the design in terms of the modules that are part of this whole software then we can go for the next level of design using DFD where we can show the modules.

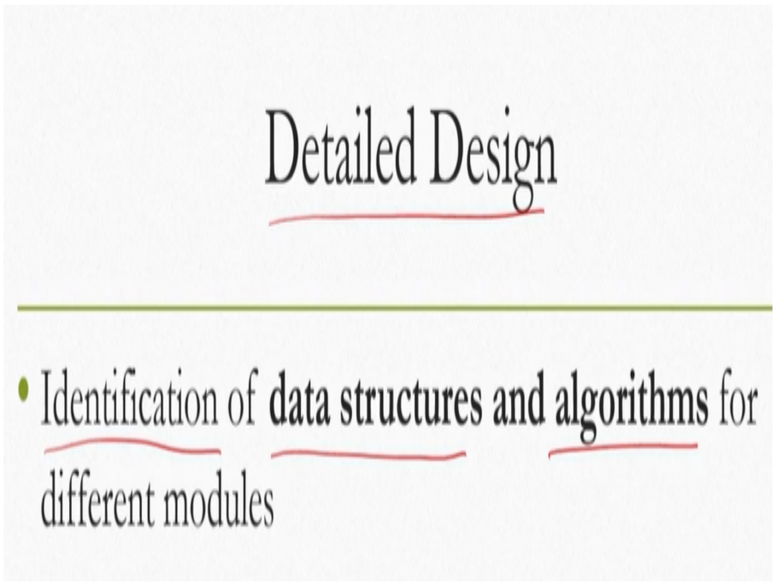
Now these processes will require some data to operate on so that database also requires something to represent so we can represent it using the rectangles as shown here. Then interaction between modules again in the form of leveled arrows to show the direction and content of interaction. So all this things can be represented with DFD we can go to even lower levels, so one of the modules from the second level can be broken down into some modules and shown as separate level of the DFD for that particular like one third level DFD is shown for the modules shown in the earlier design of the system.

Now it may be noted that conventions used are the same like to represent process we use this circle this notation to represent databases we use this notation leveled arrows. But this whole thing here is actually representing one of the modules at the higher level. So one module is divided into some module with interaction between them and the database they use and that is that forms separate level of DFD.

We will learn about all these things in details in the later lecture so that is just to give you some idea of just to express our design in the form of graphical language. Like DFD there are other languages partly graphical languages are also there so later on we will learn about this languages in more details. So that is about high level design note that in the high level design that is where we are talking about modules and their interaction we are not telling how the modules are implemented.

Instead we are simply saying that ok in the system there will be these many modules they will interact with each other in this particular. In that direction and there will be some databases which will provide data to the module and get data from the modules so on and so forth.

(Refer Slide Time: 23:45)



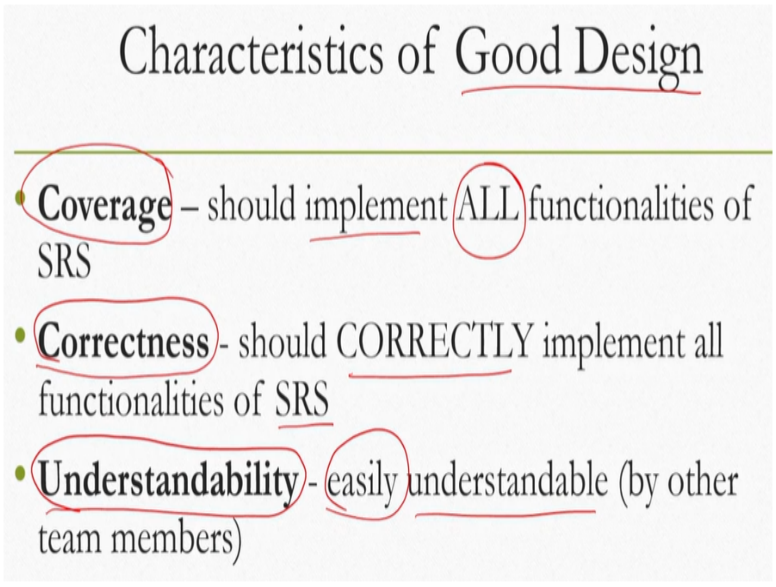
Detailed Design

- Identification of **data structures and algorithms** for different modules

Now in the detailed design phase what we do? We go for detailed design of the modules and sub modules that are part of the overall system. So here we identify data structures and algorithms that are required for implementing different modules so that is the detailed design phase. So that is in a nutshell what is system design so we have 2 phases of the design in the first phase we go for designing the high level concepts in the form of modules and their interaction.

And in the second phase we go for detailed design of the modules in terms of the data structures and algorithms that they require to be implemented. So once we have a design definitely the question comes whether this is a good design or bad design. So how do we categorize whether some design is good or bad. For that we need to know what characterizes are good design?

(Refer Slide Time: 24:48)



Characteristics of Good Design

- **Coverage** – should implement ALL functionalities of SRS
- **Correctness** - should CORRECTLY implement all functionalities of SRS
- **Understandability** - easily understandable (by other team members)

So let us see the characteristics that define a good design first thing is coverage. Now this particular characteristic tells us that a good design should implement all the functionalities that are specified in the SRS. So you have created a software requirement specification document containing functionalities. When we are going for a system design the design will be considered to be good if all the functionalities that are there in the SRS are part of the design.

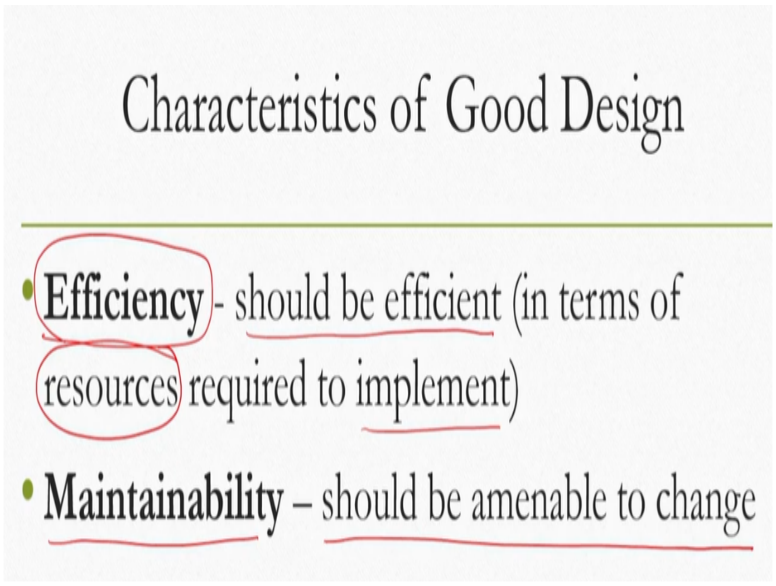
If we miss some functionalities then this is not a good design the second characteristics is correctness that means a good design should be able to correctly implement all functionality of SRS. Now here correctly means that it should be able to produce the desired output when a specific output is given. So correct implementation of all functionalities is another hallmark of good design third important characteristics is understandability what it means?

Any system design that you come up with a represent using some language should be, easily understanding now this is very important. So the whole idea is that you divide the whole problem into manageable smaller sub-problems in the form of modules and some modules. So it is not necessary or generally it is not the practice that all the modules and sub modules are implemented by the same team which designs the system.

It may happen and which is most often the case that the design is distributed to several teams and they are asked to implement different modules and then finally those are combined together. So whenever we are creating a design and expressing it with some language the design should be easily understandable to people who are not part of the design. Otherwise it will be very difficult to implement the design in a team.

So understandability is very essential characteristics of a good system design where understandability means it should be understandable by other team members who are not the part of design team.

(Refer Slide Time: 27: 58)



Characteristics of Good Design

- **Efficiency** - should be efficient (in terms of resources required to implement)
- **Maintainability** – should be amenable to change

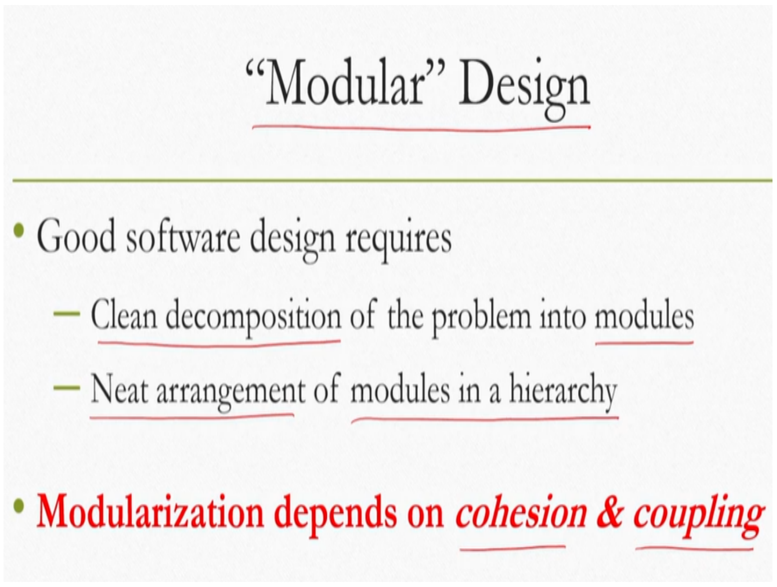
But maybe asked to implement it the fourth characteristic is efficiency this is another important characteristic. The whole design should be efficient now here efficiency means that the whole design should be made in a way such that it is possible to implement the design idea with efficient usage of available resources. So a design should not require things that are not available then that are not a good design.

So whenever we are going for designing something we should be aware of the resources available and accordingly we should design so that those resources are used optimally. Fifth characteristics is maintainability the design should be amenable to change so if there is some change required after brainstorming over the design it should be easy to that without having to recreate the whole design from scratch.

So we should be able to conceptualize and represent the design in a manner such that changes can be performed at a local level without having to redo everything in the design. So those are essential characteristics that define whether we can level a design as a good design or a bad design. If this 5 characteristics are not there as per our evaluation of the design then; there is a need to refine the design so that these 5 characteristics are maintained in the final design.

So that gives us some idea of whether our design that we have come up with can be considered to be a good design or a bad design. Now let us try to understand another important concept in relation to design of a system.

(Refer Slide Time: 30:26)



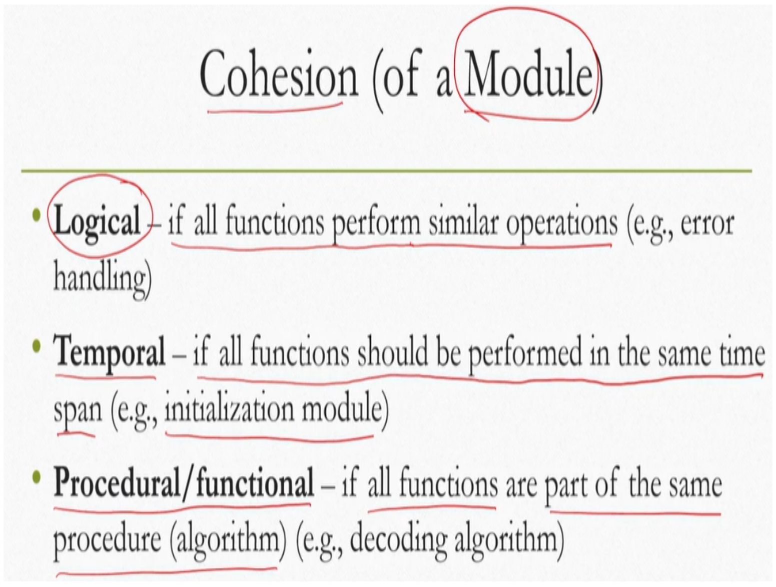
"Modular" Design

- Good software design requires
  - Clean decomposition of the problem into modules
  - Neat arrangement of modules in a hierarchy
- **Modularization depends on *cohesion & coupling***

Cohesion and coupling what it is and what are these concepts and how they are related to good designs? As we have repeatedly said our goal is to come with the modular design so that we can distribute the implementation work for faster turnaround time. So good software design requires clean decomposition of the problems into modules this is very important we should be able to very clearly and easily decompose the problem into sub problems or modules that is not the only thing.

Also we should be able to neatly arrange the modules in the form of a hierarchy that is also required. Now whether we will be able to do that? Whether we will be able to go for the modular design or not depends on 2 properties of the design cohesion and coupling. So; essentially modular design depends on cohesion and coupling.

(Refer Slide Time: 32:00)



Cohesion (of a Module)

- **Logical** – if all functions perform similar operations (e.g., error handling)
- **Temporal** – if all functions should be performed in the same time span (e.g., initialization module)
- **Procedural/functional** – if all functions are part of the same procedure (algorithm) (e.g., decoding algorithm)

So let us start with the idea of cohesion now cohesion typically is the property of one module. Whenever we are trying to design a module it has a property of cohesion what this property indicates? Several things determine the cohesiveness of a module and there are several types of cohesion. One is logical cohesion that is if all the functions in the module perform similar operations.

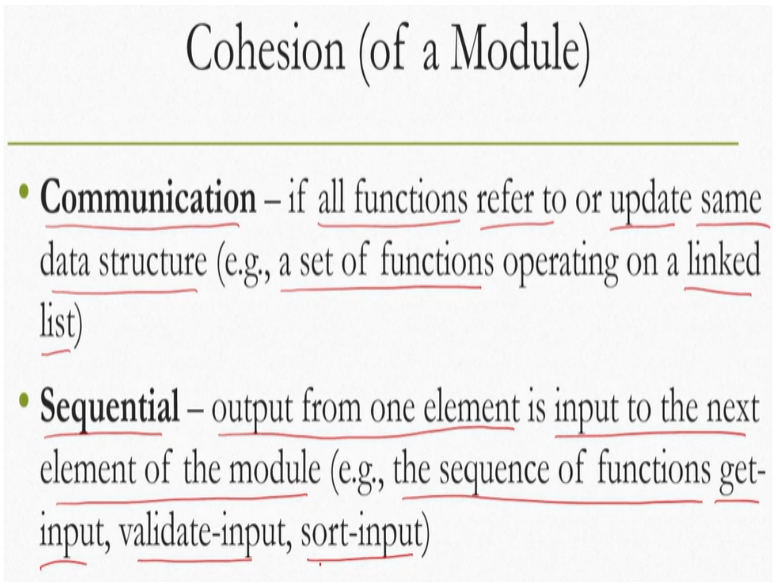
For example suppose there are several functions which are part of the error handling module so essentially their objective is to deal with different types of errors. So all the functions are performing error handling then we say that this error handling module is logically cohesive. Then there is temporal cohesion that is if all the functions of a module should be performed in the same time span.

For example whenever a system starts we have defined an initialization module so there are several functions as part of the module and those functions initializes several maybe data structures or storage elements databases. So all the functions are doing the initialization at the same time span which; is at the beginning of starting of the operations of a system. If that is the case then we say that the module in this case the initialization module is temporally cohesive.

Then there is procedural functional cohesion which indicates or which happens when all functions of a module are part of the same procedure or algorithm. Suppose there is a deciding algorithm used in some image processing task image processing software. Now decoding algorithm need not be implemented in the form of a single function instead we can implement it as a group of functions all are part of the same module of decoding and they are all representing the different part of the same decoding algorithm.

Then if such a situation exists then we say that particular module is functionally or procedurally cohesive.

(Refer Slide Time: 34:45)



Cohesion (of a Module)

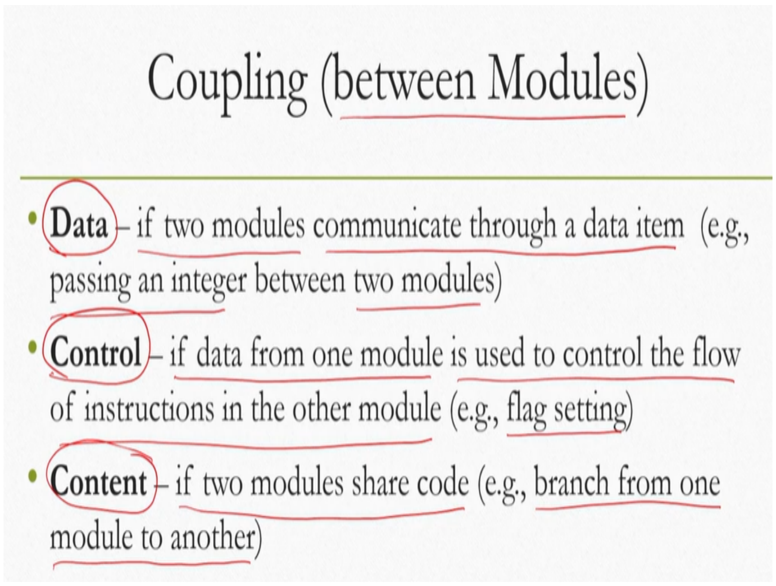
- **Communication** – if all functions refer to or update same data structure (e.g., a set of functions operating on a linked list)
- **Sequential** – output from one element is input to the next element of the module (e.g., the sequence of functions get-input, validate-input, sort-input)

Then we have communication cohesion it happens when all functions refer to or update the same data structure. As an example if a set of functions are there that are operating on a linked list data structure. So that means they are referring to the same data structure as well as updating the same database structure then those functions are said to having communication cohesion property.

And finally we have sequential cohesion ouput from one element is input to the next element of the module. For example the sequence of functions get input validate input and short input if these 3 together form a module then the output of get input is fed as input to the validate input function. Validated input output is fed to the short input function so then they are sequentially cohesive module then that particular module is called sequentially cohesive because the functions that are part of the module follow the sequential cohesion property.

So essentially this cohesion property what they say is that a module is a collection of functions and how the functions behave in terms of input or communication between them or execution, behavior. So those; behaviour of the functions in a module represent the cohesion property of the module and there are 5 types of cohesion depending on how the functions behave that is about the cohesion

(Refer Slide Time: 36:48)



Coupling (between Modules)

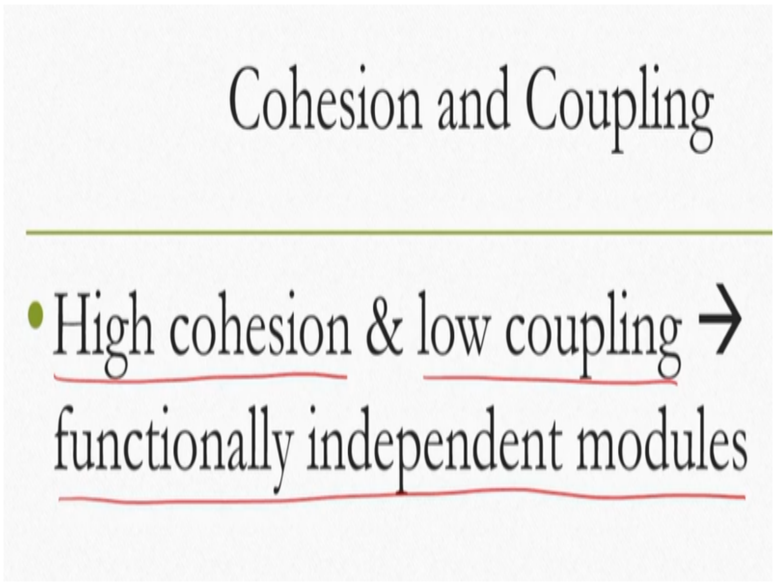
- **Data** – if two modules communicate through a data item (e.g., passing an integer between two modules)
- **Control** – if data from one module is used to control the flow of instructions in the other module (e.g., flag setting)
- **Content** – if two modules share code (e.g., branch from one module to another)

Next is the concept of coupling now cohesion is property of the single module so the functions that are part of the module coupling in contrast is the property between modules. So when we say coupling we say that it is the property between 2 or more modules. Now there are likewise there are different types of coupling possible one is data coupling what it says is that? If 2 modules communicate through data item then we say that they are having data coupling property.

For example passing an integer between 2 modules so if we are designing modules in a way such that some integers need to passed between them then they are having data coupling property. Then we have control coupling if data from 1 module is used to control the flow of instruction in the other module then we say that there is this property of control coupling. For example if there is a flag data which we said in one module and that is used to control the flow of operations in some other module then there is a control coupling between these modules.

Then we have content coupling if 2 modules share code then they are said to have content coupling property. For example branch from one module leads to execution of coding in another module. If that is there then we say that these 2 modules share the content coupling property.

(Refer Slide Time: 38:49)



Cohesion and Coupling

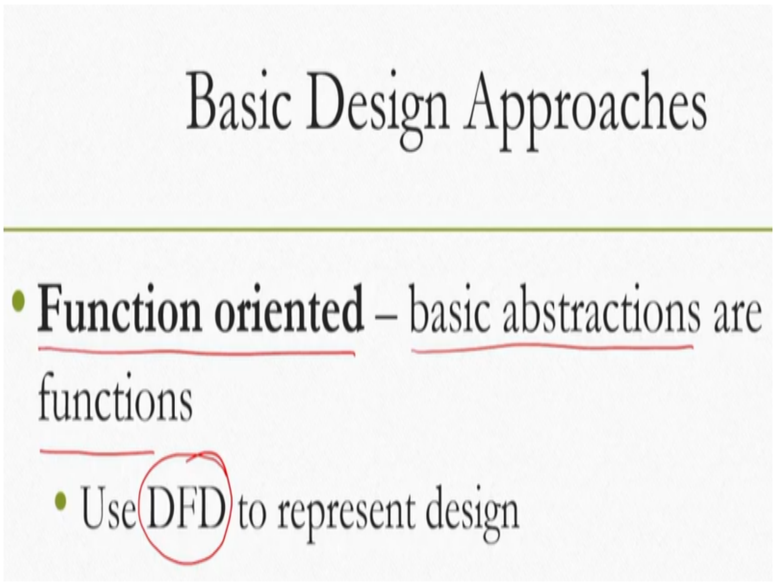
- High cohesion & low coupling → functionally independent modules

So to recollect we have these 2 important concepts which are nothing but properties of modules one is cohesion which is a property of single module and one is coupling which is property between modules. Now one important thing that we should note is that high cohesion and low coupling leads to functionally independent modules. So our objective is to have clean and neat the composition of the problem into modules.

And we should be able to represent them in a hierarchical manner now that is possible when we have functionally independent modules and that is possible in turn when we have high cohesion within modules and low coupling between modules. So essentially whether we are able to achieve our goal of a very good modular design depends on whether we have been able to design modules such that the modules have high cohesion and low coupling.

If that is satisfied then we will be able to go for a very nice modular design otherwise it will not be high quality modular design. So our goal should be to have as little coupling as possible and as high cohesion as per possible.

(Refer Slide Time: 40:35)



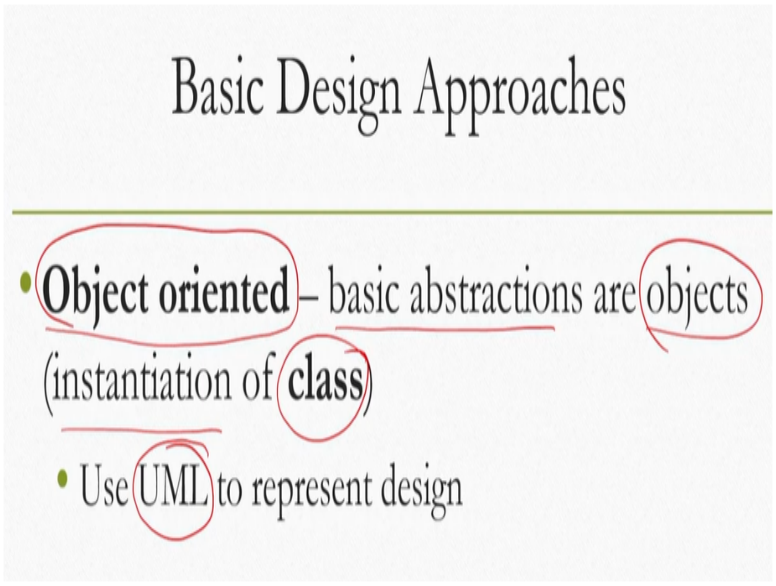
Basic Design Approaches

- **Function oriented** – basic abstractions are functions
  - Use DFD to represent design

So with that we got some idea of what is the aim of system design what we need to know and what we should be aware of? Now next task is to understand how we can go for designing system. So what are the approaches available broadly there are 2 approaches. One of these approaches is function oriented approaches. In this approach what we do when we go for design of a system is that we use functions as basic abstraction.

So whole system is designed is based on functions if we are going for such a design then we call it function oriented design approach. Now when we are using oriented design approach to represent that design we rely on generally we rely on DFD or data flow diagram the example that we have seen in the earlier part of this lecture. As I said on DFD we have detailed discussion later.

(Refer Slide Time: 41:51)



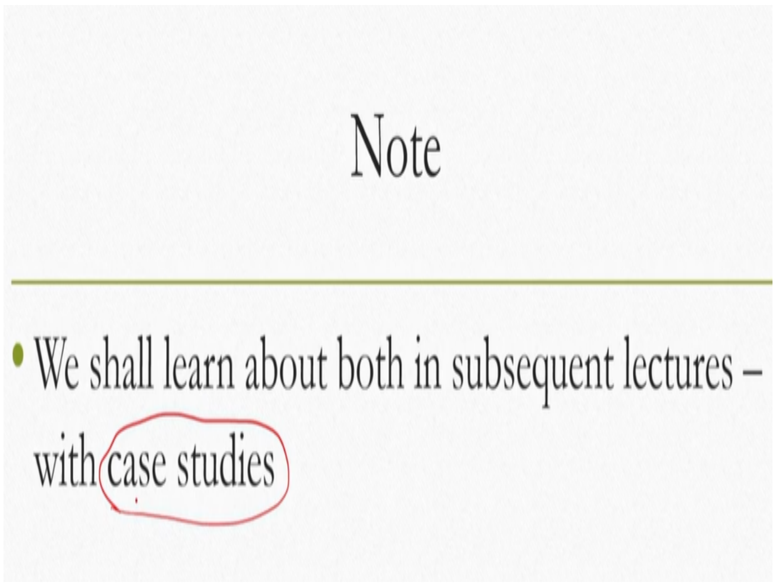
Basic Design Approaches

- **Object oriented** – basic abstractions are objects (instantiation of **class**)
  - Use UML to represent design

The other broad approach for system design is called object oriented design. Here instead of function the basic abstractions that we use are objects which are instantiation of a concept called class. So in object oriented design approach we rely on objects as the basic abstraction or basic unit and these objects are instantiations of concept called class. So that is the basic idea on which the object oriented design approach works.

Here also we rely on some language to express the design typically we go for UML or unified modeling language to represent the object oriented design or a system.

(Refer Slide Time: 43:05)



Note

- We shall learn about both in subsequent lectures – with case studies

So in subsequent lectures we learn about both these approaches namely procedural approach and DFD as well as object oriented design approach and UML to express those designs. And we will learn those with case studies so that these become easier to understand and remember. So with that we have come to the end of the lecture so in this lecture we will quickly review what we covered?

So we started our discussion on how to for system design now we learnt what is the main idea behind system design and what are the key concerns. So there are 2 concerns one is where to start and how to represent where to start? Is basically; the SRS document at the end of requirement specification stage that is our starting point. So our objective is to implement all the functions and to represent we need to make use of different languages one example we have seen graphical language called DFD and there are other languages as well.

Now design happen in 2 phases one is the high level design phase one is the detailed design phase while going for design our mina objective is to go for modular design so that the code is manageable. Now to ensure that we have to ensure that whatever design we have come up with as high cohesion and low coupling. Cohesion and coupling concepts we have discussed including different types of cohesion and different type of coupling.

And just to emphasize cohesive is the module property whereas coupling is a property between modules and our objective is to have a design that supports high cohesion within a module and low coupling between modules then only it will be possible to go for a very good modular design otherwise our objective of modular design may be compromised. Then we talked about 2 broad approaches to design one is function oriented approach other one is objected oriented approach.

For function oriented approach we need to use DFD to express the design for object oriented approach we go for UML to express the design. In the subsequent lectures we are going to talk about these design approaches as well as the languages to express them in more details with case studies for better understanding So that is all for this lecture hope you have enjoyed and learnt the concept looking forward to meet you all in the next lecture thank you and good bye.