Software Engineering

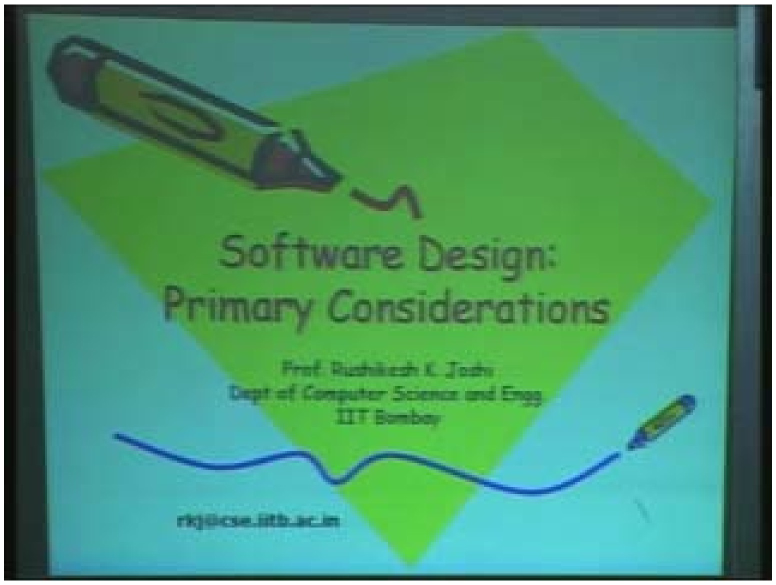
Prof. R.K. Joshi
Computer Science & Engineering
Indian Institute of Technology, Bombay

Lecture-14

Software Design - Primary Consideration

The topic of this lecture is software design, primary consideration. We are going to discuss what the preliminary considerations are when we design the software and what the ingredients for the design process are. The first question that we have to address is: What is design? We have used this term in real life often.

(Refer Slide Time: 01:03)



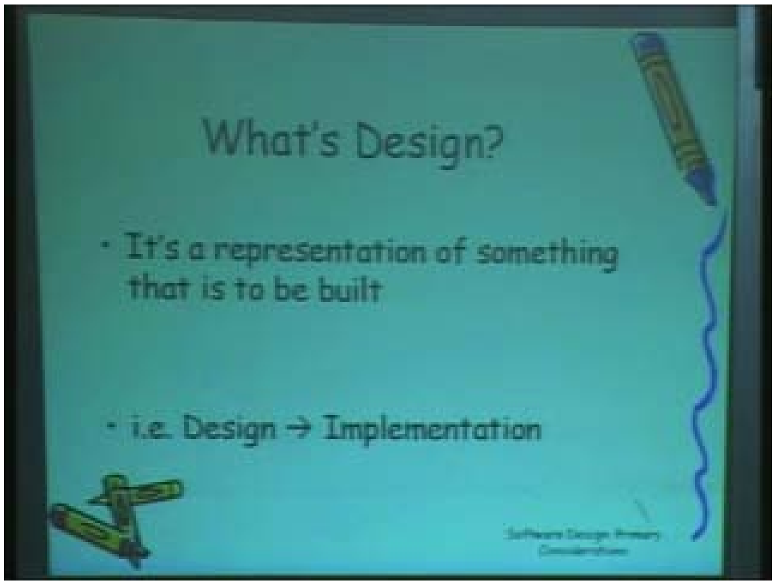
Software Design:
Primary Considerations

Prof. Rushikesh K. Joshi
Dept of Computer Science and Engg.
IIT Bombay

rkj@cse.iitb.ac.in

The first question that we have to address is: What is design? We have used this term in real life often. We first want to model the actual and it is a conceptual model that we call a design. This design has to be then converted into implementation. So it is very important that we design such that the design can be easily converted into Implementation. Let us look at some of the examples of design in real life. We often see blue prints for a building architecture. If you have gone to an architect you would see blue prints in front of him.

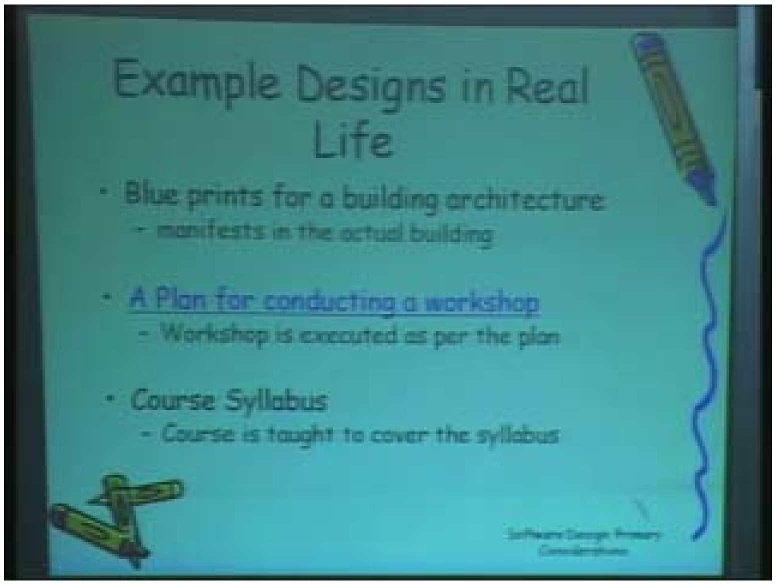
(Refer Slide Time: 01:55)



What's Design?

- It's a representation of something that is to be built
- i.e. Design → Implementation

(Refer Slide Time: 05:15)



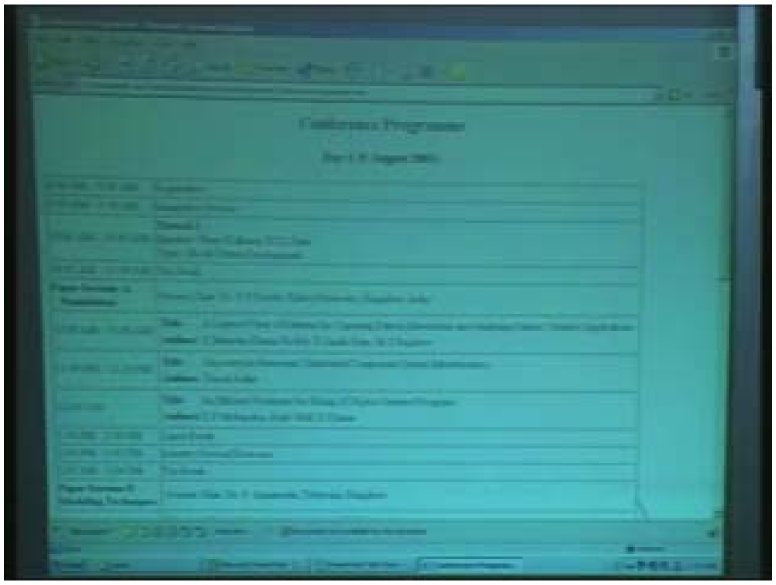
Example Designs in Real Life

- Blue prints for a building architecture
  - manifests in the actual building
- A Plan for conducting a workshop
  - Workshop is executed as per the plan
- Course Syllabus
  - Course is taught to cover the syllabus

In your early engineering drawing classes or lectures you must have created blue prints for few designs, your plans for a building for example. Or in electrical engineering you design your machines. Design is not the actual implementation or design is not actual machine. It is the model of the machine that we want to build where its considerations go in to this design process. The design then gets transformed into the actual implementation. The blue prints for a building architecture is an example and this manifests in the actual building later. The actual project has to be executed as per the design.

Another example of a design is; a plan for conducting a workshop. You are conducting an event and the actual event is your actual project or you can say it is an implementation. But first you want to have the design of that event and then as per the design you want to execute it. We have an example here for the plan for a workshop. This is a conference program. This is a program for a national conference on object oriented technologies which was conducted last year.

(Refer Slide Time: 04:00)



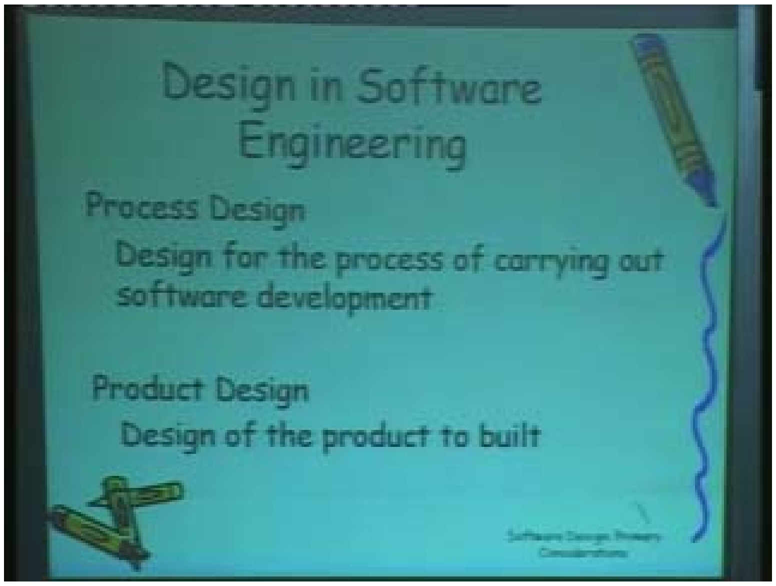
Conference Programme

[illegible]

This has time slots and various sessions have been organized into these time slots. There is a time slot for ‘Tutorial on model between the developments’ and there is a time slot for papers and there are time slots for tea breaks. There is also an evening 2 and day 2 plan and there are slots for other papers and so on. This is an example design which is a plan for conducting a workshop. It is also a design and it is not the actual workshop that we see. First we have to plan for it and then we have to execute the workshop according to the plan.

This design is slightly different from the design of a building architecture. This example is about events, building architecture is about structure. Designs have to be built for various kinds of activities with the dynamics, static and throughput. Another example design is the ‘course syllabus’. We have course syllabus for all the courses. That is the design of the course and then the course has to be taught accordingly. The course is taught to cover the syllabus. The design then gets converted into implementation, which is the actual teaching of the course. These are some example designs in real life. Designs are not actual implementation. Designs have to be converted into implementation. Now we will focus our attention on ‘Design in software engineering’. We can talk of the software development process as such and also the product that we want to build. The process of building a product, sequence of steps and the iteration in building a product, documentation, testing and verification, review process etc are the process followed for software development in general.

(Refer Slide Time: 07:57)



Design in Software Engineering

Process Design

Design for the process of carrying out software development

Product Design

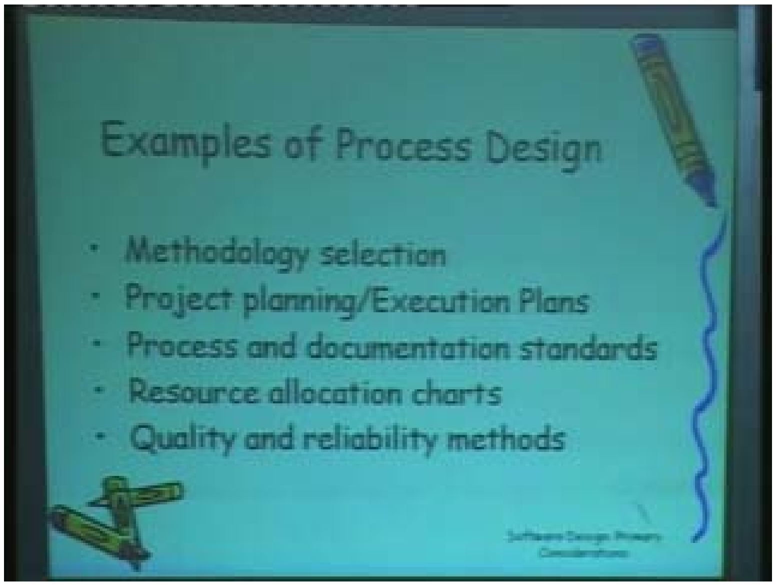
Design of the product to built

In the process of software development, one of the processes you have is the design process. When do you actually design software and where do you start? How many people work on the design? In order to answer these questions, we have to design this process itself. Any process for that matter in the entire life cycle, first you have to design. Process design is an important discipline. Process design talks about design for the process of carrying out software development.

Designing a software process is just like designing a product itself. The process is the one that brings out the product, but you have to also design the process. When you actually carry out the process then that is the implementation of the process. Similarly you plan for the product and you actually build the product which is the implementation of the product. So you can talk of process design and you can also distinguish product design from process design.

The process design is the design of the product to be built. In this lecture, we are going to focus on product design. But we will also look at some examples of process design, so that you will have an idea about the ingredients of a process design. Methodology Selection: You have to select the methodology that you want to apply in your processes and you should select the tools that you are going to use. When you select the methodology your tools also might follow your selection. Project Planning by Execution Plans: The entire project planning and how it is going to be executed.

(Refer Slide Time: 09:30)



Examples of Process Design

- Methodology selection
- Project planning/Execution Plans
- Process and documentation standards
- Resource allocation charts
- Quality and reliability methods

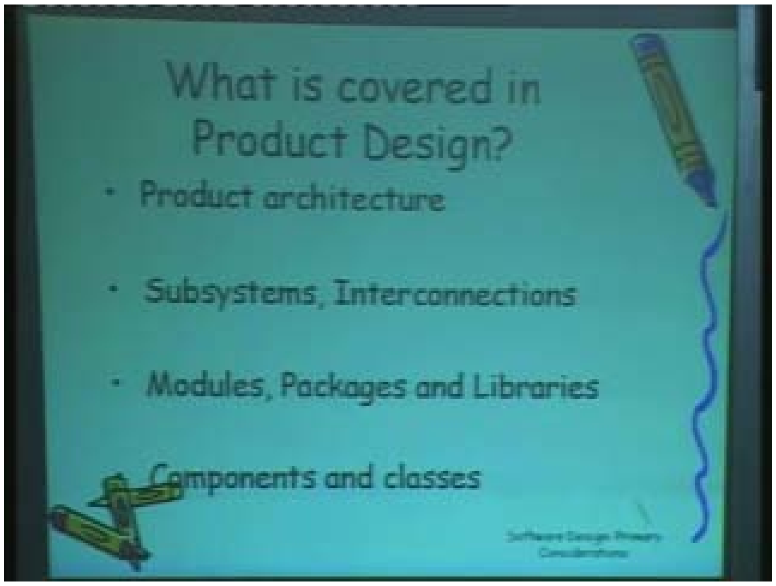
Software Design Primary Considerations

Process and documentation standards: The process by documentation standards that you are going to follow in your entire software development lifecycle. Resource allocation chart: Different things that will be considered in resource allocation chart. For example, allocation of specific software engineer, when they will be deployed? Where are they going to work? In which site they are going to work? Which are the machines that will be kept aside for the development? Which cabins that are going to be allotted for the team? Where are they going to discuss? Etc

Quality and Reliability methods: How to ensure the quality of the software development process and reliability of the product? And what are the different methods that would be employed for quality and reliability. These are all different ingredients of the process design. So when you design your process you should take all these considerations into account. Finally when you follow this design and follow this method in your software development you would get the good quality product which implements the desired requirements specified in software requirement specification. Now we will switch on to product design. We have seen process is different from the product. The process is the activity that you carry out to build a product.

When you want describe the design of the product, what are the considerations? The first one could be the product architecture. Architecture is an overall design. It is a high level view of the design. Product architecture might be in terms of sub systems and its components or various layers in your software. How to describe the architecture would be a separate topic of discussion which is not covered here. On this slide we are going to look at different aspects of design and the first one is architecture. Then we have individual sub systems. You can design individual sub systems, the ingredients of individual sub system and how they are interconnected.

(Refer Slide Time: 12:25)



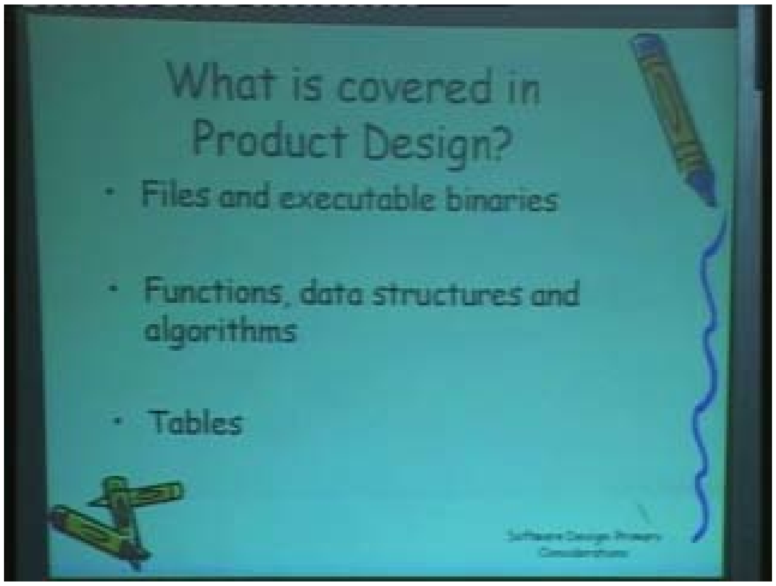
What is covered in Product Design?

- Product architecture
- Subsystems, Interconnections
- Modules, Packages and Libraries
- Components and classes

Software Design: Primary Considerations

Another design consideration would be different modules in the software, packages and the libraries that we are going to develop as the part of software. Your design might consist of module design, package design, and design of the libraries below the sub system. At top level you have high level product architecture. Then you can also talk of the design at the level of components and at the level of classes. Basically, about the specific individual classes that can be used, how they are connected and how do they collaborate, what do they do etc. You can also talk of file level design and executable binaries. When you want to actually deploy your software, you talk of executable binary.

(Refer Slide Time: 14:15)



What is covered in Product Design?

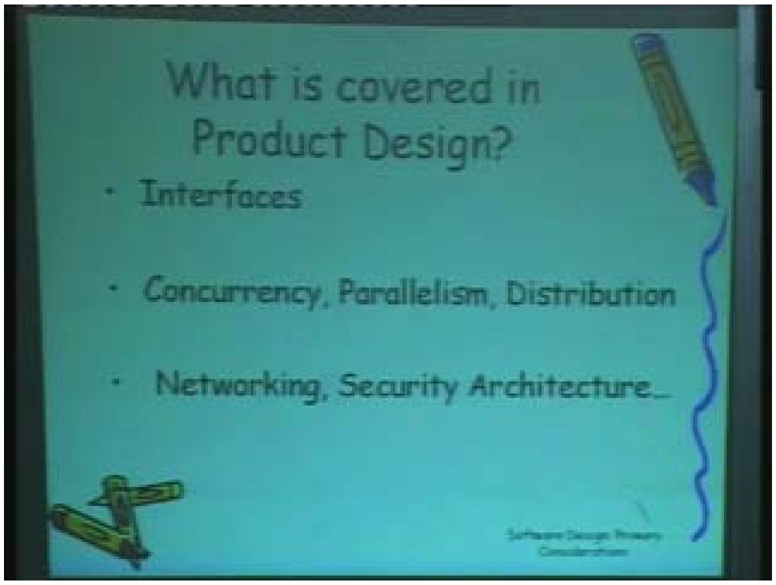
- Files and executable binaries
- Functions, data structures and algorithms
- Tables

Software Design: Primary Considerations

In a typical UNIX system you would have by bin, by user, by include and so on. You distribute or deploy these software binaries and sources under different directories and so on. The deployment architecture or the specific design of your different files and how they are interconnected, how the executables are organized etc are very important. Then you can talk of low level entities such as functions, data structures and also algorithms. You have to design your functions, data structures and algorithms. This is different from actually implementing them. Design is not a code in fact it should not be coded. Design is a high level specification of the implementation that you are going to build.

If you have a very data centric system where you have persistent data in a typical business application, for example, you are developing your library information system or you are developing your academic system, then you have lot of data and there are again relations among them. For that, you would need to have tables, you can convert your models into your relational table, you would design them, and the data design is also an important aspect of your product design. Then at the high level of the product which is the user level, how does the user interact with the product? What is the design of that interface layer? That is an important aspect to see how the interfaces are organized and how the user is going to interact with the software. Interface design is an important aspect of your entire software design.

(Refer Slide Time: 16:37)


What is covered in Product Design?

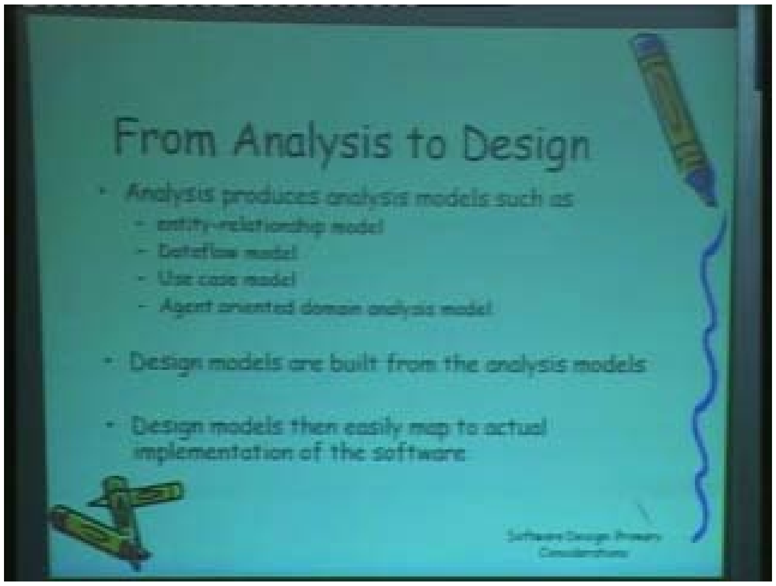
- Interfaces
- Concurrency, Parallelism, Distribution
- Networking, Security Architecture...

The other important considerations are concurrency, parallelism, etc. Which are the concurrent activities in your software? Are they also parallel and is there distribution? How are you going to distribute? What kind of architecture you have for the system? Is it a bus architecture or do you have ring architecture or do you have mesh architecture? If it is a parallel computing system, you may have further architecture such as hypercube, specialized grids and so on. Basically the aspects of design from the point of view of concurrency, parallelism distribution, we can add to it the application for tolerance and so on then another consideration is networking.

How are you going to network your systems? What are the protocols that you are going to follow and also the physical and logical layer? What is the security architecture? Apart from the very core functionality of the software, these are all very important aspects. You have to design for all types of requirements of the software, both the functional and the non functional requirements. Functional are those which are related to the actual semantics of the data and operations on data. And the non functional requirements would be requirements such as distribution, parallelism, networking architecture, the security, how the user will be allowed to login and what would be the encryption mechanism and so on.

All aspects of software development should be considered in design process and everything has to be designed so that at the implementation time you do not get any surprises. It is very important to design everything before you actually implement. Sometimes it may not happen in practice. When you are implementing, you have to change your design or you have to perhaps redo some part of your design. So your designs should be such that one should be able to refine them and modify them. Now we will look at those aspects of the design. How should the designs be? What are the properties which must be satisfied by your design?

(Refer Slide Time: 20:48)



From Analysis to Design

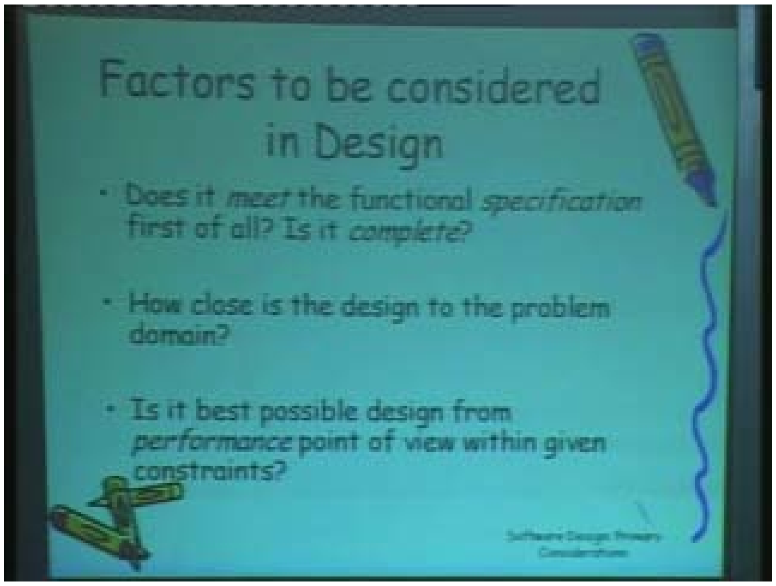
- Analysis produces analysis models such as
  - entity-relationship model
  - Dataflow model
  - Use case model
  - Agent oriented domain analysis model
- Design models are built from the analysis models
- Design models then easily map to actual implementation of the software

Software Design Primary Considerations

Before that we have to note that the designs should be done after the analysis. You have to come to design from the analysis and from the design you should go on to implementation. During analysis you have these products such as the entity relationship model or a data flow model or a use case model for the high level functionality point of view. The entity relationship model identifies the entities in the domain and the connectivity and relationships between them. Data flow model identifies the data and the flow of the data through different processes. Use case model identifies how the user is going to interact with the system. There also other different methods of analysis I have included one here, which is agent oriented domain analysis model.

What are the agents in the system and how they carry out different activities? What activities do they carry out? Agents are mainly the animate agents; basically the human being. If you want to automate an activity say business model, then you have to first find out what are the different entities and agents, how is it carried out currently and then you have to analyze the system. Then you can go on to design your software to automate that system which is already in existence. Based on the existing system, analysis models can be produced and then using these analysis models you can build the design which you can translate into software implementation. Design models are built from analysis models, so analysis model is very important. The design models can then be easily mapped to actual implementation of the software. Now we will come to the factors that we have to consider during design.

(Refer Slide Time: 21:05)



Factors to be considered in Design

- Does it *meet* the functional *specification* first of all? Is it *complete*?
- How close is the design to the problem domain?
- Is it best possible design from *performance* point of view within given constraints?

Software Design Primary Considerations

First and the most important factor is this: Does your design meet the functional specification first of all? Is it complete? Suppose in your library information system you want that the fines must be automatically calculated and design does not account for it, then the design is not complete. It is the most important fact that you have design for everything. How do you make sure of it and who is going to test the specification which may be in say plain English or in the IEEE software requirement standard? Somebody has to review and find out whether the design is meeting the requirement. There are review processes to make sure that design meets your specification and whether it is complete.

How close is the design to the problem domain? This is an important aspect of a design. If the design is very close to the problem domain itself, say for example you have objects such as books, racks, users, different floors, journals, CD ROMs, videos CDs in your library system. In the design of the software, you could also have objects which have the same name. Then it will be very easy for even for the implementer to view and understand what the system is going to do. It is very close to the actual real life system.

This is an important aspect and this makes your design more understandable because it is very close to the actual system and this is also sometimes called seamlessness. Seamlessness is a property of having no bump between two different stages. If there is a seamless transition from one activity to another you do not experience the bump or feel the difference in the activity. When the design is very close to the actually real system, if you have a real life system and if you have the software domain even the domain experts or actual users who are currently operating the real system will be able to understand the design or will be able to review it or will be also able to comment on it. The abstractions that you have used in the design have been taken directly from the real system. If it is close to the system the real system or the problem domain, then its easy to understand the design, easy to comprehend it and also it is also easy to come up with the design.

Many times you can have very similar entities and the connection as you see in the real life. But this is not always true. For example you have abstract classes in your design, where do abstract classes come from. You are seeing some commonality amongst many different types of objects and that commonality are capturing your design, but it is not there in the real life. But the actual object, the concrete objects are there. The concrete classes will be the corresponding entities and real objects can be mapped to the specific concrete classes in the design which are very close to the actual object.

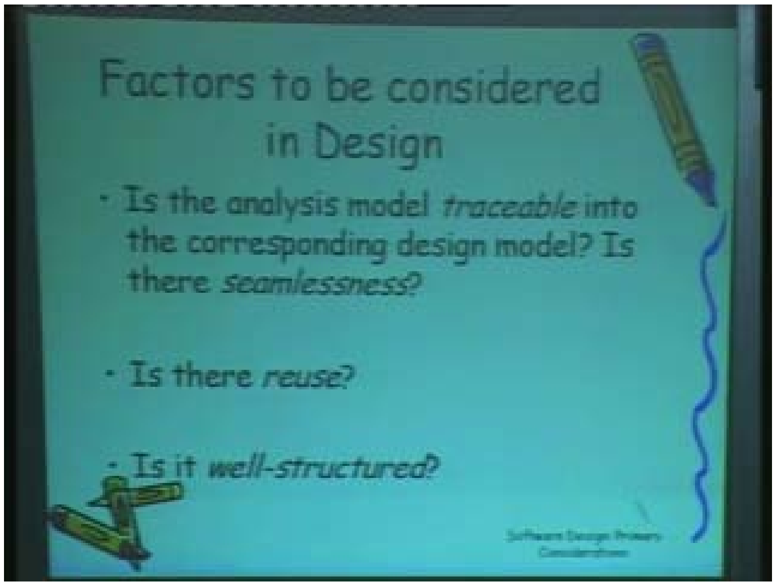
The abstract class which might be a super class of your many subclasses, that abstract class you do not see in the real life which is mainly in your mind, which is at the level of your logical organization of the object. Some of it might get manifested in the library system through your classification system. The designer adds these additional abstractions in the design which are not there in the real system.

Whatever is there in the real system, if you find corresponding entities in the design, it is easy to comprehend such a design, to understand design and also to build the design and then to convert it to an implementation. Another factor that you need to consider in the design is performance. Is your design, the best possible design from the performance point of view? You may not always get the best design because your resources are limited you have a deadline in front of you. So you have to make a best attempt to get a high performance design which does not have performance bottleneck. For example there is a server that accepts request from across the internet. Everything has to come at the server and the server itself executes the responses and sends them out. This one server which accepts all the requests and computes the responses will be a bottleneck.

Instead you could have gone for parallelism and also distribution. So that in parallel you could have carried out many activities and responded to many requests simultaneously. You can make sure that your low level data is in consistent state and you must get a high performance design even with your resources and time constraints. In fact, expected performance of the software would be considered at the stage of requirements specification and you have to design for it. Performance is an important criterion. Again to summarize, the completeness of the design, closeness to the problem domain and performance aspects should be taken into account when you design.

Another important factor to be considered in design is, is the analysis model traceable into corresponding design model? For example, you have done data flow analysis and you have done entity relationship. Can you trace them into a design?

(Refer Slide Time: 36:50)



Factors to be considered in Design

- Is the analysis model *traceable* into the corresponding design model? Is there *seamlessness*?
- Is there *reuse*?
- Is it *well-structured*?

Can you show how an entity in the entity relationship model has been mapped on to another design entity? You should be able to show whether it has gone as an attribute in a class or has it become a class itself or some collections of entities and relations being mapped to tables or they being designed as separate table. In your data flow diagrams or the data flow model, are the processes being mapped to functions or they processes servers which are active servers? A server typically goes in a loop. It is open for request as soon as it gets a request it operates on the request and sends response back. So you have to trace this analysis into your design.

Why is this important? First of all the design is banking on the analysis model. The analysis has to be converted into design and then whoever wants to understand analysis and design, has to go through this if it is traceable directly. Again this traceability has to be seamless. If you use the same methodology for design and analysis you might benefit from seamlessness. For example if you apply object oriented analysis and design process, then use the same notations and same meta models for both your analysis and your design. Then there is seamlessness. When you further refine the analysis that you have already came up with, it will get transformed into design. So it has to be first traceable. If possible, it could become seamless which is an additional advantage.

Now, when you design are you designing everything from scratch or are you going to reuse something which you already have? This is another important factor that you consider when you design your software. When a software firm takes up a project, it may not be the first time that it has taken up a project unless it is a startup firm. You carry out design often for different kind of projects and different products.

So when you get something new, you would look at your earlier results or earlier designs rather than starting from scratch to design. Are you going to benefit from your earlier design? If you have encountered similar situation earlier, you have to bank on some of the results of your earlier design process. But how do you reuse is another question.

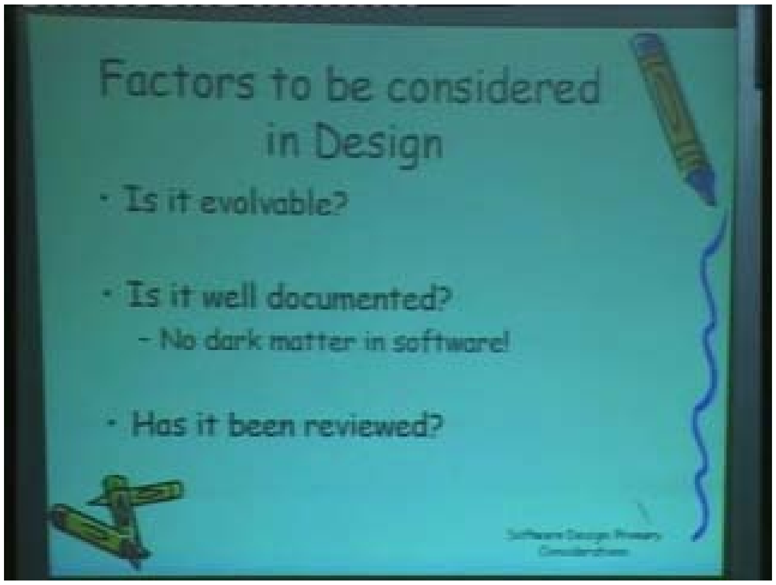
If you have designed for a specific protocol, you can simply take that design if the protocol is recurring. You have to benefit from your old design. There are mechanisms for reuse. There is a separate discussion on design patterns. Design patterns are the object oriented design which can be reused again and again for different domains in different applications. For example, you have a common design requirement that customers subscribe or consumers subscribe and the publishers publish the results or publish the events. Whenever something is published, the subscribers must get the results of the publication or the events are to be delivered.

If there are services built for example abstraction service, it looks at different newspapers and then it gives you an abstract of what has happened today to the subscriber. This is a common requirement that there are publishers and subscribers. If you have encountered such requirements in a design earlier in another project and if this recurs in a different new project, then you have to now use the results of your earlier design and directly apply it in the new project. The names of the entities might be different. There might be specialized behavior, but you should be able to make use of most of what you have already encountered. So object orientation does allow you to reuse lot of your design ideas and there is a nice book by the gang of four authors (Design patterns: elements of reusable object-oriented software). Erich Gamma is the one who actually initiated this book.

This book talks about reusable design. Incidentally reusable design does not directly imply that corresponding code is also reusable as it is. But at the level of design, at least you should be able to reuse and if there is code reuse that is also possible. Most often your entities might be different and you have specialized behavior. The amount of code reuse that you can have in your system sometimes is limited because of various considerations. But the point is that one has to learn to reuse your results of your old design processes if there are similar requirement. So you are reusing design and you are also reusing the design processes, because once you do design again and again, you also develop the process of designing. You also know how to go about decomposition. So the process also gets reused along with the product. So reuse is an important factor in your design.

Now the next factor is; is the design well structured? How good the design is? Is there a good decomposite or is it very haphazard? Is it very hard to understand the design? Are there are two many interactions? One does not know where to start such designs is not good. You need to have a well structured design. We are going to see later in this lecture about what kind of structure they are. Another factor that goes into designing is the evolvability of your design. Is it evolvable? This is very important because the requirements also keep on changing quite often

(Refer Slide Time: 40:32)



Factors to be considered in Design

- Is it evolvable?
- Is it well documented?
  - No dark matter in software!
- Has it been reviewed?

Software Design: Primary Considerations

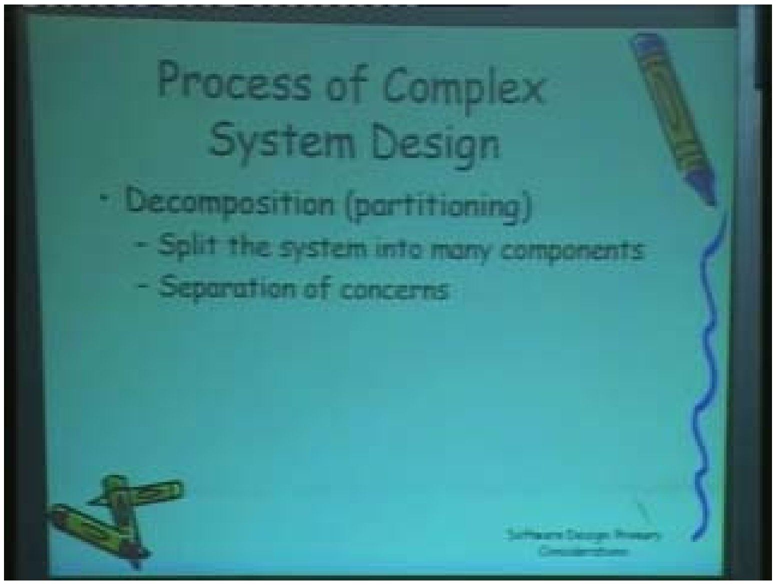
Sometimes you have not considered all of them in one shot. You will know it down the line and you need to evolve your design. So when you are designing, during that time itself you must design such that the design is evolvable and one should be able to make a change into it very easily. There are techniques to build evolvable a design. For example, if you have already structured your different entities into a nice inheritance hierarchy, you will be able to add a new subclass into inheritance hierarchy and evolve the design. If you already have a modular design, you will be able to add a new module into design. You have to take into account future additions and future changes.

There are different properties of specific design that goes into these main or major high level properties such as evolvability, for example coupling, cohesion and so on. These are actually software metrics and which is not focus of today's lecture. We are going look at the basic core properties or the basic considerations in the design. In order to achieve an evolvable design, it has to be well documented. You must produce all the important documents of the design. There are diagrammatic notations, you may also have text based description, and you may also specify your designs. There must not be any dark matter left out. If you document well, the rest can follow up on the documentation.

Document should not be written for the sake of documentation. It is for the sake of all the goodness properties that you want to extract from a design such as, evolvability maintainability and understandability. You have to produce proper documentation. How to document top down approach must be taken. You must have high level document so that first quickly you can grasp what the system is. After you grasp the whole system, you can then zoom on to specific sub systems and then understand the system. Another factor is; has the design been reviewed? This in fact is about the process of design. You have to reveal your design so that you must make sure that it is complete and you are not missed out anything. So this is an important aspect of your design process. Now, we will look at the important process that you apply in designing complex system.

The important criteria or the important activity that you carry out when you want to defeat the complexity is by decomposition. You cannot get the entire design in one shot. You cannot start writing down the lowest level details of your design. Decomposition is very important and that is the way to beat any complexity.

(Refer Slide Time: 42:35)



Process of Complex System Design

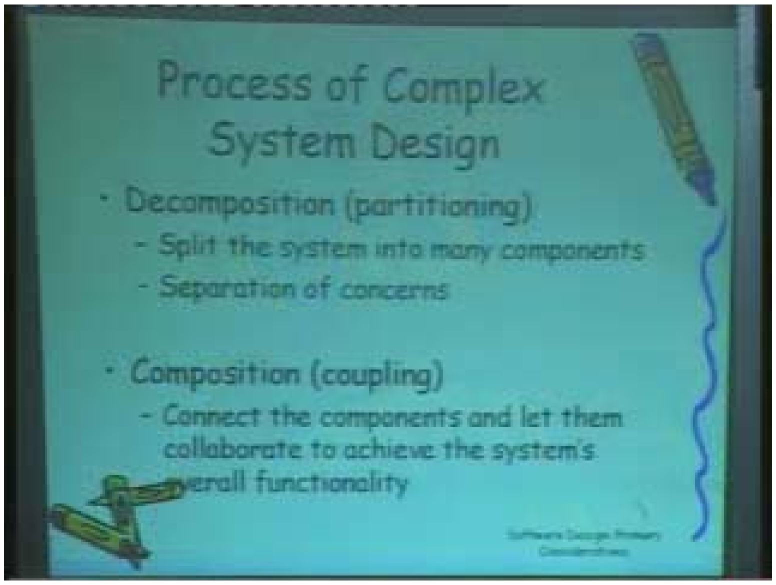
- Decomposition (partitioning)
  - Split the system into many components
  - Separation of concerns

Software Design: Primary Considerations

Any problem that you want to solve, you should be able to decompose the problem into sub problems solve them and then connect it back. Decomposition can also be seen as partitioning, split the system into many components and separate the concerns. You might split this system into many components, but then there might be redundancies and lots of overlaps and it might result into a chaotic situation. It has been decomposed and some of haphazard allocations of task have been done, but then there is no good protocol or there is no consistency among them in what they do. Such a system is not usable. So you have to separate the concern to have that modularity. Everyone does some specific task, it is not repeated elsewhere and then they collaborate with each other.

Of course when you decompose, the decomposed components or the subsystem or the partitions or the parts must collaborate with each other. There must be methods of composing them or connecting them or getting them collaborate or coupling them. Decomposition and composition are very important activities for doing complex systems design. You have to decompose, you have to separate the concerns and then again connect all your design artifacts together. Connect the components and let them collaborate to achieve a systems overall functionality.

(Refer Slide Time: 43:32)

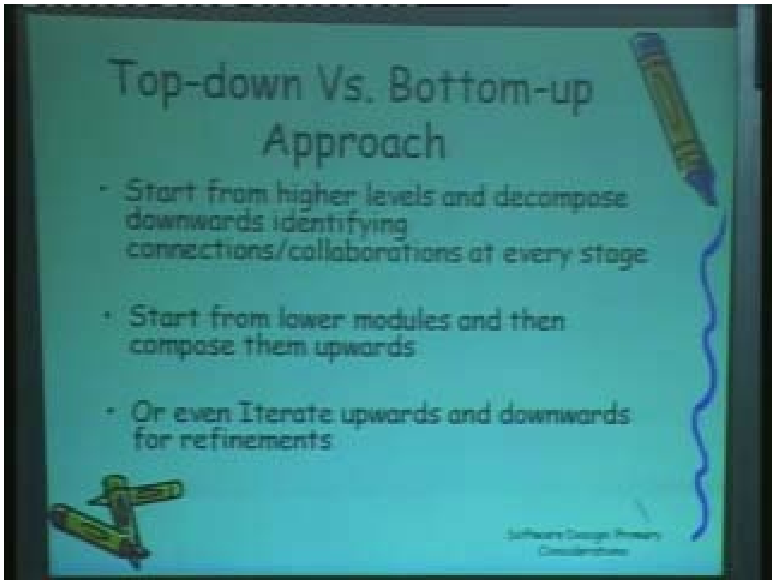


Process of Complex System Design

- Decomposition (partitioning)
  - Split the system into many components
  - Separation of concerns
- Composition (coupling)
  - Connect the components and let them collaborate to achieve the system's overall functionality

One can take a top down approach to this or can take bottom up approach. For top down approach, you start from a very high level design description and then break it down into further different sub systems or modules and then further break it down and go on designing that way.

(Refer Slide Time: 44:41)



Top-down Vs. Bottom-up Approach

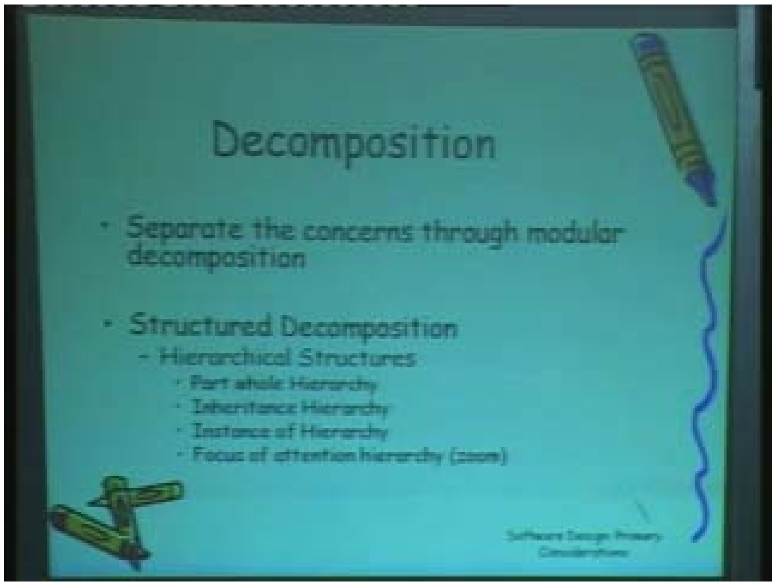
- Start from higher levels and decompose downwards identifying connections/collaborations at every stage
- Start from lower modules and then compose them upwards
- Or even Iterate upwards and downwards for refinements

You can also start from very low level module and then build them up put them into packages and then go on to build the architecture. op down versus bottom up approach Top down approach is good for starting from a high level description and understanding. It is good for documentation, but most often if you do not have the clear idea about your design, top down design also may not be possible.

You may have ideas about the low level module and then you want to connect them. Then you will go about bottom up design. But usually in practice you may want to go back and forth. You may want to start at some top level go down and then you get something new there and then again build up connect it back. So you may want to go back and forth that is iterate upwards and downwards for different refinement which is very practical approach. You start top down and if you feel that now you have something more in the down or lower layers then you want to come back and you want to modify your top level designs and so on. You can start from both directions. Top down design is in the practice if you have the top level vision of the design. One important property of top down decomposition is that it gives you the good understanding of the problem.

For documentation even if you have not practiced exact top down process for your design you should document from a top down perspective. Top level design document is the first one that the reader might want to read and might want to understand and then they can go about looking at lower level module and further zoom on. Top down documentation is very useful even if you have not followed the top down design. We now know that composition and decomposition are the two important activities in designing. Now we will now look at this decomposition activity in detail.

(Refer Slide Time: 47:04)



Decomposition

- Separate the concerns through modular decomposition
- Structured Decomposition
  - Hierarchical Structures
    - Part whole Hierarchy
    - Inheritance Hierarchy
    - Instance of Hierarchy
    - Focus of attention hierarchy (zoom)

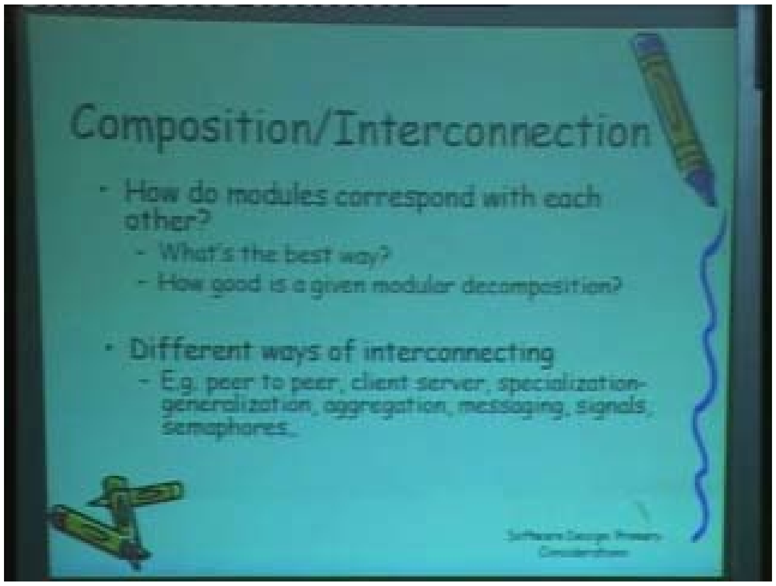
Software Design Primary Considerations

We have already seen that we need to separate the concerns through the modular decomposition and we have also seen that there are different ways for decomposition. For example, the hierarchical structures you can build. The decomposition gives you a hierarchical structure. You can decompose a system into a part whole hierarchy. There is a big system and then there are parts inside the system. A part whole hierarchy for example there is a computer and in the computer there are different parts such as cards, monitors, keyboard and or different IC chips and so on. You give design in a part whole and then further it can be decomposed as what are the parts of the card and what are the parts in the chip and so on.

Or you can take a classification hierarchy or inheritance hierarchy or a IS-A hierarchy. Another way of decomposing is instance of hierarchy, it uses types. For example, a is a type of b and b is a type of c and so on In languages like Smalltalk, you have objects, classes and there are Meta classes. This is another way of decomposing a system. Another orthogonal perspective is that of focus of attention which is zooming. You have high level design and then you go on to a lower level and zoom on to that and then you further zoom on to that to get this structured decomposition. These are different ways of structuring your design. In UML state chart, you can zoom on to a state and give a separate state chart for the stage. This is the way to decompose your system so that you will be able to understand request at the high level at the top level and then further go on downwards and zoom on to your sub systems or your sub design.

We now know the different ways of decomposition. Now we will move on to composition. What about composition, interconnection or collaboration and coupling? How do modules correspond with each other? What is the best way for them to correspond with each other? Should they interact with each other too much or should you minimize that or should there be no interaction? Without interaction how can they collaborate?

(Refer Slide Time: 50:43)



Composition/Interconnection

- How do modules correspond with each other?
  - What's the best way?
  - How good is a given modular decomposition?
- Different ways of interconnecting
  - E.g. peer to peer, client server, specialization-generalization, aggregation, messaging, signals, Semaphores.

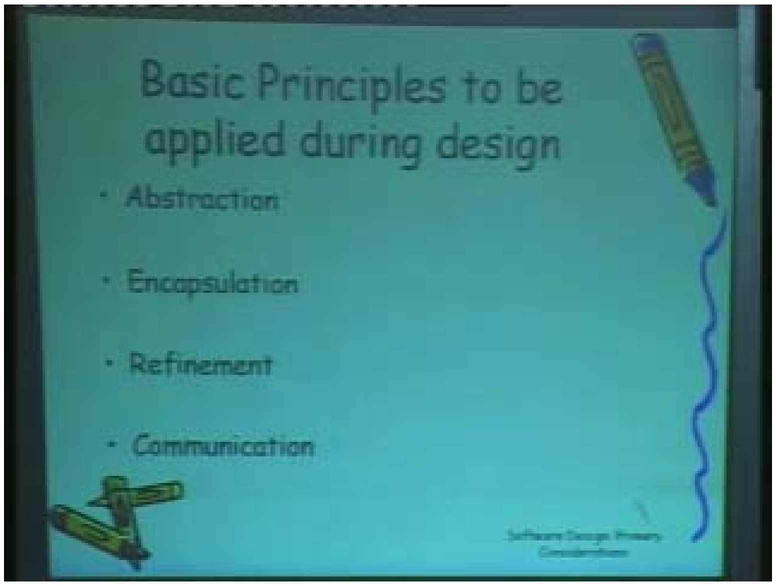
Software Design: Primary Considerations

How can you have a system with many modules? How good is a given modular decomposition? What is the goodness property? There are metrics to measure the goodness of design. There are different ways for modules to interact. You can have functional interfaces, you can have processes, you can have sockets, or you have remote method invocations or you have services on internet, or you can call functions, there are recursions etc. There are different ways of interaction and some of these have been listed here.

There could be peer to peer interaction ‘A’ communicates with ‘B’ and ‘B’ also communicates with ‘A’. ‘A’ can send a message to ‘B’ and ‘B’ can also send a message to ‘A’. This is not about just reply. This is a message that originates and asks for a service. A asks for B’s service and B asks for A’s service. That is peer to peer way of connecting.

You can have client server model for interconnection. The client asks the server to provide specific service. They send out the message and get the reply in return. You can have specialization-generalization for interconnection. Inheritance, aggregation for part whole, you have different ways of messaging, message queues, sockets and black boards. You have synchronization methods such as, signaling, semaphore, monitor etc. Hence, there are different ways of interconnecting system or composing them. Once you have decomposed, you have to connect them. So composition and decomposition, or decomposition and then the composition of the decomposed modules or the entities, interconnections in the collaborations among them are very important. There are different ways of making them collaborate and there are different ways of connecting them. Those have to be considered in your design. Now we will look at the basic principles that you apply during the design.

(Refer Slide Time: 53:04)



Basic Principles to be applied during design

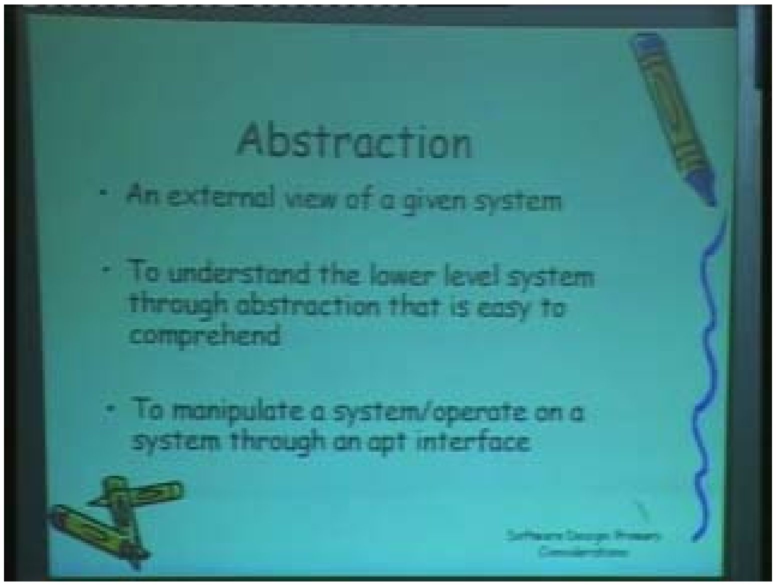
- Abstraction
- Encapsulation
- Refinement
- Communication

Software Design Primary Considerations

What are these guiding principles? We must have come across these terms in real life or in your software engineering or in your object oriented programming classes or basic programming classes. Abstraction, Encapsulation, Refinement and Communication, if you apply these principles in your design process you will get many of the good properties of design. Let us look at the properties of these basic guiding principles one by one. What is an abstraction? Abstraction is an external view of a given system. You want to understand the system from the high level perspective first and then go on to. Basically you want to abstract it out. Abstract what the system does from the external point of view.

The operating specific buttons, for example you have washing machines to operate or you want to operate a microwave oven. So you have the external perspective or the interface. There is an external abstraction.

(Refer Slide Time: 55:05)



Abstraction

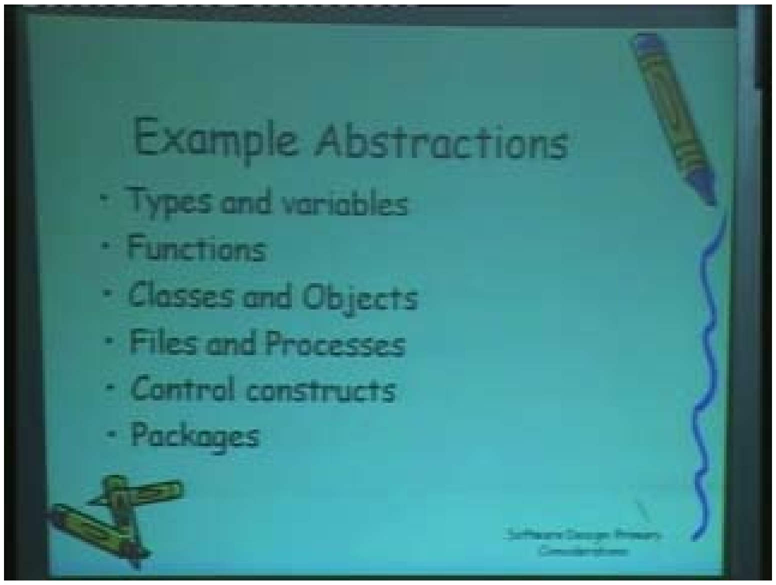
- An external view of a given system
- To understand the lower level system through abstraction that is easy to comprehend
- To manipulate a system/operate on a system through an apt interface

So the user's abstraction is different. The person who actually builds the microwave, his abstractions may be different and the principle of abstraction has to be applied for different roles. To understand the lower level system through abstraction that is easy to comprehend and is very important. Any system if it is seen from an abstraction point of view, that is external view of the system, you can understand your system well. Not only understand, but also for manipulation abstraction is important. For example, data structures like stack. You can think of a stack in terms of behavior of its push and pop method. Whatever is push goes on top of the stack and whatever is pop comes out from top of the stack.

When you describe this behavior, you do not worry about how it is implemented inside. For example, what is the data structure used for this stack? Is there a linked list or is there a static data structure? How many pointers are you using? How are you implementing them? How are you initializing? Are they automatically initialized by the language itself? These are the internal views. But when you want to describe the data structure you want to describe it as an abstract rather than an implementation.

So given an abstraction, you understand the system and then use the abstraction to manipulate the actual instance of the system for which you have described the abstract. So we use this principle of abstraction in your day to day programming practice. For example, types and variables, functions, classes and objects, files and processes, packages, control constructs such as 'if..then..else'; 'repeat..until'; 'for loop' or sequel statements and so on.

(Refer Slide Time: 56:36)

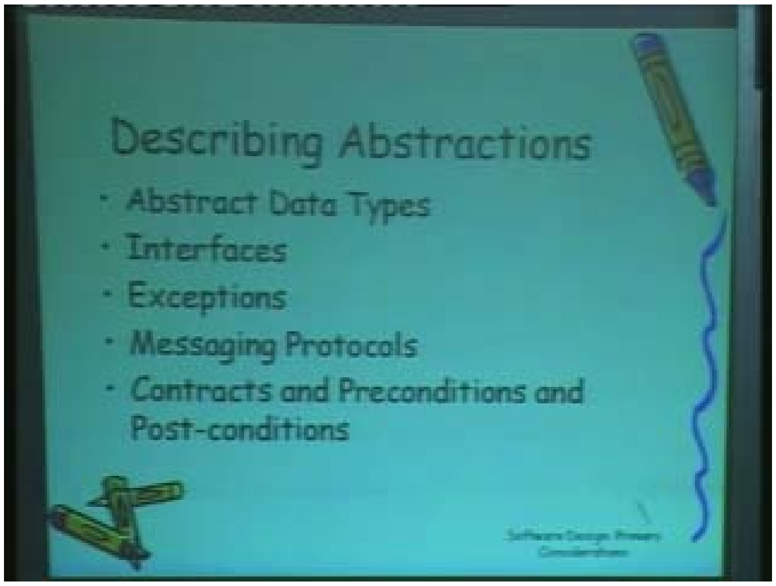


Example Abstractions

- Types and variables
- Functions
- Classes and Objects
- Files and Processes
- Control constructs
- Packages

Use these abstractions and then build similar programming which is abstract programming. But for your given problem domain you have to find out abstractions in the problem domain. How do you describe these abstractions? You can use abstract data types, you can describe them into interfaces, and you can also describe the exceptions or the exceptional behavior that the entity will project.

(Refer Slide Time: 57:48)



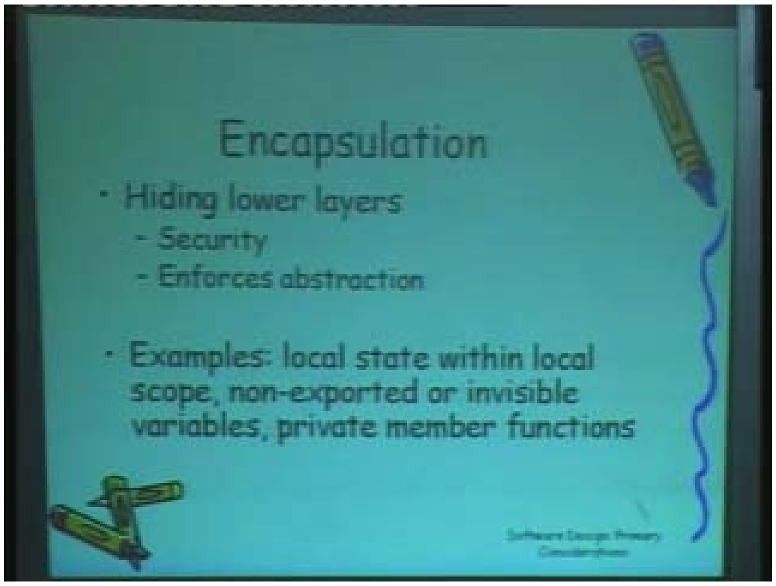
Describing Abstractions

- Abstract Data Types
- Interfaces
- Exceptions
- Messaging Protocols
- Contracts and Preconditions and Post-conditions

Suppose you are asking for a service and if the server is unable to provide that service, what is the behavior that you expect from server is also the part of abstract. It is an externally observable behavior exception that you are going to expect from server.

Messaging Protocols: What are the messaging protocols? What we send first and what do you send later and what is expected? The protocol, handshakes are also can also be part of the high level view. Contracts that are provided, pre conditions and post conditions: When you ask for a service, what are the parameters that are expected from you? And what is guaranteed to you in return? Pre conditions and post conditions are also part of contract or abstraction of the given entity. The next one is principle of encapsulation.

(Refer Slide Time: 58:35)



Encapsulation

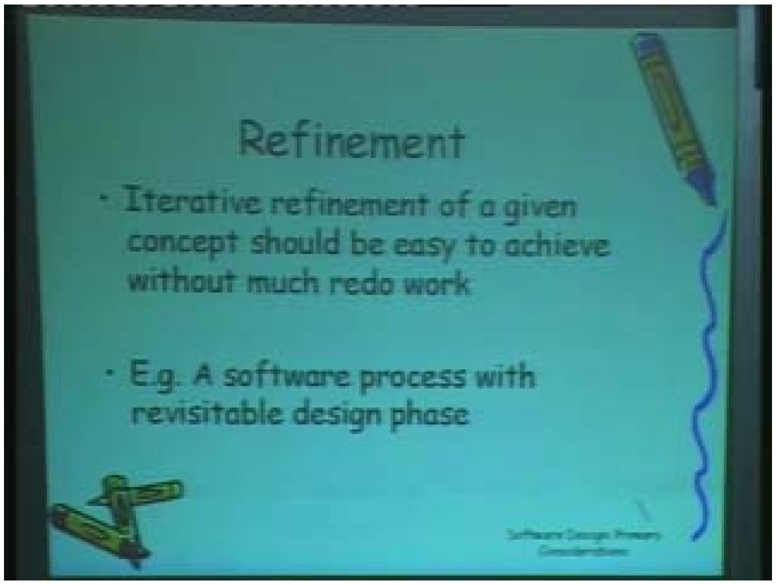
- Hiding lower layers
  - Security
  - Enforces abstraction
- Examples: local state within local scope, non-exported or invisible variables, private member functions

Software Design: Primary Considerations

Encapsulation talks of hiding lower layers. This is for security and also for enforcing the abstraction. You would hide lower layers on top of the higher level. Examples of encapsulation is local state within local scope, non exported or invisible variables, private member functions in your classes or in your object oriented program. So encapsulation is manifest. You want to hide something to project the external abstraction that you have thought of to the user and the internal implementations are hidden. You have to enforce it, it is not enough to simply implement the abstraction, but it is also required that you enforce the abstraction through the principle of encapsulation. Refinement: Iterative refinement of a given concept should be easy to achieve without much redo work.

How do you refine? What are the principles of refining? What are the ways of refinement? How are you going to apply this into your designs? For example, a software process with revisitable design phase you can come back again and again in the process and you apply this iterative refinement. You can also apply a refinement to inheritance. If you want to refine your subclass, your super classes to sub classes. You can apply this open close principle for components. Implementation is open for refinements but the interface is closed.

(Refer Slide Time: 59:30)

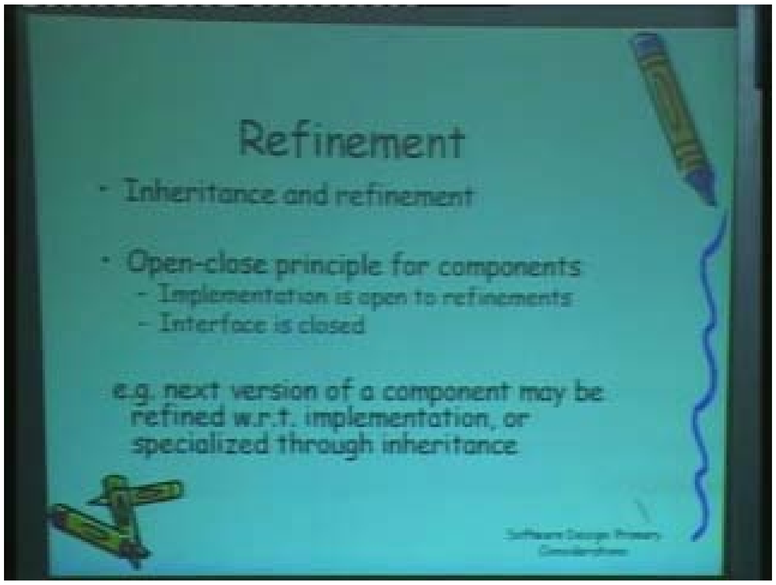


Refinement

- Iterative refinement of a given concept should be easy to achieve without much redo work
- E.g. A software process with revisitable design phase

Software Design: Primary Considerations

(Refer Slide Time: 59:59)



Refinement

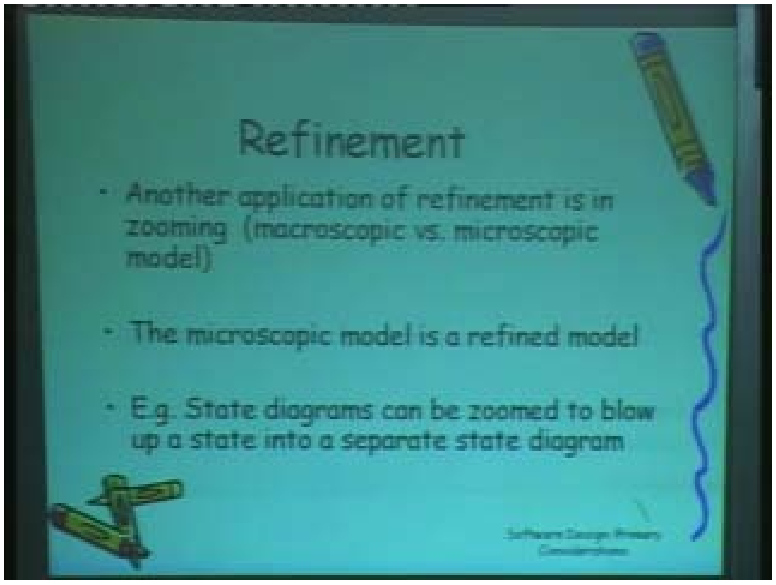
- Inheritance and refinement
- Open-close principle for components
  - Implementation is open to refinements
  - Interface is closed

e.g. next version of a component may be refined w.r.t. implementation, or specialized through inheritance

Software Design: Primary Considerations

A contract is guaranteed that you can change the implementation and you can make it better. You can refine your implementation, which means in the next version of a component you give a refined version of the component. The implementation is better or it might also be specialized through inheritance. So refinement can take place through inheritance or open close principle by changing the implementation without affecting the contract that you have exported or guaranteed for the outside user. Another application of refinement is zooming. You want to zoom and refine that means from macroscopic model you want to go to microscopic model. A microscopic model is the refined model.

(Refer Slide Time: 1:00:33)



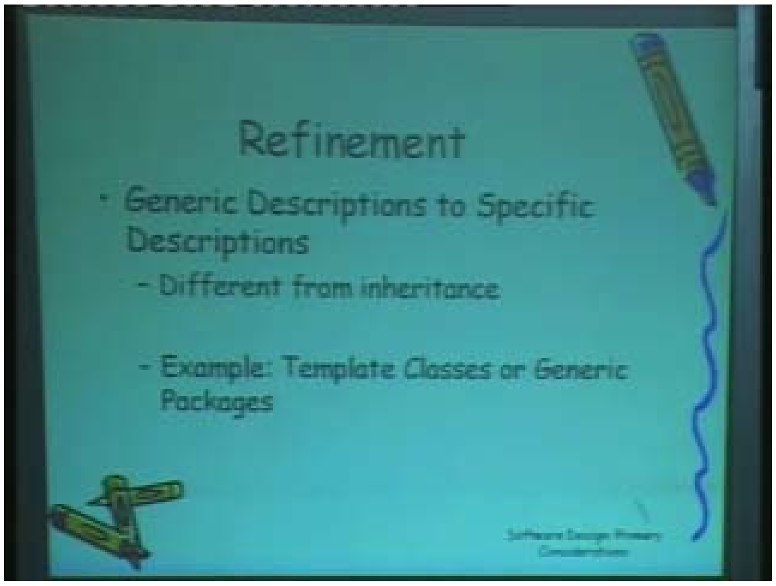
Refinement

- Another application of refinement is in zooming (macroscopic vs. microscopic model)
- The microscopic model is a refined model
- E.g. State diagrams can be zoomed to blow up a state into a separate state diagram

Software Design: Primary Considerations

State diagrams for example in UML. They can be refined or can be zoomed on to. There is fourth way of refinement that is generic description. We have generic classes you can refine them for specific types. This is different from inheritance. It is template classes in C++.

(Refer Slide Time: 1:00:49)



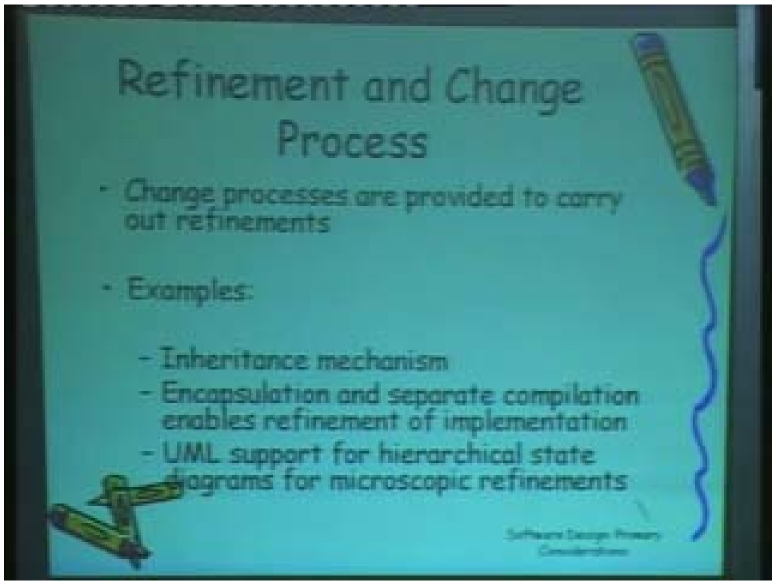
Refinement

- Generic Descriptions to Specific Descriptions
  - Different from inheritance
  - Example: Template Classes or Generic Packages

Software Design: Primary Considerations

Refinement is very important for a change process or even for expansion for evolution and as well for understanding an organization. The change processes are provided to carry out refinements. These are also depends on what are the abstractions that we use for the refinement.

(Refer Slide Time: 1:01:25)



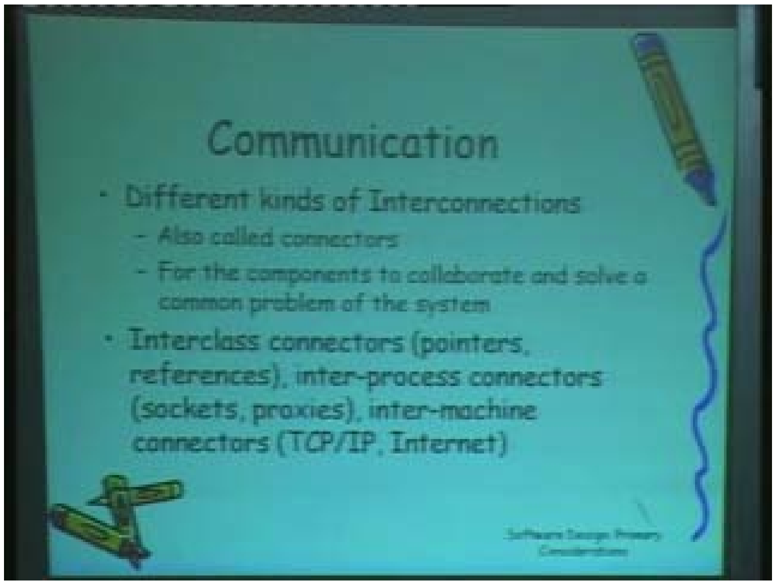
Refinement and Change Process

- Change processes are provided to carry out refinements
- Examples:
  - Inheritance mechanism
  - Encapsulation and separate compilation enables refinement of implementation
  - UML support for hierarchical state diagrams for microscopic refinements

Software Design: Primary Considerations

So the examples are inheritance mechanism, encapsulation and separate compilation of different files, you can refine independent components separately or you can blow up your models and refine them from macroscopic to microscopic point of view etc. Finally we are going to see the aspect of communication. Different kinds of interconnections and you have to note the different types of connectors and how to make use of them effectively in your design.

(Refer Slide Time: 1:02:12)



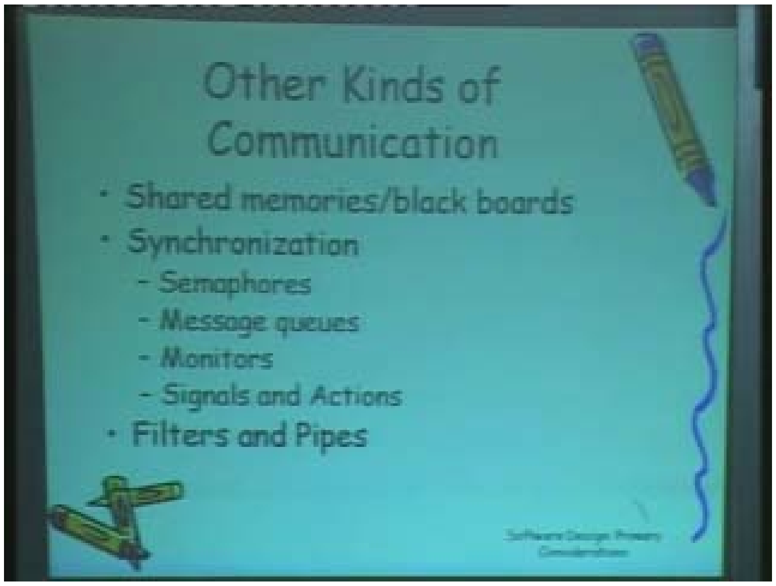
Communication

- Different kinds of Interconnections
  - Also called connectors
  - For the components to collaborate and solve a common problem of the system
- Interclass connectors (pointers, references), inter-process connectors (sockets, proxies), inter-machine connectors (TCP/IP, Internet)

Software Design: Primary Considerations

Interconnections: They are also called different kinds of connectors; the components collaborate through these connectors. So we have to use the right kind of connectors. For interclass connectors, you can use pointers, references etc. For inter process connectors, you can use sockets, proxies etc. For inter machine connectors you can use TCP IP, internet and so on. For synchronization there are different kinds of connectors, example, pipes and filters and some more connectors have listed out here. They are shared memories, black boards, semaphores, message queues, monitors, signals and actions, filters and pipes.

(Refer Slide Time: 1:02:33)

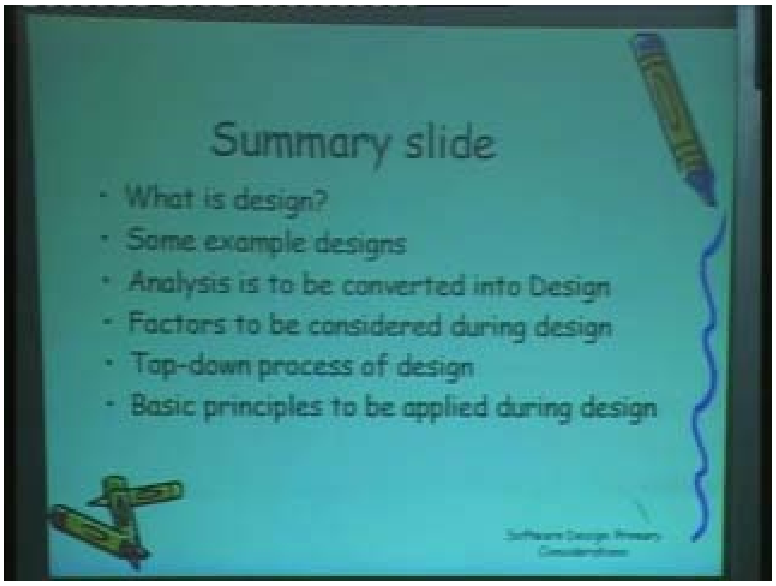


Other Kinds of Communication

- Shared memories/black boards
- Synchronization
  - Semaphores
  - Message queues
  - Monitors
  - Signals and Actions
- Filters and Pipes

Software Design Primary Considerations

(Refer Slide Time: 1:03:40)



Summary slide

- What is design?
- Some example designs
- Analysis is to be converted into Design
- Factors to be considered during design
- Top-down process of design
- Basic principles to be applied during design

Software Design Primary Considerations

So there are different ways of using these connectors amongst those the components that you have designed. Finally we will look at this summary slide what we have seen in today's lecture. We looked at what is design, and then we looked at some examples of design. We had seen one example of design of a workshop or a real life process. Then we learned that the analysis is to be converted in to design. We have seen different factors to be considered during design. We have seen the advantage of a top down process of design and we have also seen that you can apply or you can go back and forth in a design process. Then we looked at the basic principles to be applied during design; the abstraction encapsulation refinement and communication.

Thank you