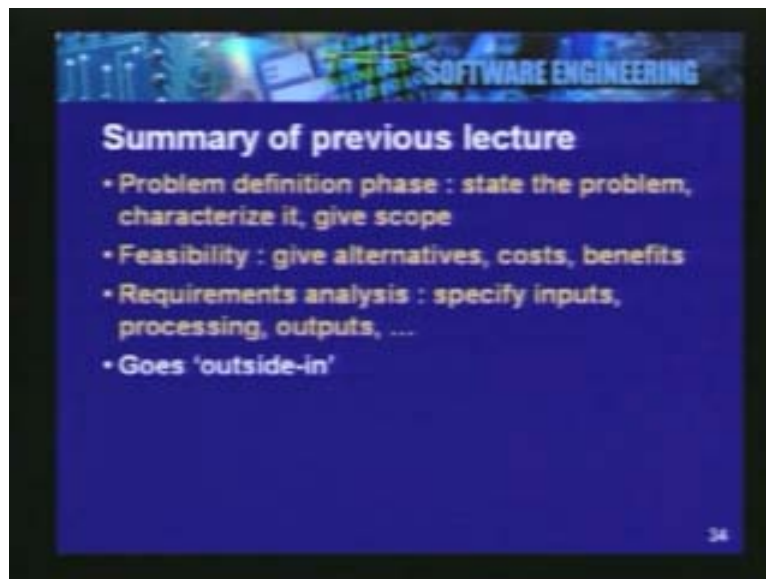


Software Engineering
Prof. N.L. Sarda
Computer Science & Engineering
Indian Institute of Technology, Bombay
Lecture-4
Overview of Phases
(Part - II)

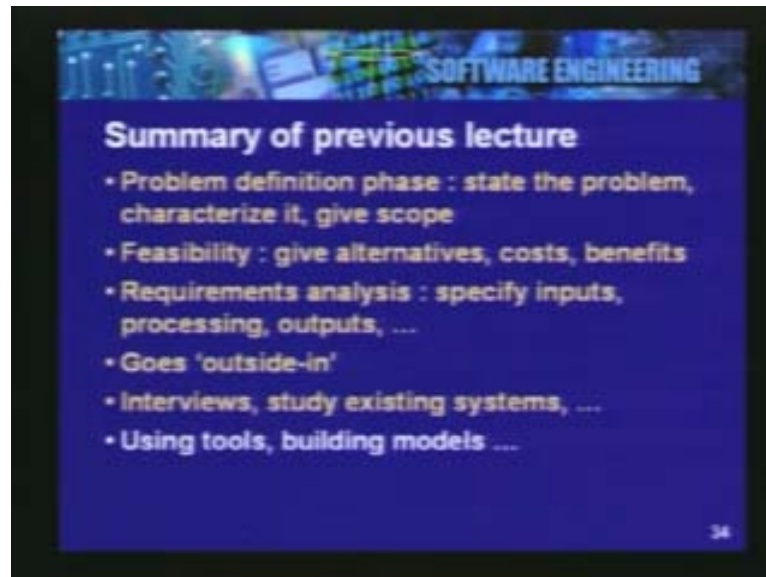
We studied the problem definition phase, with which we generally start our development life cycle where we tried to state the problem clearly try to characterize it, and give a scope of effort and time to the user. The next step we discussed was the feasibility step, where we give different alternatives to the user we work out their costs and benefits which mean include both the tangible benefits as well as the intangible benefits. And then we started discussing the important phase of requirements analysis: where we specify inputs, processing and outputs. Primarily we said that the requirement analysis phase goes 'outside-in' where we start by looking at different outputs that the system has to provide to the users or different type of interactions that the system has to support for the users.

(Refer Slide Time: 01:46)



Starting from the outside boundary of the software, we work towards the inside and try to identify the inputs and the processing which is necessary to meet the functions of the systems. We said that the requirement analysis phase consist of meeting users, finding out detailed requirements as to see them and in fact it may be also study the existing systems by following various activities which happen in the real world.

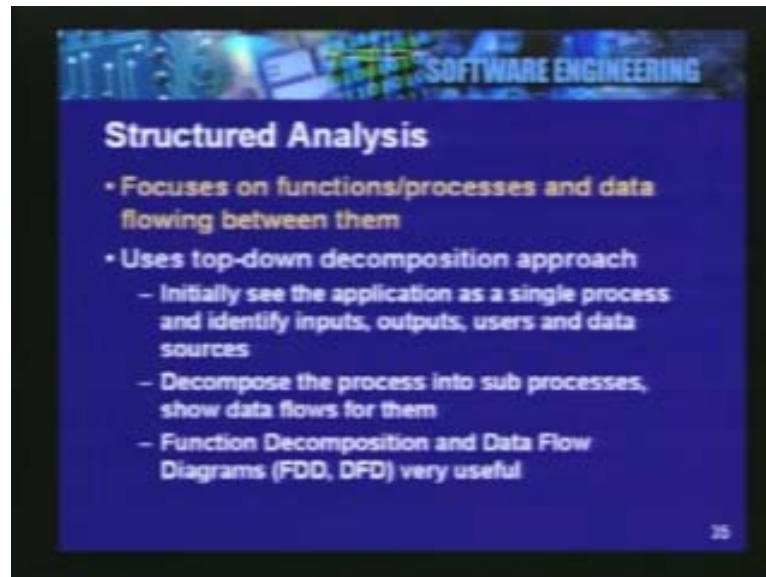
(Refer Slide Time: 02:30)



And since these activities is a fairly long drawn activity, many people are involved lot of record keeping has to be done. It's important that we use appropriate tools and also build different types of models through which our understanding can be concisely stated and can be verified with the users. So these are the different phases we have discussed last time. Let us continue further and see what is available further in requirement analysis phase and what kind of documentations standard are prescribed for recording the finding during analysis phase.

Now structured analysis has often been accepted as one of the common technique to use during requirement analysis. It consist of focusing on various functions and processes existing in the user environment and then identifying the data flowing between these functions. We generally use the top down decomposition approach in this where initially the whole applications is seen as a single process. For these processes we identify the different inputs, outputs, who are the users of the systems, what kind of data is stored and which need to be used and which needs to be made available to the applications. So at the single process view of the applications, the external inter faces in terms of the inputs, outputs and users are possible data sources are identified. This is the initial step or the first step. Hereafter we decompose this process in to sub processes and we identified the different data which flow between them. The two techniques which we commonly used in the structured analysis method are the function decomposition technique and the data flow diagrams.

(Refer Slide Time: 03:45)



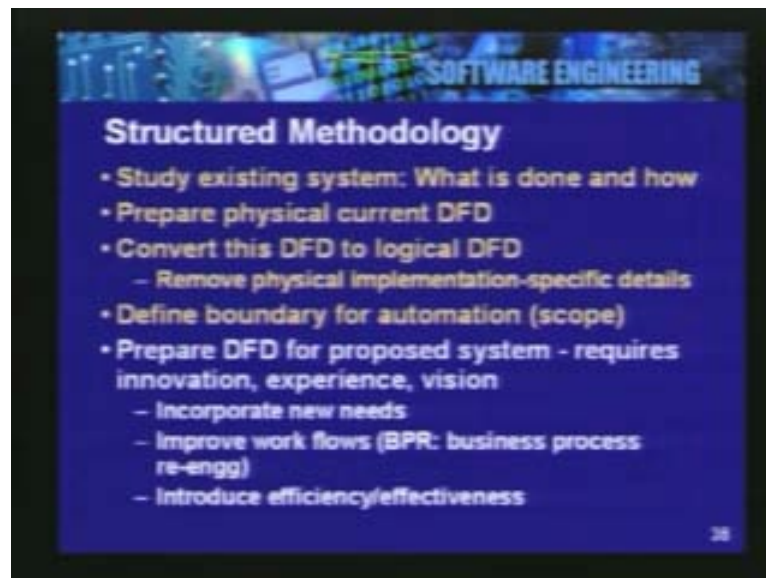
We will study them in more details subsequently but basically the idea here in the functional decomposition is that the overall functionality of the software system we plan to develop is broken down into subtasks. For example in a railway reservations system we may have one sub function called reservations the next one called cancellations the third which may consist of producing various types of reports and the fourth could be maintaining the data about various strains, schedules and time tables and so on. So we decompose the overall functionality in to sub task or sub processes this is the function decomposition.

And was the function decomposition has been done and the sub processes have been identified we also can establish what kind of data flow text place among these sub processes. So this is the metrology in the case of structured techniques and subsequent to doing this kind of analysis where we start from single process view and decompose it we precede further to establish what is called the current physical data flow diagram. So we start by studying the existing systems we understand the sub processes we try to find out what is done and where and how it is being done. So the beginning point is the existing systems. From here the data flow diagram that we produce naturally corresponds to the current systems and it corresponds to the physical view of what is happening in the real world. So we call it physical current data flow diagram.

From here we prepare a logical data flow diagram by removing the current implementation-specific details. Basically we want to now focus on the what specification what exactly done and what is require to be done rather than how it is being done. So we try to construct the logical data flow diagram from the physical current data flow diagram. We did also identify the boundary of automation. What is the scope for the software system which processes and sub processes would be included in the scope of the software. Once this is done we prepare the data flow diagram for the proposed system.

In fact the difference between the existing system and proposed system has to be clearly worked out and this would depend on not only the user requirements but it would benefit from the experience of the analyst the innovations that he can suggest in the existing processes so that the proposed system will effectively solve the problems which were felt by the user and which has initiated the whole project.

(Refer Slide Time: 07:22)

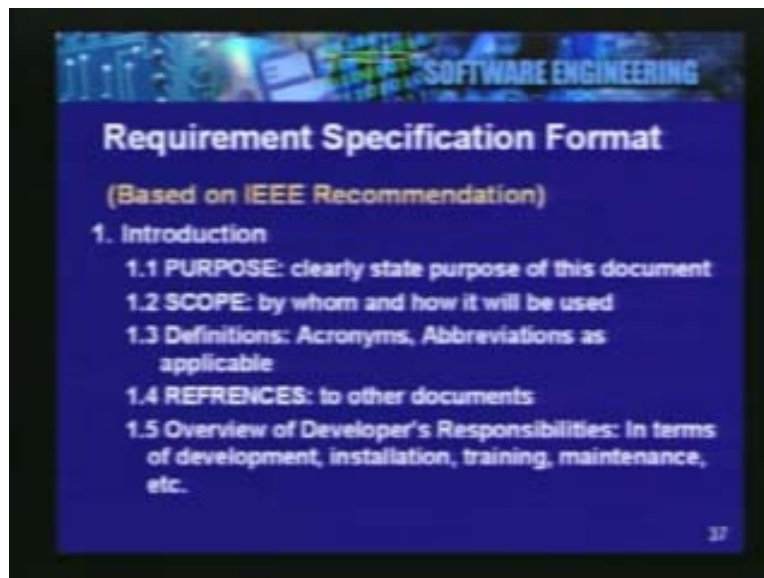


So the proposed system is not just a duplication of the existing systems. We are not only changing the way the things are done today but we may even extend the scope of the system we may suggest new task, new functions, new output from the system. So this would depend on the **vision** and the experience of both the users in terms of what they need in order to solve their business problems effectively and the experience and the **vision** of the developer or the analyst who is assisting the user in identifying the requirements. So naturally this will include new needs of the users the proposed system would improve the work flows it might do some business process re engineering it will try to **introduce** efficiency and most effectiveness in the way things are done at present. So idea is to repair the data flow diagram for the proposed system. This proposed system will become the basis for software implementations subsequently and once these proposed systems is clearly defined in terms of inputs, processing, outputs.

Then we are ready to develop or to prepare the documentation for this particular phase which results in the preparation of what is called the software requirements specification document. We will now look at the format of the Requirement specification document. Is very important to understand these clearly because this is one of the very important documents as we said earlier it become a baseline for a contract between the user and the developer. It is expected to contain the complete requirements which are adequate to tell the user what he would get from software and it also adequate to tell the designers and developers subsequently for the complete implementations of the software solutions.

So let us understand the documentation format which has been standardized by IEEE this is the format that we are discussing is as per the recommendation given by the IEEE organizations which has made lot of recommendation for software's standardization. The first section is the introduction section. It first establishes the purpose of this document; what exactly is the purpose and what is covered in this document?

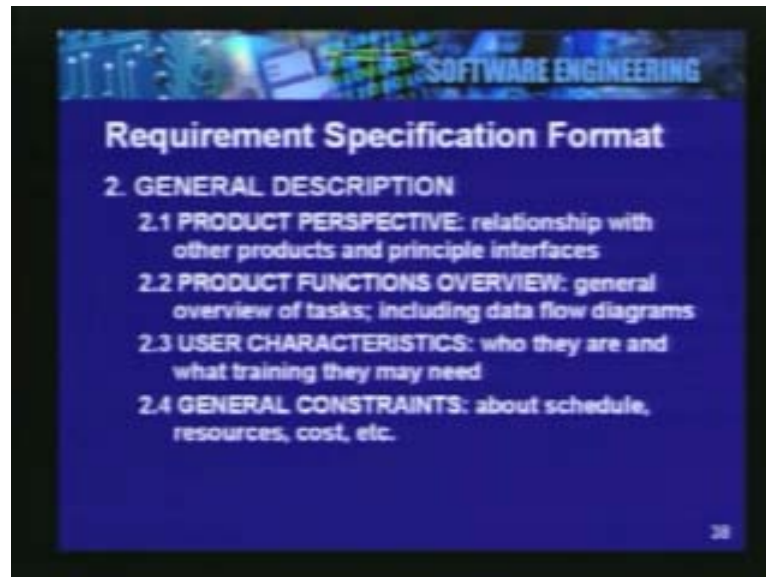
(Refer Slide Time: 10:58)



Then it defines the scope of usage; who will use these documents and for what purpose? It gives various definitions, abbreviations, acronyms as they may be applicable and which have been used in the rest of the document. It may also refer to other documents which have been prepared and which have been referred by the user. For example, these may be documents made available by the user which define some parts of the requirements or these may be documents which are prepared during feasibility step. All these documents can be referred here clearly in the SRS document. Finally the introduction will also briefly specify the responsibilities of the developer; what exactly is the scope of the project? Does it include development, installation, training, maintenance, and support for a limited time etc?

All these responsibilities are also clearly stated briefly in this section so introduction section is an overview of the documents purpose, and the references responsibilities. These are clearly stated and this is the first section of the SRS document. The next section is the general description of the software that we plan to develop. It first contains the product perspective. Here we see the product as a single unit and try to establish its overall context. We see it as a black box; how does it relate to possibly other software products? What are its interfaces with the other products?

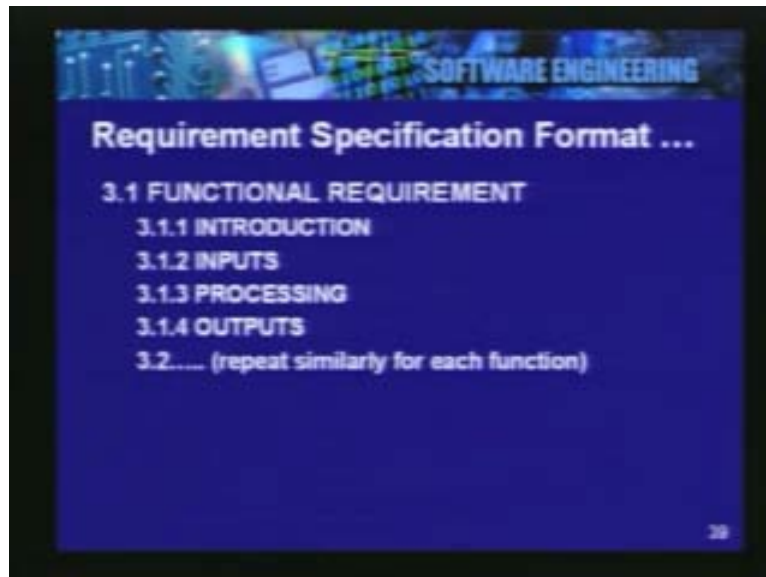
(Refer Slide Time: 12:53)



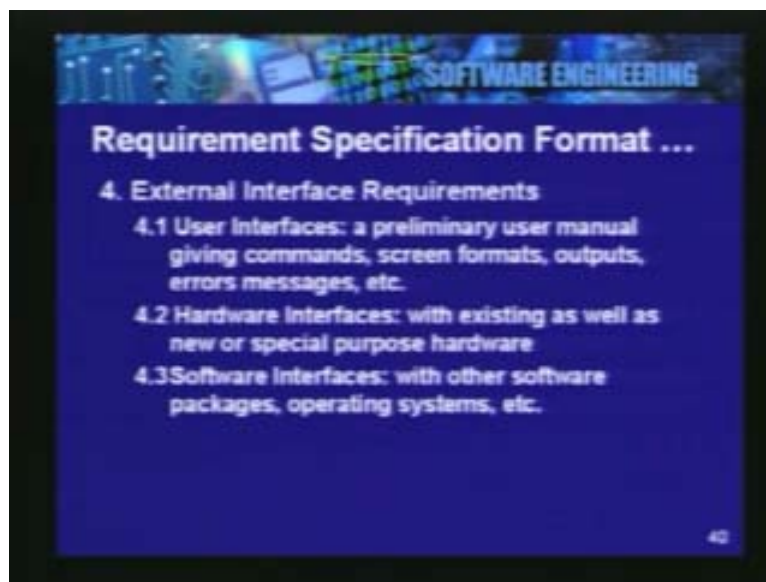
Then it states the functions of the product briefly. These are the general overview of various tasks that the product is supposed to perform, including references to data flow diagrams. This product function overview is an overview of the functionality of the software. We then try to characterize the users for whom the software is being developed and we also mention the different constraints which are placed. These constraints may be about the schedule of development, resources such as the existing system and software packages, constraints about the cost and other resources. The section 2 of the SRS document is a general description given not only the overview of the product's features, but also placing it in its proper context in terms of other products, in terms of users who will use what characteristics we can assign to these users, in terms their expertise and training and so on.

Section 3 is really the core part of the document. In this part we give functional details. We define every function of the software, by giving a brief introduction to this function. Then we describe the inputs to the function, processing and the outputs. So these functions may be different functions that the software will perform. For example, it may be a function related to reserving a seat on a train or cancellation of a ticket already booked. So these are different functions or tasks. Each of them needs certain inputs, performs some processing and produces some output. For every such function, we give description in this particular section. So section 3 is really the body of the SRS document where functions are described in full details.

(Refer Slide Time: 14:36)



(Refer Slide Time: 15:32)

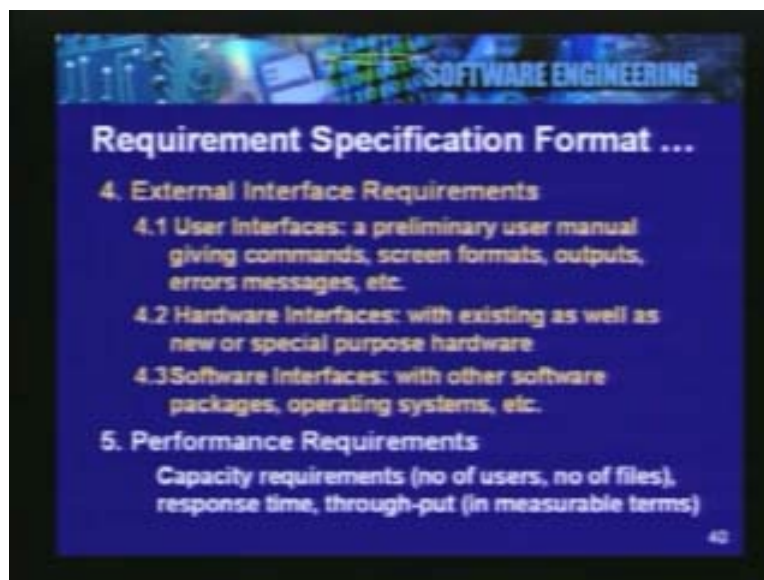


In section 4, we give summary of external interfaces of the software. We first define the user interfaces; that is the human users, what kind of screens or outputs they would receive, what kind of commands they can give, what kind of error messages may be thrown up by the software. In fact this section of the document can be treated as a preliminary user manual. In fact it is this section which will give a good idea to the user about the kind of system he is likely to get at the end of development. We should note here that we are not doing any design. At this point, the user interfaces are specified more in terms of contents.

Meaning, what are the logical contents of different interfaces or commands that will be made available to the user in order to use the software we are developing. So we must remember this point that, exact screen format, screen layout report format are not essential. If the logical contents of these components which are specified and in terms of which the user will try to get a good idea of what kind of system is being developed. Then we also describe the different hardware interfaces. Of course these interfaces are applicable when we are interfacing with specialized hardware for meeting some functionality of the software.

For example this might be a hardware interface to control some chemical process. So these hardware processes are also clearly specified. Then we have software interface specifications. In case the product that we are developing has to interface with some existing packages. Then those interfaces are clearly specified here. For instance the railway reservation system may interface with the accounting system of the railways, so that all the fund collections are automatically transferred to the accounting system. Now, this accounting system will be an already existing system. So our railway reservation system package that we might be developing will have to interface with such an accounting package.

(Refer Slide Time: 18:09)

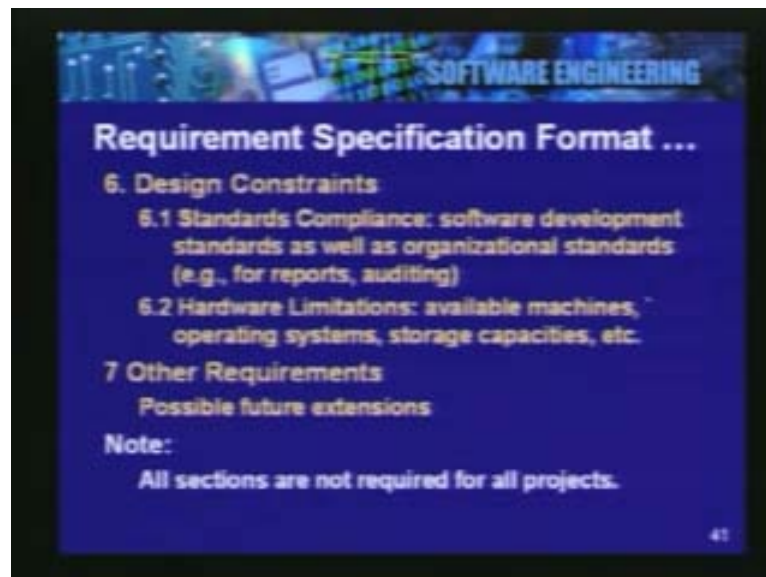


So these software packages are also clearly specified. We also define here performance requirements. These performance requirements would be used subsequently for defining the sizing of the various elements of the software or the hardware. For example we will identify the number of users, volumes of data, the response time requirements, the throughput requirements. These requirements are specified in terms of user's requirements. That is what kind of environment in which the system will function, what kind of work load it will have, and what response time is expected from the system. So if the system is a real time kind of a system where it is processing online transactions, then we need to define the throughput and the response time. Consider the example of say banking, when the banking software has to accept transactions from the customers.

Then we need to identify, in how much time the system must react to the input of the transaction. Or totally in a particular given time which may be the peak time, how many transactions should the system process in a given slot, which could be one hour or something. So these are the capacity requirements which are important to be specified and they are specified from the user's point of view. And these are defined by taking into account, what are the actual requirements of the organization. So after identifying these, we identify the various design constraints. Now these constraints may be in terms of the development standards. The organization may have very specific standards for developing the software. Or it may be standards for various reports. For example in the financial world lot of reports are required by regulatory authorities. So these reports must be as per the standards. Then there are auditing requirements. So these standards needs to be complied with and those must be clearly identified and they are identified in this section.

We also identify any hardware limitations. For example the user may have already some machines, operating systems; database systems etc and we may need to develop the software for the available environment. So these would be treated as hardware constraints or system constraints. And finally we state any other requirements possibly mentioned here the future extensions for the software. This is the overall format for the SRS document which is giving a complete specification of the software in terms of what it would contain. We must note here that all sections which are listed here may not be required in all projects, because some projects may be very small in scope. But in most cases, these different sections would be necessary because they cover various aspects not only the functions, but they cover constraints, they define functional requirements, they define performance requirements.

(Refer Slide Time: 21:04)

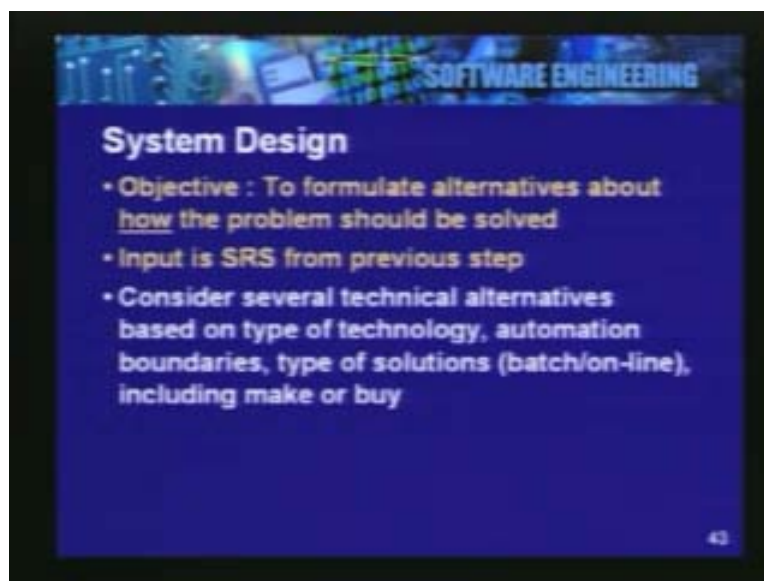


So one can see here that SRS format has taken into account various aspects and we can see that by using this specification format, we are meeting our expectation that the document will clearly tell the user what features and functions the software will provide and how he will be able to use it. Similarly we have made it detailed enough, so that from here we can hand over this document

to our design team. And they will be able to convert the specifications into a design. So SRS document which is a baseline document needs to be detailed and we must ensure that we have collected all these data and put it in the form of a document. Once the SRS is prepared, which can be an extensive task for large software, we need to review this with users. There has to be a formal review meeting in which the entire SRS document would be reviewed. There should be a sign off, where the users should say that; yes, the SRS document clearly defines what the system needs to do. And also there would be some peer review, which will ensure that the document contains enough details for the design activity to be carried out. So we are going to do the design subsequently. It must contain the enough details, so that the design can be carried out.

We can define the platforms, we can define the capacities, and we can work out the sizes required for the different systems and so on. All these must be possible from the SRS document and we see that it is detailed enough to give us all these data. It is very important that the analysis phase results in preparation of such a detailed and standard document. So we proceed further only when the document is accepted by the users as being complete in all aspects. Once it is signed off, we proceed to the next phase, which is the design phase. In the design phase our objective is to formulate the implementation strategies for the problem that has been identified. So now we move to the “how” part of the solution. In the SRS phase or in the requirement analysis phase, we have defined “what” part of the problem. Now we are trying to address – “How” the problem should be addressed, how it should be solved? Input to this phase is naturally the SRS document from the previous step. We now try to consider different technical alternatives.

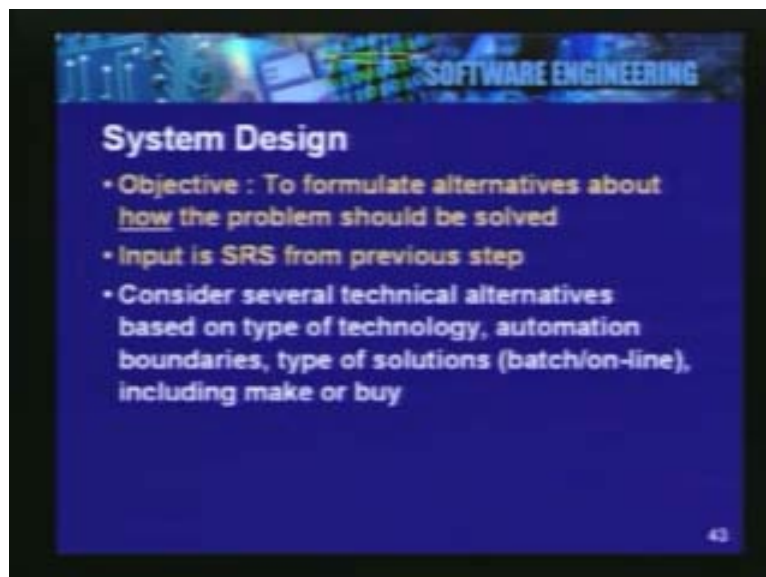
(Refer Slide Time: 24:49)



We must remember that SRS is a logical document; it specifies the requirements, without dictating how those requirements must be implemented. Design phase is the first phase in which we make a transition towards the solution. Different alternatives may be available. These are primarily technical alternatives, what kind of technology we should use for solving the problem. Now after this point we are going to increase our efforts from phase to phase in order to build the

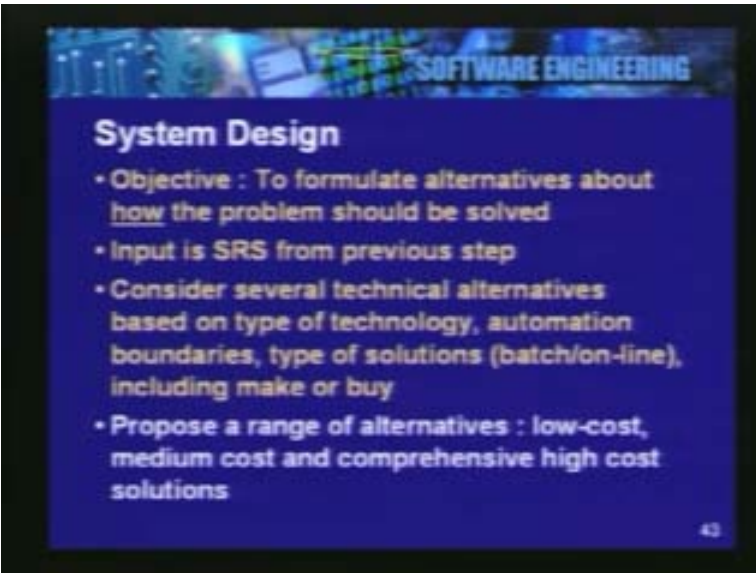
software. It is important to review at this point once again whether the automation boundary that we have defined is the appropriate one or not. So we look at not only the different types of solutions which may be possible based on the technology options, but we may once again consider the business functions which needs to be put in the scope of the system. We also consider what type of solutions to create, whether they would be batch or online type of solution, including whether we should make the software or buy something which is readily available in the market. In fact these are different alternatives in order to realize the goals which have been stated in the SRS document. So this is a point at which again we consider different alternatives and these alternatives may have different costs and different efforts.

(Refer Slide Time: 26:22)



They may need different time to complete the project. So it is important to consider these technical alternatives once again. Whereas, in the feasibility study we did consider various alternatives which were primarily based on business alternatives, now we explicitly consider additional technical alternatives. We are doing it again because we have a very good understanding of requirements now.

(Refer Slide Time: 27:02)



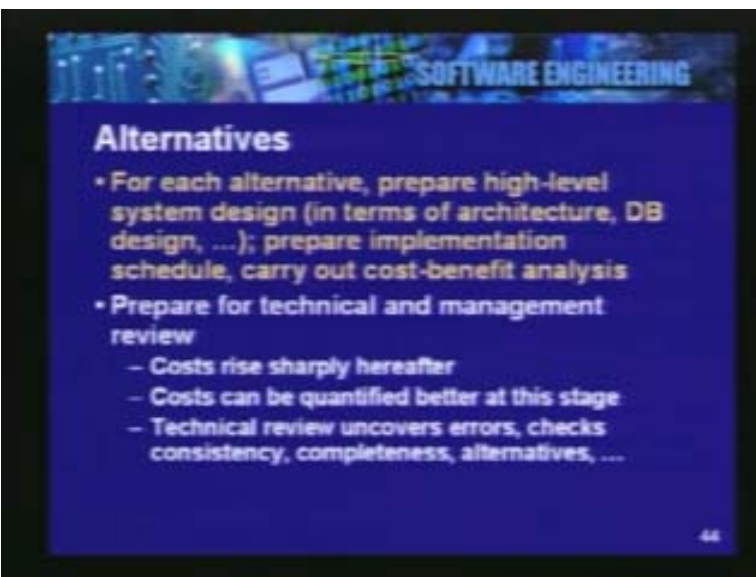
System Design

- Objective : To formulate alternatives about how the problem should be solved
- Input is SRS from previous step
- Consider several technical alternatives based on type of technology, automation boundaries, type of solutions (batch/on-line), including make or buy
- Propose a range of alternatives : low-cost, medium cost and comprehensive high cost solutions

43

We have completed the requirements specification. This is appropriate point to consider different technical alternatives. The purpose of coming up with different alternative is to provide user different options, which may be starting from low cost options to medium cost, high cost options. Naturally, giving different cost effectiveness, different benefit to the user, in terms of meeting overall requirements. For each alternative we prepare a system specification we prepare implementation schedule. And if necessary we carry out the cost benefit analysis, because the costs are going to significantly rise from this point onwards. Since we understand the scope better now, we will be able to do a better cost benefit analysis at this point.

(Refer Slide Time: 28:00)



Alternatives

- For each alternative, prepare high-level system design (in terms of architecture, DB design, ...); prepare implementation schedule, carry out cost-benefit analysis
- Prepare for technical and management review
 - Costs rise sharply hereafter
 - Costs can be quantified better at this stage
 - Technical review uncovers errors, checks consistency, completeness, alternatives, ...

44

We then prepare the design document and which is then reviewed both by technical people as well as the management people. The technical people who review the document would ensure that different technological options considered are meaningful under all important or all applicable options have been covered. Whereas the management review would be concerned with identifying whether there are implications on cost, benefits and so on. The purpose of the review is to ensure that we are within the proposed costs and we will be able to meet the schedules. The technical review would ensure that there are no errors in the proposed design, that it is consistent and complete and various applicable alternatives have been worked out. This phase should end with a clear choice. So the different alternatives prepared in terms of different system designs are reviewed both by the management and the users, and it ends with a clear choice which can be taken further into detailed design and implementation. What are the design goals in this phase? In this phase we are trying to define or do a design of the processing component, because the requirements are clearly defined in the processing needs of the user. So these processing components will be converted into software. We have a functional approach as well as we have an object oriented approach. Different alternatives are available for design methodologies. They may be conventional functional development or it could be object oriented development. There are different methodologies available for the paradigms.

So if it is a functional paradigm you may use structured system analysis and design methodology, which is based on taking data flow diagrams and converting them into software architecture. Taking entity relationship diagrams and converting them into database design. Or it could be an object oriented paradigm and the associated object oriented methodology in which you convert it into an object oriented implementation. So the kind of design methodology you choose would depend on the paradigm that you want to employ for implementing the software. If you are designing the functional components separately you also need to design the data component. The data component is generally handled by designing a database for the given application. A database design consists of multiple steps.

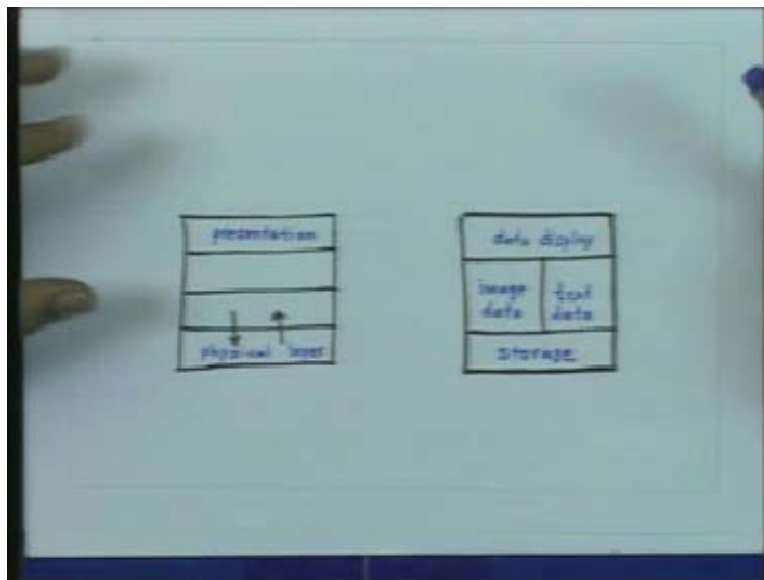
(Refer Slide Time: 30:52)



For example the first step is generally the step in which the normalized database design is worked out from the entity relationship model. Now all database systems have a particular data model such as the relational database model. We will study this issue in some more details subsequently. But when we prepare the entity relationship model to understand the information domain of the user, this is a conceptual data model and this needs to be converted into a database design. After doing a normalized or conceptual database design, we modify that if necessary for getting good performance from the system. This often is called de-normalization. And finally, a step in database design is consisting of choosing the right storage techniques through which data can be accessed efficiently. This might consist of creating different indices to the data stored in a database. So in general the design of software consists of designing the processing component as well as designing the data component.

These two components may be designed separately or they may merge into a single design dimension when we use object oriented technology. Usually large complex software will be decomposed into what we may call partitions or layers. We do not implement large software as a single module or a routine. It consists of different segments or partitions as shown in the diagram here in front of me.

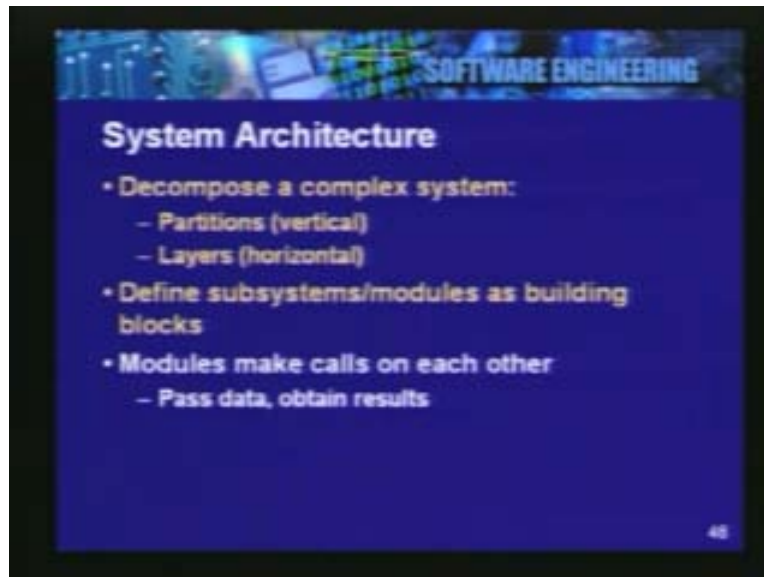
(Refer Slide Time: 33:13)



As you see here the overall software represented by a boundary as a rectangle consists of layers in this case. And these layers have different functions and they pass messages or they make calls on features provided by a lower level in order to carry out their tasks. Similarly you may also decompose the software into partitions, where each partition carries out a specific responsibility. So the overall complex software is built by decomposing it into partitions and layers. And each partition and layer is given a specific responsibility. We define different subsystems at the software level. We also define modules; in fact modules are the building blocks of the overall software. Each module has a specific function to perform. Module may be a piece of code, which when executed carries out a specific task. Such modules together make a subsystem. Multiple subsystems like this will make up the overall system.

So a subsystem may be representing a partition or may be representing a layer. So this vertical and horizontal decomposition of the overall software is necessary in order to divide it and to break it into realizable components. These modules will naturally make a call on each other; they will pass data and collect results.

(Refer Slide Time: 34:47)

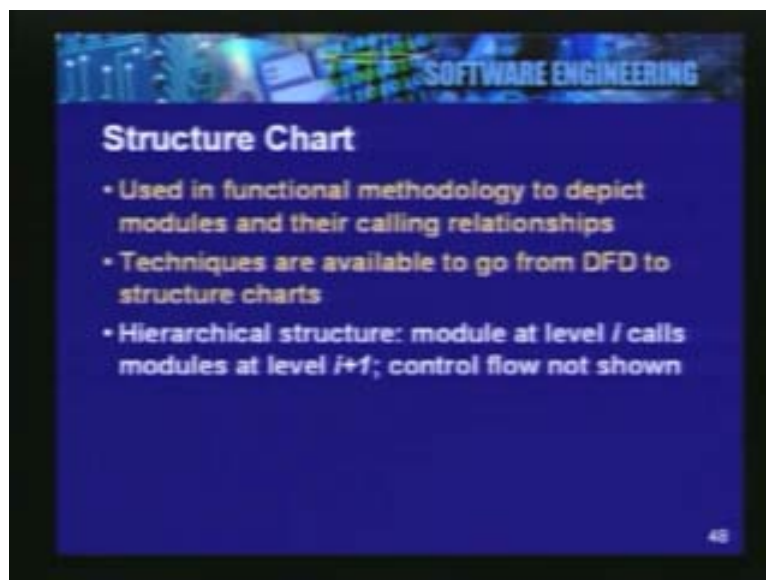


So module is a building block and the different modules are interrelated in terms of execution. A module may invoke another module and pass certain data to this module and expect certain computations. For example, a module may solve the data in an ascending order or it may compute the tax for a given employee. Now this module expects certain data to be given as input and it produces results. Such a module will be called from another module. So we are now talking about the execution architecture of given software which is defined in terms of the modules and the calls that they make on each other. Now these modules are important elements as we just now said that these are the building blocks. Good software architecture should make these modules as independent as possible. An independent module is one which has a clear responsibility, which has a well defined task to do. It does its function, which is a cohesive kind of a function.

So modules should be independent. They should have minimum interdependence. These characteristics are important from the maintenance point of view. This is a basis for handling complex tasks. Complex tasks must be broken into sub units. These sub units must be cohesive. They should have little interference from other modules. There should be minimum interdependences, so that a module can be replaced by an equivalent module without disturbing the overall functioning, and this replacement may be done for various reasons during the maintenance. We decompose the modules or subsystems until we reach units which are implementable, which can be coded and tested. The software architecture that we tried to define in terms of modules and interrelationships among them can be captured through a notation or a diagramming tool called structure chart. Structure chart is used in functional methodology.

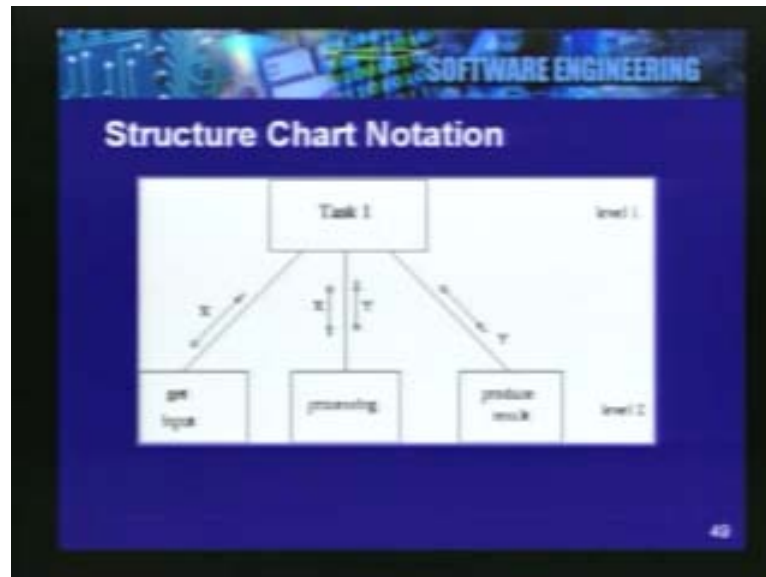
It depicts different modules and their calling relationships. There are techniques available to arrive at such a design consisting of modules and calls among them and produce a structure chart. There are techniques available to go from data flow diagrams to structure charts. A structure chart basically is a hierarchical chart. The modules can be organized at different levels. So that module at level I calls the modules at level $I + 1$. A hierarchically structured set of modules, have modules organized at different levels and the control flow is not shown, only the calling relationships is shown. Here we are putting some kind of a discipline in the software architecture that we are designing. The architecture will have a hierarchical structure, and modules at one level will call modules only at the next level and there will not be arbitrary calls among the modules. This ensures a systematic architecture for the software and is a very important design guideline. Here is a diagram which shows an example for structured chart.

(Refer Slide Time: 38:40)



As you see, we have 'Task one' which is shown as a module at the top. This task has been written as a software module or a unit. It calls on three modules at the next level. These three modules have been named as, "get input", "processing" and "produce output". The lines which are connecting these modules basically represent the call that the module Task 1 makes to "get input" or "processing" in order to perform its own job. So the structure chart basically shows modules as rectangles and links among them representing the call relationships. These modules call each other to perform their own overall tasks. As you see here the Task 1 invokes the get input module.

(Refer Slide Time: 38:55)



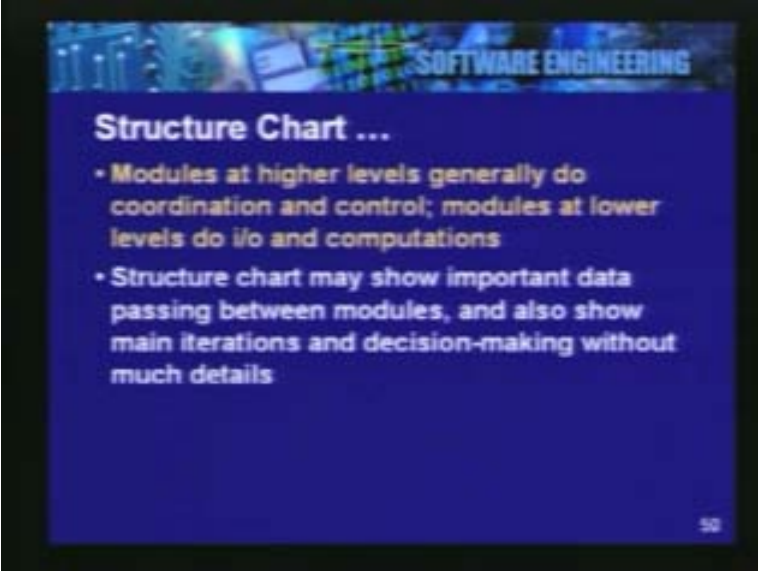
There is a data item named *x* which is flowing from “get input” and going towards Task 1. Basically here, what it implies is that the Task 1 needs this *x* and it has given this responsibility to the “get input” module. So it calls the “get input” module and receives from this model. The data item *x* which it may then pass to the “processing” module. As you see here, the *x* data item is traveling from Task 1 to “processing”. The *y* data item is actually being produced as a result from the “processing” and it is being sent back to the Task 1. The Task 1 is then passing this *y* to the “produce result” module.

So in general the modules which call each other may have some data items flowing from the calling module to the called module or there may be some results which are going back to the calling module. This is the data which flows as parameters between the modules when one module executes or invokes another module. As you see here again the modules are organized at different levels. The Task 1 is a module at level one and the other three modules are at level two. The module at level one, which is the Task 1, is the one which invokes or which calls the modules at the next level and not the other way round. This is the hierarchical structure that we enforce on the software design. Software design then therefore consists of identifying these modules and the hierarchy of the software. This hierarchy represents the architecture of the software.

If necessary the modules on the level two such as the “processing” module can be decomposed further. So the hierarchy can consist of multiple levels. Depending on the complexity we can keep decomposing till we arrive at a module which is clearly specified, and which can be implemented in terms of a coding “code module” which can be converted into a program. The modules at higher level generally do the coordination and control, whereas the modules at the lower levels do input/output and computations. This is exactly what we saw in the previous diagram. The module at the higher level which was the Task 1 is the one which gets work done from the modules at lower level. So it obtains inputs, it obtains computational results and it passes them from one module to another module. So the task of coordination and control usually

gets done at the modules at the higher level, whereas the modules at the lower level you can think of them as worker modules. They actually perform the tasks such as the input/output tasks and the computational tasks.

(Refer Slide Time: 43:25)



SOFTWARE ENGINEERING

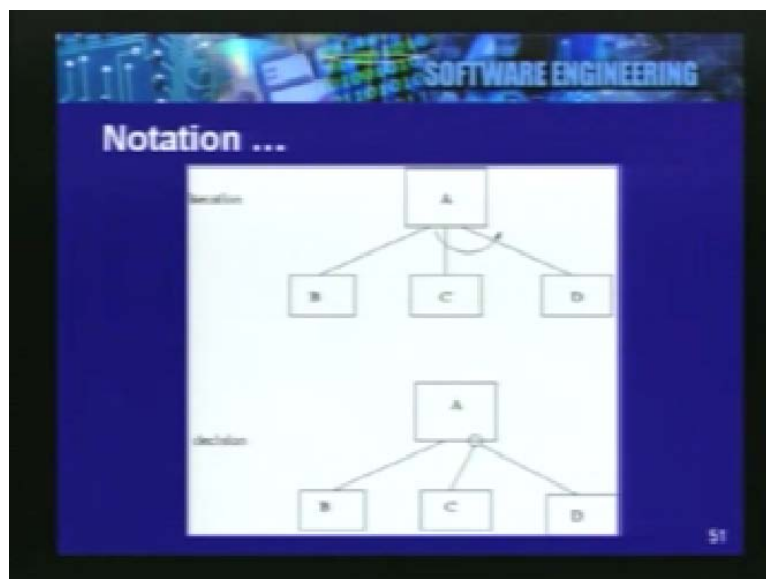
Structure Chart ...

- Modules at higher levels generally do coordination and control; modules at lower levels do i/o and computations
- Structure chart may show important data passing between modules, and also show main iterations and decision-making without much details

50

Structure chart also shows important data. As we just now saw these data are shown as flowing between the different modules. There may be either data coming into the module or the data going out from the module to the calling module. Usually you do not show the control flow in the structure chart. However some important control flow or structures such as decision making and iterations may also be shown on a structure chart without giving too many details.

(Refer Slide Time: 44:21)

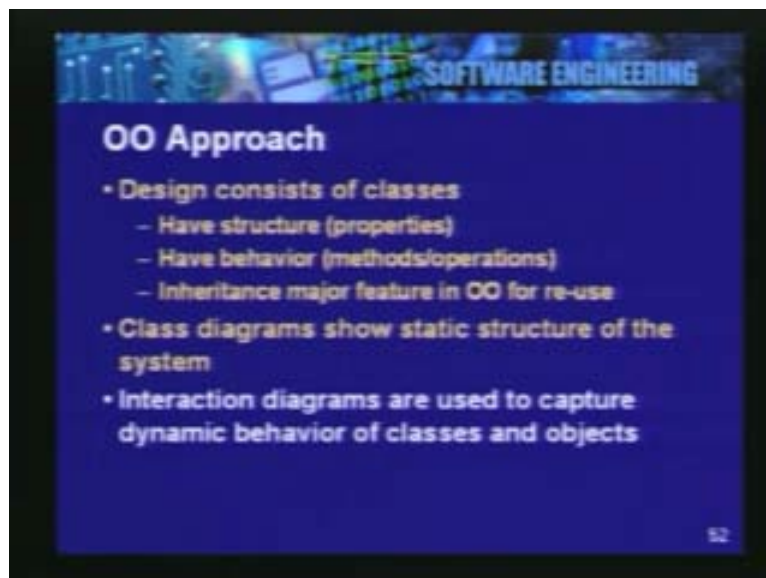


In fact this is only to indicate to the reader of the structure chart, that some modules may be repeatedly executed, whereas some modules may be executed only depending on certain condition. The few notational extensions are made to the structure chart as shown in the diagram. Here the diagram at the top shows iteration. It shows that module A may be repeatedly call on modules C and D. The diagram at the bottom has a small diamond here which indicates that module A will call module C or D depending on some condition. In fact the details of the condition are details of how often the iteration would be done are not given on the structure chart. It only shows that such an iteration or decision making is present in the software. And these are the important repetitions or iterations and the important decisions which are made in the software functions.

On the object oriented approach the software consists of different classes. In fact class is the main concept in the object oriented approach, where it has both the data associated with the class and the executable functions associated with the class. So we call these as structure and behavior for a class. This is a paradigm which combines the data and the processing to get them on a single dimension and identifies classes which have structural properties and which has a behavior or methods defined for them. It is a completely different paradigm and we will see more details of this subsequently.

Object oriented paradigm is very useful and has occupied very important position in the design of software. And it simplifies greatly, the complex software development. It has not only a new notion of a class as a unit for organizing the processing as well as the structure, but it provides a concept of inheritance which allows us to reuse the software from other existing components. We use class diagrams to show the static structure of the system. The class diagram identifies what are these different class components.

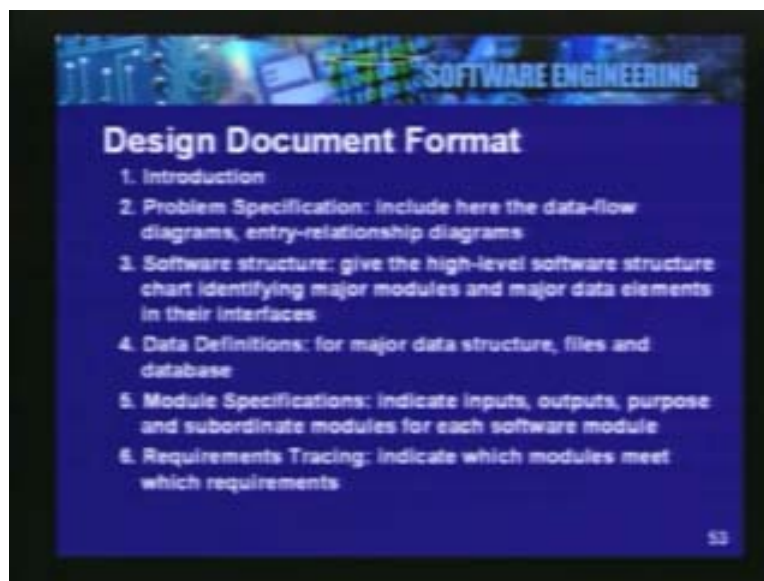
(Refer Slide Time: 47:02)



And besides the class diagram, we use interaction diagram which indicate the dynamic behavior of the system in terms of classes interacting with each other. So we will see this object oriented approach subsequently in much more details. The only point we want to make here is that there are different ways in which the design can be approached. It could be the conventional approach where the architecture consists of functional modules depicted in the form of a structure chart or it could be an object oriented approach. Object oriented approach will decompose also large system into various packages. So the principles are similar that you use the decomposition technique into convert the large software into smaller components these components may be modules or they may be classes and we organize the data and processing among these.

Once we have completed the design and also have finalized the approach we can prepare our design document which also has a well defined structure. The design document will generally include these different sections which will specify the design completely. We start by giving an overall introduction when we define the problem for which the design has been prepared. If necessary we indicate here the different such as data flow diagrams, entity relationship diagrams or class diagrams which have been prepared by us and we enclose them as appropriate annexure. Then we define the software structure in terms of the architectural diagrams. We define the data in terms of various database structures, the files or any other important data structures.

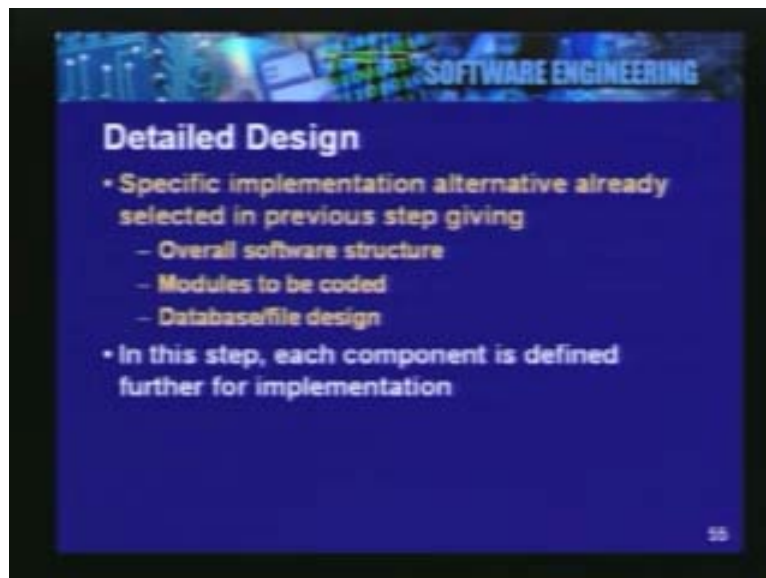
(Refer Slide Time: 49:03)



We give the module specification which indicates what input they take, what processing they perform, and what the subordinate modules they use. Finally we do the tracing of the requirement. We indicate how these different modules meet the various requirements that have been identified in the SRS document. So the design document is fairly a technical document. It consists of our approach towards building the software. We have identified the components which will be implemented as part of the overall software. This document needs to be reviewed by the technical people who will see that the document is detailed, it is comprehensive, it is complete and also that it can be traced and can be mapped to the SRS document. We ensure through this requirement tracing, that the design document covers all the functions which were

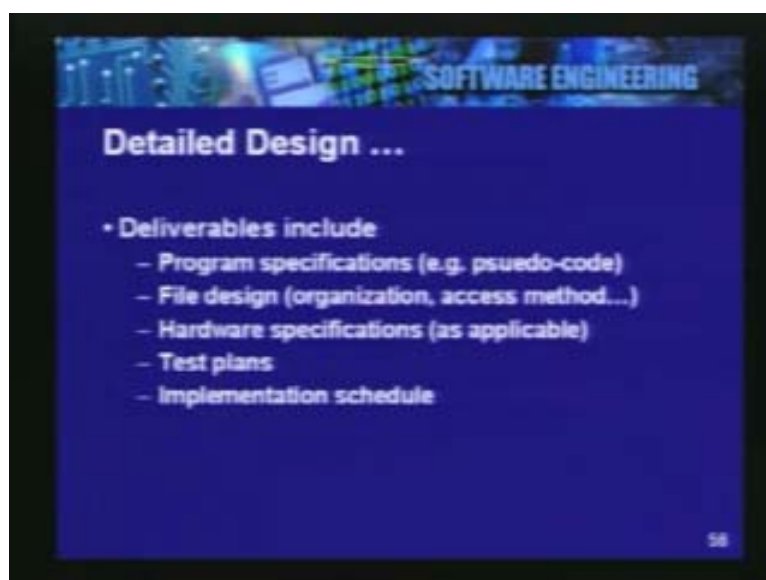
identified in the SRS document. After doing the overall design or the high level design, we perform the detailed design. In the detailed design, we go towards details of making these modules implementable. We give the inputs which have been prepared from the previous design document are made available, which consists of software architecture, the different modules and the different database and file design.

(Refer Slide Time: 50:50)



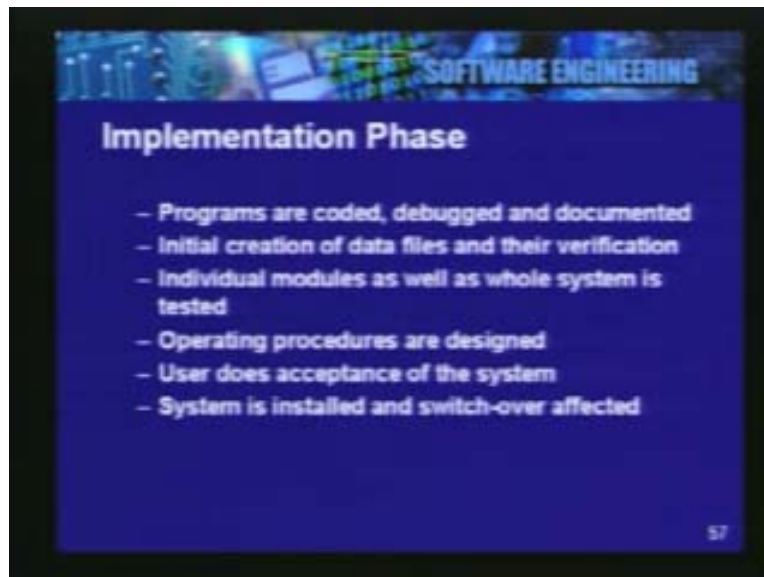
In this detailed design step, each component will be further refined, so that we make it ready for implementation. Here the deliverables will include detailed program specification, where we will indicate the logic or the algorithm for different modules.

(Refer Slide Time: 51:19)



We will give the file and the database design, we will give the hardware specifications if they are applicable, we will make detailed test plans and also we will give the implementation schedule. So these are the deliverables of the detailed design. And the input to the detailed design is the design document which comes from the first high level design step. This detailed design document will be prepared giving all these specifications and it will end in a technical review. Technical review at this point consists of walking through the different specifications and ensuring that the specifications are complete for us to begin implementation. We go to the implementation phase which naturally consists of coding the programs, testing them, documenting them. After the code has been prepared, we also have the responsibility of converting the existing data which may be in a manual system and creating initial file and database. We prepare the operating procedures for the users. We do the overall testing, where not only individual modules are tested but the whole system is tested. Then we carry out the user acceptance, where the user will do a functional walk through for the system that we are ready to deliver to him.

(Refer Slide Time: 52:34)



So functionally it should be acceptable to the user. Moreover it should also be acceptable in terms of performance. In a SRS document the performance criteria were also specified. So at this point the SRS document will be used by the user to carry out the final acceptance of the system. Once the user has accepted the system we are ready to switch over. The system can be installed. And once the system is installed we are ready to use the software in day-to-day operations. The next phase which is the operations and maintenance phase, we will ensure that the system will continue to meet the user needs. And in order to do this we may have to take up maintenance activities. The maintenance activity consists of, removing any errors that the user may find in the software, or extending the present features in the software for new requirements that the user may have. So we might make some extensions or we may even add entirely new features.

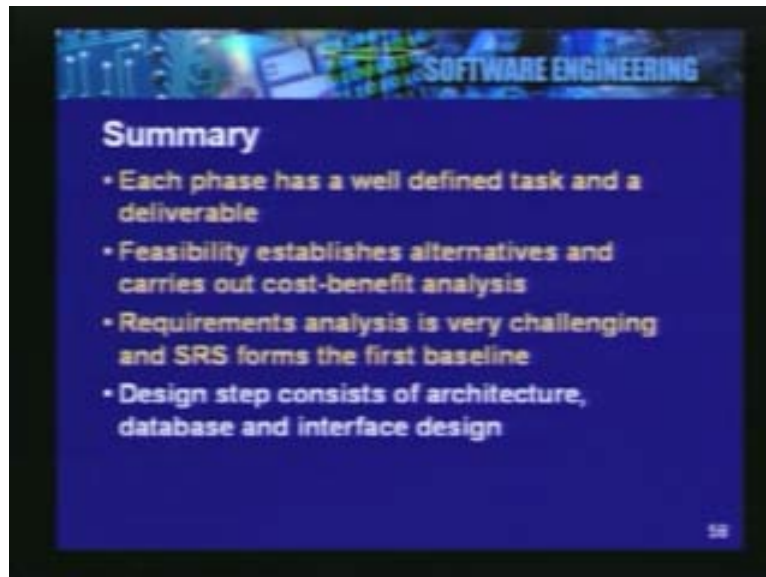
Occasionally it may also be necessary to convert the software to run on new platforms or new systems, because the software might have a fairly long life and during this time the hardware or the systems may change significantly and we may want to change the platform of the software. So these are the maintenance activities and this generally continues throughout the lifecycle.

(Refer Slide Time: 54:13)



Let us summarize the various phases we have carried out. We have seen that there are different phases and each phase has a well-defined task and has a well-defined deliverables. In many cases these deliverables have a standard format. Now we have seen that feasibility was one of the important steps. It establishes different alternatives and these alternatives are analyzed for the cost effectiveness. Requirement analysis is a very important step and also very challenging where the analyst or the developer has to prepare the complete specifications for the software by meeting the users and by going through the existing system if necessary. So this is going to be the first baseline. And we saw that there is an IEEE format which is very comprehensive and which defines the various aspects of the software which needs to be developed.

(Refer Slide Time: 55:32)



Then we saw the design step. The design step consists of designing the architecture of the software, also designing the databases. And naturally it will also consist of designing the various interfaces that the users have with the software. In the rest of the course on software engineering, we will take you through some of the details of the design techniques. Similarly we will take you through details of some of the modeling techniques. In this particular lecture, we have only given a broad summary of different phases, through which the different steps are carried out in different phases of the life cycle. And we saw that each phase ends in a technical or management review.