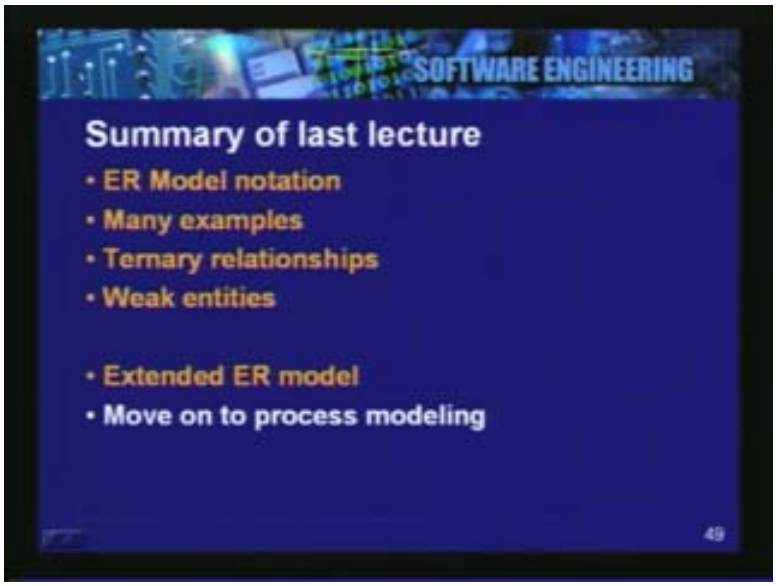
# Software Engineering

**Prof .N.L. Sarda**
**Computer Science & Engineering**
**Indian Institute of Technology, Bombay**

**Lecture - 11**
**Data Modeling- ER diagrams,**
**Mapping to relational model**
**(Part -1)**

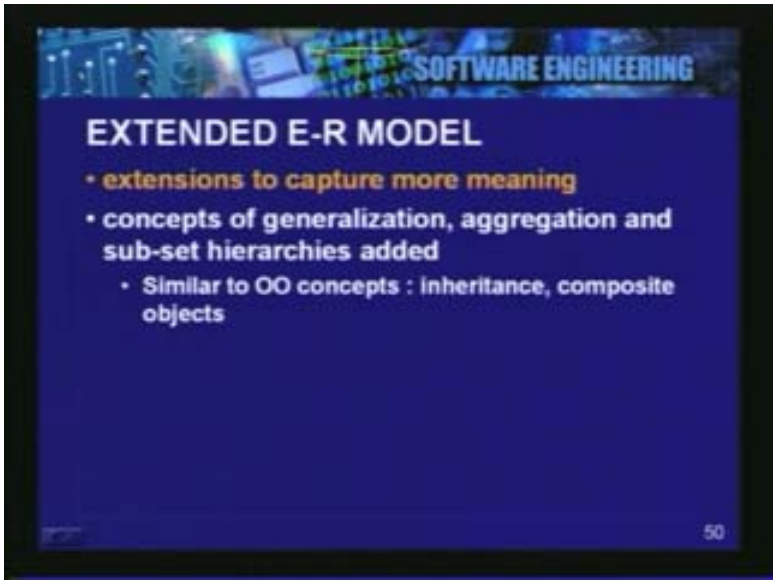
Let us continue with our module on data modeling. Let us summarize what we have seen so far. We have seen the ER model notation. Of course, before ER model notation, we had seen that basic concepts of entity and relation ship. We saw the different types of symbols used for representing different concepts.

(Refer Slide Time: 02:15)



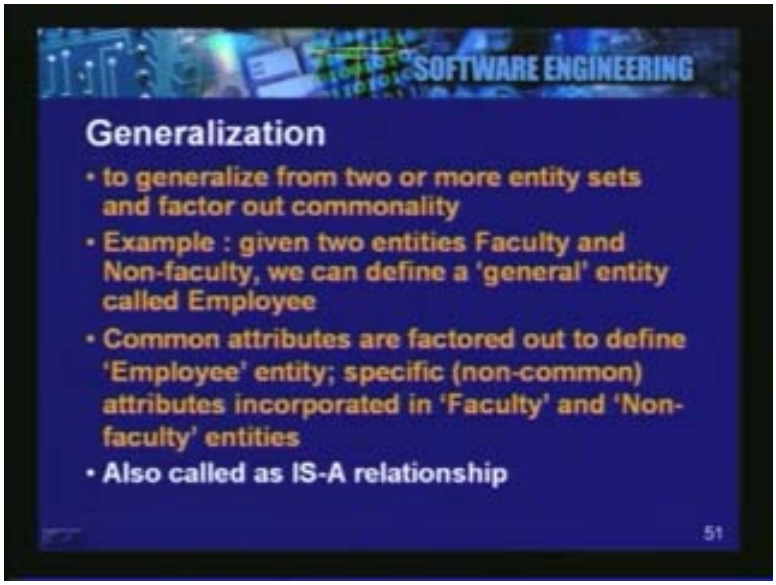

We saw many examples, and then especially we discussed ternary relationship and how we have to take adequate care in modeling complex relationships we saw that ternary relationships cannot be replaced by two binary relationships so they must be properly understood, properly analyzed, and also verified that our model of a relationship is correct. We then saw the concept of weak entities and we said that the weak entities are encountered quite frequently and generally they depend on some strong entities and they can be defined in context of some stronger entity. We will now continue with the extended ER model and after that we will move on to process modeling. Let us now discuss the extended ER model in which last time we had said that primarily the intention is to provide more semantics, more concepts for capturing the real world with its full meaning.

(Refer Slide Time: 02:55)




We specially have new concepts, namely the generalization, aggregation and subset hierarchies which are also called specialization. We will find that some of these concepts are quite closely related to the concepts in object oriented paradigm and they relate to inheritance in particular and also to composite objects. Let us first study the generalization.

(Refer Slide Time: 06:15)




The idea here is to find out from the entities that we have already identified whether something can be factored out as a common element. So it is possible that you might have defined two or more entities and you see that there is some common aspects to those and you would like to extract that common aspect into an entity by itself. This is called generalization.

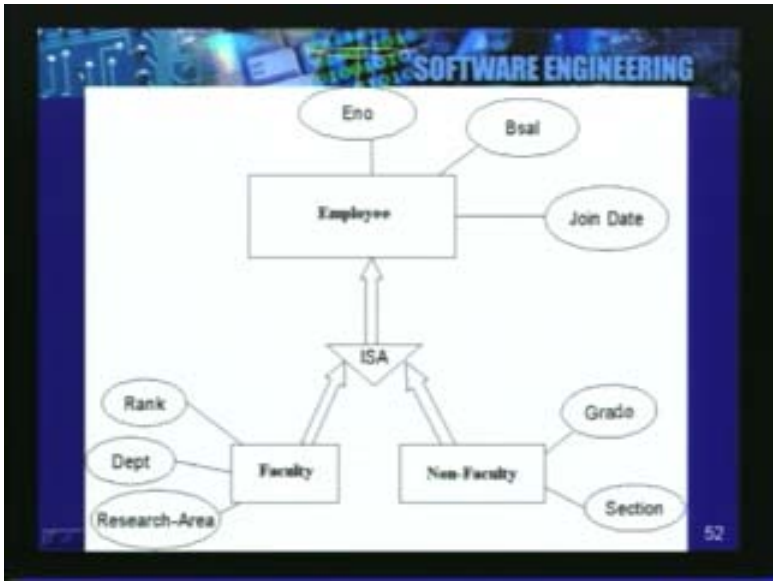
Here is an example. Suppose we have identified two entities and we have called them as faculty entity and non faculty entity. The purpose could be that faculty entity participates in taking courses. Then we also have non faculty who do not teach courses. So we have defined these 2 entities. Looking at these entities and their attributes, we find that they also have lot of common attributes and there may be some applications where the two of them are treated together. For example, a payroll application naturally will have to make salary payments to both faculty and non faculty. In that case they might need to be considered together.

So we find that these two entities have some common elements and we would like to generalize these two and create a new entity possibly called employee. Hence we say that employee is a generalization of the two entities: faculty and non-faculty. We started by saying that these were the two entities we had identified earlier, then considering the application context we find a lot of commonness among them and that commonness we have extracted into a general entity called 'employee' entity. In this case, the common attributes which are present at both the faculty and the non faculty level are extracted and are factored out. These common attributes are treated as the attributes of the general entity employee.

These attributes for example could be the employee identification, employee name, date of joining, basic pay rate. These are attributes which will apply to both faculty as well as non faculty. These can be extracted and defined at the level of the employee itself. There may be some attributes which are applicable to faculty, but not applicable to non faculty and vice versa. These specific or non common attributes are left with the original entities. The common attributes are extracted and defined at the general level and the specific attributes are left with the specific entities.

Such a generalization is also known as IS-A relationship. We would have a notation which is somewhat similar to notation of relationship, but here the relationships would be named as IS-A relationship and we will see some examples of that. Here is the example we were discussing where we started by saying that we have the faculty entity and we have the non-faculty entity. They had some common attributes. These common attributes have been extracted and we have created a general entity, a more generalized entity called employee and the common attributes have been identified. The common attributes include the employee number which is the key attribute itself, and then we have the basic salary and the joining date. These are applicable to both the faculty as well as non faculty therefore they have been pulled out here.

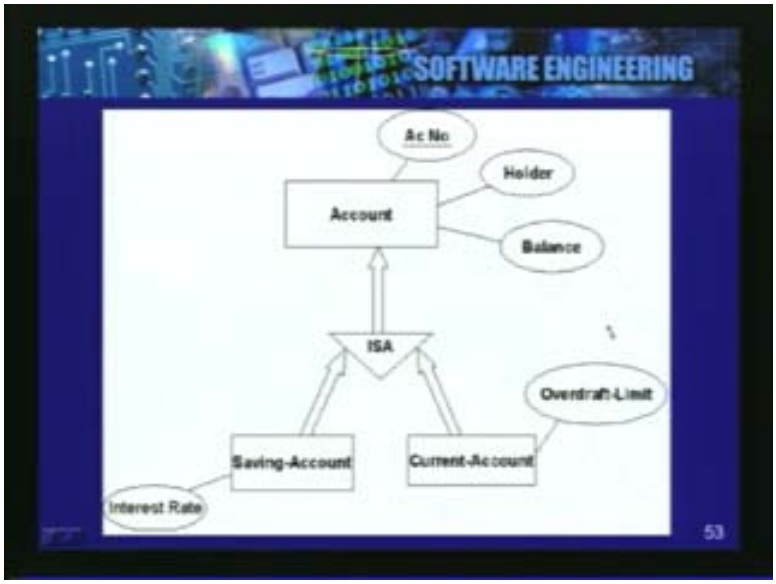
(Refer Slide Time: 06:28)




Non common attributes are left with the entities. Faculty has attributes such as rank, department research area etc, whereas non faculty has rank, grade and the section in which they work. The fact that we have started with these entities: faculty and non faculty, and from here we have extracted and we have created the employee as a more general entity, is indicated by the IS-A relationship between these entities. The direction of arrows indicates the direction of generalization. So we have generalized and created the employee entity from these two entities. Note here that IS-A relationship is shown as a triangle with the words IS-A written and the direction of arrows are from more specific entities to the more generalized entity. These are the notations that we use for generalization. Here is one more example. We have two types of accounts: A saving account and current account. Many of you would know that these are accounts which you can open in the banks.

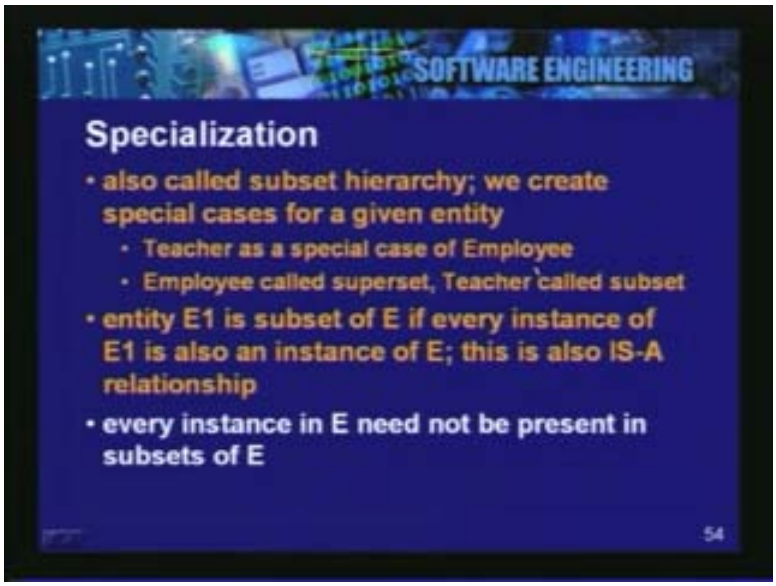
These two accounts have a few different attributes and also there are few attributes which are common. The common attributes have been extracted into a general entity called account and the account has attributes: account number, the holder of the account and the balance in the account. Whereas the nonspecific attributes or the non common attributes which are specific to these entities have been left here. Saving accounts has the ‘interest rate’ attribute and the current account has the ‘overdraft limit’ attribute. These are the specific attributes and the common attributes are defined at the level of the generic entity.

(Refer Slide Time: 08:21)




Again here you see that we have named the relationship IS-A and you can read it in this way - Savings account IS-A account, current account IS-A account. That is why the direction of reading is implied and the IS-A relationship reads in this way that we are saying that a specific entity such as the saving account is also an account. This is the generalization example. Let us consider the specialization example which is the reverse of generalization. This is also called subset hierarchy. Here we start with an entity and then we find that we can create some special cases of that entity. So that we can prepare better ER diagram and we can indicate the roles that that was special entities play.

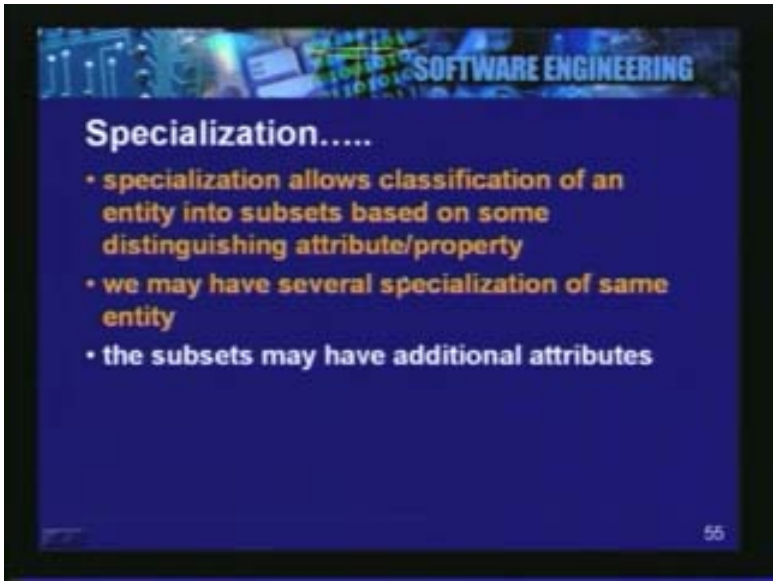
(Refer Slide Time: 12:39)



Here is an example. Let us assume that we have created the employee entity and we find that there are certain types of employees whom we may want to distinguish. So we create another special case of the employee entity called teacher. Here we begin with employee then we find that are some special employees who need to be treated special differently in our model. The reason for separating teachers from other employees and treat them as a special case is, we may want to define the teaching relationship where only teachers teach courses. We started by identifying employee as an entity in our real world and then we created teacher as a special case. Here the employee is called superset and teacher is called subset or they are also called super type and sub type.

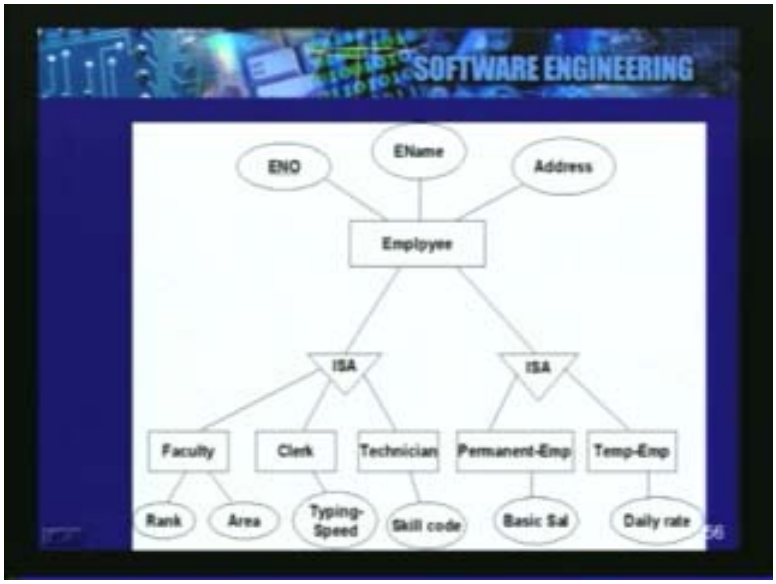
In general an entity E1 is called as a subset of some other entity E, if every instance of E1 is also an instance of E. we also call this relationship as a IS-A relationship. This means every teacher has to be an employee. That is every instance of teacher is also an instance of employee and there cannot be a teacher who is not an employee. Of course the other case may hold that there may be employees who are not teachers. So the reverse is not implied. Every entity E1 which is a subset, every occurrence of such entity E1 must also be an occurrence of E. Every instance of E on the other hand need not be subset of E. Teacher is a special case of employee. Being a special case that there may be more employees who are not teachers but every teacher has to be an employee. This is a precise meaning of specialization. Specialization allows classification of an entity in to different subsets based on some important attributes. It allows us to categorize a given entity into different types or different classes.

(Refer Slide Time: 13:30)




We may have in fact, several classifications or several specializations of the same entity based on different attributes. The subsets may have additional attributes. For example, we create teacher as subset of employee and we may want to give some additional attributes to teacher which do not apply to other employees. Here is the example of the specialization. Remember that here we start our modeling by identifying the employee as an entity.

(Refer Slide Time: 13:35)




Then we considered in the further modeling that different employees need to be treated differently. So we identify the attributes which define that basis. Here we have two categorizations. It is possible that employee is treated differently for payroll depending on whether they are permanent employees or temporary employees. So we have a specialization of employee into two categories: Permanent employee and Temporary employee. An attribute such as basic salary may apply to permanent employee whereas daily rate may pertain to temporary employee. There are some common attributes to both types of employees: Employee number employee name and the address.

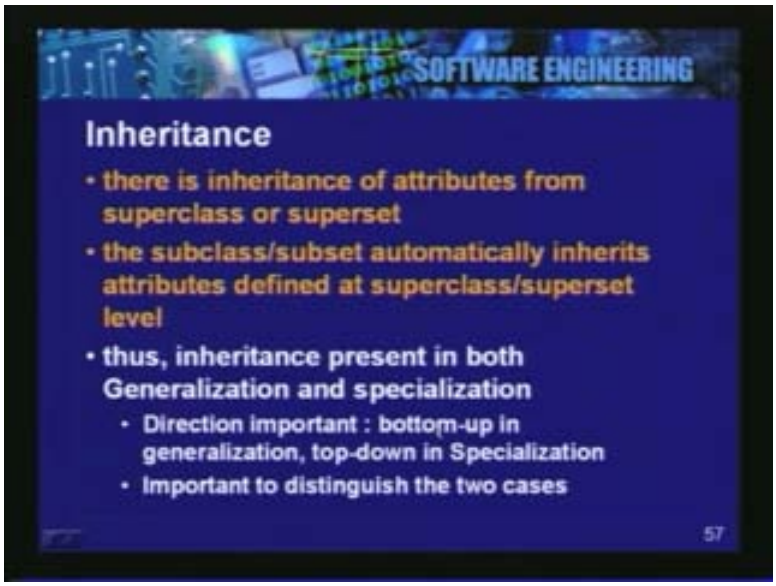
This is one way of classifying employees. The other way could be based on their function based on the type of work they do. Here we have created another specialization into categories called faculty, clerk and technician. We have also identified few attributes. For faculty we have rank and area of research, for clerk we may have typing speed and for technician we may have skill code. Always remember that we do specialization or generalization with some objective in mind. It is possible that you would define a relationship between the faculty and a course called 'teach' relationship.

It is best to create the teach relationship with faculty and not with the employee because there are some employees who do not teach. The purpose of this specialization therefore is to indicate additional behavior or some specific behavior of these specialized entities and you also identify attributes which are meaningful.

Remember that what we said earlier that every employee need not fall into the categories given here every faculty obviously IS-A employee. This is the way you read the relationship 'faculty IS-A employee'. But in the case of specialization every employee need not be in the categories that you have defined. We may have an employee who is neither faculty nor clerk nor technician. He has a role which is not relevant for our modeling. We have not identified additional roles. It is possible that only these three roles are meaningful to us in the model that we are creating.

Where as in this case we have some employees who are permanent employees and some who are temporary, it is possible that these two categories cover all the employees and there are no employees who are left out. Because they are the binary classification, either it has to be permanent or it has to be temporary. There is nothing other than these two and hence this classification is complete. Specialization is going from an entity and creating special categories. Whereas generalization is going from a given set of entities, finding out what is common, then extracting it and defining that as a general kind of an entity. Note the difference in the notation also in the case of generalization. We had arrows pointing towards general entity and here we have the classification or specialization going from top to down direction. Let us examine these two concept generalization and specialization in the context of a concept in object orientation, the concept of inheritance.

(Refer Slide Time: 19:45)




How are these two concepts related with inheritance? Obviously we see that there is an inheritance implied here from the super class or the generalized class to the special cases. Because the common attributes are defined at the general level and the specific attributes are defined at a specialized level. Naturally the attributes at the generic level are inherited by the entities which are at the next level. So there is an inheritance, the attributes are inherited by the more specialized entities from the base entity.

The sub class or sub set automatically inherits the attributes defined at the super class and super set level. There is an inheritance in both the cases whether it is a generalization or whether it is a specialization. The attributes of the generic entity are inherited by the subclass or the subset entity. Hence inheritance is present in both the cases. However, the direction of the inheritance or the direction in which we have created the model is an important difference. That is the reason we have distinguished generalization from specialization. We do generalization in the ‘bottom up’ direction, we have gone from more specialized entities and created a general entity, whereas in the case of specialization we started with an entity and created specific cases of that entity.
It is important to distinguish between these two cases.

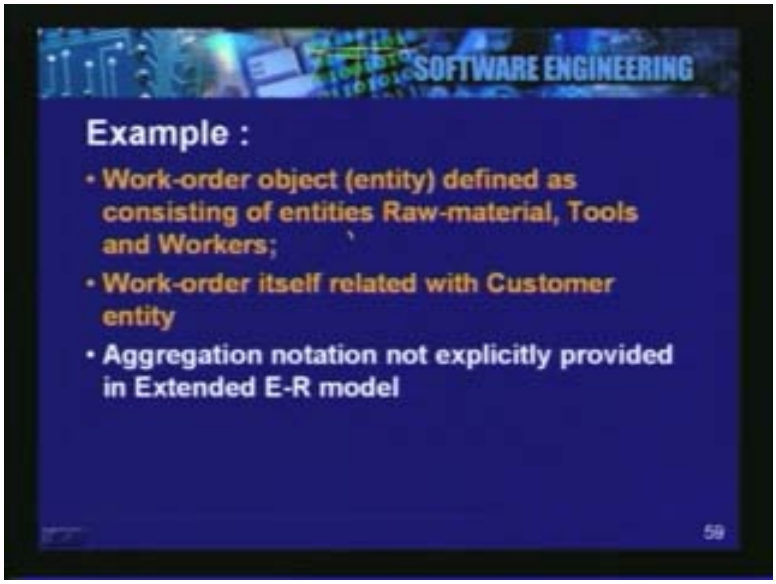
Let us next examine the concept of aggregation. Aggregation is used for building complex entities from existing entities or possibly existing entities and relationships between them. We are creating a complex entity and that would be called as an aggregated entity. There are two ways by which we can do that. We can create an attribute whose value itself is another entity. Usually we have talked about the attributes and we said that attributes have values and so far we have seen attributes which are primarily simple attributes like age or roll number or hostel number. But there can be attributes whose value is itself an entity.

(Refer Slide Time: 21:50)


For example, when student study a course and let us say he has course attribute. We can say course attribute value is CS 101, which is the course code but that is not the same thing as saying course as a value. When we say that the whole course itself is a value then we say that the attribute in this case is an attribute whose value is another entity, the course entity. In this case we are creating a complex entity with values as entities themselves. The second example is when we create an entity from an existing set of entities which you may already have identified. So you club them together and create a more complex entity. We will see an example of the second case.

Consider the example of the work order. Work order is something which will be received by a company or an organization and it is supposed to manufacture that particular item for a given customer. When a work order is received, the work order will need to be translated into some raw materials and tools which will be worked upon by the workers to create that particular piece of product. A work order is primarily represented in terms of three basic entities: the raw materials, tools and workers. It is really a complex entity and that complex entity is formed by a relationship between raw material, tool and workers.

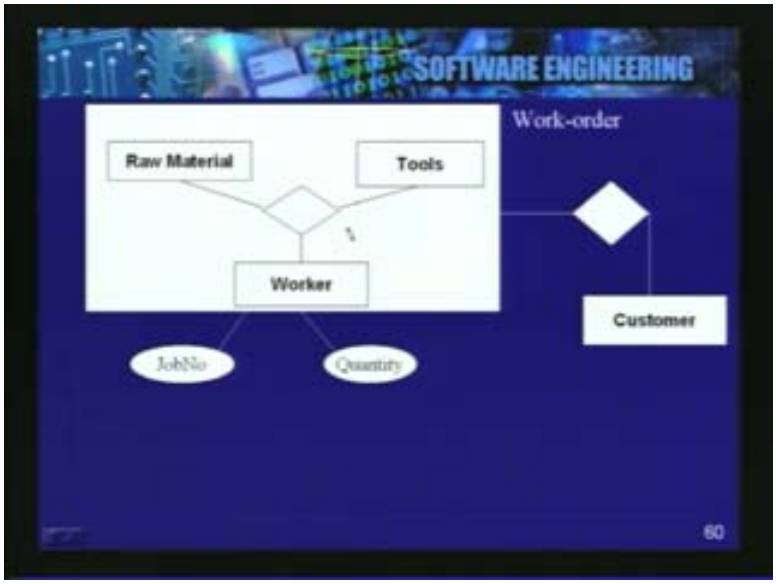
(Refer Slide Time 23:30)




On the other hand, the work order itself has to be treated as an entity because it is related to customer entity. Customer would be shown as having a relationship, a binary relationship with the work order. The work order itself will get decomposed into entities showing its relationship to raw material, tools and workers. Work order also may have some useful attributes for itself. Like the work order number or the quantity in which the customer has ordered that particular product. Now, we want to show this in the form of an ER diagram or some suitable notation.

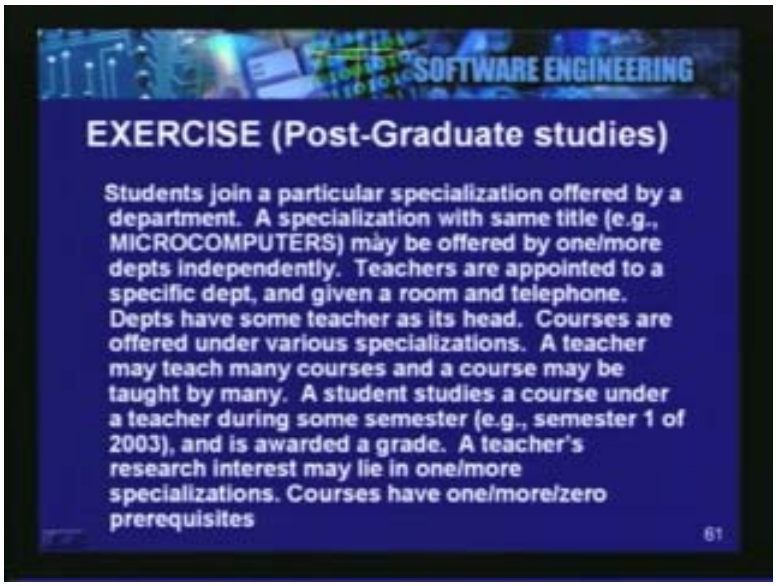
Unfortunately there is not any standard notation provided for this. We will see an example which we can use for representing aggregation. Here is the work order which is represented as a large rectangle. That large rectangle itself consists of a small ER diagram on its own, defining the relationship between the raw material entity, the tools entity and the worker entity. This complex object has two attributes job number and quantity note that these attributes are connected to the outer rectangle. They are not connected to these inner entities which forms the complex or the aggregate entity.

(Refer Slide Time 23:41)




This work order is related to customer through a binary relationship. In this example we are showing an aggregate entity like this which can further participate in additional relationships and which has its own attributes. Let us now take a more complex and complete exercise. There is a statement of the problem given here. It is concerned with the post graduate studies. I will show you a very concise description of problem in this case.

(Refer Slide Time 25:30)



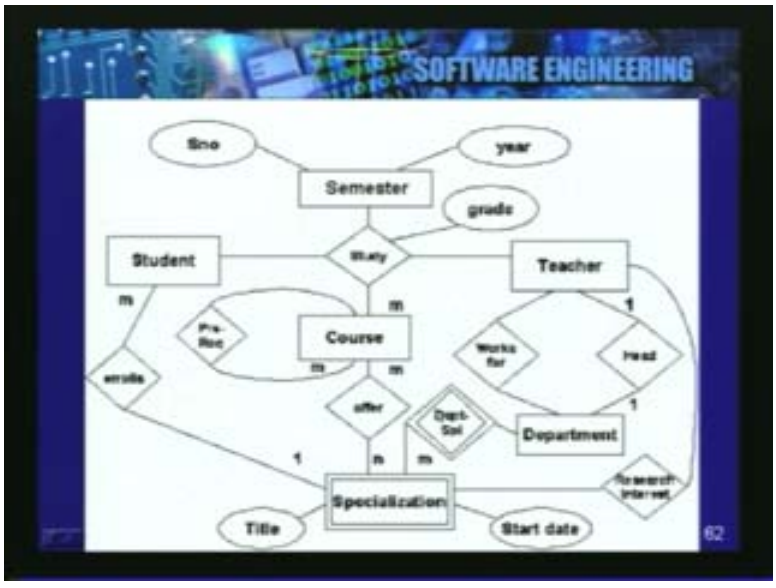

You should remember that this concise description has been prepared possibly as a result of analysis that we must have done during the analysis phase and we condensate in the form of a brief statement so that we can analyze it for identifying the required data model.

This should be read carefully and should be understood. Since it comes from a university environment, I expect all of you would be somewhat familiar with this. We have students who join a particular specialization that may be offered by a department. We have an academic institute such as IIT, which has many departments and these departments offer specialization at the post graduate level. “A specialization with same title may be offered by different department independently. Teachers are appointed to specific department and they have a room and telephone number. Departments have some teacher as its head. Courses are offered under various specializations. A teacher may teach many courses and a course may be taught by many teachers at post graduate level. This is quite common that a course is taught jointly.

A student studies a course under a teacher during some semester and these semesters are distinguished from each other. We have a semester 1 of 2003 and then there may be semester 2 and so on. These semesters are also properly identified and the students are given a grade in every course. Teachers have research interest, and this research interest may lie in one or more specializations. Courses may have zero or more prerequisites.”

This is the general statement of the problem. This is to some extent readymade for preparing an ER model. Preparing such concise description is always very challenging and needs a good understanding of the problem domain. After you have done analysis phase, you develop this understanding, you put it in the form of a concise description, so that it can be converted into a data model. You should try doing this exercise yourself. However we will show you a solution here and hopefully the solution that you come up with, should be somewhat similar. Here is the solution and I will point out some basic entities and relationships which you can check with the description that we have just now seen.

(Refer Slide Time 27:55)



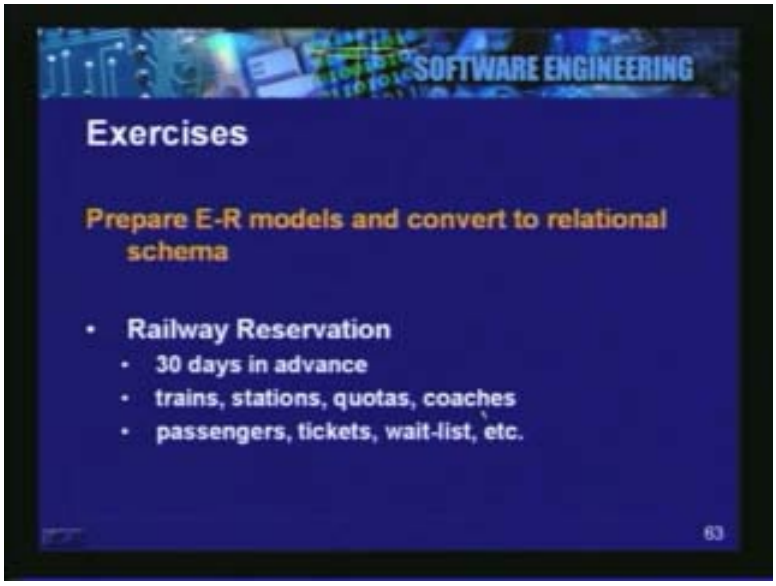
In this diagram, we have a department entity and departments offer specializations.

- Specializations here are shown as weak entities which depend on the departments. This will take care of specializations which are named similarly. We will now know that in what context it can be treated on its own. For example, a specialization such as micro computers may be offered by electrical department which is different from same specialization offered by let us say computer science department. The weak entity would nicely capture this requirement.
- Then we have teacher entity. Teachers who are working for departments and one teacher may also head the department which is shown as the different relationship.

- These teachers also have research interest. So we have a direct relationship from teacher to specialization and this naturally would be one to many type of relationship. It will permit teachers to have many specializations possibly across departments.
- We have course entity. Courses which are offered under different specialization.
- We have student entity.
- We have created semester as the entity.
- The study relationship in this case has been defined as the four way relationship. It is between student, semester, teacher and course. And the study relationship has the grade attribute. This study relationship will permit the student to enroll in different courses which are taught by multiple teachers in different semesters and if the student fails he can possibly repeat that course in another semester. This would be permitted by our model here.
- We have a student enrolling in specific specialization.

So in this case we have we have all the main entities, we have the various relationships which are indicated in the scenario and a few important attributes are shown. I will leave you with a few more exercises where you are expected to prepare the ER models and if you have studied the database systems, you can try converting these ER models into a database design.

(Refer Slide Time 31:35)



I will just give a few examples briefly. Hopefully they are familiar situations. You would be able to expand on them and create an interesting problem where you would be able to apply the ER modeling concepts properly. The first example is a railway reservation.

- Railway reservation is where we permit reservations to be done in advance.
- We have trains that go to various stations.
- We have quotas on various trains for various stations.
- A train when it is scheduled to run has some coaches. These coaches are identified as you typically would have seen. You have sleeper coaches identified as S1, S2, and S3 so on.
- These coaches would have the passenger list associated with them. They have certain capacities against which the people make reservations.
- We have people who are booked for the various train journeys. They are given tickets. A ticket might list one or more passengers
- Tickets may be confirming ticket or there may be waitlisted tickets and so on.

These are different things that we would like to cover in the railway reservation situation. You should prepare a short description of what we would like the reservation system to handle. For example we have not listed here cancellation, but they should also be forming an important part. Based on this, then you can try identifying the entities, relationships even the generalizations and specializations if they are applicable. There is another example of an old car mart which buys and sells old cars. The cars are either purchased directly from the people and they are sold to people. All the data has to be kept.

(Refer Slide Time 34:00)


They may sell the cars either on a full payment or on installment basis. They also provide service to cars and they may do some pre sale repairs and they have commission agents who help us in buying.

Basically, this is a small second hand car dealer who would like you to develop a data model in order to develop an application. Let us take another example which many of you would naturally have lot of interest in. It is cricket database.

- In this case we would like to store data about various matches which are played between countries over a period of time.
- We have countries represented by players and the countries have teams which keep changing from time to time.
- Matches are arranged and then every match may have some result.
- We keep record of scores for each and every match. Of course the scores can be very detailed. But in this case, let us say that we would like to know for each player and in each role that he has played as bowler or batsman, the aggregate score that he achieved.

Prepare an ER model and I would like you to also prepare some sample data to validate your model. Create some instance diagrams to illustrate your ER model. You should try to do these different exercises as the data modeling is a very fundamental. Given an ER diagram done by somebody else, it is easy to read. But it is very challenging to prepare one on your own even in this popular situation such as cricket database or railway reservation system.

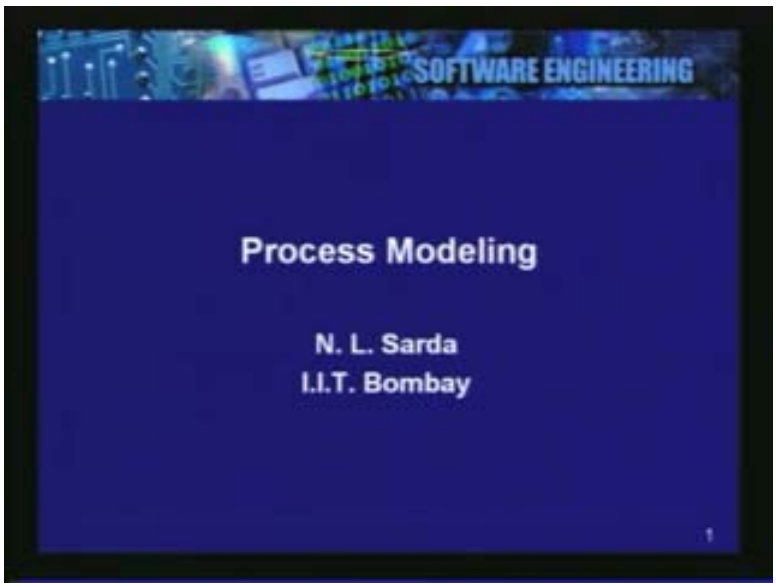
These are familiar examples and you will still find that it is quite challenging to prepare a good ER model which can be understood by somebody else without any explanation from you. In fact that is very critical test of how good your model is. You should not be required to explain to some body else what your entities mean and what relationships mean. In fact the names and the constraints that you represent in your ER model should be straight forward to understand once the person who is looking at your model is familiar with the problem you are trying to solve. So a real test of your ER model would be its understandability by a third person and of course with reference to the scope that you would have defined in a short description.

Let us now summarize. We have seen data modeling as a very important element of understanding a user application. We identify the different components and understand relationships between the concepts that we find in a given application. ER model is a conceptual model and that is what we developed. First we try to understand the data domain or the information domain of a given application at a conceptual level. At this point we are not concerned with implementing this on a computer system. The ER model uses very intuitive kind of concepts. Those are entity relationship and attribute. They are straight forward to understand and they are also straight forward to communicate with others. It also provides a number of constraints by which we can define clearly the meaning of data as it should be in a given situation. These were the concepts of primary key cardinalities of the relationship and so on.

(Refer Slide Time 37:43)

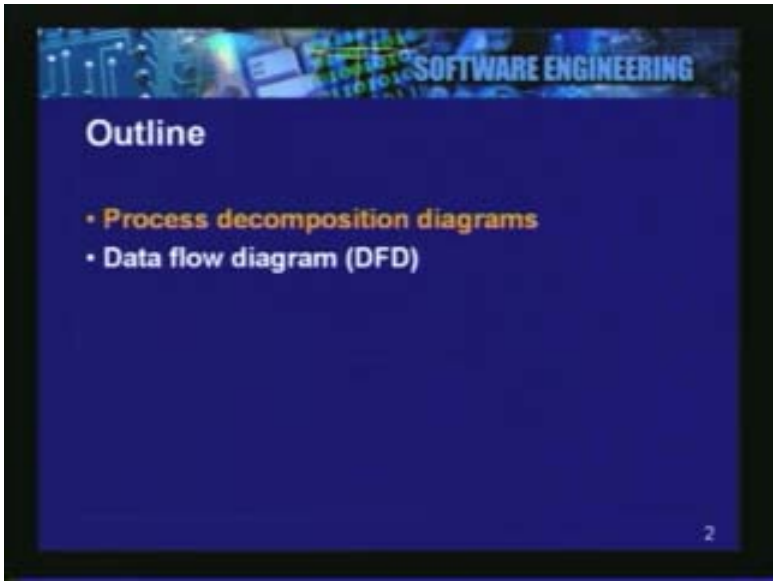

It has a very simple diagramming notation which is the strength of the ER model and it can be understood easily by the users and that would be very helpful to us in validating our model. ER model is extensively used by analyst and even by designers as a first step towards the design of a data model or a database solution. Let us next study process modeling. As we had said earlier also, there are two types of models we can really make while carrying out the analysis of the given application. They are the data modeling and the process modeling.

(Refer Slide Time 38:04)

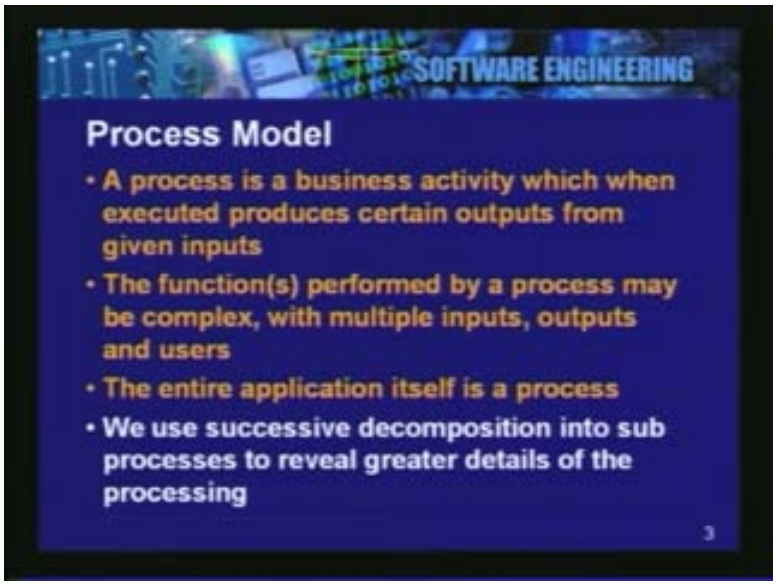


We have already seen data modeling. We will now get into the details of process modeling. We will talk about the process decomposition as one of the important step in handling complex process and we will also see a diagramming notation for representing decomposition of a process. Then we will talk about the data flow diagrams which are the important notation available for representing complex processes. In fact that would be the thirst of this module. After studying the process decomposition diagrams, we will look at data flow diagrams in details. Let us first understand the process model.

(Refer Slide Time 39:00)



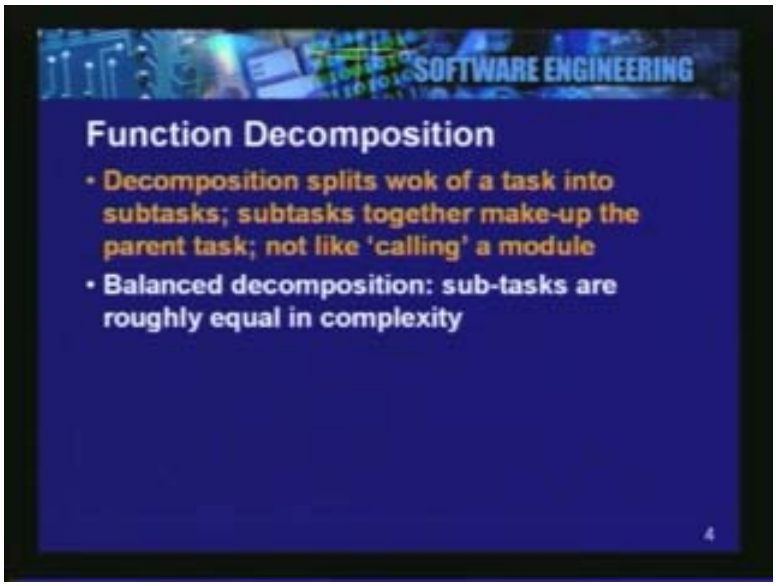

(Refer Slide Time 40:30)



A process is a business activity which when executed produces the required outputs. It takes few inputs, carries out some processing on them and produces some meaningful output. We say that the process represent some kind of a business activity. For example, the process may be for reservation of a ticket or the process may correspond to registering for a course. These are business activities and each one can be treated as a process. The function performed by a process could be quite complex and may need multiple input and may have more than one output and there may be many users involved in a given process itself. The processes can be simple to complex. The entire application which we are trying to develop can be treated as a process because it has certain inputs, certain outputs and it performs some processing.

We try to handle complexity by successively decomposing a given process in to sub processes. The main idea in the process modeling is to decompose a process into smaller processes or sub processes and in the process, we reveal more and more details of what happens inside that process. If we say that there is a process called reserving a ticket. Then we identify the sub processes in the given process. For example, checking availability on a particular date of travel is a sub process. This is called the function decomposition. Decomposition splits the work of a task into subtasks. We have a task which has some overall objective and we break that into subtasks.

(Refer Slide Time 42:30)




When we do that, then these subtasks taken together do what the parent task is required to do. So the subtasks really add up and they make up the parent task. We must distinguish this situation from the structure chart kind of a situation or programming kind of a situation where you have one program calling a subroutine or a program calling a module to do some other work. In fact the decomposition replaces a task by a collection of its subtask. This is really the splitting of a task. We generally try to do a very balanced decomposition. So when you start, you have a complex task at hand and in order to identify its subtasks, you will decompose into two or more. The question would be how many subtasks you should break it in to? Or how they match with each other in the complexity?

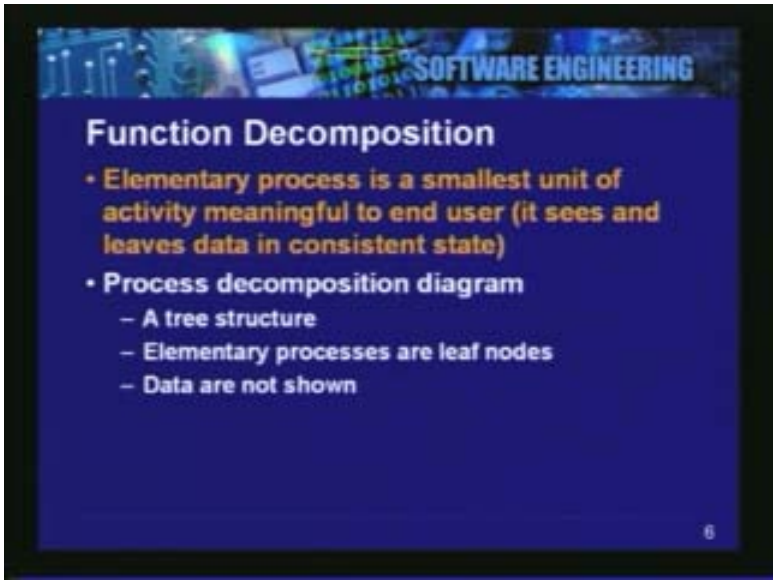
Suppose you take a task T1 and you break it into two, say T11 and T12. We roughly expect that these T 11 and T12 would be similar in complexity. It should not happen that the out of two, one does very little and all the other is expected to be done by the second. It should be a well balanced decomposition where the subtasks are broken into roughly equal complexities subtasks. We start from a given task and continue decomposition. This is the top down decomposition which produces a hierarchical structure.

(Refer Slide Time 45:01)


A hierarchical structure is a level by level structure. At the top most level you will have the single process and the next level it shows its sub processes. Then one of those sub processes may be further decomposed. This gives you a tree type of a structure or a hierarchical structure. You generally decompose into two or more. But a thumb rule is not to decompose a task in to more than five because our ability to comprehend more than five things at a time is limited. So we generally try to decompose a task into up to five subtasks, roughly of similar complexity. While doing this, the two principles which we should keep in mind are the high cohesion and minimum coupling. Each subtask should be a well defined subtask, should have precise goal and should have maximum independency. It should be as independent of the other subtask as possible. That is the principle of high cohesion.

The next is minimum coupling. That means the subtasks should have minimum interdependence among themselves. High cohesion and minimum coupling are fundamental criteria of decomposing anything complex into its subtasks. We continue this decomposition process until we identify elementary processes. These elementary processes are fairly simple and straight forward. We generally would not require any further decomposition of these. An elementary process can be defined as a smallest unit of meaningful activity. The users would consider them as a well defined smallest piece of action. In terms of relating it to database state, we can say that these elementary processes see the data in consistent state and when they are finished they leave the data in the consistent state.

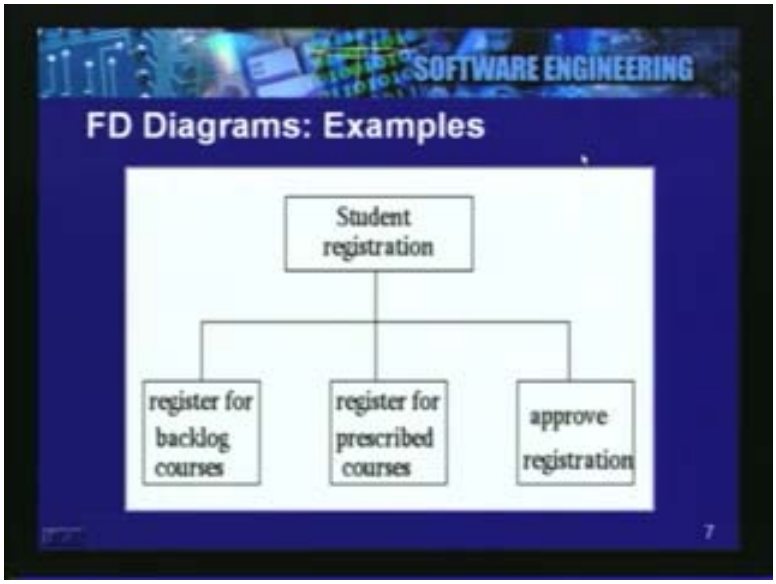
(Refer Slide Time 47:00)




They do not introduce any inconsistency after they finish their task. There are situations where it may not always be so. Consider the example of a bank. When you transfer money from one account to another, there are two steps here. You reduce one account and you credit another account. Both of them have to be done together otherwise the data would be in inconsistent state. Money would have been reduced, but would not have been credited. Hence elementary process would be a fund transfer process. You are transferring money from one account to another. You cannot or you would not like to decompose it further because the whole thing has to be taken as a single activity. The process sees the data in consistent state when it starts execution and when it finishes execution it leaves the data in the consistent state.

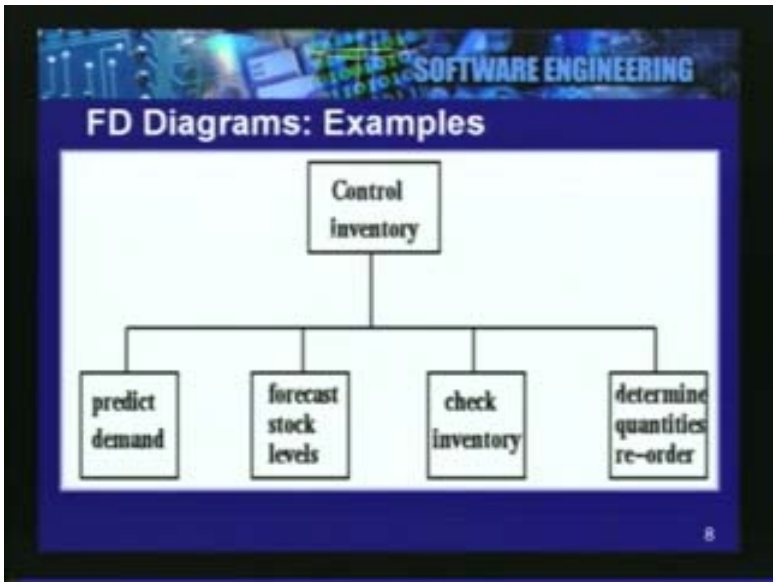
We now have seen that a process would be decomposed and we will do decomposition at multiple levels till we reach elementary processes. A diagram can be used for representing this decomposition. It would be having structure of a tree and we will have this elementary process at the leaf level. In doing the process decomposition diagram we are showing only activities. We are not showing any data in these diagrams. So remember that process decomposition diagram is only giving decomposition of the various processes. Here is a fairly simple example of a function decomposition diagram. The process that we want to consider is the process called student registration. Student registration may be a fairly complex process. So we are dividing it into three sub processes. The first process is called register for backlog courses. If a student has any failed courses in the past, then he must first register for them. Then the second process is, he must register for prescribed courses and then the third subtask is to get the registration approved.

(Refer Slide Time 47:40)




Hence student registration consists of these three processes: backlog registration, prescribed course registration, and approval of the registration by the faculty advisor of the student. Here is another example, inventory control. In the inventory control, an organization wants to ensure that stocks that it keeps in its warehouse are of appropriate quantity. It has to mange that inventory. Student registration may be a fairly complex process. So we are dividing it into three sub processes. The first process is called register for backlog courses. If a student has any failed courses in the past, then he must first register for them. Then the second process is, he must register for prescribed courses and then the third subtask is to get the registration approved.

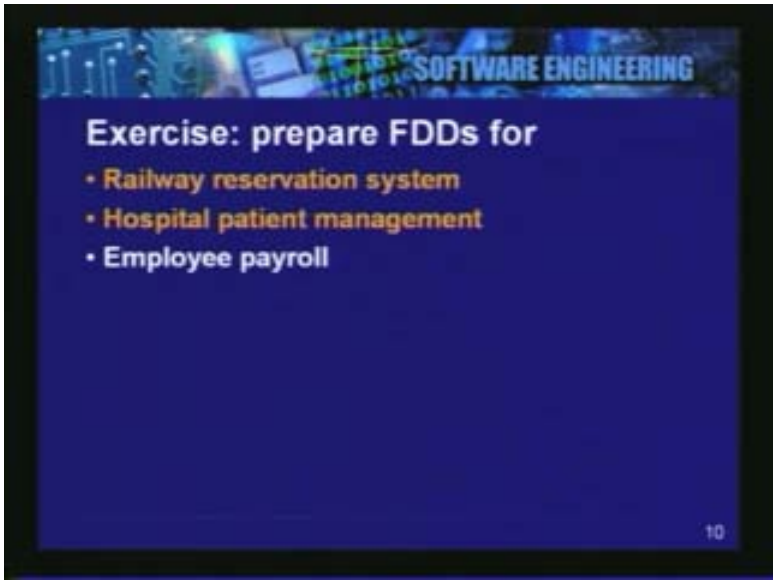
(Refer Slide Time 48:45)



The first of them is predicting the demand, then there is sub process called forecast stock levels, then the third sub process is check the inventory and the next process is to determine what quantities to order. Hence reordering, checking inventory, forecasting stock levels, predicting demand these four sub activities make the overall activity of control inventory, What we do in the process decomposition or function decomposition is to look at a process, study what happens in that, and then identify the sub processes which taken together indicate what really happens in this process. It gives a better understanding of what is required. For example, in control inventory we need to carry out these subtasks. This is what the function diagram represents.

Here is a small exercise. You can prepare such functional decomposition diagrams for some common situations which are quite well familiar to you. Function decomposition diagram for a railway reservation system, for a hospital patient management system, and for the payroll system of employees. In each of these cases, try to create the scenarios for these applications and prepare the function decomposition diagrams for them.

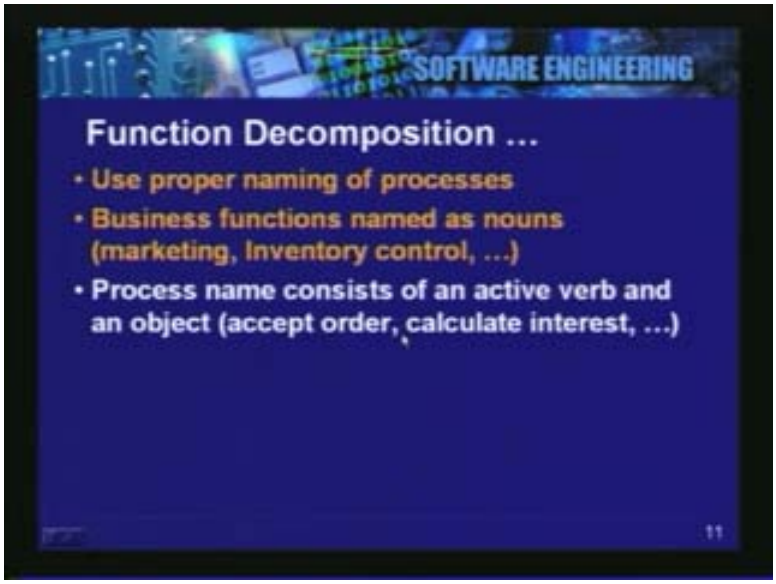
(Refer Slide Time 50:55)



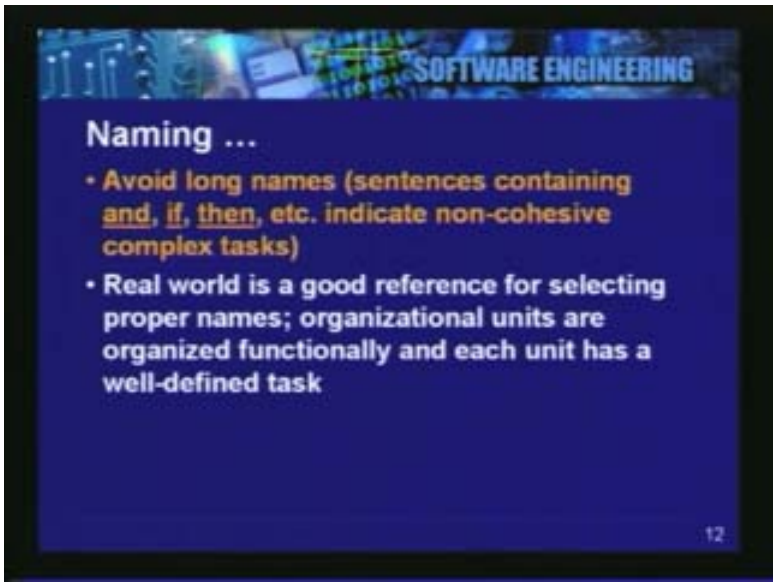

Here are some general guidelines of preparing the function decomposition model.

- Use proper naming of processes. This is very important, because through proper naming you are conveying the purpose of a process.
- Business functions are generally named as nouns. These could be marketing inventory control and so on. These are typical business functions and we use nouns for representing them. And it should be either a single word or a simple phrase such as marketing or inventory control.
- Processes are named consisting of active verb followed by an object, because process represent some action therefore we have a verb representing that action and action is having some purpose. It has a target; we represent that as an object. Some of the good examples are ‘accept order’, ‘calculate interest’ for process names.

(Refer Slide Time 52:15)



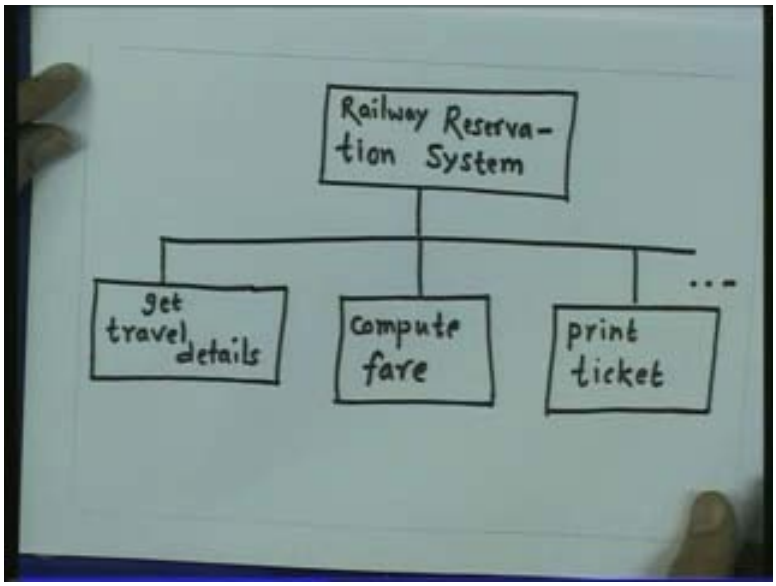

(Refer Slide Time 53:10)




Now, the names should be concise. They should be simple phrases or best if it is a single word. You should avoid long names or do not describe a process in the box that you are preparing for representing that process. Any time you use words such as ‘and’, ‘if’, ‘then’ etc, it clearly is an indication that it is a complex process and you have not done the decomposition which is cohesive and with minimum coupling. So you must avoid long names. Real world is a good reference for selecting the proper names. Very often organizations have sub divisions and these sub division carry some names. They are a good source of identifying proper names for our processes.

So naming is the most important and the challenging task in a properly defined function decomposition diagram. Let us conclude this lecture with an example. This problem I had mentioned earlier as an exercise for you. Let me illustrate a possible solution to this problem. This is familiar problem of railway reservation that I have mentioned earlier. I want to illustrate the typical mistakes we make in function decomposition. Let us look at an example that I have here. We have a railway reservation system as a function to be implemented.

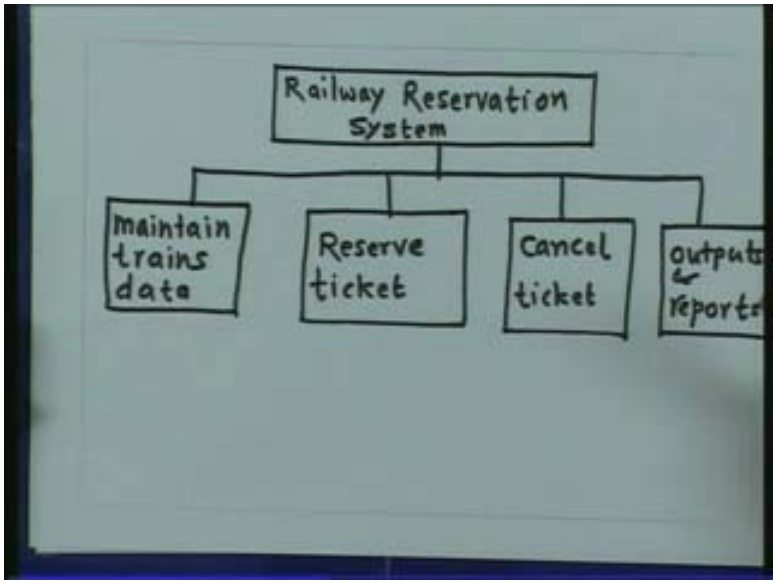
(Refer Slide Time 56:18)




This is the root function which we want to decompose and also some decomposition that has been shown. There is a ‘get travel details’ as a sub function of this, another function is ‘compute the fare’ and the third sub function is ‘print ticket’ and so on. Here we are identifying some sub functions which are required in the railway reservation system. But when we look at these, they are some small functions which come to our mind and we start writing them without considering whether they are forming a proper part of the whole system. And whether the guidelines that we had mentioned earlier, that the functions the sub functions that you prepare should be of roughly same complexity. Moreover they should add up to the main function that we are decomposing.

These functions like get travel details, compute fare, and print ticket etc will be definitely required. But they are almost like random thoughts and we do not know in what way they add up and what is left out here. We need to properly decompose the railway reservation system function at a higher level and not directly jump into some minor functions which do something very specific. Individually they may still very cohesive, but they are at a very lower level of granularity and they are very small subtasks which need not come at the next level of decomposition itself. So at the next level we should have a larger breakdown of this overall task, where the subtask will be of sufficient complexity and they would represent a first level of decomposition appropriately. Let us look at another alternative which is shown here. We have now the railway reservation system which has been decomposed into four sub processes.

(Refer Slide Time 58:39)

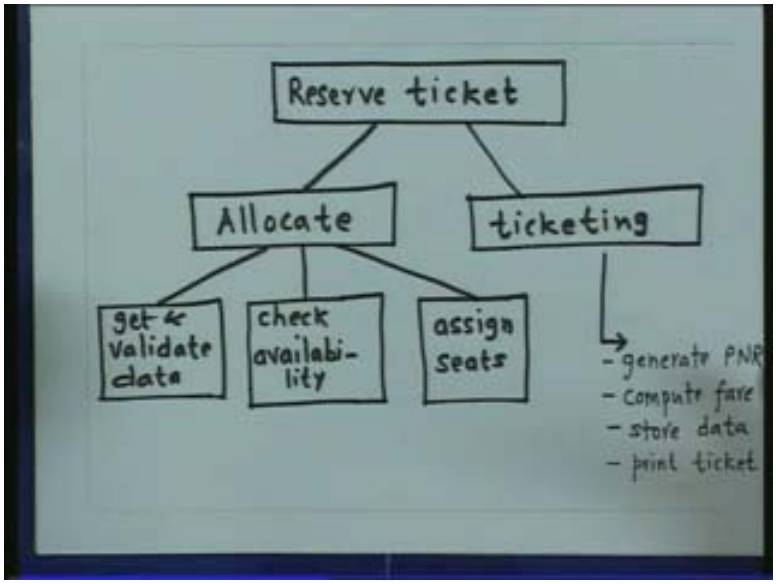



When we read them, we will find that these four subtasks together cover everything that we have in mind about the railway reservation system.

- The first one is sub function which says maintain data about the trains. This is something which will be required. This contains all the trains, their stations, fair details, the sizes of these trains in terms of types of coaches and number of coaches that they have. All these data needs to be maintained and we can define one sub function for that.
- The second one is about the reservation of the ticket. We do not go right away into the details of like printing a ticket or computing a fare at this point. We first decompose it at a higher level called reserve ticket.
- The third one is cancel ticket function.
- Then we have output and reports sub function. In this we will include various types of reports that need to be prepared. This could be reports about accounting, this could be reports which represents routine outputs such as giving the coach charts that we put up at the outside the gate of the coach where you list the names of passengers who have booking on that particular coach etc. All these outputs and reports can be delegated to a separate sub function and these outputs and reports will be obtained at a specific point in time. For example the coach charts will be printed just one hour before the train journey begins.

This way if you look at all these four functions, we see that collectively they perform all the tasks that are implied by the railway reservation systems. This is the first level of decomposition and it is complete at this level. At every level of decomposition, the two levels should match in terms of functions that they perform. Let us take one of these for further decomposition. Let us take reservation of a ticket as a function to be decomposed further and this decomposition is shown in a separate diagram here.

(Refer Slide Time 59:39)




We are decomposing the ‘reserve ticket’ sub process which was identified in the previous diagram. We are decomposing it into only two subtasks here called allocate and ticketing. As the name suggests the purpose of ‘allocate’ is depending on the preferences and the number of passengers listed, the date of journey etc. This function will take all that and give an allocation for that particular request. We have further decomposed the ‘allocate’ function into get validate data, check availability on the train, assign seats, and so on This is the way at every level you should decompose a function and ensure that it matches perfectly with a level with the earlier level function I would urge you to try out many such examples and you can use this function decomposition notion very well because this is the basis for developing models subsequently.