

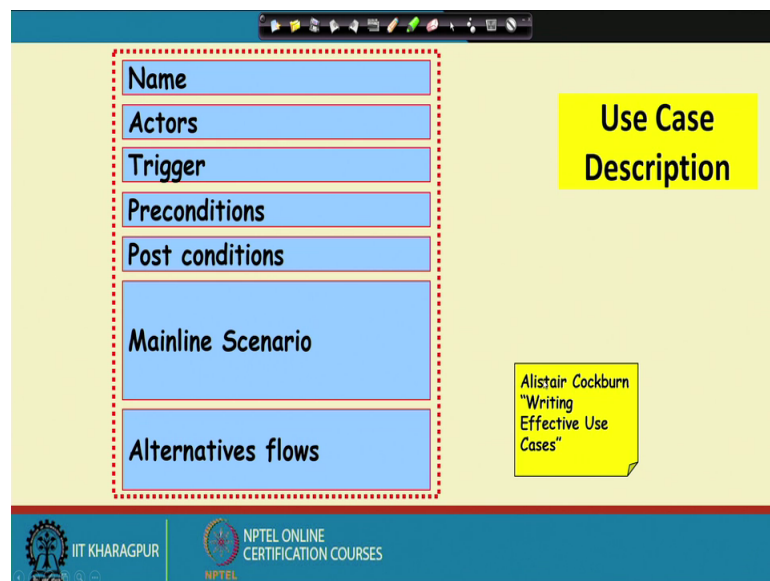
**Software Engineering**  
**Prof. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 33**  
**Overview of Class Diagram**

Welcome to this lecture. In the last lecture, we had seen how to develop Use Case models, the graphical model and also how to decompose the complex use cases using the 3 relations; generalization, specialisation and include and extend. In this lecture, let us see how to document a use case because just the graphical model by itself does not convey the complete picture; it just tells about what are the use cases and if they are decomposed into some use cases who are the actors and so on.

But details of the use cases are not given and that is the reason why in a use case model not only the graphical use case model should be given, but also the text description. As far as the text description is concerned UML does not impose any standard way of documenting the use cases. But then, some good practices have come up and those are normally followed even if even though there is no hard and fast requirement to follow the same documentation technique. Let us see how to document use case.

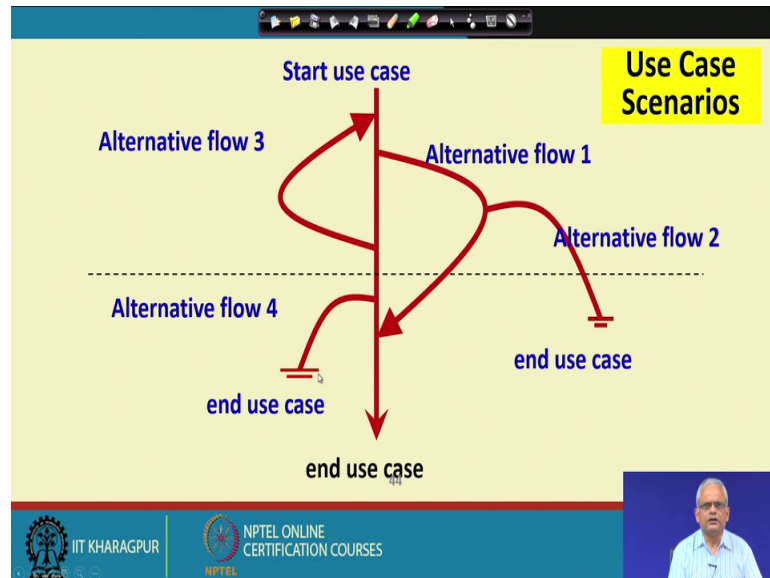
(Refer Slide Time: 01:44)



The documentation that is recommended by Alistair Cockburn in his “Writing Effective Use Cases” book says that every Use Case we must have the following documentation to

accompany. The name of the use case; the actors that would use the use case; the trigger when the use case will start to execute; the preconditions that is what must hold before the use case can be executed; the post condition that is after the use case completes what conditions must hold; the mainline scenario, this is the typical interaction sequence between the computer and the user and the alternative flows.

(Refer Slide Time: 02:48)



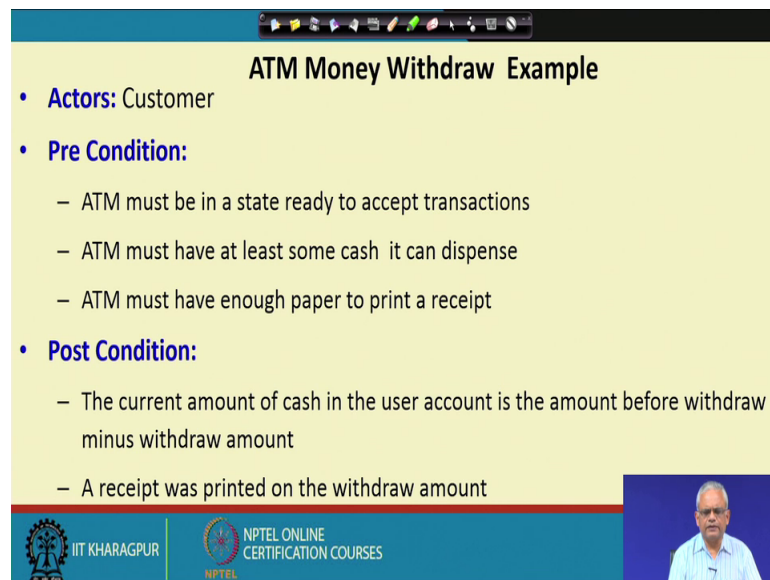
Will see what are the different scenarios and what is the mainline scenario the alternative flow and so on. In every Use Case there is a normal interaction between the computer and the user and sometimes there are alternative flows. Let us take an example to explain this. Let us assume that we have a bank ATM. Let us take the case of a bank ATM. The user goes to the bank; inserts the ATM card and then, the system prompts for the password or the pin code the user enters the pin code; the computer prompts do you want to withdraw from savings account or current account?

User choose a savings account and then the asks for the amount user enters let say 2000 rupees and then the system asks do you want a printed receipt and the user enters yes and then, the cash is dispensed and the printout is generated and the use case terminates. So, that is like a mainline scenario; but then sometime the user when it prompts to the user please enter the amount the user enters 550 rupees and then, it prompts says that please enter in multiple of 100 rupees. So, again the user enters and so on.

Sometimes as it enters 2000 rupees and then, it says want a printout and so on. But then, the system generates a message out of cash and it terminates. Sometimes, it proceeds and the amount is entered and it says that account does not have sufficient balance and it ends and so on.

So, the mainline scenario is the typical interaction sequence between the user and the computer. The alternate scenarios occurred, but these are not so frequent. The frequent case is the mainline scenario, but then there are other scenarios the which we call as Alternate flows or Alternate scenarios may occur. While documenting a use case we need to document not only the mainline scenario, but also the alternate flows. Let us see how do we go about documenting that.

(Refer Slide Time: 06:20)



**ATM Money Withdraw Example**

- **Actors:** Customer
- **Pre Condition:**
  - ATM must be in a state ready to accept transactions
  - ATM must have at least some cash it can dispense
  - ATM must have enough paper to print a receipt
- **Post Condition:**
  - The current amount of cash in the user account is the amount before withdraw minus withdraw amount
  - A receipt was printed on the withdraw amount

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us take the ATM money withdraw example. Here, document the actor is the customer; the precondition is that the ATM must be switched on ready to accept transaction. ATM must have at least some cash it can dispense; otherwise, it would be displaying out of cash. ATM must have enough paper to print a receipt. So, this is the precondition before the user can start to interact.

The post condition is that the current amount of cash in the user account is the amount before withdraw minus withdraw amount. So, if the use case completes successfully, the post condition says that what must hold true that is the amount in the balance in the

customer's account will be the amount before the withdrawal minus the withdrawn amount and receipt was printed on the withdrawal amount.

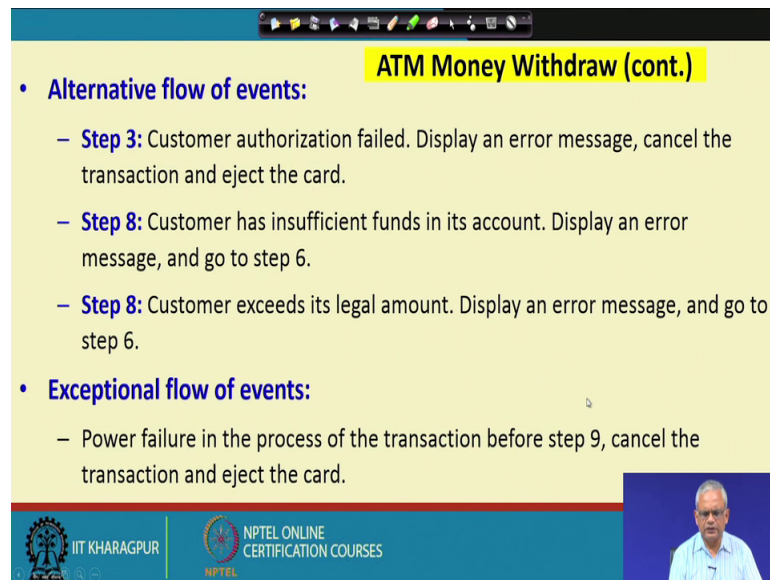
(Refer Slide Time: 07:37)

Actor Actions	System Actions
1. Begins when a Customer arrives at ATM	
2. Customer inserts a Credit card into ATM	3. System verifies the customer ID and status
5. Customer chooses "Withdraw" operation	4. System asks for an operation type
7. Customer enters the cash amount	6. System asks for the withdraw amount
	8. System checks if withdraw amount is legal
	9. System dispenses the cash
	10. System deduces the withdraw amount from account
	11. System prints a receipt
13. Customer takes the cash and the receipt	12. System ejects the cash card

And here, we list the mainline scenario and alternate scenario. The different scenarios basically are interactions between the actor actions and then there is a corresponding system action. The customer inserts the debit card; the system verifies customer Id. The customer chooses withdraw system asks for an operation time, operation type; the customer enters the cash amount.

The system asks for withdraw amount; the system checks if withdraw amount is legal; system dispenses the cash. System deduces the withdraw amount from the account and system prints a receipt. The customer takes the cash and the system ejects the cash and the card. This is the mainline scenario; but then there can be several alternate scenarios.

(Refer Slide Time: 08:40)



**ATM Money Withdraw (cont.)**

- **Alternative flow of events:**
  - **Step 3:** Customer authorization failed. Display an error message, cancel the transaction and eject the card.
  - **Step 8:** Customer has insufficient funds in its account. Display an error message, and go to step 6.
  - **Step 8:** Customer exceeds its legal amount. Display an error message, and go to step 6.
- **Exceptional flow of events:**
  - Power failure in the process of the transaction before step 9, cancel the transaction and eject the card.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let see, how do we document the alternate scenario. In the step 3, the customer authorization fails; then the customer has entered a wrong pin code. Then, display an error message; cancel the transaction and eject the card. So, this is one alternate flow.

Another alternate flow can occur at step 8, the customer has insufficient fund in the account; display an error message and go to step 6. Step 8, customer exceeds the legal amount that can be dispensed for customer may be 50000 is the legal amount; the customer entered 55000; display an error message and go to step 6. There can be exceptional flow of events. For example, while the transaction is going on power failure and then the transaction is cancelled and the card is ejected.

(Refer Slide Time: 09:41)

**Use Case Description: Change Flight**

- **Actors:** traveler
- **Preconditions:**
  - Traveler has logged on to the system and selected 'change flight itinerary' option
- **Basic course**
  1. System retrieves traveler's account and flight itinerary from client account database
  2. System asks traveler to select itinerary segment she wants to change; traveler selects itinerary segment.
  3. System asks traveler for new departure and destination information; traveler provides information.
  4. If flights are available then
  5. ...
  6. System displays transaction summary.
- **Alternative courses**
  4. If no flights are available then ...

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

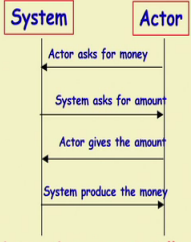
Let us take another example of a Use Case and how it can be documented. So, this is the change flight in a flight reservation system. The actor is the traveller. Precondition is that the traveller has logged into the system and selected the change flight option. The basic course or the mainline sequence is that the system retrieves the travellers account and the flight itinerary from the client account database.

The system asks to select the segment that wants to change system asks the traveller for the new departure and destination information and if the flight is available; then do the change and then the system displays the transaction summary. This is the mainline sequence and then, there can be alternate sequences if there are no flight available for the segment; then error message is displayed and so on.

(Refer Slide Time: 10:40)


**Guidelines for Effective Use Case Writing**

- Use simple sentence
- Do not have both system and actor doing something in a single step
  - Bad: “Get the amount from the user and give him the receipt.”
- Any step should lead to some tangible progress:
  - Bad: “User clicks a key”



```
sequenceDiagram
    actor Actor
    participant System
    Actor->>System: Actor asks for money
    System->>Actor: System asks for amount
    Actor->>System: Actor gives the amount
    System->>Actor: System produce the money
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



So, we just saw some templates for documenting use case, but UML gives flexibility in not following the exactly if the same standard.

Same template, we can tailor it, but then the one that we discussed are considered as good practices and many developers follow that. Let us see some guidelines for writing effective Use Cases. The Use Case is a document to be read by a large number of stake holders and therefore, it should be very readable. We have to use simple sentences and we must have separately this actor action and the system action documented because as you can see here the system and the actor their actions alternate the system initially the actor initiates the system responds and asks for something the customer enters it. The user enters the system responds and so on.

And therefore, we must write it in the same way what does the actor action and what is a corresponding system action? We should not mix that up that get the amount from the user and give him the receipt; we should not club both of these prompt the user for the amount, the user enters the amount; then the system action is dispense the account and print the receipt. Also we should not try it trivial steps in the use case documentation. For example, the user clicks a key that is not a progress not say tangible progress in the use case execution. So, every step that we document for use case must result in some tangible progress in the use case execution.

(Refer Slide Time: 13:03)

**Identification of Use Cases**

**1. Actor-based:**

- Identify the actors related to a system or organization.
- For each actor, identify the processes they initiate or participate in.

**2. Event-based**

- Identify the external events that the system must respond to.
- Relate the events to actors and use cases.

The slide also features a small diagram of a use case diagram with a box labeled 'Library' and several actors connected to it. The slide footer includes the IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES logos, and a small video inset of a speaker.

Now, let us see that given a problem description; how do we identify the use cases. A popular way is to find out from the problem description who are the actors and then, for each actor we can find out what are the functionalities that the actor invokes and then against that actor, we had seen the case of the library software. We identify at the actor is the member from the problem description, the librarian and the account and then for the member, we can identify the use cases that the member needs to execute that is a query book, issue book, written book.

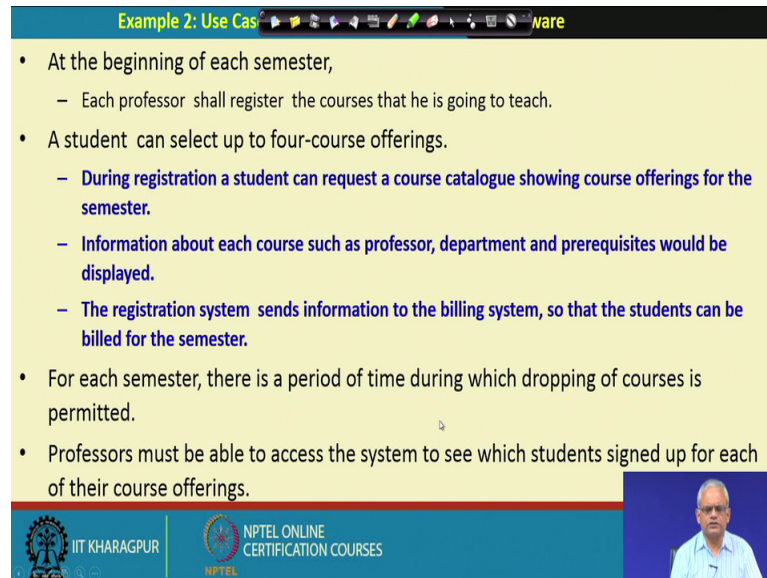
Similarly the librarian create member, delete member, create book, delete book, cheque fine etcetera and the account check the current balance, enter book procurement price and check the fine collected and so on; membership fee collected and so on. So, this is a popular way to find out what are the, who are the actors and each actor needs to invoke which functionalities.

The second way is a event based, where we identify what are the invocations of the system. What events the system must respond to and then we relate these events who generate these events and then what are the use cases that execute? For some systems like let say the embedded controllers and so on. It is easy to identify the events based on the state machine specification of the controller and then, relate who are the actors who generate that event and what is the corresponding use case. So, for some systems like



embedded controllers and so on event based may be preferable, but then the actor based one is very popular across all systems.

(Refer Slide Time: 16:04)



**Example 2: Use Cases**

- At the beginning of each semester,
  - Each professor shall register the courses that he is going to teach.
- A student can select up to four-course offerings.
  - During registration a student can request a course catalogue showing course offerings for the semester.
  - Information about each course such as professor, department and prerequisites would be displayed.
  - The registration system sends information to the billing system, so that the students can be billed for the semester.
- For each semester, there is a period of time during which dropping of courses is permitted.
- Professors must be able to access the system to see which students signed up for each of their course offerings.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see one example. This is a course registration software; let me just read the description of the software and let us see how do we go about identifying the actors, the corresponding use cases and then document it. At the beginning of each semester each professors shall register the courses that he is going to teach. So, into the software each professor will enter the course details that he is going to teach and then, once the professors are entered their courses they will teach, the students can select up to four-course offerings.

And during the registration, a student can request a catalogue showing various course offerings for that semester. Information about each course professor, which department, what are the prerequisites would be displayed at part of this catalogue. The registration system sends information to the billing system, once the student completes registration. So, that the student can be billed for the semester based on the which courses he is taking.

For each semester there is a time period during which dropping of courses is permitted and the professor must be able to access the system to see which students have signed up for their course offerings. How do we go about developing the use case diagram for this

problem description? We need to identify the actors, which is very easily identified. Here, it says that the professor shall enter the course details.

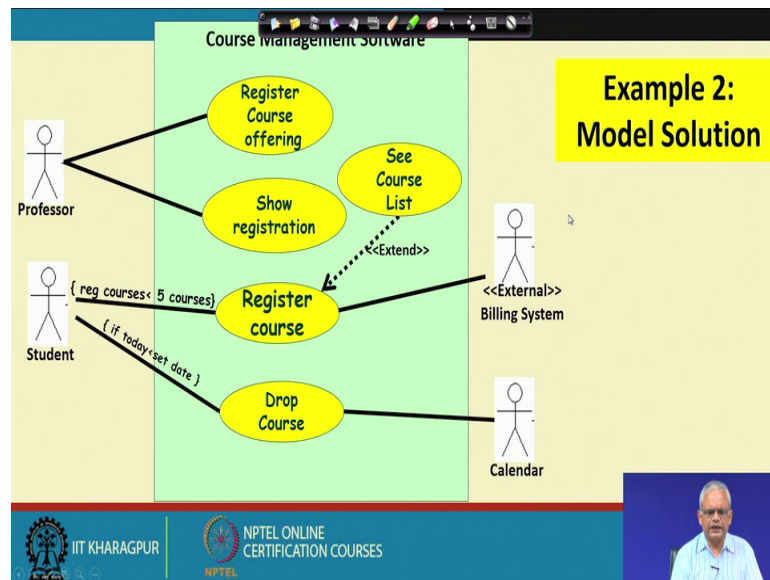
So, the professor is a actor. The student can select up to four-course offerings. The student is a actor and the registration system sends information to the billing system. The billing system is a external software may be running on a different computer or at least it is different software and this is also an actor. So, we can start identifying what are the use cases that each of these actors can invoke. The processor can register the course detail; the student can select course offering.

And during the course offering, during selecting a course he can request for a catalogue. So, this is the optional he may or may not request a catalogue and looks like an extension. So, course catalogue is an extension of the use case which is registered for courses. For each semester there is a period of time during which dropping of courses is permitted. So, that dropping of course, is a use case for which the student is the actor because the student initiates it; but how do we represent that there is a period of time during which dropping of courses is permitted.

When there is element of time here, we need to have an external system which is a calendar or a time that is the typical convention even though it is the system time. But if we represent a calendar actor or a timer actor time actor, then the design process become simpler as we proceed will see; but then, at the moment whenever there is a element of time, we will have another actor which is a calendar and this is a use case which the professor will access to find which students have signed up.

Now, let us see this is the overall diagram that we discussed. Now, let us see how do we draw the diagram.

(Refer Slide Time: 20:46)



So, this is the diagram, the professor registers the course offering. He can find out how many students have which students have registered and the student can register for courses which is less than 5 courses. We just write here in the form of a constraint on this line the communication line and during the course registration on successful registration, the billing system is sent information to generate the bill and we are written here external system, the billing system.

And for dropping courses, there is a time limit and we have the calendar as an external actor and then we have written the constraint here if today is less than set date. So, as long as the deadline is not passed the student can drop a course.

(Refer Slide Time: 21:50)

• Use case name should begin with a verb.

• While use cases do not explicitly imply timing:  
– Order use cases from top to bottom to imply timing -- it improves readability.

• **The primary actors should appear in the left.**

• Actors are associated with one or more use cases.

• Do not use arrows on the actor-use case relationship.

• **To initiate scheduled events include an actor called “time”, or “calendar”**

• **Do not show actors interacting with each other.**

• <<include>> should rarely nest more than 2 levels deep.

**Style Notes  
(Ambler, 2005)**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Scott Ambler, in his book in 2005, gave some style notes how to document a use case. Well, all use cases must be verbs and even though, when we document the different use cases; when we document the different use cases, even though we can document that in any order. But according to Scott Ambler as far as possible we must have an ordering of the use cases in the diagrams and that should correspond to the order in which these use cases are in invoked that helps in understand ability.

For example first is query book, issue book, written book and so on. We must order the use cases from top to bottom to imply timing wherever possible and the primary actor must be on the left. For example, if the student is registering for the course, the student initiates the registration this is a primary actor; whereas the external billing system is a secondary actor, the primary actor must appear in the left; the secondary actors must appear in the right.

The relation between the actor and the use case drawn by just a line this do not use an arrow and whenever there a time we must have a “time” or “calendar” as an external actor. External system which is the actor and if there is a interaction between two customers. For example, the user gives some information to the clerk and the clerk enters it.

We do not have to show that; we need to only represent the information that the actors input to the system. The interaction between actors, we do not have to capture here in

this diagram and when decomposing the use cases, the complex use cases, we should not decompose into a very deep nested sequence of use cases. The include and extend should be rarely nested to 2 level deep. The use case names should be in the perspective of the users we should use the users language.

(Refer Slide Time: 24:36)

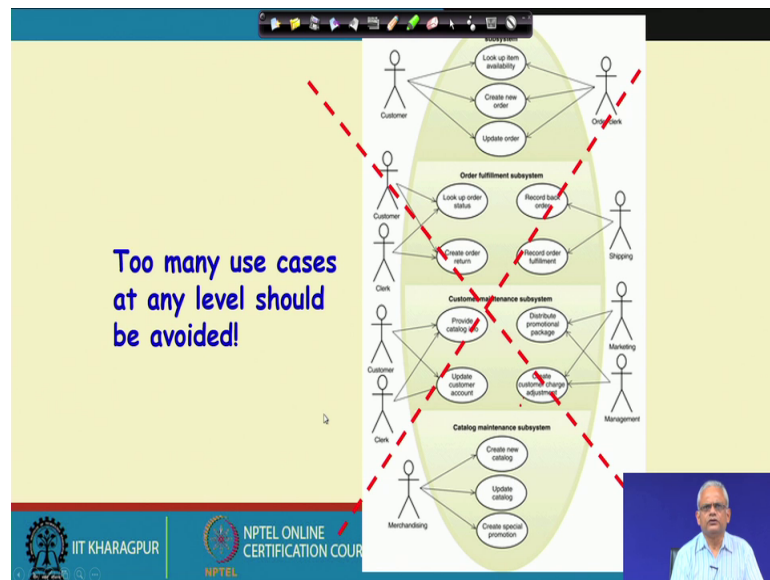
Effective Use Case Modelling

- Use cases should be named and organized from the perspective of the users.
- Use cases should start off simple and at as much high view as possible.
  - Can be refined and detailed further.
- Use case diagrams represent functionality:
  - **Should focus on the "what" and not the "how".**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

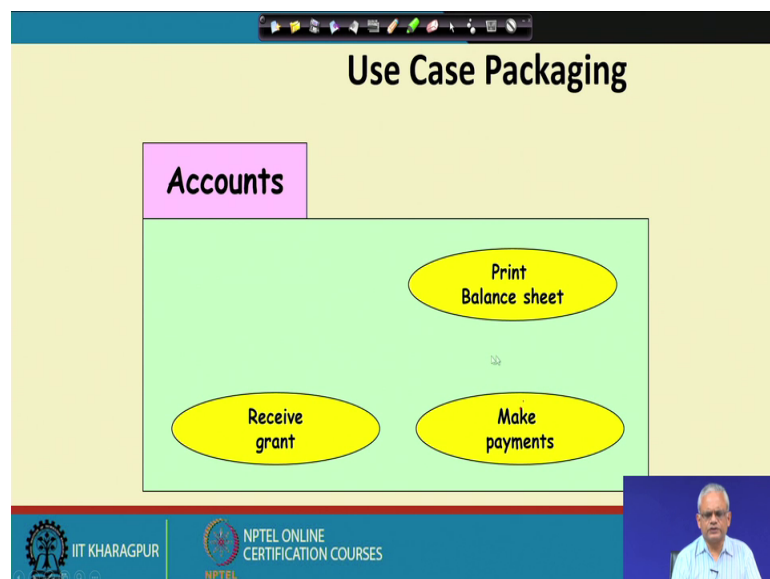
The use cases should be simple to understand and this can be decomposed wherever required and also remember that these are requirements, the use cases are basically functional requirements and just like the requirements they focus on what aspects and not how or the design aspect same holds for the Use Case diagrams as well.

(Refer Slide Time: 25:32)



We should not have a Use Case diagram that has too many use cases that becomes very confusing; when we have a situation where we have too many use cases we should use packaging.

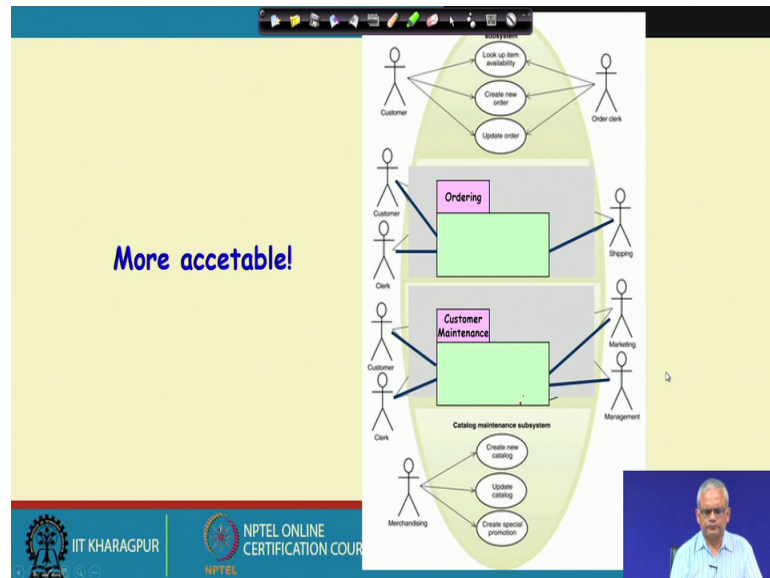
(Refer Slide Time: 25:49)



In Use Case packaging we use this folder symbol which is supported by UML and related use cases we can put inside the package and we write the name of the package here, all the use cases pertaining to the accounts actions.

For example: printing the balance sheet, making payment, receiving grant etcetera. These are in the accounts package. We can use the packaging to reduce the number of use cases in a Use Case diagram.

(Refer Slide Time: 26:36)



Just see here the same one; we can just have the ordering here. The ordering as one of the package; the ordering related use cases are here. The customer maintenance related use cases are here and then, in a separate diagram we can show the use cases in the ordering and that makes the diagram easier to understand.

(Refer Slide Time: 27:03)

## Class Diagram

- Classes:
  - Entities with common features, i.e. attributes and operations.
  - Represented as solid outline rectangle with compartments.
  - Compartments for **name, attributes, and operations**.
  - Attribute and operation compartments are optional depending on the purpose of a diagram.

So, far we have been looking at the use case diagram that is a crucial diagram; but it is also one of the simplest diagrams to develop. But then, it requires little bit of skill with respect to identifying the use cases; with respect to decomposing the use cases the documentation of each use case. And request you to please practice some of the assignments that we will give on developing the use case diagram to specific problems because just by understanding the concepts here, you may not get the expertise to develop a use case model.

You need to practice and several problems to develop the expertise of developing good use cases. Now, let us look at the class diagram. We know that classes are entities with common features, this have attributes and operations. Each class can be instantiated into objects; all objects have the similar attributes and operations. The classes this typically accepted that these are solid outline rectangle with compartments and the compartments are name, attribute and operations.

But we will see that there are various ways of representing the class diagram; for example, you might just write the name of the class. We might write the name and attribute into compartments or you might write all the name, attribute and operations.

We are almost at the end of this lecture and in the next lecture, we will discuss the class diagram in more detail. How to document a class diagram; the class relations and how do you represent that in a class diagram, we will discuss in the next lecture.

Thank you.