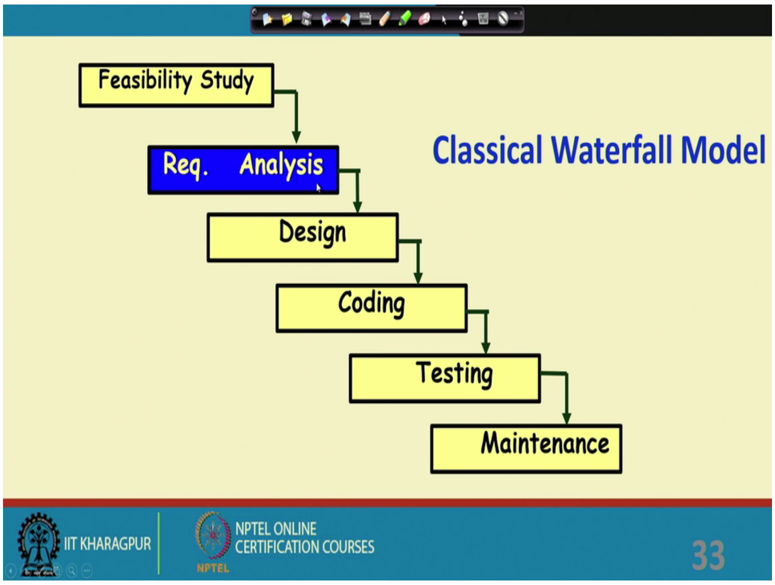
# Software Engineering
## Prof. Rajib Mall
## Department of Computer Science and Engineering
## Indian Institute of Technology, Kharagpur

## Lecture – 08
## Waterfall Model

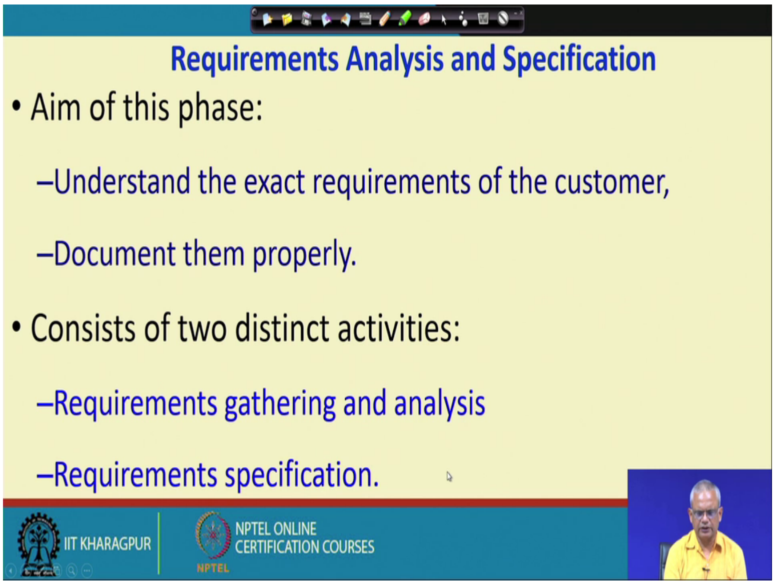
Welcome to this lecture will continue from where we ended last time and we will look at the second phase of the classical Waterfall Model which is requirement analysis.

(Refer Slide Time: 00:38)



So, look at requirement analysis and specification.

(Refer Slide Time: 00:41)



Requirements Analysis and Specification

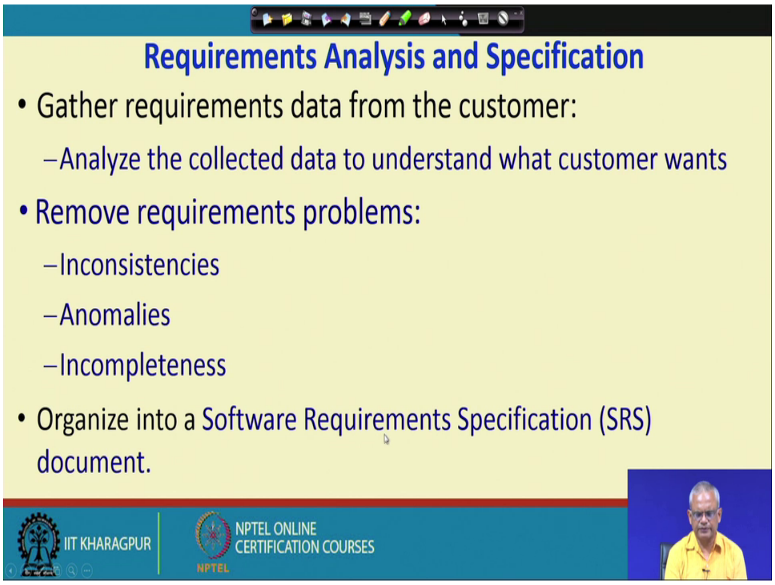
- Aim of this phase:
  - –Understand the exact requirements of the customer,
  - –Document them properly.
- Consists of two distinct activities:
  - –Requirements gathering and analysis
  - –Requirements specification.

The main aim of this page is to understand the exact requirements of the customer and then analyze the requirements. Find out if there are any difficulties in the requirements, problems, eliminate all the problems and then document this properly.

Almost every developer will have to do this some time or other; it is an important skill because how will the requirements analysis and specification is done; will determine whether the project will succeed or fail. So, this is a very important skill to understand how exactly to gather the requirements analyze the requirements and then document this properly; we will look at some standard ways of documenting. As we can see from the name of this phase there are basically two main activities; one is the requirement analysis and the other is requirement specification.

So, the first activity is requirements gathering and analysis; the second activity is requirement specification. Let us look at what is involved in requirements gathering and analysis and what is involved in requirement specification.

(Refer Slide Time: 02:39)



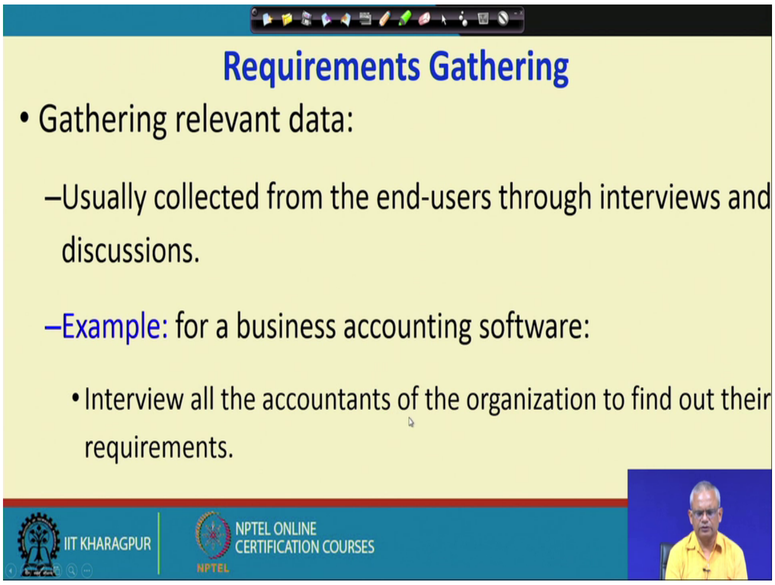
## Requirements Analysis and Specification

- Gather requirements data from the customer:
  - –Analyze the collected data to understand what customer wants
- Remove requirements problems:
  - –Inconsistencies
  - –Anomalies
  - –Incompleteness
- Organize into a Software Requirements Specification (SRS) document.

In requirements gathering, we need to gather the requirements, collect the data, understand what the customer wants. And during the analysis we analyze the collected requirement; identify there are problems in the requirement, but what sort of problems can be there in a requirement? There are 3 main types of problems that are normally exist in a gathered requirement one is called as inconsistency, anomalie and incompleteness.

Inconsistencies means, that one part of the requirement contradicts with some other part of the requirement. Anomaly is ambiguity some requirements are anomalous or not clear, in completeness some requirements are missed this should be there, but somehow the requirement has been missed. So, these are 3 main problems they need to be eliminated and then finally, that is to be documented in the form of a requirement specification document called as the SRS document or the software requirement specification document.

(Refer Slide Time: 04:10)

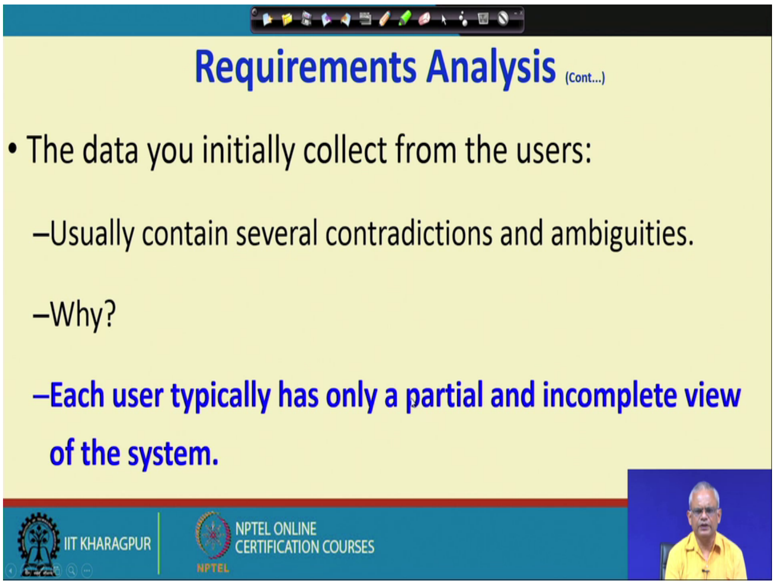


## Requirements Gathering

- Gathering relevant data:
  - –Usually collected from the end-users through interviews and discussions.
  - –Example: for a business accounting software:
    - Interview all the accountants of the organization to find out their requirements.

During the requirements gathering; the relevant gather data is gathered from the end users, normally done through interviews and discussions. For a business accounting software might have to interview, meet all the accountants in the organization who are doing in the manual mode, find out what exactly they are doing right now, what is the requirement from the software?

(Refer Slide Time: 04:45)



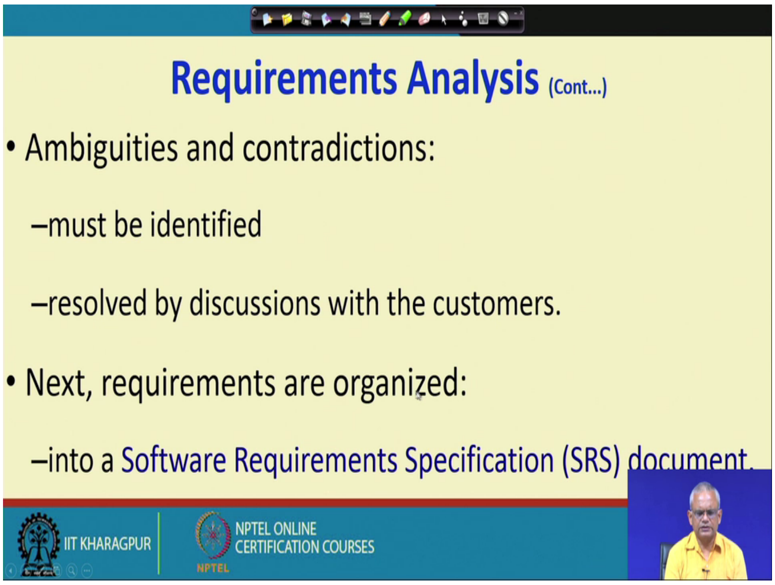
## Requirements Analysis (Cont...)

- The data you initially collect from the users:
  - –Usually contain several contradictions and ambiguities.
  - –Why?
  - **–Each user typically has only a partial and incomplete view of the system.**

And then once the data is gathered then there will be several contradictions and ambiguities. Why is it that the gathered requirements will always have difficulties problems like contradictions, ambiguities?

The main reason is that you are gathering requirement from a set of end user and each user as is different view of the software. And sometimes the views may contradict each other and also they might miss out some of their requirements or they may give a ambiguous statement. Each user has partial and incomplete view of the system and therefore, that is a main reason why there are difficulties in the gathered requirements.

(Refer Slide Time: 05:51)



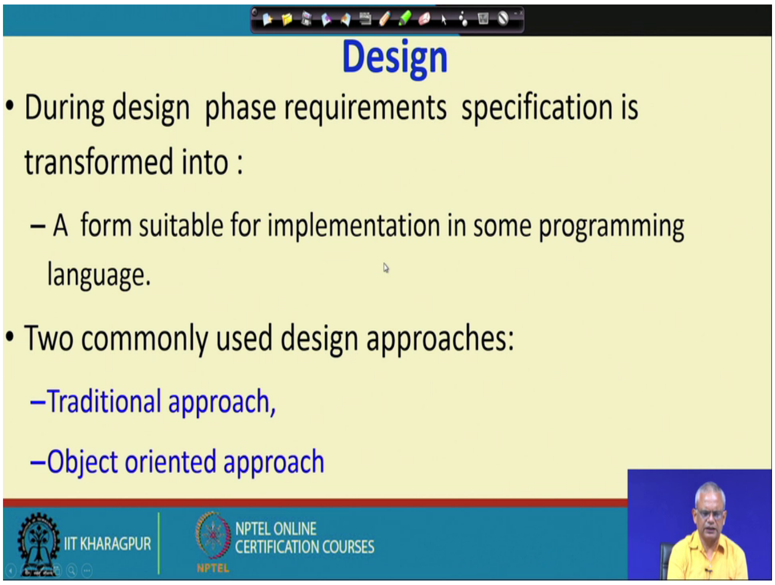
Requirements Analysis (Cont...)

- Ambiguities and contradictions:
  - –must be identified
  - –resolved by discussions with the customers.
- Next, requirements are organized:
  - –into a Software Requirements Specification (SRS) document.

All these problems in the requirement have to be identified and resolved, but how will this be resolved? By discussing with the customers that how exactly that problem will be addressed. And then once all the difficulties are eliminated from the requirements gathered requirements, then these are documented in the SRS document.

After few lectures we will look at the exact techniques by which requirement gathering analysis is done and then the format in which the requirement specification that is the SRS document will be written. And after the requirements analysis phase is the design phase; the design phase based on the requirements, the design is done and the design document is produced. Let us very briefly look at what is exactly involved in the design activity.

(Refer Slide Time: 07:06)

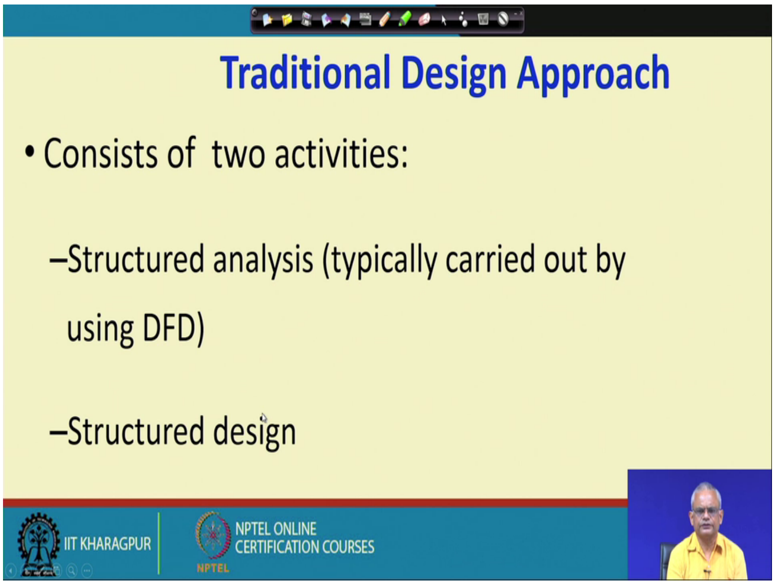


## Design

- During design phase requirements specification is transformed into :
  - A form suitable for implementation in some programming language.
- Two commonly used design approaches:
  - Traditional approach,
  - Object oriented approach

The design activity typically uses the requirement specification document; the designers consult the SRS document and then they come up with a design which will be easily implemented in some programming language; two main approaches are being used the traditional approach and object oriented approach.

(Refer Slide Time: 07:35)

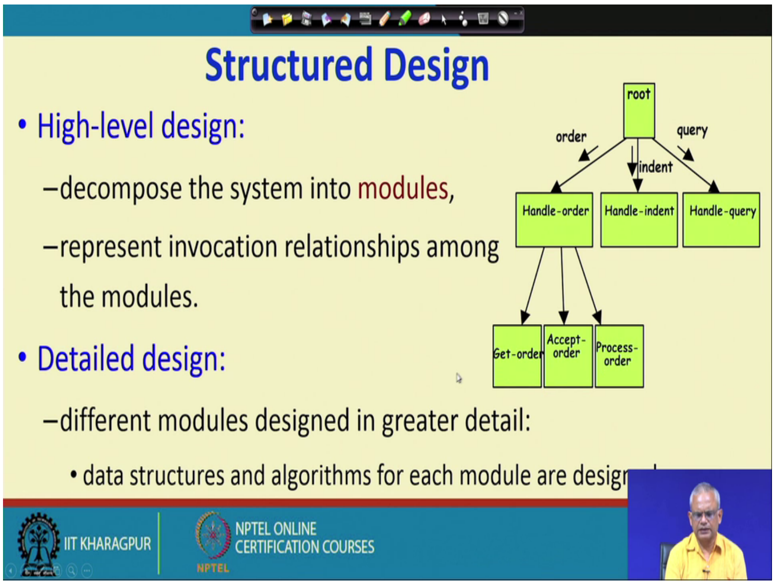


## Traditional Design Approach

- Consists of two activities:
  - Structured analysis (typically carried out by using DFD)
  - Structured design

In the traditional approach, there are two main activities one is to develop the DFD representation that is called as the structured analysis. And then once the DFD representation or the dataflow diagram representation is completely done; then it is translated into a structured designer we will look at this later.

(Refer Slide Time: 08:07)

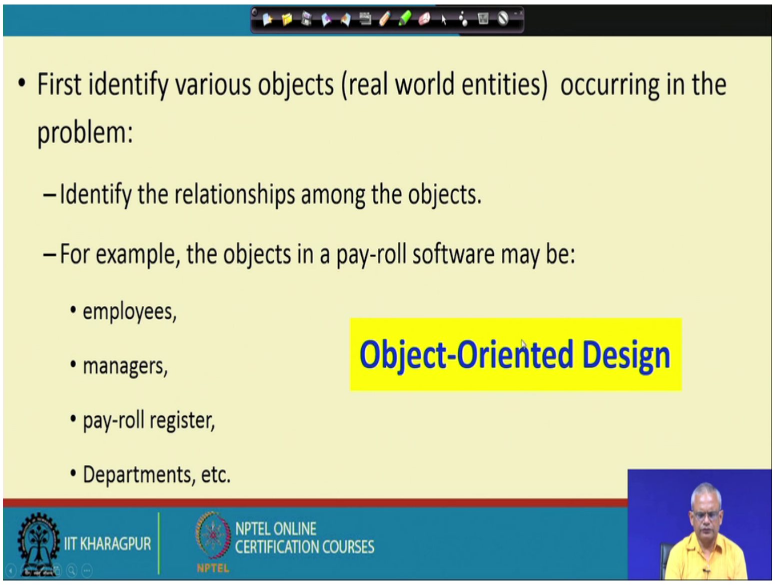


Structured Design

- High-level design:
  - decompose the system into modules,
  - represent invocation relationships among the modules.
- Detailed design:
  - different modules designed in greater detail:
    - data structures and algorithms for each module are design

At the end of the structure design, we will have a high level design in terms of the module structure. So, this is the module structure what are the modules and what are the called relation between the modules? And once the module structure is ready; then the detailed design is done where the data structure for each module and also the functions algorithms etcetera are designed.

(Refer Slide Time: 08:47)

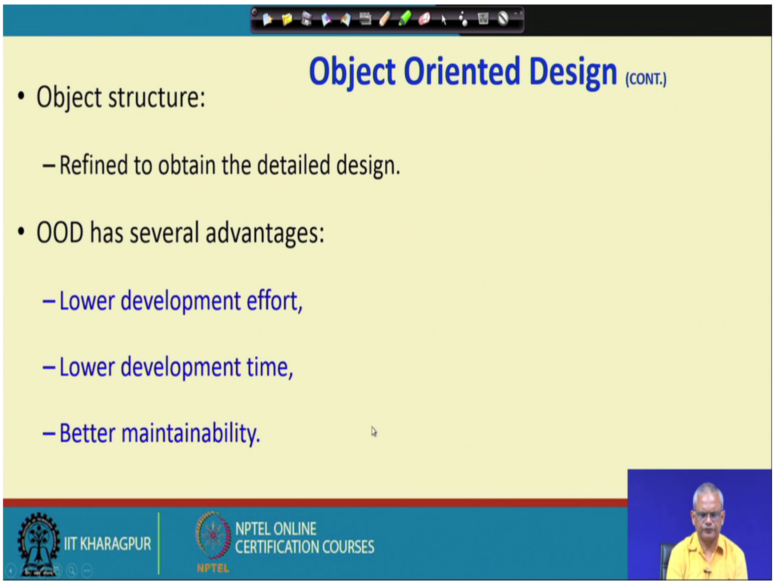


- First identify various objects (real world entities) occurring in the problem:
  - Identify the relationships among the objects.
  - For example, the objects in a pay-roll software may be:
    - employees,
    - managers,
    - pay-roll register,
    - Departments, etc.

Object-Oriented Design

On the other hand, in the object oriented design which we will also examine in some depth is that we identify the relations among objects, first identify the objects what are the relations among them. And then based on that we develop the design for example, the objects in a payroll software may be the employees, managers, payroll register, departments etcetera.

(Refer Slide Time: 09:22)



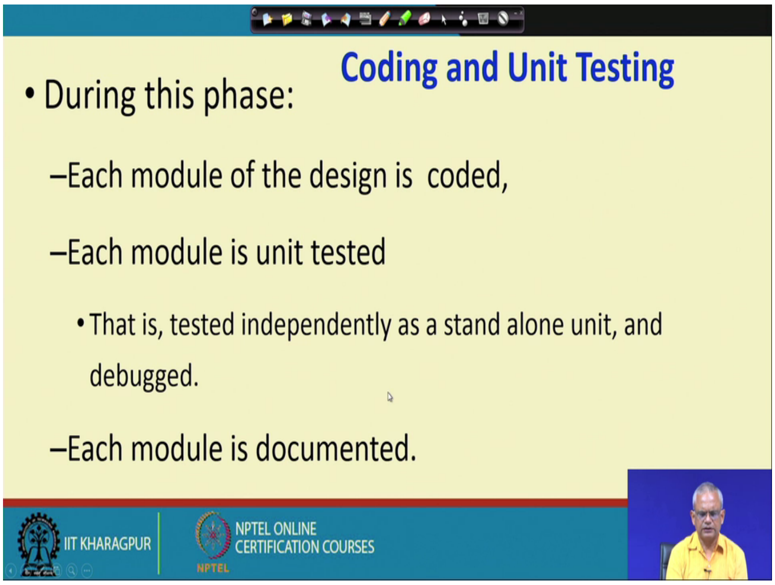
Object Oriented Design (CONT.)

- Object structure:
  - Refined to obtain the detailed design.
- OOD has several advantages:
  - Lower development effort,
  - Lower development time,
  - Better maintainability.

And the object structure is refined into the detailed design; after a few lecture we will see how exactly the objects are identified? How these are refined into the detailed design? But then the object oriented design technique has become very popular, it offers several advantages lower development effort, lower development time and better maintainability.

Once the design phase is complete; next the coding is undertaken, here based on the design document the coding is done and after that the testing activities are done.

(Refer Slide Time: 10:15)



## Coding and Unit Testing

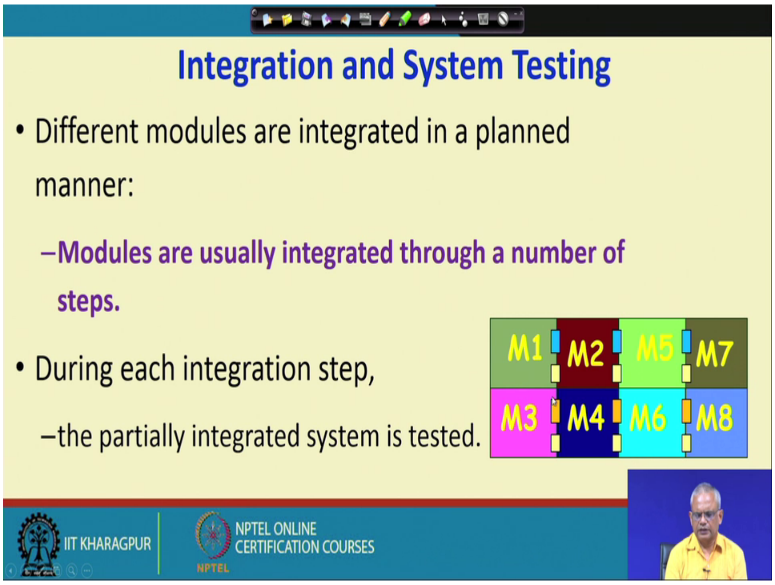
- During this phase:
  - –Each module of the design is coded,
  - –Each module is unit tested
    - That is, tested independently as a stand alone unit, and debugged.
  - –Each module is documented.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

During the coding phase each module of the design is coded typically its developer is given couple of modules to code and then completes the code and also does the unit testing. And once this passed the unit testing then the modules are documented and this phase completes.

The next stage is testing, so during coding not only the code is written, but the modules are unit tested. The next stage is testing where integration and system testing is done.

(Refer Slide Time: 11:03)



## Integration and System Testing

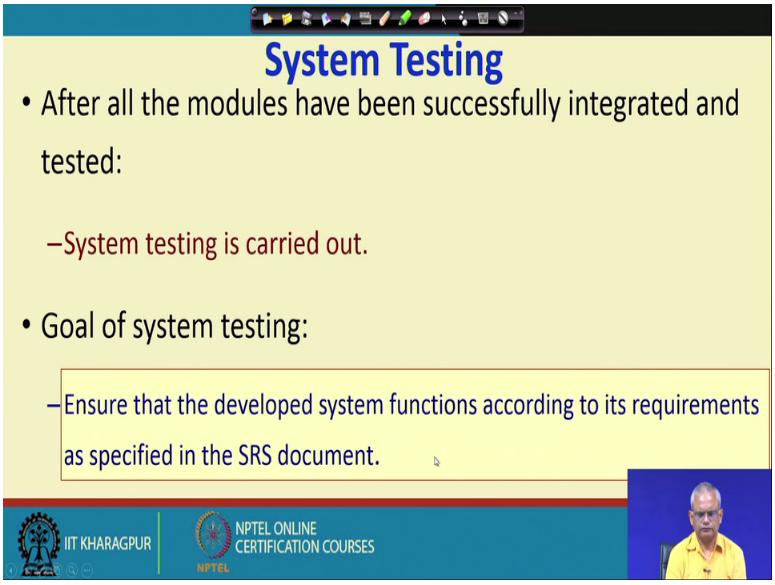
- Different modules are integrated in a planned manner:
  - –Modules are usually integrated through a number of steps.
- During each integration step,
  - –the partially integrated system is tested.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

During integration testing the different modules are integrated in a planned manner that is there are number of steps through which the modules are integrated. Maybe initially these 4 modules are integrated; to integrate these modules the main idea is that we check whether they are interacting properly.

The interface bugs are the main focus in the integration testing; during unit testing the bugs in the different modules are identified. And the integration testing the main focus is determining if there are any interface bugs. And in the next step more modules may be integrated and again the interfacing between them is checked. So, the system is integrated over a number of steps and each time; it is tested to check if there are any bugs and the typical focus of the integration testing is to identify the interface bugs.

(Refer Slide Time: 12:29)



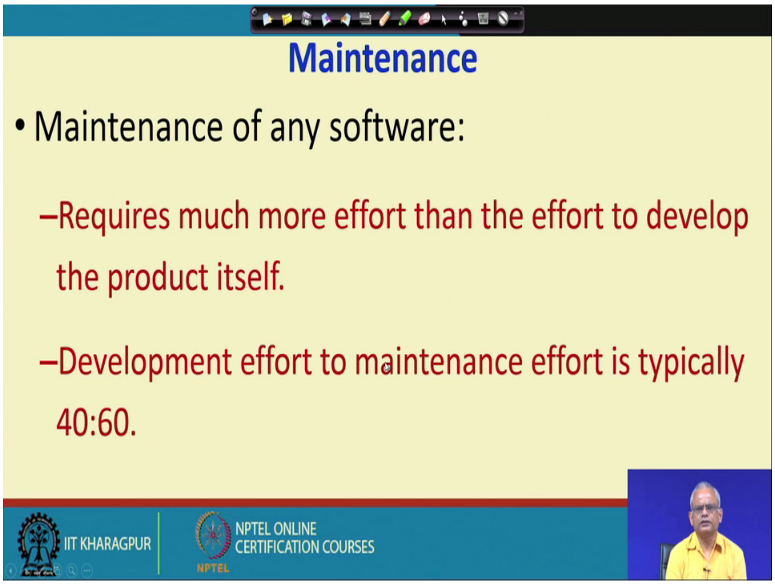
System Testing

- After all the modules have been successfully integrated and tested:
  - System testing is carried out.
- Goal of system testing:
  - Ensure that the developed system functions according to its requirements as specified in the SRS document.

Once all the modules have been integrated and the system testing is carried out; system testing the fully integrated system is checked to see if it meets the requirements that are experienced by the customer.

So, the goal of system testing is to check if the developed software satisfies all the requirements that have been expressed in the SRS document. And once the system testing is done, the software is delivered to the customer and then is the starting of the maintenance phase; where if there are any bugs reported by the customer, they may get fixed enhancements and so on; the maintenance phase continues for quite long time.

(Refer Slide Time: 13:38)

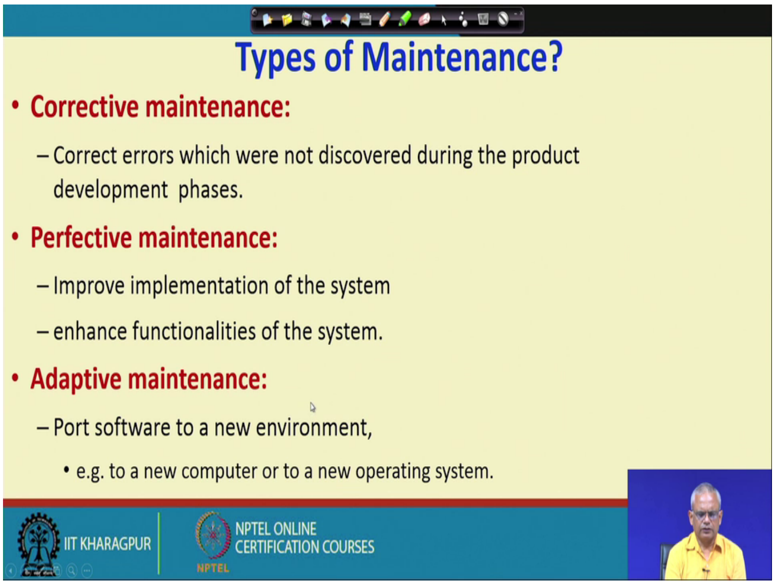


## Maintenance

- Maintenance of any software:
  - Requires much more effort than the effort to develop the product itself.
  - Development effort to maintenance effort is typically 40:60.

We said that earlier that maintenance takes maximum effect; much more than the effort required to develop the product itself. Because maintenance occurs over a long time; the lifetime of the software is large, the development time is only a small fraction of the lifetime. And typically the effort for maintenance is much more and the development effort to maintenance effort is typically 40 to 60; that is 40 percent development effort and 60 percent maintenance effort.

(Refer Slide Time: 14:20)



## Types of Maintenance?

- **Corrective maintenance:**
  - Correct errors which were not discovered during the product development phases.
- **Perfective maintenance:**
  - Improve implementation of the system
  - enhance functionalities of the system.
- **Adaptive maintenance:**
  - Port software to a new environment,
    - e.g. to a new computer or to a new operating system.

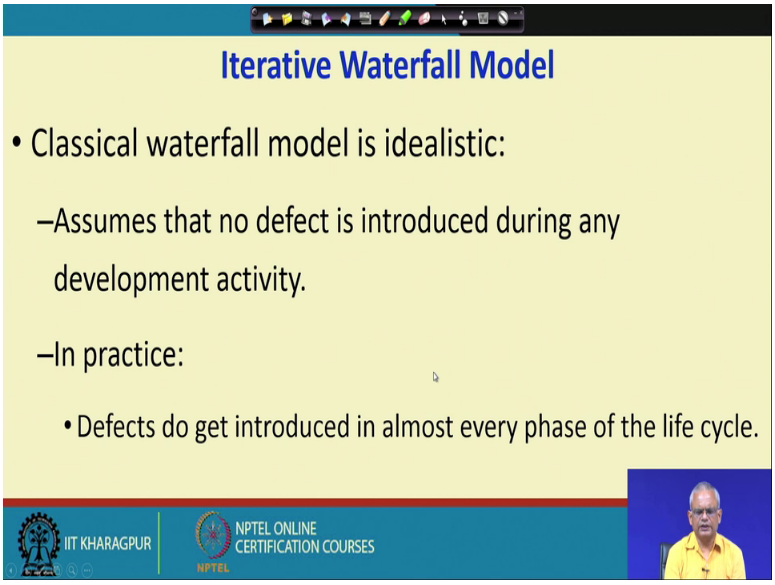
But let us look at what are the reasons why maintenance may be needed? That we call as the types of maintenance. There are 3 main reasons why maintenance may be needed; one is that bugs may be reported and need to correct those that is called as corrective maintenance.

It may be necessary to enhance the functionalities that is hard new functionalities which were not visualized earlier or improve the implementation in some way maybe the response time is not satisfactory; try to improve the response time and so on; that is called as perfective maintenance. And the third type of maintenance is adaptive maintenance.

In adaptive maintenance the software may be required to work with a new hardware; may be that the company procured a new server and it has to be installed on that; needs to work on this. Or maybe there is a new operating system on the software sorry on the new computer. And it needs to work with a new operating system and therefore, the program may have to be change little bit. So, this is called this kind of maintenance is called as adaptive maintenance.

So, there are 3 reasons why maintenance may be needed for a software one is called as corrective maintenance. This maintenance required to correct bugs perfective main maintenance to make the software more perfect like add new features, make the implementation, more efficient and adaptive maintenance where need to make it work with other types of software hardware and so on.

(Refer Slide Time: 16:34)



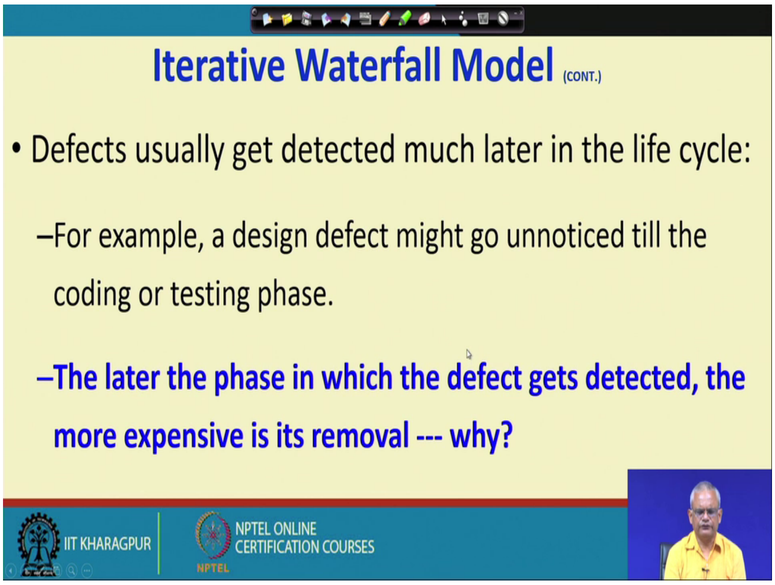
## Iterative Waterfall Model

- Classical waterfall model is idealistic:
  - –Assumes that no defect is introduced during any development activity.
  - –In practice:
    - Defects do get introduced in almost every phase of the life cycle.

The classical waterfall model is very simple; it matches with the intuition, but then it cannot be used or it is very difficult to use in a real life project. The main difficulty that will be observed when if we try to use the classical model in a real project is that it has a waterfall, where it can only go from one stage to the other stage.

Once a stage is complete no further activity on that stage occurs it transits to the next stage. But what if there was a design error which was discovered during let us say testing. Because every programmer makes mistakes and it may not be that all mistakes are discovered within that phase; maybe a mistake will be discussed much later. And therefore, the classical waterfall model is idealistic; assumes that no mistake is done or no mistake escapes a phase. In practice there are many mistakes which are committed and escaped that phase.

(Refer Slide Time: 18:01)



## Iterative Waterfall Model (CONT.)

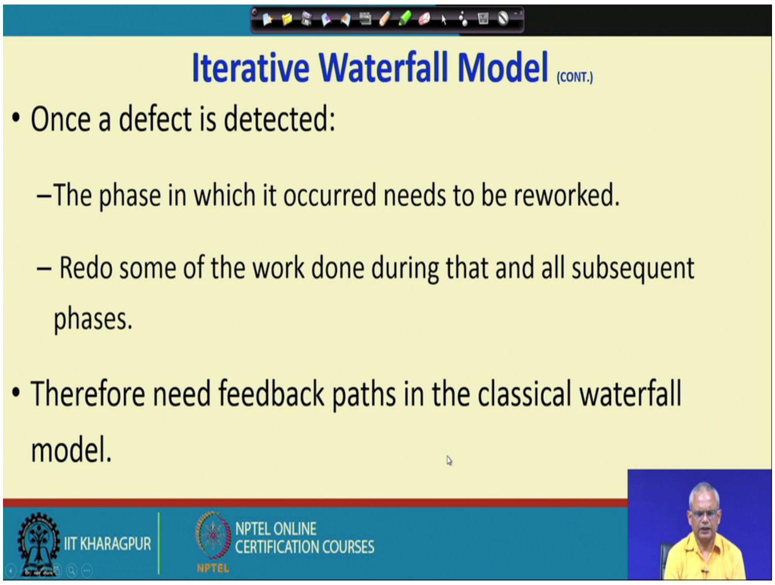
- Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase.
  - The later the phase in which the defect gets detected, the more expensive is its removal --- why?

And therefore, we need to as soon as there is a problem reported or might have to repeat some activities in the phase in which it was done; correct those not only correct that correct the subsequent document also. For example, if there is a requirements problem not only correct the requirement, but also may have to do the redesign; redesign parts and change the code and so on. But then if we understood that much let me ask this question that the later the phase in which the defect gets detected; the more expensive is its removal. So; that means, if a requirement is let us say faulty, there is a mistake in doing the requirements and let us say we discovered the mistake in the requirement during the design phase.

Then we need to rework the requirements and that has some cost, but let us say we had a mistake in the requirements; in the design phase we did not discover nor in the coding phase, but we discovered the requirements problem in the testing phase. Then it will be much more expensive that is what we are asking here that why is the latter the phase the defect gets detected, the more expensive is the removal.

The answer is that we need to rework the results of many phases; if we discussed, if we detect the problem requirement problem during testing; then we not only have to take up the requirements document and change all the requirements that are in problem, need to redesign, need to change the code and again need to test. Obviously, that is much more then if we had discovered the problem during the design stage; we might have to just change the requirements and some part of the design.

(Refer Slide Time: 20:34)

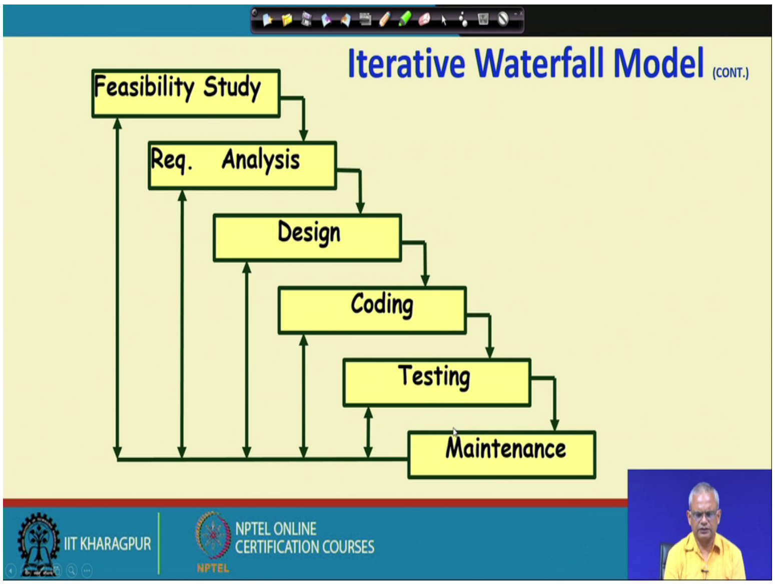


Iterative Waterfall Model (CONT.)

- Once a defect is detected:
  - The phase in which it occurred needs to be reworked.
  - Redo some of the work done during that and all subsequent phases.
- Therefore need feedback paths in the classical waterfall model.

So, once a defect is detected in a latter phase; we need feedback paths in the classical waterfall model.

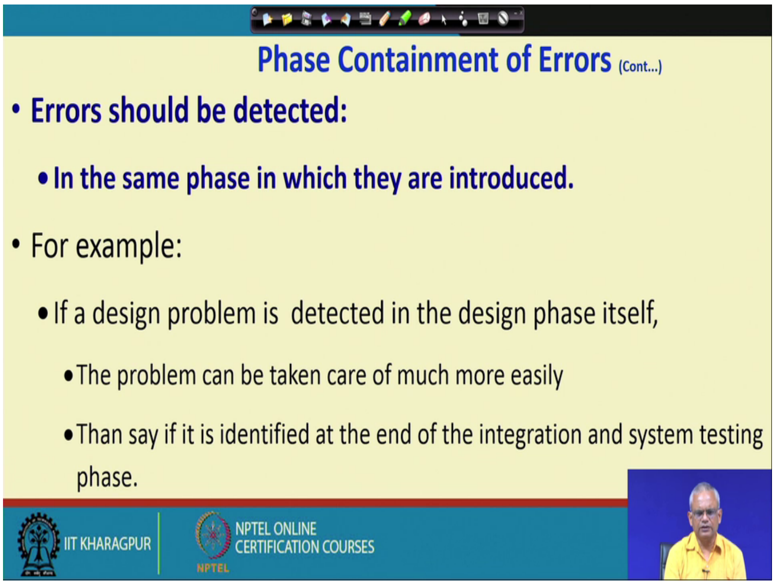
(Refer Slide Time: 20:51)



Iterative Waterfall Model (CONT.)

A defect of one phase may get detected in any of the later phase and as it is detected; we need to rework the results produced at that phase. And therefore, we need the feedback paths need we should be able to revisit that phase and that is basically the representation of the iterative waterfall model, where we have added feedback paths to the classical waterfall model.

(Refer Slide Time: 21:28)



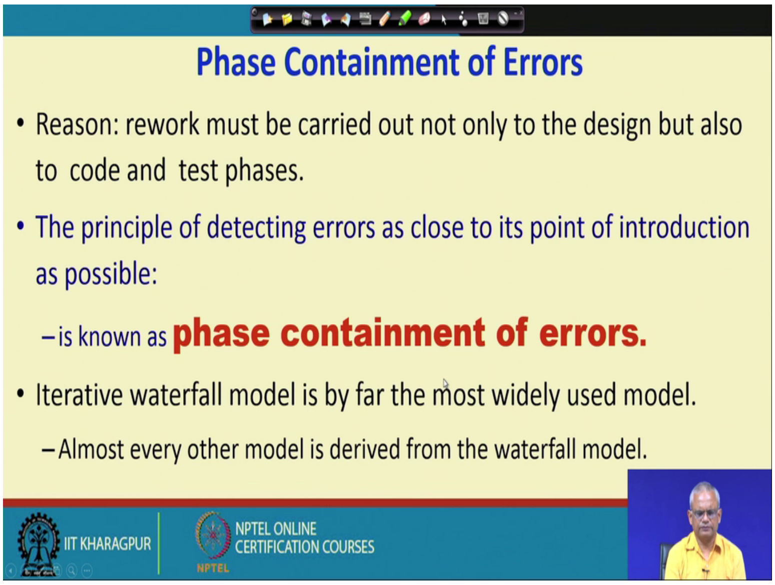
Phase Containment of Errors (Cont...)

- Errors should be detected:
  - In the same phase in which they are introduced.
- For example:
  - If a design problem is detected in the design phase itself,
    - The problem can be taken care of much more easily
    - Than say if it is identified at the end of the integration and system testing phase.

But one thing is that the principle that the later the defect is detected, the more expensive it is. That gives us an idea is that if we detect the defect in the same phase itself, then it will cost the least. For example, if the design problem is detected in the design phase; then we just correct that there itself. But if we detected this during coding and unit testing not only we need to change the design, but also change the code.

So, the programmers commit mistakes that is universal; that programmers do commit mistakes. But then the main idea here is that we must be able to detect those mistakes as quickly as possible and within the same phase for the cost of the correction to be low. And this is called as the phase containment of errors the errors must be detected within the same phase in which they are introduced. And the reason we want to have phased containment of errors is that it will cost the least; this is a very important principle the phase containment of errors.

(Refer Slide Time: 23:09)

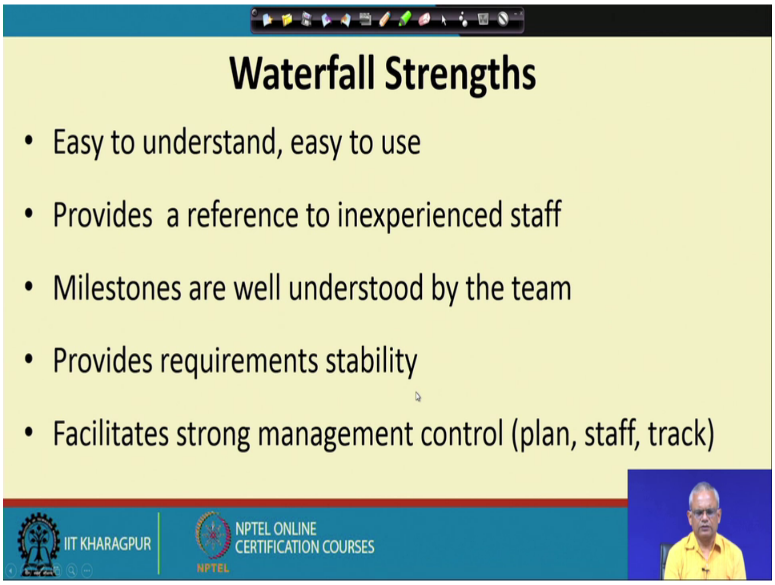


## Phase Containment of Errors

- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible:
  - is known as **phase containment of errors.**
- Iterative waterfall model is by far the most widely used model.
  - Almost every other model is derived from the waterfall model.

We will already discuss the phase containment of errors; we will not spend more time here. And need to mention that the iterative waterfall model was used very heavily and it is one of the fundamental model in the sense that all other models are derived from this model.

(Refer Slide Time: 23:43)



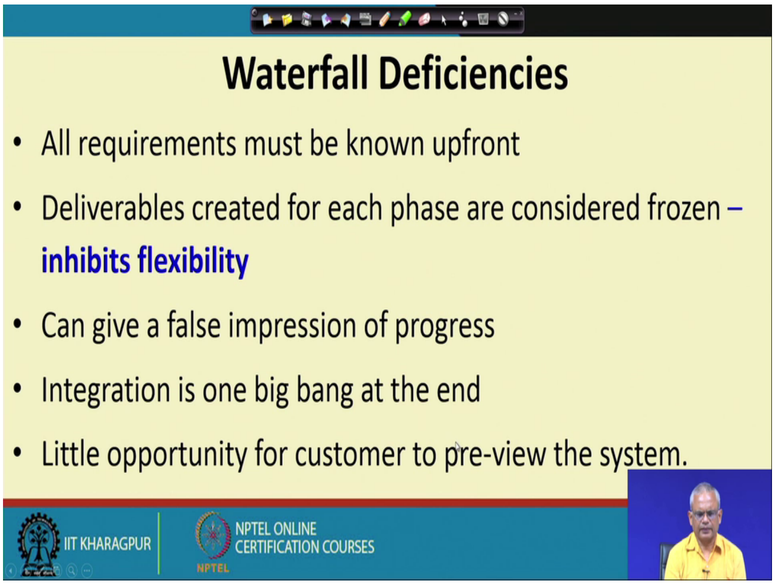
## Waterfall Strengths

- Easy to understand, easy to use
- Provides a reference to inexperienced staff
- Milestones are well understood by the team
- Provides requirements stability
- Facilitates strong management control (plan, staff, track)

The waterfall models are easy to understand and easy to use, the inexperienced staff who never had any project experience. If they work in a project using the waterfall model they can easily relate to the model and feel comfortable. As far as the project manager is concerned the milestones here are well understood phase entry phase starting and so on. The requirements have collected in the beginning and based on that the development starts.

So, the requirements do not change during the development and also the strong management control. In the sense that based on the requirement have been completed it can plan how long the design will take, coding will take and then track the project.

(Refer Slide Time: 24:49)



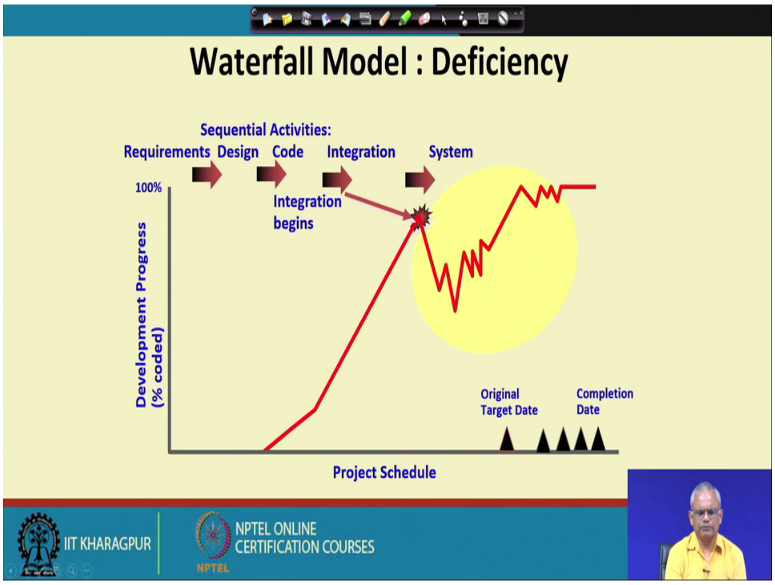
## Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – **inhibits flexibility**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

But what are the deficiencies of the waterfall model? One of the major deficiency of the waterfall model is that it requires all the requirements to be gathered and documented. And there is no scope to change this later on, but it is often the case that all requirements cannot be given; somebody can miss out on requirements, requirements may change and that cannot be accommodated in a pure waterfall model of development.

It lacks flexibility because requirement changes etcetera cannot be accommodated. It can give a false impression of progress because we find that requirements complete, design complete, coding complete.

(Refer Slide Time: 25:54)

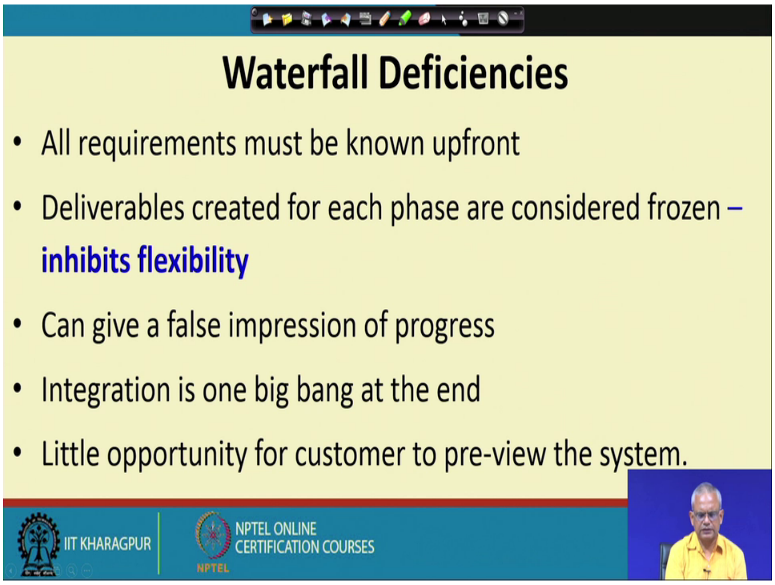


Waterfall Model : Deficiency

But then during integration problems may occur. Typically this is the scenario that the project is on time, everything is under control, the project manager is happy, requirements; design everything is progressing fine and then the integration begins. And from here; the problem starts actually, the project starts getting delayed. This was the time by which the system testing should be over, but unfortunately the integration problems occur here. Because till now it was under the grasp of one developer who has done the unit testing, but then the different developers they have made different assumptions and they do not integrate well; this typical projects scenario.

And the problem start during integration testing and the project starts getting delayed; delayed. The project manager announces the new delivery date again has to change it and so on. This is a typical problem of the waterfall model is about late integration.

(Refer Slide Time: 27:46)



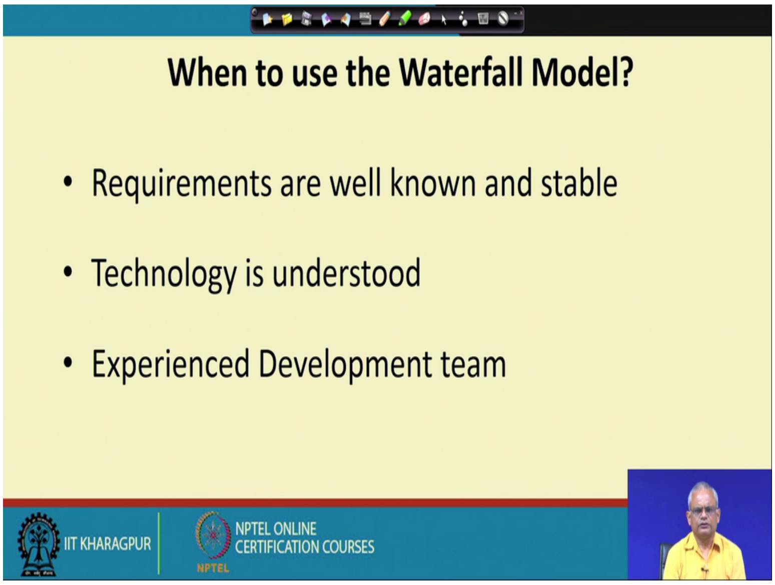
# Waterfall Deficiencies

- All requirements must be known upfront
- Deliverables created for each phase are considered frozen – **inhibits flexibility**
- Can give a false impression of progress
- Integration is one big bang at the end
- Little opportunity for customer to pre-view the system.

And this the project manager initially gives a gets a impression that everything is under control, project is proceeding as per time, but then suddenly the project starts getting delayed. So, the waterfall model often observed that it gives a false impression of progress.

One of the main reason is that integration is only at the end and many times, the problem start once the integration activity start. And also during the time that the development system is under development; the customer is entirely under dark. Only after the development completes is the system installed, but then may typical development may take several months until that time the customer has no clue what has happened to the software, will it be delivered, what will it be like and so on.

(Refer Slide Time: 29:06)



## When to use the Waterfall Model?

- Requirements are well known and stable
- Technology is understood
- Experienced Development team

The waterfall model is used now also for projects which are well known the development team are very familiar with this kind of projects, they have done many projects. The requirements are known and stable, the technology is understood, the developers have done this many times. The development is experienced in developing this kind of projects.

If this is the situation, these basically simple projects which has been done by the development organization many times and the requirements are known and stable, they do not change. Let us say an accounting package, let us say a company develops the accounting solution for different customers.

And they have experience solved many problems and now they want to you do an accounting software for a new customer. The developers are experienced on accounting software technology well understood; the accounting requirements are almost stable. And then in this case a waterfall model will be a suitable model. We will stop at this point and we will look at the other models, life cycle models and the type of projects for which they are suitable, how they differ from the waterfall model and so on; we will stop here.

Thank you.