

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

**Lecture – 02**  
**Introduction- II**

Welcome back. In the last lecture we were discussing some very basic issues. We are debating whether program writing is a art or a engineering. We said that, initially every technology starts in a art form slowly becomes a craft and then graduates into an engineering approach.

Software is no different and in the engineering approach we said that the past experience of large number of program developers has been systematically investigated, documented, scientific basis techniques have been provided, but then some are thumb rules; we look at those thumb rules as we proceed and the techniques.

Now, let us continue with what we are discussing.

(Refer Slide Time: 01:20)

• Early programmers used **exploratory** (also called **build and fix**) style.

- A `dirty' program is quickly developed.
- The bugs are fixed as and when they are noticed.
- Similar to how a junior student develops programs...

**What is Exploratory Software Development?**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

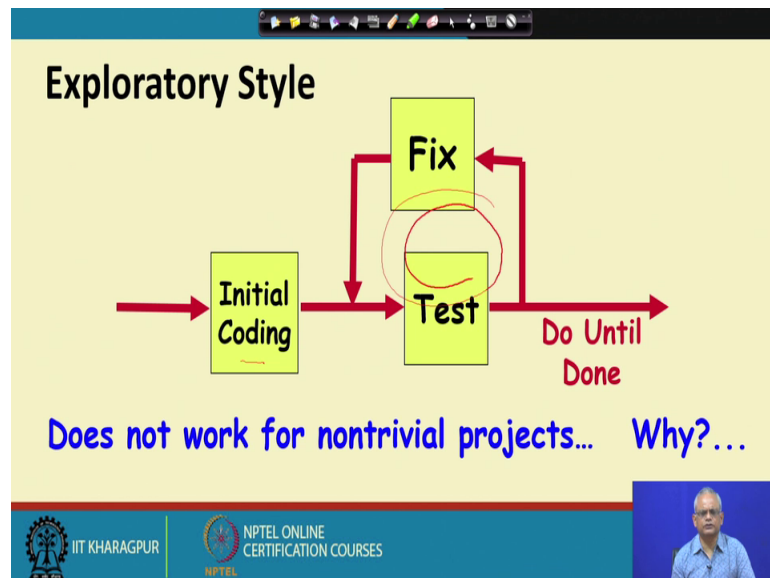
Let us try to understand; what is exploratory software development. Exploratory software development is the software development practice of a (Refer Time: 01:37). Let us say somebody who just starting to program, just learning programming and he is doing small assignments; how will he develop the program: that we call as the

exploratory software development or the build and fix style of development. This is the same style which the early programmers used, because there were no other techniques, they had to use their intuition and develop and that is called as a build and fix style.

In this style as soon as the problem statement is given based on the understanding of the problem statement start writing the program. And normally that is a dirty program in the sense that it will have lot of bugs. And then the programmer tests it, the bugs are fixed, as and when they are noticed.

If we document this development style the build and fix style in the form of a schematic diagram, we will come up with this diagram.

(Refer Slide Time: 02:58)



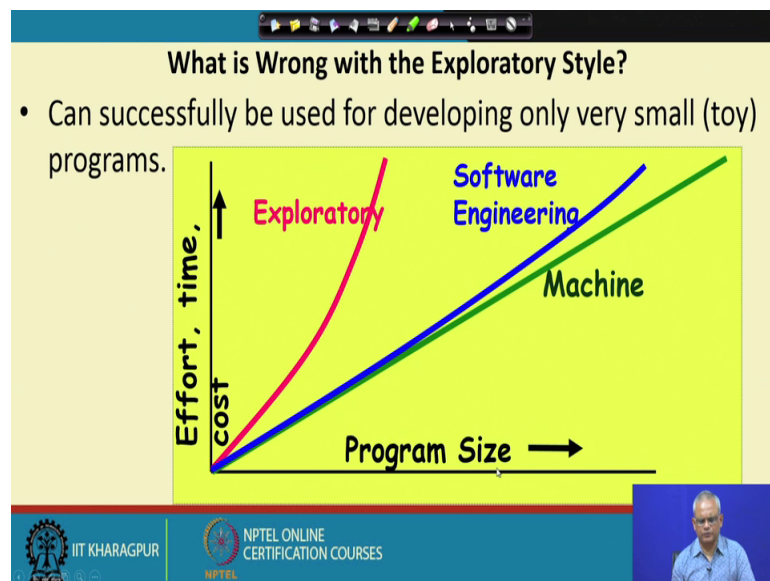
As soon as the problem is given we document we do the initial coding, and after the initial coding is complete start testing, find that the large number of bugs getting reported. Maybe some of them are cross, the program crosses or may be just large number of bugs compilation errors and so on and then just fix it. Again compile test and again see that there are errors keep on this cycle: fixing the program and testing. We keep on doing it until we find that, it is as per your understanding of the requirement for the program.

So, this is the model of an exploratory style of development. Let us keep this in mind, because as we proceed we will once in a while contrast with this basic style of development with the engineering style of development.

But one thing is that if we use the exploratory style as most of the new programmers use they can develop small programs. But then if the project is nontrivial, it is a large project requiring hundreds of thousands of lines of code and you do this that initial coding finds errors keep on fixing this will not work, the project will never complete. And even if somebody really puts lots and lots of effort and completes it somehow, he will come up with a very poor quality of software.

But then what is your basic reason let us try to investigate that; why is it that this style of development the exploratory style does not work for large projects, only for very trivial programs we can use the exploratory style.

(Refer Slide Time: 05:45)



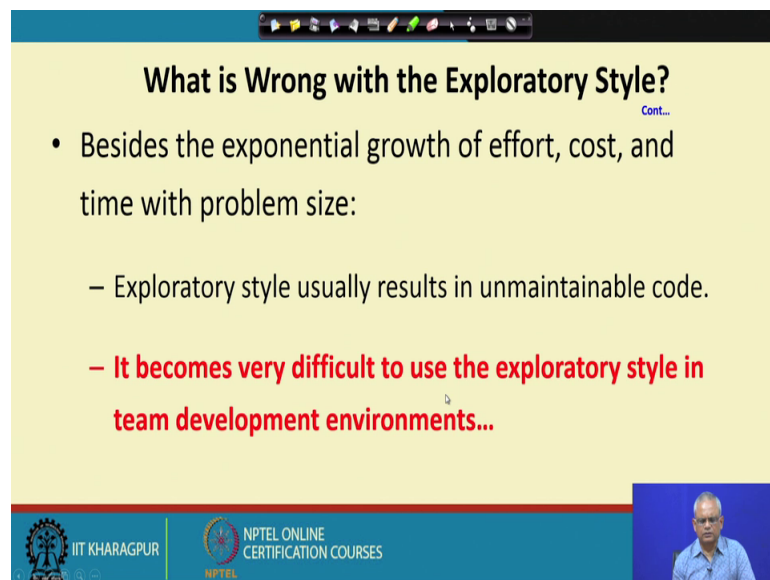
Let us try to understand what is wrong with the exploratory style. To understand this let us look at this plot. As the program size increases you develop large and larger programs and still keep using the exploratory style or the build and fix style. You will see that for small programs, maybe you are successful, but there is the program size increases; the effort, time, cost all increase exponentially. Maybe, to develop let us say a 100 line program you just took couple of hours, but to develop a 1000 line program you took several months.

But then, after some size the exploratory style just breaks down. Even if you try your best using the exploratory style still you cannot develop after a certain size, whereas the software engineering approaches let us look at here. There is a almost a linear increase in the effort, time, cost compared to the program size. So, if a 1000 line program took couple of hours a 10000 line program we will should take only about 10 times of that, not like 1000 times or 100 times.

But then, let us assume that there exists a machine to which you can give the problem statement it will come up with the code. It is a robotic machine which can write code. If that can be realized then it will be exactly linear. If you take some time some cost is incurred to give 10 times exactly 10 times will be incurred, whereas using software engineering principle it just increases slightly, we will see why.

But then, one very basic thing we need to remember from this slide is that the exploratory style, the cost effort time they increase almost exponentially, whereas using software engineering principles we are keeping them down to almost a linear increase.

(Refer Slide Time: 08:51)



The slide is titled "What is Wrong with the Exploratory Style?" and includes the following content:

- Besides the exponential growth of effort, cost, and time with problem size:
  - Exploratory style usually results in unmaintainable code.
  - **It becomes very difficult to use the exploratory style in team development environments...**

The slide also features a video inset of a speaker in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

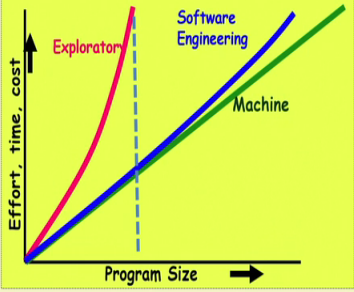
But then, the question naturally arises is that why is it so, what is wrong with the exploratory style, why is it that the exploratory style the cost effort time to develop increase exponentially, and also break down after certain size? And also, the other symptoms are that: the programs are poor quality unmaintainable and it is very difficult to use the exploratory style in the team development environment. But let us investigate

why it is so, what is wrong with the exploratory style. Whereas, in using the software engineering principles we are able to overcome this.

(Refer Slide Time: 09:49)

**What is Wrong with the Exploratory Style?** Cont...

- Why does the effort required to develop a software grow exponentially with size?
- Why does the approach completely break down when the size of software becomes large?



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 16

So, that is a basic question we must understand. The answer to that: you should be able to answer that why is it that the exploratory style there is exponential increase in the effort, time, cost, and we keep that down to a linear using software engineering principles. What is really the thing that is working here for software engineering?

(Refer Slide Time: 10:25)

**An Interpretation Based on Human Cognition Mechanism**

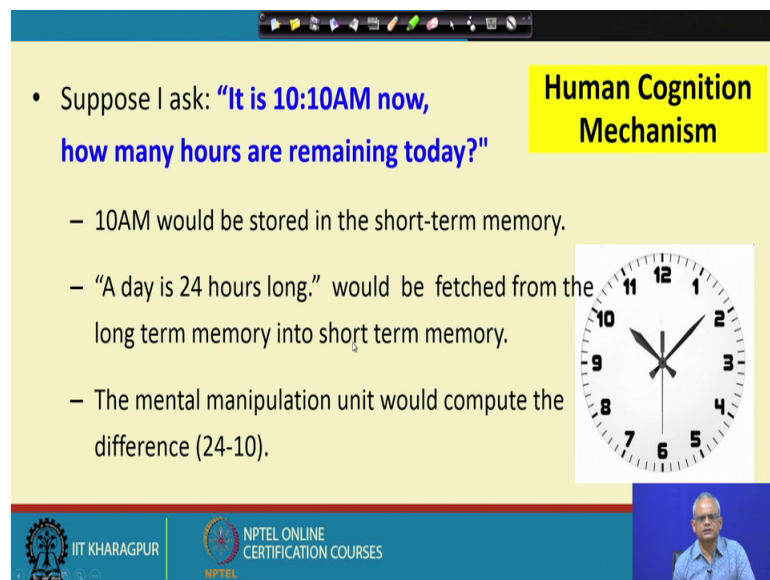
- Human memory can be thought to be made up of two distinct parts [Miller 56]:
  - Short term memory and
  - Long term memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 17

Let us investigate that, because that is a very fundamental principle. Why software engineering works, whereas the exploratory style does not.

To understand that we need to little bit understand the human cognitive mechanism. Long back result from Miller 1956 said that human memory can be thought of made of two distinct parts: one which is short term memory and the other is a long term memory. And as the name says, the short term memory is one where we remember something, but only a small duration time may be several seconds or just minutes. Whereas, long term memory remember it for months years and so on, short term memory if it is there you tend to forget it after some time.

(Refer Slide Time: 11:26)

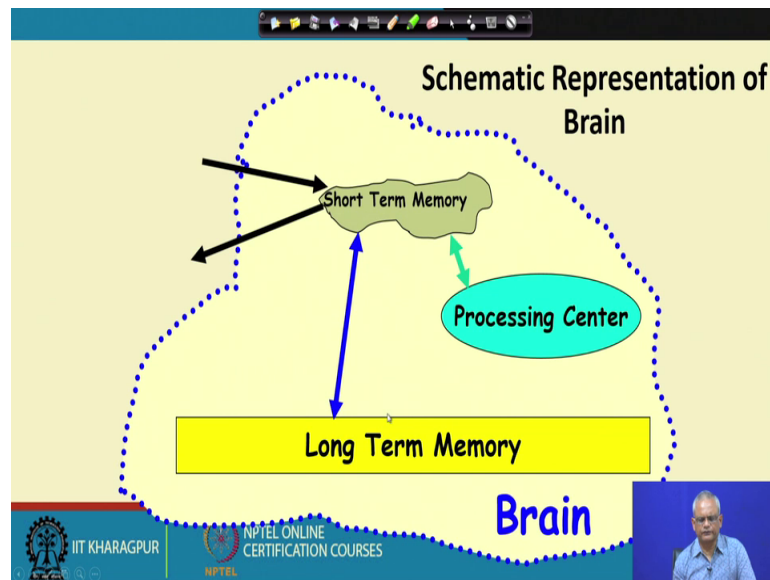


The slide is titled "Human Cognition Mechanism" in a yellow box. It contains a list of three bullet points explaining the cognitive process for solving a time-related problem. To the right of the text is an image of a clock showing 10:10. At the bottom left are logos for IIT Kharagpur and NPTEL Online Certification Courses. At the bottom right is a small video inset of a man speaking.

- Suppose I ask: **"It is 10:10AM now, how many hours are remaining today?"**
  - 10AM would be stored in the short-term memory.
  - "A day is 24 hours long." would be fetched from the long term memory into short term memory.
  - The mental manipulation unit would compute the difference (24-10).

Let us see the working of the human cognition mechanism. Let us say as I say that see look at the watch it is 10:10. How many hours remaining today? It is 10:10 by the watch, how many hours remaining today? The way the human mind will work is that 10 AM that is in the short term memory you saw the time it is 10 AM and from your long term memory you will fix that one day is 24 hours. And now that is also in your short term memory that 24 hours is a day, and then there will be a mental manipulation unit which will compute the difference 24 minus 10. And then you will say that its 14 hours or 13 hour 15 minutes.

(Refer Slide Time: 12:37)



If we put that down in the form of a diagram, we will see that there are the long term memory holds large number of items. Depending on the person it can hold billions trillions of item, whereas a short term memory can hold only few items. And then these are fixed from the long term memory, the short term memory, from the observation looking at the clock and so on, 10 AM. And then the processing center where some computation can be done and the answer is given.

(Refer Slide Time: 13:26)

- An item stored in the short term memory can get lost:
  - Either due to decay with time or
  - Displacement by newer information.
- This restricts the time for which an item is stored in short term memory:
  - Typically few tens of seconds.
  - However, an item can be retained longer in the short term memory by recycling.

**Short Term Memory**

The footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES".

But then what exactly is the short term memory, what can it store, how long; let us look at it. An item stored in a short term memory gets lost; with time may be several seconds minutes or hour. It gets lost from the short term memory, because there are too many new information that has come into the short term memory or that it just play and decays with time. But then, the main thing is that it stays for very short time and depending on whether there are many new information coming the time duration can be even very short, can be typically tens of seconds.

But then, somebody can remember something longer by recycling in memory. For example, you went to the market wanted to buy some things, just keep on recollecting again and again that what you need to buy. So, you were recycling in the short term memory.

(Refer Slide Time: 14:45)

**What is an Item?**

- **An item is any set of related information.**
  - A character such as 'a' or a digit such as '5'.
  - A word, a sentence, a story, or even a picture.
- Each item normally occupies one place in memory.
- When you are able to relate several different items together (**chunking**):
  - The information that should normally occupy several places, takes only one place in memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But then we said that the software memory stores items, but what exactly is an item. An item can be a character, it can be a digit character like a b etcetera, digit like 5 6 etcetera. It can be a word, a sentence, a story, and even a picture. If somebody remember some numbers let us say 5 9 7 etcetera each one is considered one item. But then, if there is a relation between them then it can be considered as a one item. Let us say 1 to 9: 1 2 3 4 5 6 7 8 9. So that will be one item, because you recognized that there is a relation it is 1 to 9.



Similarly a word, individual letters if there is no relation there will be separate items, but if they are related they reward mean something then that is just one item. The sentence is one item, story is one item and even a picture is one item. And they occupy one place in the short term memory.

One conclusion here we can make is that if you remembering something and you are able to build the relation between the things that you want to remember, then it becomes easier to remember. That is called as the Chunking.

Let us just look at an example of chunking.

(Refer Slide Time: 16:27)

**Chunking**

- If I ask you to remember the number **110010101001**
  - It may prove very hard for you to understand and remember.
  - But, the octal form of **6251 (110)(010)(101)(001)** would be easier.
  - You have managed to create chunks of three items each.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 22

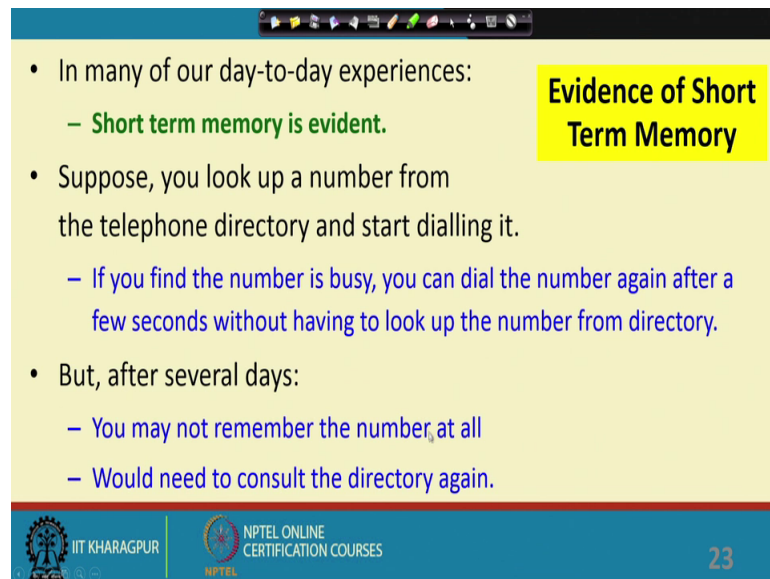
Let us say I ask you to remember the number 110010101001. You will struggle to remember this, because there are too many digits and then the relationship among them is not so obvious.

But then, let us say just group these into three and I give you the octal form, 6251 then it becomes easy for you to remember, because the short term memory about 7 items it can easily remember. And anything more than 7 there is a exponential increase in effort to remember you will have to look at various relations between them try to reduce let us say you might say that initially 2 or 1 then two 0s 101 etcetera. You will struggle to find relations, reduce it, chunks, and then try to remember so that becomes very hard to

remember a large set of items. That is because of the limitation of the human cognition mechanism that the short term memory can hold only 7 items.

And as long as you are able to find relations and so on it becomes easy for remembering that as long as it is less than 7. So, that is the principle of chunking.

(Refer Slide Time: 18:09)



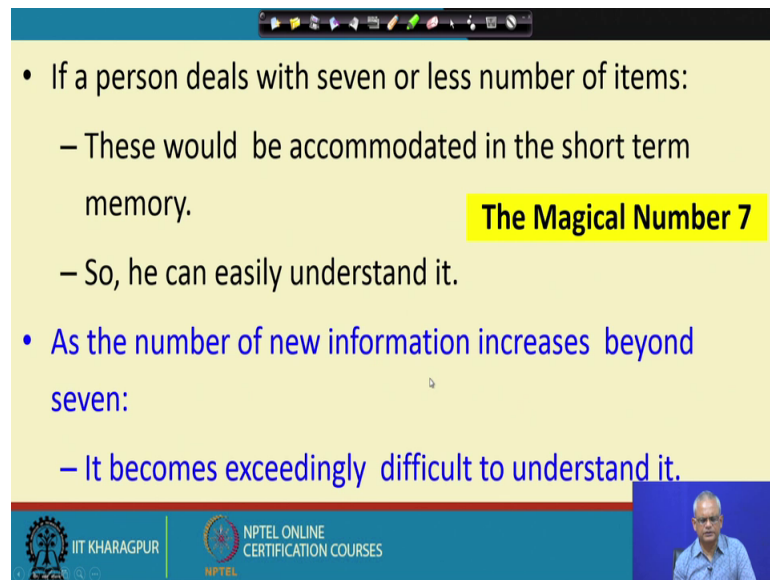
The slide is titled "Evidence of Short Term Memory" in a yellow box. It contains the following text:

- In many of our day-to-day experiences:
  - Short term memory is evident.
- Suppose, you look up a number from the telephone directory and start dialling it.
  - If you find the number is busy, you can dial the number again after a few seconds without having to look up the number from directory.
- But, after several days:
  - You may not remember the number at all
  - Would need to consult the directory again.

The slide footer includes the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and the number 23.

There are many day to day experience where you can the short term memory is evident. For example: you looked up a number telephone number and then you found that you not able to dial it, it is busy. But then, you will see that after few minutes you are almost able to remember the number. But what about after several days, you will hardly remember anything.

(Refer Slide Time: 18:42)



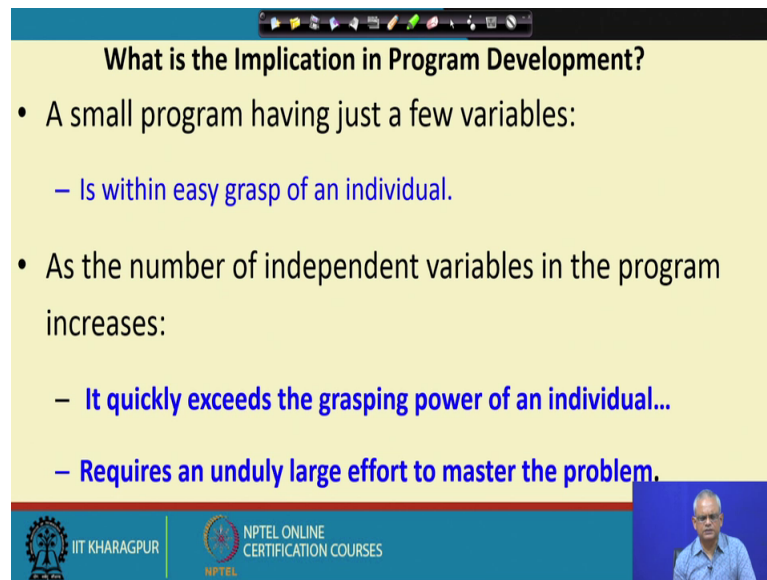
The slide features a yellow background with a blue header and footer. At the top, there is a navigation bar with various icons. The main content is a bulleted list. A yellow box highlights the title 'The Magical Number 7'. The footer contains logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a man speaking.

- If a person deals with seven or less number of items:
  - These would be accommodated in the short term memory.
- **The Magical Number 7**
  - So, he can easily understand it.
- As the number of new information increases beyond seven:
  - It becomes exceedingly difficult to understand it.

But then, the number of items that somebody can store in short term memory is about 7 and that is called as the magical number 7. Any set of information which has 7 items in it for an ordinary person it becomes easy to understand that. But if the number of items are large let us say tens of thousands let us just look at in very trivial example. Let us say you met 5 people somewhere you will almost remember them. But let us say you met 10000 people for 10 minutes, hardly you will remember anybody. That is because of the number 7, that is if the short term memory capability is restricted to 7.

So, if the information has more than 7 items it becomes exceedingly difficult to understand and to remember. If it has let us say 15 items to understand and remember you take exponential time compared to there are 5 items. If there are 100 items then it becomes real difficult.

(Refer Slide Time: 20:18)



**What is the Implication in Program Development?**

- A small program having just a few variables:
  - Is within easy grasp of an individual.
- As the number of independent variables in the program increases:
  - It quickly exceeds the grasping power of an individual...
  - Requires an unduly large effort to master the problem.

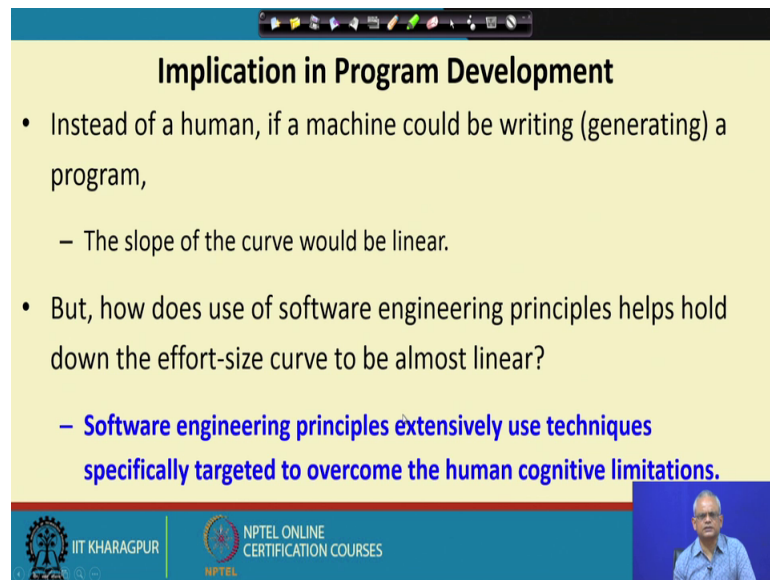
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But then you would ask that what is this has got to do with programming. And how does it provide a answer to the exploratory programming style, the effort, cost, time duration etcetera increasing exponentially breaking down after sometime, whereas, software engineering is able to hold it linearly with respect to the problem size.

The answer is that: to understand a program we must look at what are the variables, how do they interact. Let us say you are writing a small program, having only a couple of variables, you can easily look through the code understand what is happening. But let us say you are looking at a program which has thousands of variables there, and each variable is set by some programming constructs being used and so on. For the human mind it becomes extremely difficult to understand that what is going on in the code, how does it work, what it is achieving, and so on?

So, as long as the number of independent variables in a program is small it becomes easy to understand, but as the number of variables in the code increases it becomes very difficult to grasp, to understand and requires and unduly large effort to master the problem. But then, the next question that arises is that if that is the case, that if the number of independent variables and so on are more than 7 it is a large problem dealing with many things, then how does the software engineering principles they contain the complexity of this problem.

(Refer Slide Time: 22:40)



**Implication in Program Development**

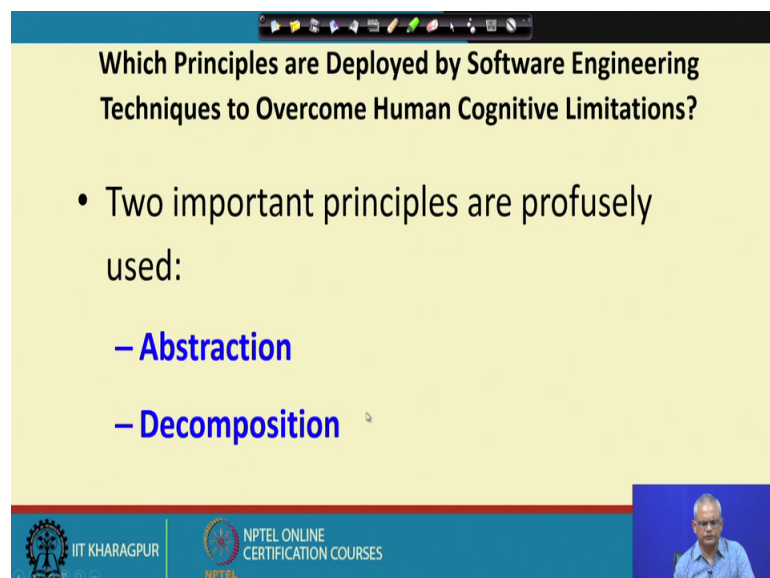
- Instead of a human, if a machine could be writing (generating) a program,
  - The slope of the curve would be linear.
- But, how does use of software engineering principles helps hold down the effort-size curve to be almost linear?
  - **Software engineering principles extensively use techniques specifically targeted to overcome the human cognitive limitations.**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Actually, before we would answer that question just like to mention that for machines there is no such problem as short term memory, of course they have RAM and so on, but then the restrictions are not so severe. So, if the machine would like write the code then the slope would be almost linear.

Now, let us understand how the software engineering principles contain the complexity. Because, we said that as the complexity increases the human mind by itself would take exponential time effort to understand to solve the problem.

(Refer Slide Time: 23:35)



**Which Principles are Deployed by Software Engineering Techniques to Overcome Human Cognitive Limitations?**

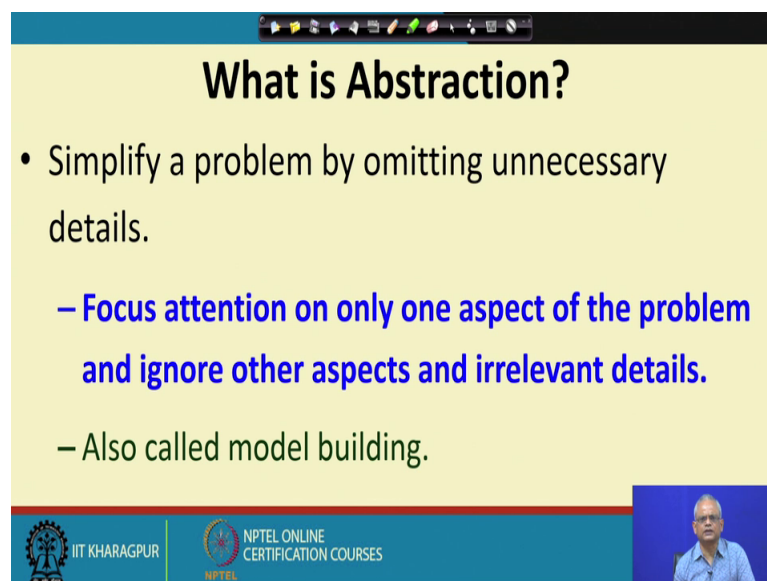
- Two important principles are profusely used:
  - **Abstraction**
  - **Decomposition**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Actually there are two major techniques that are used to overcome the cognitive limitation of 7. These two important principles are used in almost every technique that we discuss in our lectures: one is called as obstruction and the other is called as decomposition. Let us understand what exactly the obstruction and what is decomposition, it is the two independent techniques, how do this reduce the complexity of a problem. And then later when we use these techniques in various software engineering tools and techniques it will become clear to us that whether we are using obstruction or decomposition.

Now, let us investigate these two fundamental techniques to handle complexity. Because, software engineering is after all developing programs efficiently, as the size increases we want to develop only with linear time cost effort and so on. And for that we need to effectively handle complexity increase in complexity. And these two techniques to handle complexity are used in almost every software engineering principle. Let us look at these two techniques.

(Refer Slide Time: 25:08)



**What is Abstraction?**

- Simplify a problem by omitting unnecessary details.
  - **Focus attention on only one aspect of the problem and ignore other aspects and irrelevant details.**
  - Also called model building.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

One is the obstruction: let us see what is meant by obstruction. In obstruction we focus our attention only on some aspect of the problem and ignore the rest. This is also called as model building.

Let us say we want to develop a large building. Now we want to see that how does it look like. We will ignore all other aspect like what are the strength, what is the wall

thickness, what is the internal plan and so on; we just want to see how it will appear. Then we will concentrate only on its frontal appearance and we will ignore everything and we can construct a model of that; the frontal view of the building. Similarly we can, let us say we want to see what will be the floor plan for a building. We will just concentrate on that, we will not bother about how does it appear externally is a thickness of the wall and so on; we were just concerned about the floor plan.

If we are let us say concentrating on let us say what is the hitting requirement of a room; we will just look at how much is the, where is it located, is it in the top floor, how much heat it is getting etcetera, and we will omit other unimportant aspects, for every problem we can construct an abstraction. The abstraction is also called as a model building if we create a model we concentrate on some aspect and ignore the rest. So, model building is the same as an abstraction. And this is an important technique in software engineering for we want to build models of everything, requirements, design, code, etcetera; and that is one way to tackle complexity.

We will stop here and we will continue in the next lecture with this abstraction decomposition and so on. And we will look at the other issues in software engineering.

Thank you.