

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

Lecture – 01
Introduction- I

Welcome to this course on Software Engineering. Now we will just have a brief Introduction to Various Issues in Software Engineering Motivation, why we need to study this course, what are the benefits that we will accrued from here and so on. So, let us get started.

(Refer Slide Time: 00:41)



About Myself

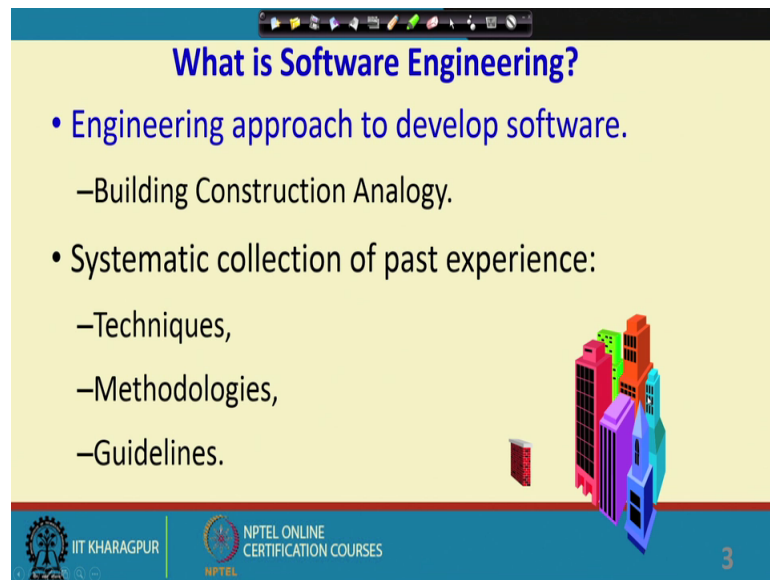
- RAJIB MALL
- B.E. , M.E., Ph.D from Indian Institute of Science, Bangalore
- Worked with Motorola (India)
- Shifted to IIT, Kharagpur in 1994
 - Currently Professor at CSE department

The slide includes two photographs: one of a large, ornate building with a clock tower (likely IIT Kharagpur) and another of a modern, multi-story building with a glass facade (also IIT Kharagpur). The slide footer contains the IIT Kharagpur logo and the NPTEL Online Certification Courses logo.

First is about myself: I will be the instructor for this course, my name is Rajib Mall. Completed all my education, Bachelors, Masters, Ph.D from the Indian Institute of Science Bangalore; that is quite some years back. That is the institute where I graduated. And then worked few years with Motorola India and then have been with IIT Kharagpur since 1994 and currently a Professor at CSE department

So that is the institute, Indian Institute of Technology, Kharagpur.

(Refer Slide Time: 01:32)



What is Software Engineering?

- Engineering approach to develop software.
 - Building Construction Analogy.
- Systematic collection of past experience:
 - Techniques,
 - Methodologies,
 - Guidelines.

The slide features a yellow background with a blue header and footer. On the right side, there is a 3D illustration of several colorful buildings in shades of red, green, blue, and purple. The footer contains the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with the number 3.

To start with any course we need to know what the course deals with, we will also do the same thing. We start with the basic definition of software engineering. If somebody asks that what is software engineering, we should be able to answer based on this. The simplest answer what is software engineering is: an engineering approach to develop software. But then, the question arises that what exactly is the engineering approach to develop software, how is it different from traditional approach to develop software.

To appreciate this distinction between engineering approach to develop software and traditional, now his approach to develop software; we will just look at a analogy with a building construction. Let us say you want to construct a building what would you do? First let us consider that you want to build a small wall like this, that you will be very easily be able to do from your basic intuition that you need some bricks, cement, colour and so on. And you will succeed in developing a good small wall like this. But let us say, you are now asked to build a large building like this, having 30-40 floors a complex. Your basic intuition about construction will not work here. If you actually get started building this with your basic intuition building may collapse, it will never complete, it will be a poor quality and so on.

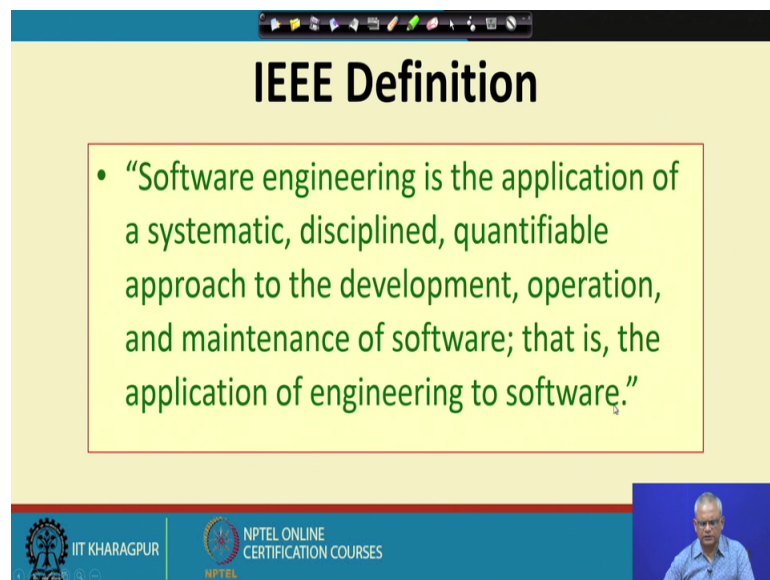
The same thing is with respect to software. If somebody is asked to write a small program let us say 10-20 50 lines of code he will be able to do it based on this basic intuition. But let us consider developing a software which is 100-1000 lines of code.

Now, if he wants to extend the intuition and develop a large software again he will face the same effect as constructing a large building from the basic intuition. The engineering approach to develop software, let us one develop software based on some techniques and tools. So, we look at those techniques and tools as part of the software engineering.

So, an alternate definition of software engineering can be that it is a collection of past experience which has resulted in various techniques for development of large software, methodologies for development, and some guidelines, and tools.

As we proceed with this course we will be clear about, what are the techniques, what are the methodologies, and guideline, tools and so on.

(Refer Slide Time: 05:50)



IEEE Definition

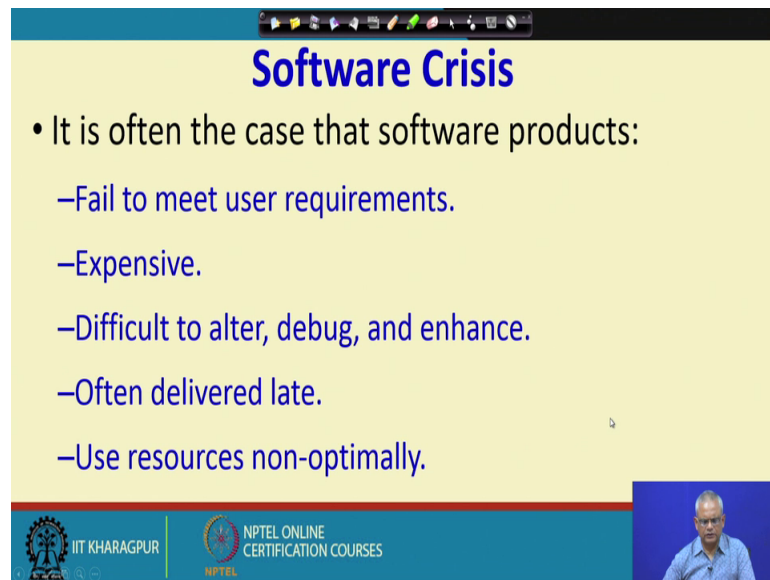
- “Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.”

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at the IEEE definition of what software engineering is. The IEEE definition is “software engineering is the application of systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; and that is the application of engineering to software”.



So, the IEEE definition also says that, software engineering is the engineering approach to develop software and it further elaborates this by saying: application of systematic disciplined quantifiable approach. And all this systematic disciplined quantifiable approach have as you will see been developed from experience of programmers over the last several years.

(Refer Slide Time: 06:54)



Software Crisis

- It is often the case that software products:
 - Fail to meet user requirements.
 - Expensive.
 - Difficult to alter, debug, and enhance.
 - Often delivered late.
 - Use resources non-optimally.

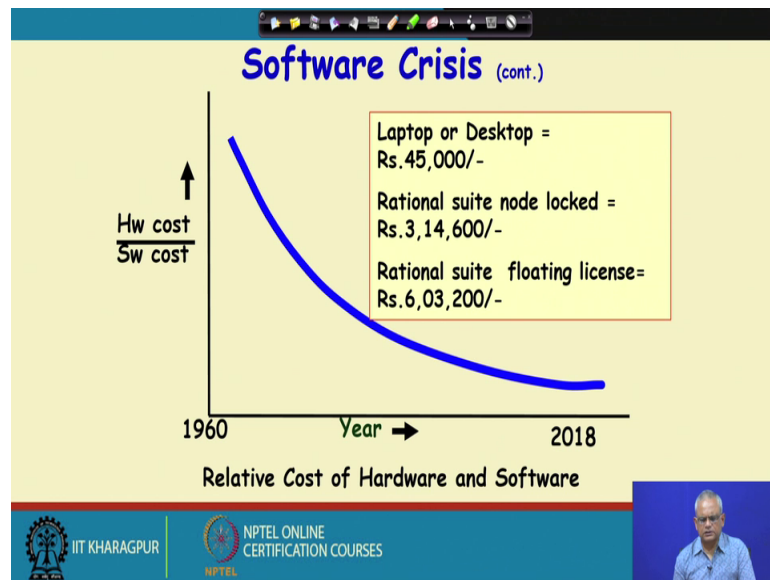
 

To motivate the need for software engineering why it is important for the software developers to learn software engineering; let us just understand what the software crisis is, because everybody says that there is a software crisis.

The software crisis, the symptoms are that they fail to meet user requirements. Software is delivered and then the users feel that it is not what they wanted. Software is expensive much more expensive than hardware and every year the difference between software and hardware prices have been increasing. It is difficult to make any changes to the software to debug if there is a problem reported, it takes long time to correct it. Sometimes the bugs cannot be corrected, difficult to enhance, add new features. It is very usual for software to be delivered late and use resources non-optimally

For most of the software that you order or develop a smarter projects have these symptoms, that they rarely meet the user requirements, turn out to be very expensive, difficult to alter debug and enhance, delivered late, and use resources non-optimally.

(Refer Slide Time: 08:52)

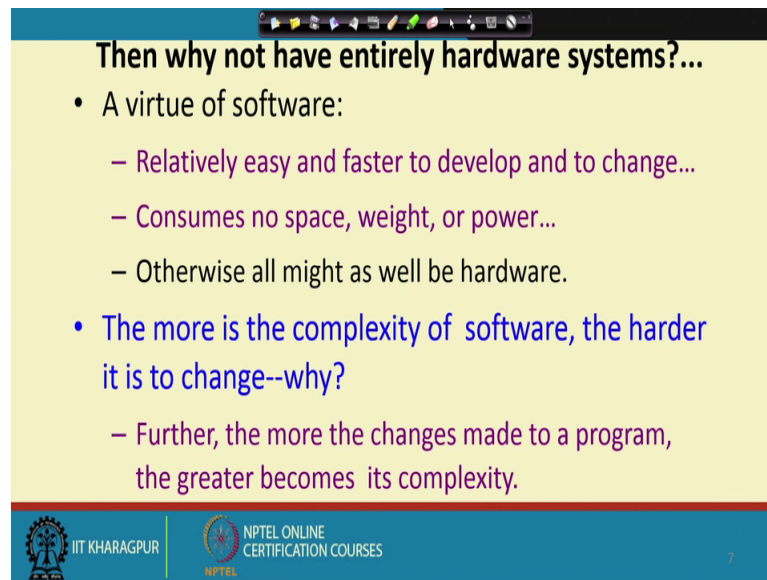


Just to explain how expensive is the software compared to hardware. Over the years if we compute how much a company spent on hardware and how much spent on software. If we plot the ratio: we will see that 1960s are and so, the hardware was the major aspects millions of dollars of hardware, and then software was very inexpensive compared to the hardware. But then, the hardware costs have dropped dramatically and software costs are reasonable.

So, now the hardware cost divided by software cost is almost nominal approaching 0. Just to give a feel. Thus, let us say that a laptop or desktop costs 45000 rupees, and let us say some software let us say rationally it costs 3 lakh rupees; if it is node locked and floating license costs 6 lakh rupees which runs on a desktop. So, just see that cost of one software is many times the cost of the hardware and then we need to run various types of software and hardware.

So, now the software is much more expensive, companies spend much more on software than on hardware. But then the question arises is that why is that the cost of hardware has dropped so much, but then software has not really the cost has not dropped and actually increased.

(Refer Slide Time: 11:03)



Then why not have entirely hardware systems?...

- A virtue of software:
 - Relatively easy and faster to develop and to change...
 - Consumes no space, weight, or power...
 - Otherwise all might as well be hardware.
- The more is the complexity of software, the harder it is to change--why?
 - Further, the more the changes made to a program, the greater becomes its complexity.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

There are many reasons for this. We will examine the reasons, but before that just want to ask a very fundamental question: that if hardware is so good delivered in time, extremely reliable, inexpensive, then why not have everything in hardware; because, after all whatever can be done by software can also be done by hardware. There must be some reasons otherwise people would only be developing hardware and using it and there would be no software. Let us look at the reasons why software is indispensable. And actually the use of software is increasing as compared to hardware.

There are two or three major virtues or strong points of software which actually make it a preferred solution compared to hardware. One of the most of the most important thing is that developing software is easy, faster and also if you find anything to be changed you can easily change it. Hardware development is a long run process, if it is a VLSI chip that you are considering then it is multiyear, starting from requirements to fabrication it is a long process. And if you later want to make a small change again you have to make that long process; change requirements, change design, mass, and so on fabricate, test. So that is a long duration several years.

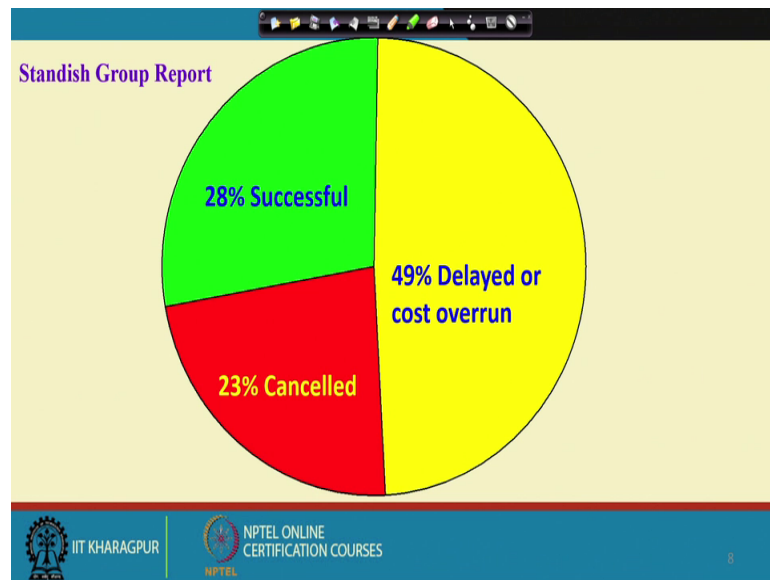
Compared to software making a fix a small change may be just a few days. And also software consumes no space weight or power. If you want to develop something that works in software let us say rational enterprise so, let us consider any software or may be let us say the word processing software let us consider Microsoft Word. It is possible to

have this developed in hardware, but then the size of the hardware will be enormous. It will be extremely expensive and it will consume huge power. But then if you add software it just resides in your hard disk and just gets executed feature by feature. And that is the reason why software is the preferred mode of solution for applications as compared to hardware.

But then, one thing we need to appreciate that software is becoming more complex, and as we change software it also the complexity increases. But then, what is the basic reason? We are trying to answer some fundamental questions here, and as we proceed these questions are actually the basics based on which the other techniques are built up. Must understand these basic principles, that the more complex is the software the hardware it is to change, and also if we change it the complexity of the software further increases; what is the reason. The main reason is that to change the software we must first understand it, without understanding how it works, where exactly we have to change we cannot change it. And if it is very complex software will take lot of effort to understand and then make the change. And also if we make a small change it changes the basic underline design modularity and so on.

As we will see later that, every change actually degrades the structure of the software and makes it much more complex for somebody else to understand. As changes keep on happening to a software may be due to additional features to be supported, may be to correct bugs and so on the software becomes more complex; becomes much more difficult for somebody to understand and make further changes. This is one of the very basic principles as we proceed will need these principles. And it will help us understand why some techniques are the way that they are.

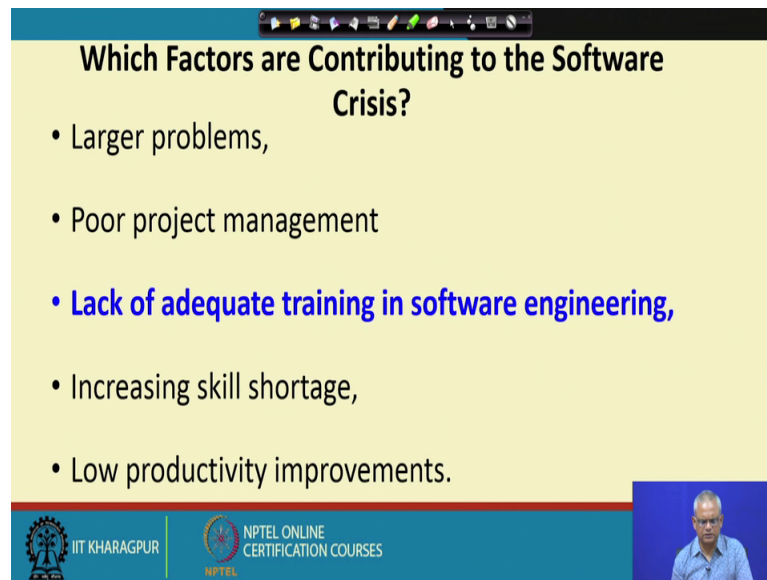
(Refer Slide Time: 16:59)



Now, let us look at the large number of projects that are done. What is the statistics about the success rate of the projects? The success rate is very surprising, that only a quarter is successful: just 28 percent is successful. A quarter is never sees the light of the day they are cancelled. And half of the projects they are delayed, delivered run, delivered late and also the cost overrun the price increases.

But then, why is it that only quarter of the software is actually successful, half of the software is challenged, and another quarter is actually failure, they are to be cancelled. Let us try to understand what is the reason.

(Refer Slide Time: 18:19)



Which Factors are Contributing to the Software Crisis?

- Larger problems,
- Poor project management
- **Lack of adequate training in software engineering,**
- Increasing skill shortage,
- Low productivity improvements.

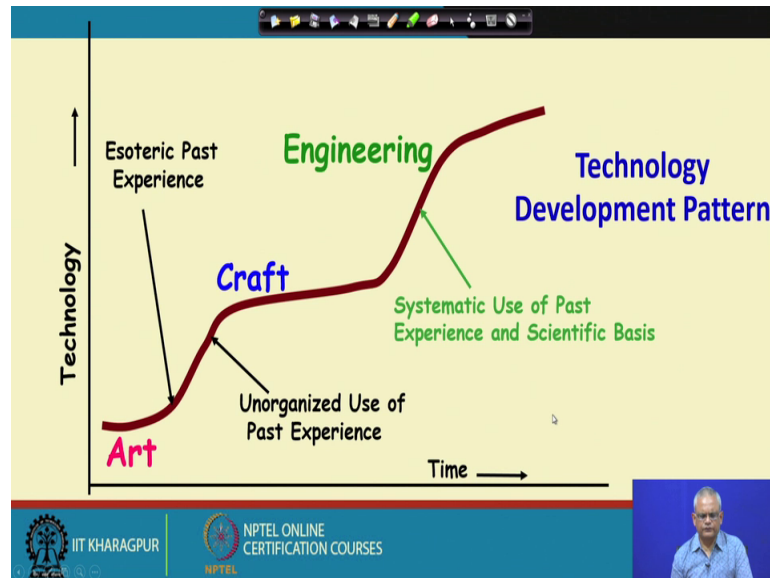
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The reasons are that: now there is a demand for more and more software, the hardware is simpler and the software is much more complex due to obvious reason that any complexity in the application has to be done in the software. Because, otherwise hardware will be very difficult to develop, it will be large; it will consume lots of power volume and so on. So, the problem sizes of software is increasing year after year.

Poor project management, because these are after all entirely manual efforts, the programmers have to develop they have to write the code test and so on. But, surprisingly one of the major reasons that has been identified for the software crisis is the lack of adequate training in software engineering. The developers they are not very conversant with the latest software engineering techniques. When they graduate from a college they do not have proper background in software engineering, they have studied some things that do not really help in developing good quality software. There is a increasing skills or tests as far as software development, software engineering principles are concerned.

And even with software engineering the skill improvements are not really very drastic, it improves moderately, very slowly over the years. Compared to the way the problem sizes are increasing the productivity development has not really kept pace.

(Refer Slide Time: 20:29)



Let us just try to understand another very basic aspect. Might have heard that somebody saying programming is an art, but then they said software engineering and there is an engineering approach to develop software. So, in software, an art or engineering; let us just try to understand the issues concerned.

Now let us look at how the technology develops for any specific domain, whether it is steel making, whether it is paper making. Let us see how the technology develops. We look at the technology development pattern, initially somebody comes up with some idea and then follows it, comes up with solution. Let us say paper making was known to some people in China, and only they could develop good quality paper. The others tried they do not know how to make paper, possibly came up with some very bad quality paper. And the thought that paper making is an art.

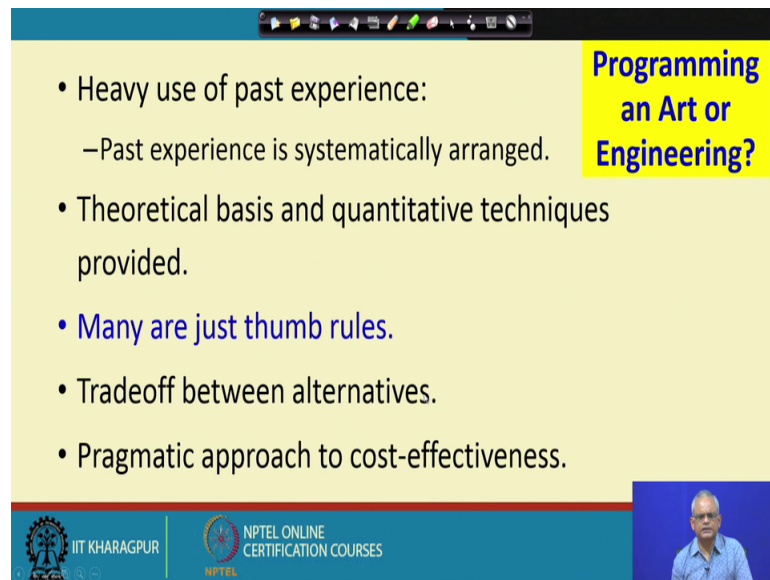
And artist as you know that if you ask somebody that how come he paints so well. Let us say an artist is a draws a nice drawing and you ask him how did you come up with such a nice drawing. You will say that ok, it just comes to me I just draw it like that. Same thing with technology, initially they say that we just know how to make it. But then slowly the art form in technology development graduates to a craft form. In a craft form, there are some techniques they can identify how they make it and they do not share it actually with lot of people, they share with only their apprentice.

It is basically a hidden secret. Let us say paper making or steel making, do not like to share with many people. They know the techniques, but they do not share it. And there are craftsman who know how to do it. So, they are basically the apprentices and then they also share with very less number of people. And slowly it transits to an engineering approach. In engineering the techniques are all investigated. The past experience is analyzed and scientific basis is given to this techniques. These are well documented. And, this can be studied by anybody is their open knowledge and then it has graduated into engineering. But what about program writing: yes, it is same pattern the program writing has followed.

In the 1960s, there were good programmers and there were bad programmers. The good programmers they just wrote good programs and they do not know how they write good programs, but then they just wrote good programs. And other programmers they just struggled and could not come up with good programs. But then slowly the techniques that they were using in writing the good programs were identified and shared with few and that was the craft form. A good programmers to train other programmers, who can become good programmers.

But later the techniques that they were using for writing good programs were all systematically investigated, scientific basis given to them and published in the literature, in the book form and so on, so that anybody can read and then it has graduated to an engineering form where somebody can read what are required to really develop good software, large software, the basic principles one need to know and so on. And our focus is looking at this software engineering techniques.

(Refer Slide Time: 25:35)



The slide features a yellow background with a blue header and footer. The title 'Programming an Art or Engineering?' is in a yellow box on the right. A list of five bullet points is on the left. A small video inset of a man speaking is in the bottom right corner. The footer contains logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

- Heavy use of past experience:
 - Past experience is systematically arranged.
- Theoretical basis and quantitative techniques provided.
- Many are just thumb rules.
- Tradeoff between alternatives.
- Pragmatic approach to cost-effectiveness.

If we look at the engineering approach to develop software there is heavy use of past experience, the past experience is systematically arranged. And, for this experience investigation has been made theoretical basis and quantifiable techniques; quantitative techniques have been provided. But then, for many of the techniques that has come from experience quantitative techniques have not been provided, could not be provided and these are just thumb rules.

There is tradeoff between alternatives, because when we design a software we have to look at various kinds of tradeoff: the complexity of the software, the cost and so on; and because that the tradeoff between alternatives we have a pragmatic approach to cost effectiveness.

Just to summarize our discussion: software engineering is actually an engineering approach to develop software, and this engineering approach has developed from past experience of large number of programmers, over the years the programmers have innovated new ways of looking at how to write program good programs and so on. All this techniques have been systematically organized so that somebody can go through this topics and be a good programmer, we able to write large programs in a team and so on.

So, that is the focus of our discussion and over the next few lectures we will discuss various techniques, issues and so on.

Thank you.