

Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 02
Complexity of Software

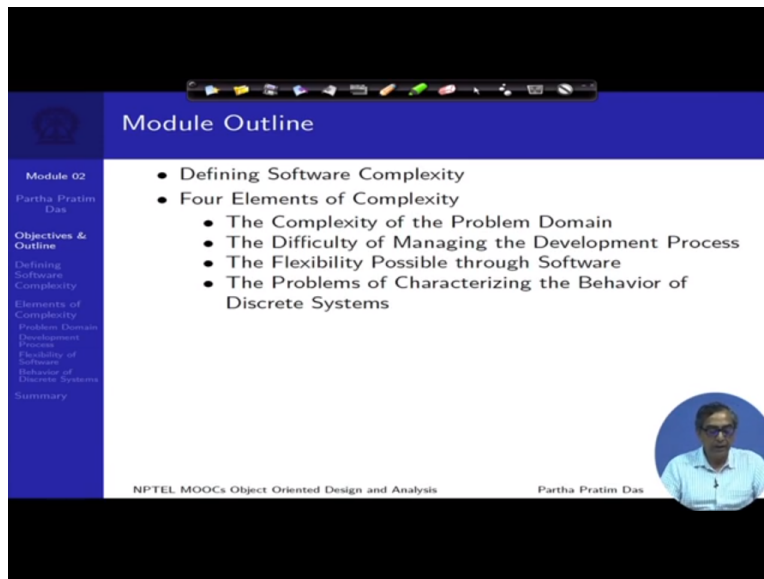
Welcome to module 2 of object-oriented analysis and design course. In module 1 we have set the stage for the requirements of OOAD. We have seen that software development is still at a nascent stage and has a long way to go for confirm success of projects and to achieve a status of a proper field of engineering which can guarantee correct construction, timely completion of projects and within budget delivery and so on. In this module, we look at what is the complexity of software.

(Refer Slide Time: 01:04)

The screenshot shows a presentation slide titled "Module Objectives" for "Module 02" by "Partha Pratim Das". The slide content includes two bullet points: "Understand why Software is Complex" and "Study the elements of Complexity". A sidebar on the left lists the module outline: "Objectives & Outline", "Defining Software Complexity", "Elements of Complexity", "Problem Domain", "Development Process", "Software of Discrete Systems", and "Summary". A small video inset in the bottom right corner shows the professor, Partha Pratim Das. The footer of the slide reads "NPTEL MOOCs Object Oriented Design and Analysis" and "Partha Pratim Das". The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 12:52 PM on 30-May-17.

That is we take a look at why software is complex and to analyze little bit more, we try to understand the elements of the complexity of software.

(Refer Slide Time: 01:19)



Module Outline

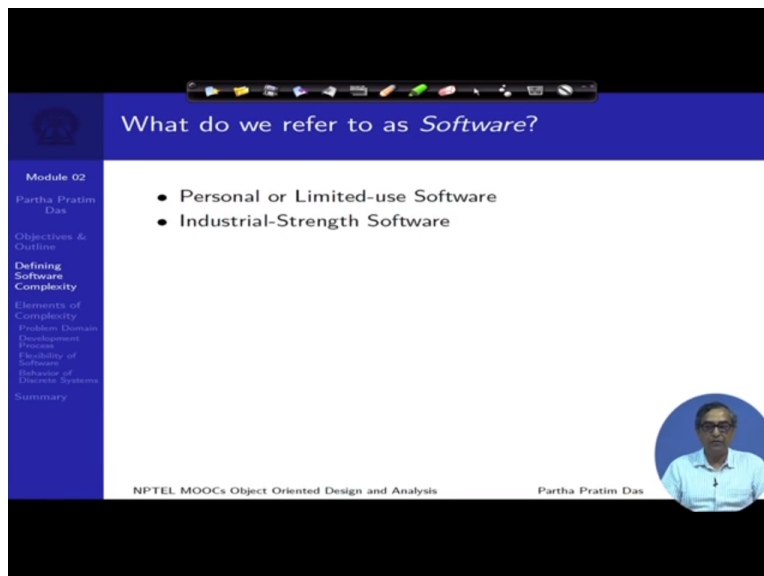
- Defining Software Complexity
- Four Elements of Complexity
 - The Complexity of the Problem Domain
 - The Difficulty of Managing the Development Process
 - The Flexibility Possible through Software
 - The Problems of Characterizing the Behavior of Discrete Systems

NPTEL MOOCs Object Oriented Design and Analysis

Partha Pratim Das

This is the outline and I have already meant mentioned earlier, you will be able to see this outline on the left bar throughout.

(Refer Slide Time: 01:34)



What do we refer to as *Software*?

- Personal or Limited-use Software
- Industrial-Strength Software

NPTEL MOOCs Object Oriented Design and Analysis

Partha Pratim Das

So before we start to take a look into the complexity of software, let us first try to understand what you refer to software. There has to be a, I mean this is just clarifying the (01:50) clarifying the understanding. So software as we know is a, you can always simplify and say that anything you say as a software is finally a program which takes inputs and generates output which is which is a quite a simplest and straightforward view of software.

But we all know that software gets lot more complex than that. So here for the purpose of this course we look at, we divide the software of the world in 2 buckets. One we say are personal or limited-use

software the other we say is industrial-strength software. So let me just quickly explain what these mean.

(Refer Slide Time: 02:35)

Personal or Limited-use Software

- Limited set of behaviors
- Not very complex
- Specified, constructed, maintained, and used by the same person or a small group
 - Amateur programmer, or
 - Professional developer working in isolation
- Tend to have short life span
- Can be thrown away and replaced with entirely new software rather than attempt to reuse them, repair them, or extend their functionality
- Generally more tedious than difficult to develop
- Techniques to develop them is of little interest here

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

The personal or limited-use software has a limited set of behavior, it is not very complex and most importantly it is specified, constructed, maintained and used possibly by the same person or at most by a very small group. For example I recall about 10, 15 years back or while working on a project 3 of us faced with a problem that at those days, there used to be floppy drives, they used to be very very small. So when we wanted to copy one big file from one system to the other.

It did not fit into one floppy. So what we needed is to divide that binary file into 4,5,10 different floppies then copy each one into a single floppy, take that to the other system, copy back and then match that, 10 different parts together into 1. Now this is something which was required which is a engineering requirement of the setup that we had. So we struggled in doing those was, that was not that day when you could just go to Google and type and get a free software which has which somebody has written for you.

So we sat down and wrote a program in c I remember and which will take a binary file and divide into as many file as you want, then copy them one by one in different floppies and take those floppies, copy the individual part and again another part of that program will combine them together. So this also so this is the software, this had functionality, this has served us for about 3, 4 years quite effectively and all, all of us some 4, 5 of us in the group used to regularly use that. Now this is a kind of software which we will say is personal or limited software.

(Refer Slide Time: 04:40)

Personal or Limited-use Software

- Limited set of behaviors
- Not very complex
- Specified, constructed, maintained, and used by the same person or a small group
 - Amateur programmer, or
 - Professional developer working in isolation
- Tend to have short life span
- Can be thrown away and replaced with entirely new software rather than attempt to reuse them, repair them, or extend their functionality
- Generally more tedious than difficult to develop
- Techniques to develop them is of little interest here

Module 02
Partha Pratim Das
Objectives & Outline
Defining Software Complexity
Elements of Complexity
Problem Domain
Development Process
Flexibility of Software
Behavior of Discrete Systems
Summary

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das 5

Because as I had described the behavior is quite simple, it is not complex at all. It is done by a small group ourselves, kind of this group is amateur programmer, we were not exactly armature programmer but we are professional developers working in isolation not as a part of a big team and such software tend to have a short life span and a of course once or twice we needed to make changes, we did that ourselves and such software since they are not very complex has not been done with a lot of effort.

So it is possible when they get dated is possible to scrap them and write new software all over again or make major changes, you do not have to worry about a lot of investment already existing in this software so that somehow even though the functionality is not matching exactly of what you need to get, you still need to continue to use that and bend backwards to write some more wrappers, some more plug-in to make that old software, the decaying legacy software to work under the present conditions.

So such softwares may be often may be more tedious but they are not very difficult to build and the reason I characterize the software is we will not be concerned with, we will not talk about the development of such software in this course. This is not of much interest.

(Refer Slide Time: 06:07)

The screenshot shows a presentation slide with a blue header and a white body. The title 'Industrial-Strength Software' is in white text on the blue background. Below the title, there is a list of bullet points describing the characteristics of industrial-strength software. On the left side, there is a vertical navigation menu with various topics. At the bottom, there is a red banner with white text and a small circular portrait of a man.

Industrial-Strength Software

- Exhibit a very rich set of behaviors – *reactive systems that drive or are driven by events in the physical world*
- Works with scarce resources – *time, space, power, ...*
- Maintains the integrity of millions of records while allowing concurrent updates and queries – *airline booking*
- Commands and controls of real-world entities – *routing of air or railway traffic*
- Tend to have a long life span
- Depended by many users over time on proper functioning
- Usually based on frameworks that simplify the creation of domain-specific applications

**Complexity of Industrial-Strength Software systems
the human intellectual capacity**

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

What we are interested in is, what we call is industrial strength software. Industrial strength software does not mean software that needs to be used in an industry. It is showing that the kind of strength that it shows is what industry you typically use. The sum of the characteristics would be it the software exhibits a very rich set of behaviors for example; it could be reactive systems that drive or are driven by events in the physical world.

For example you may be aware that a lot number of cars today when you break, press the break actually the break happens through software, there is no direct shaft which connects the pedal that you are pushing at the break and the shoes that press against the tyres, the rim of the tyre. So these are, these are different reactive systems that are driven by the events in the physical world. Second typical characteristic would be this software work with scarce resources.

By scarce resource it could mean several things, certainly typically it means time and space that is it must work in in a limited time, many of these are real time in nature, they have small memory to use and off let one of the most scarce resources stand out to be power. You are all using mobile phones, handsets, tablets and so on and am sure one common complaint, most of us have for most of the gadgets that we use, I need my battery runs out very fast.

Irrespective of what system we use irrespective of what use pattern you have, you are always very constraint on power so such software you will have to be aware of scarce set. The third is such software has to maintain integrity of millions of records I mean if you have software which manages 100 students record or 200 bookings and so on, then it is not an industrial set software but it has to be 100s

of 1000s, millions like air line bookings railway reservations and so and so.

It, the software could command and control real world entities but in like routing of airways, railway traffic and so on. Usually they should have long time span this should depend, many users should be depending on them, it will be critical for many users that the systems function properly and more often these are based on frameworks that simplify the creation of a domain-specific application. This last point would become, if you are not aware of frameworks, then this will become more clear as we go through the course.

But these factor, some of these or all of these factors decide that the software is industrial strength. So there could be simple examples like the Indian railway reservation that am sure all of us have used is will be a candidate here. AutoCAD will be a candidate here word will be a candidate here and if you think you will see that the whole match the requirements or characteristics that I am putting down here but my split and match program for transferring binary files from one pc to other will not be a candidate.

So the complexity of industrial strength software systems are important to study. Primarily because if you once you put all these conditions then the complexity exceed the human intellectual capability. It is impossible for any single individual or even for a small group of individuals to comprehend understand the system in totality. So that is a very interesting scenario, we are talking about building a system, we are talking about building a software which nobody understands in full.

But the software together works. So that is where the challenge lies and it is those kind of software that we will be interested in and the development analysis, design, implementation techniques for those software can be facilitated by object-oriented analysis and design and with those aspects we will study the course. So when we will talk about a software henceforth, it will always mean this kind of software. (Refer Slide Time: 10:45)

Software is Inherently Complex

Module 02
Partha Pratim Das

Objectives & Outline
Defining Software Complexity
Elements of Complexity
Problem Domain
Development Process
Flexibility of Software
Behavior of Discrete Systems
Summary

Inherent complexity derives from four elements:

- The Complexity of the Problem Domain
- The Difficulty of Managing the Development Process
- The Flexibility Possible through Software
- The Problems of Characterizing the Behavior of Discrete Systems

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

So the basic point is the software is inherently complex but we can look at this complexity in 4 different dimensions which are commonly known as elements of software complexity. The complexity of problem domain, difficulty of so if I just highlight the complexity arising from problem domain, the difficulty of managing development processes, the flexibility that software offers, the fact that software is soft is a strength of software and that itself is a basic complexity.

And kind of works often are the weakness of the software and finally the behavior of the discrete systems or the discrete event systems. So what we will do is we will take each of these elements and try to see what does a complexity mean in regard to this system, these elements.

(Refer Slide Time: 11:50)

The Complexity of the Problem Domain

Module 02
Partha Pratim Das

Objectives & Outline
Defining Software Complexity
Elements of Complexity
Problem Domain
Development Process
Flexibility of Software
Behavior of Discrete Systems
Summary

- **Domains are difficult to understand.** For example:
 - Electronic system of a multi-engine aircraft
 - Merchant Shipping
 - Online Trading and Reconciliation
- **Functional Requirements** are
 - *Complex to master*
 - Often are competing, even contradictory
- **Non-Functional Requirements** (*usability, performance, cost, survivability, and reliability*) are
 - *Often Implicit*
 - Difficult to justify in budget

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

So, first the complexity of problem domain, so basically if you look at the scenario then it is always the

scenario of 2 parties, on one side is the user of the software or that we in the industry, we typically call them as customers or clients and on the other side is the developer or the vendor who will develop the system. Exactly the developer would have skills, domain knowledge in software but the user would come from her own domain of expertise requirements.

And to be able to develop the software, it is the requirement of the developer to understand the domain of the user. We will just take some random examples an electronic system of a multi engine, now you do not expect the software engineer to understand this electronic system. You do not even expect the software engineer to understand the engine of an aircraft or the kind of control that the engine requires other issues that arise when there are multiple engines on the aircraft.

And the engines need to switch from on to off and speed controls and all that. So it is the big challenge for the software developer to understand this domain a different very different thing, merchant shipping, as you will be aware that 1000s and 1000s of crate is regularly transported across seas all around the world and big companies, corporations manage that. Big ships manage that, that is the activity of merchant shipping.

If you are asked to develop software to manage merchant shipping or certain aspect of merchant shipping, am not even talking about everything in merchant shipping. Certainly you will find it very very difficult; you will not even understand that domain. Online trading and reconciliation, we use net banking, many of us may be using the online trading to trade on stock market but if you want to really develop the software,

You will need understand all the aspects of what happens when you plays a transaction, when you plays you buy or sell a order and how does it go through, what kind of synchronization does it require, what is said to be a match of a order between buy and sell orders, how these are reconciliated, how is money really is taken out from the bank account or credited to the bank account, how are demats operated and so on so forth.

So am just trying to randomly talk about different domains to make you understand that it is a really difficult to understand the domain. A second issue with the complexity that comes from the problem domain, is functional requirements are complex to master. In the background of the back that you have not understood the domain, so all that you have done is possibly skim through some pages that the

customer has given you.

And then have done some blind search from google to quickly educate yourself or familiarize yourself with some jargons of that domain but when it comes to functional requirements so when you said that a dead fred will have to be reconciliated with the EOD, you will not find it easy to really understand what this whole functionality means. So they are complex to master and often the requirements are competing and contradictory.

To take it as well be contradictory just couple of days back, I have been sitting with a team to develop who are developing a digital library. This digital library should be available at a national level, several, 100s and 1000s of people will use that. So on one side, the the requirements of the security team is that each and every individual must be known to the system, each and every individual must have been authenticated through biometrics and so on

To make sure that nobody misuses the library. The other side of the requirement, the different page of the requirement says that this is the national resource, a national asset so this digital library should be assessable to one and all in the country. So you can easily make out that both of these are real requirements of the user or the client. The client is being honest but these requirements are contradictory they are competing requirements and this is just a simple example.

When you go to more and more complex domains, you have more and more complex ah situations to deal with. The third, we have a lot of non-functional requirements in a software. By non-functional, what is meant is which is not evident, that is, is the software user friendly, it could be functionally good but is it user friendly. Does it have a good performance that is does it perform the task within the expected time, what is the cost of the software.

How long with the software survive, is it reliable, how often does it crash, so and so forth. Non functional requirements are very difficult to deal with because users do not understand non functional requirements so users will tell us, the developers that, this is what I need. The user at best will give a vague idea of a functional requirement in incorrect terms but non functional requirements, most of the times the user will be silent about or will tell you something which is not at all very meaningful.

So a non functional requirements are often implicit and since they are implicit, when you will consider

them in your design and say well this is the budget that you will need to put up, the customer will come back strongly saying that well this is, this is not justifiable, am not getting any functionality in this. The customer will never understand why you want the software to survive 10 years and why there is a separate course to make the software survive 10 years.

The customer says yes I have get the software, I will use it for 10 years. So these are some of the core difficulties that happen in the, due to the complexity of the problem domain.

(Refer Slide Time: 18:34)

The Complexity of the Problem Domain

Module 02
Partha Pratim Das
Objectives & Outline
Defining Software Complexity
Elements of Complexity
Problem Domain
Process
Evolution of Software
Release of Discrete Systems
Summary

- **Communication Gap** between *Users* and *Developers*
 - Users *cannot express*; developers *cannot understand*
 - *Lack of expertise* across domains
 - *Different perspectives* on the nature of the problem leading to *different assumptions* regarding the nature of the solution
 - *Few instruments* to precisely capture requirements – large texts with some drawings is all we have
 - Leads to *External Complexity*
- **Changing / Evolving Requirements** during Development
 - Early products lead to *better understanding of needs* by the users
 - Developers get *enlightened through the process of development*

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das 9

Now let us continue further. There is a there is a notion and many people refer to it as external complexity which is basically the starting point of any software development project. So just think of the scenario, there is on one side is the user, the customer, who has some needs so would has called the vendor, the developer has gone there.

So the first thing what has to happen, the customer has to explain the requirements, the needs to the developer. Now in this, so this is a purely whatever way you do it this is a human-human communication process. And we know that any place you make a communication, the first thing that happens is there is a gap in communication. And the severe gap of communication between the users and the developers is one of the greatest bottlenecks of smooth development of software.

The first thing is users cannot express the other side is developers cannot understand. Users cannot express because users talk in their language, the language of their domain. Developers cannot understand because developers are software developers, they do not understand the language of the

domain. They talk in the language of their software. So when a broker talks to a company that I need a back office management system for demats, the broker very fluently says that well

These are the different kinds of demats it will have and there should be a possibility you should support a placing of demats and so and so forth. to the software poor software developer that is all greek the software does the developer does not understand what placing a demat mean. On the other hand when the software developer says that look you this has to be real time and so on and I will need this kind of a server, there will have to be such network router boxes and so on

Exactly the customer is totally at a loss, is to what is this going on. Where is my shares where are my demats, what is my back office. So the lack of expertise across domains is a major issue of communication gap and that gives rise to different perspective on the nature of the problem. So the perspectives of the developer and that of the customer that of the user are very different in terms of what the problem is.

The user sees one kind of a problem; the developer sees a different kind of a problem. The developer provides a solution to the problem that the developer sees and that is not the solution that the user sees. So it is a complete I mean it is a perfect recipe for disaster. Now in this process to minimize the communication gap unfortunately we have very few instruments and most projects till date works with large volume of text written in natural language with some drawing at the best is all that we have.

And certainly we know that even a simple sentence like I will go to your home tomorrow can be understood in several ways. I will go to, your home tomorrow. I will go to your, home tomorrow. I will go to your home, tomorrow. Some sentence but you all understand it means different things so when a complex requirement is written down in terms of natural language, certainly it is open to several interpretations and developers more often make that interpretation which the user never mind

And this is one specific aspect which we will expand on significantly in this course and this will be the basis for, the lack of instrument is the basis for designing using and promoting the unified modeling language which is far away from any natural language, it is very close to diagrams. Diagrams, icons or what we understand universally irrespective of culture, age, education and our mother tongue. So but this remains to be a big uhh complexity of the problem domain and all these lead to the external complexity as I started saying.

The next issue is what I touched upon also little bit in the in module 1 is changing and evolving requirements is a reality of a software development and requirements change for 2 reason for the, on the user sides, the requirements keep on changing because more and more the software I see developed. I understand actually what I needed; I understand that what I see is not what I needed, so I change the requirement. Even on the other side, on the developer side, more and more the developer develops,

She start becoming a better expert of the domain in which the software is being developed. So after 5 months or 6 months of development, then the developer realizes, understands very clearly that there is only one kind of demat, everything else is a specialization hierarchy where such and such and such can be done. Dint see that when this was introduced to her at the time of giving the requirement. So changing or evolving requirements is a reality which adds to a lot of complexity to the problem domain.

(Refer Slide Time: 24:12)

The Complexity of the Problem Domain

Module 02
Partha Pratim Das
Objectives & Outline
Defining Software Complexity
Elements of Complexity
Problem Domain
Development Process
Flexibility of Software
Behaviour of Reactive Systems
Summary

- **Large Capital Investments**
 - Make *scrapping of projects* unfeasible with changing requirements
 - Lead to *inordinate percentage of software preservation*

Maintenance	<i>Correct errors</i>
Evolution	<i>Respond to changing requirements</i>
Preservation	<i>Use extraordinary means to keep an ancient and decaying piece of software in operation</i>

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das

Last but not the least is software projects, large software projects have huge capital investment so when the requirements really change, when the scenarios really change because the business situations may have changed, the business process may have changed, the technology may have changed, it immediately does not demand practical to scrap the project it gets unfeasible with this changing requirements. So what we have to do?

We will have to somehow tweak around and make the old software work in the new way. More often these changes are referred to as maintenance and I agree with Booch the author of the book who says

that no, this is not maintenance. True, this is not. Maintenance is when you have errors in the software you correct that. That is maintenance but when genuinely you respond to changing requirements, that is evolution and finally when you bend you back, bend backwards to make the legacy software,

Decaying software, ancient software to fit the today requirement somehow you are basically doing software preservation. Naturally the software preservation requires a inordinate percentage of effort by the developers and we need to also build techniques, we need to also make our processes such you can not only analyze and design and implement but we should be able to maintain, evolve and preserve the software using our techniques. So when you focus on these, all these will have to be equally emphasized.

(Refer Slide Time: 26:11)

The slide is titled "The Difficulty of Managing the Development Process". It is part of Module 02, presented by Partha Pratim Das. The slide content is as follows:

- Create **Illusion of Simplicity** to hide external complexity. Hence code bases get large
- Code bases are **large in size** (LOC: 10^5 to 10^6 & more)
 - In spite of the *use of smart languages*
 - In spite of the *reuse of designs and codes*
 - Goes well beyond the comprehension of a single (or small group of) individual/s – even with meaningful decomposition. Hence, we need teams
- Challenges associated with **team development**
 - Large code bases mean more teams with *more developers in more geographies*
 - More developers means *more complex communication* and hence *more difficult coordination*
 - Key management challenge is to maintain a *unity and integrity of design*

NPTEL MOOCs Object Oriented Design and Analysis Partha Pratim Das 11

Let me now move on to the next element of software complexity that is difficulty of managing the development process. The whole, why do we make software? To solve problems. Why do we make software? So that complex problems can be solved in simple terms. So I need to do a, do an analysis of a stress for building a bridge. I want to do a finite element analysis which is a very complex process but I should be able to do that with the press of a button.

So the reality of what we do and the illusion of that the software has to produce are very different. Software has to be of illusion that is very easy. Matlab on instruction and you get the fu Fourier transform done but that Fourier transform is a very complex mathematics. So the software has to hide the external complexity the gap between the domain and the instrument of delivery in the software which results in the software is getting large in size, huge code basis.

Complex software typically have the loc, loc stands for lines of code ranging from 100s or thousands to a million or more and this is given that we are not today not coding in assembly language or a low level language. We are coding in smart high level languages in spite of the fact that we will take all efforts to reuse lines, reuse codes but the fact remains that code base finally turns out to be finally huge that it goes beyond the comprehension of a single individual or groups.

Even when we have done a very meaningful decomposition so what is the consequence? Unit teams. Unit teams, software cannot be developed by individuals or very small group of people. Teams preferably should be created small but unit teams and the moment unit teams, now you have a new dimension to software development which is team development. If the team does not work, the software does not work. So large code bases would mean large teams, more developers.

You will not get more developers at one place, economics will come in. You will have more developers in more geography, some in India, some in Brazil, some in Poland, some in US. More developers will mean more complex interaction, more misunderstanding, more difficult coordination and the key management challenge remain to be how do you maintain a unity, how do you maintain a integrity of the design of the software in spite of all these challenges.

So huge amount of complexity is not from the technology or from the problem domain it is from the process of development because it is a human intensive, because it is a process where the intellectual capability has to go in very strongly. I will stop here and take the remaining points separately.