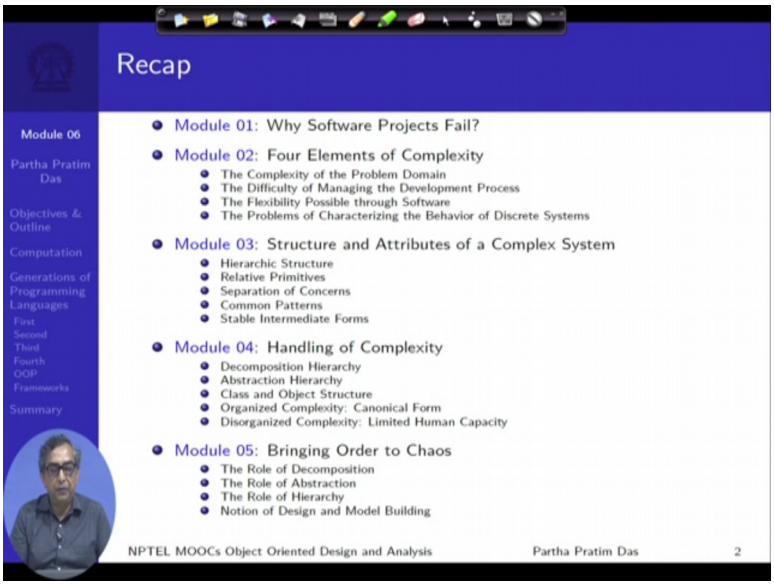
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture - 09
Evolution of Object Models - Programming Languages and Paradigms

(Refer Slide Time: 00:30)



Welcome to module 6 of object-oriented analysis and design. We have discussed in the last 5 modules, we have discussed various aspects of challenges faced by complex software development that is given a typically complex system, how to analyze the structure of that, what is the commonality between the different structural and behavioral aspects of complex systems particularly even going across domains.

And based on that we have been able to summarize into few common strategies of decomposition of abstraction, of hierarchy extraction and so on and from this module onwards we will slowly start engaging these techniques, these strategies into actually analyzing different real-world situations to develop their software systems, to design their software systems and in this regard, before we actually start doing the analysis and design.

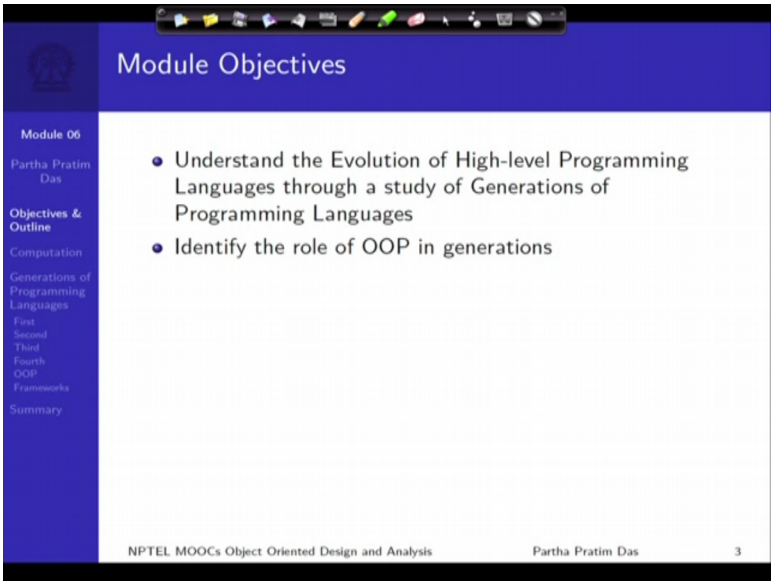
We will take a quick look in the target system on which we can actually implement this software in terms of the programming languages and programming infrastructure that has been available

to us over the period of time when all these different object analysis and design methodologies have evolved. So, once we are able to do an analysis and create a design, certainly a task remains as to how to express that design in terms of a programming a language.

So, the basic definition of a software is kind of an equivalent to in a in a certain way, equivalent to coding my thoughts, coding our thoughts in terms of certain programming languages and therefore on one side, the one part of the problem is to understand the real world, understand the problem domain and realize what needs to be done for providing a solution and the other side is understand the computing system, understand the programming language.

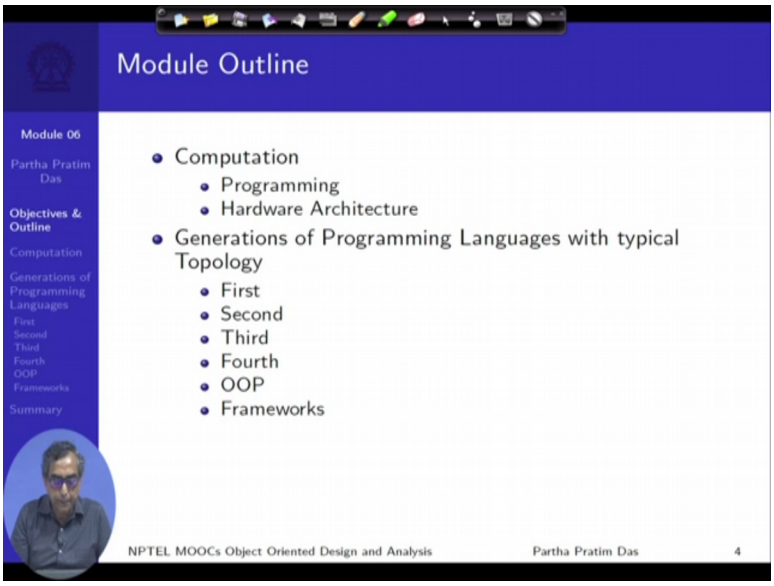
Understand the capability of expression in that language so that we can bridge the gap as software engineers and really implement the designs that we do in terms of the programming systems that we would have at our disposal.

(Refer Slide Time: 03:27)



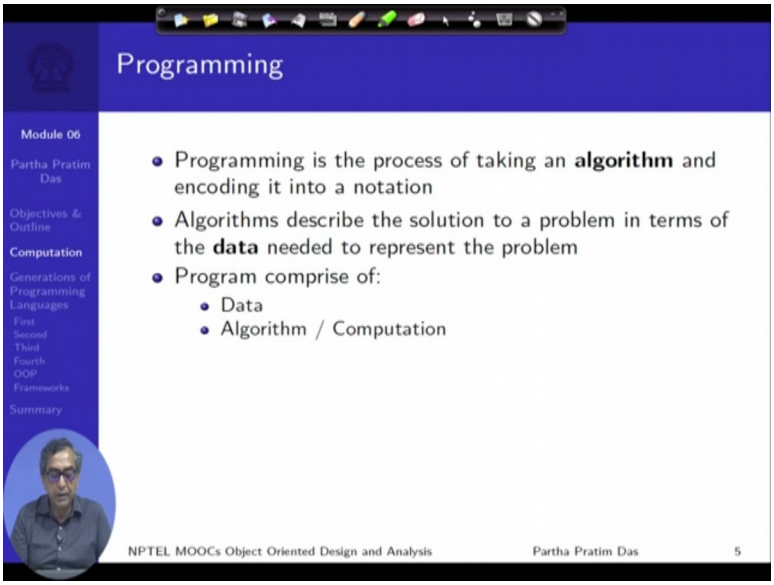
So, to have a good appreciation of what the current situation is we take a quick look into the historical evolution of high level programming languages through the study of generation of programming languages and we will try to see how oop object-oriented programming or object-oriented concepts have evolved over this period of time.

(Refer Slide Time: 03:47)



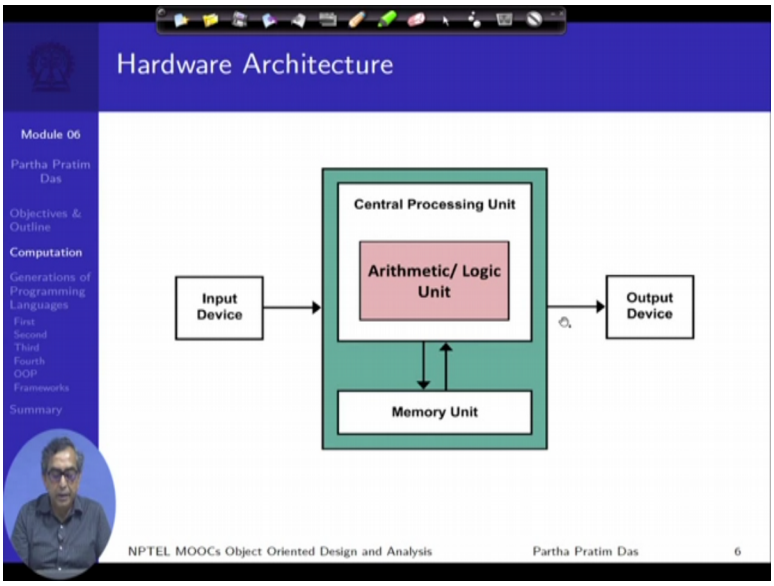
This is a module outline, it will be available on the left of our slides.

(Refer Slide Time: 04:01)



So certainly, the first aspect that we are trying to address is computation which is or commonly known as programming which is known as process of taking an algorithm that is a well-defined sequence of steps and encoding it into an notation which we typically say is a program, the notation has to belong to one well defined programming language. The algorithms describe the solution to the problem using or in terms of the data that need to represent the problem.

(Refer Slide Time: 04:35)

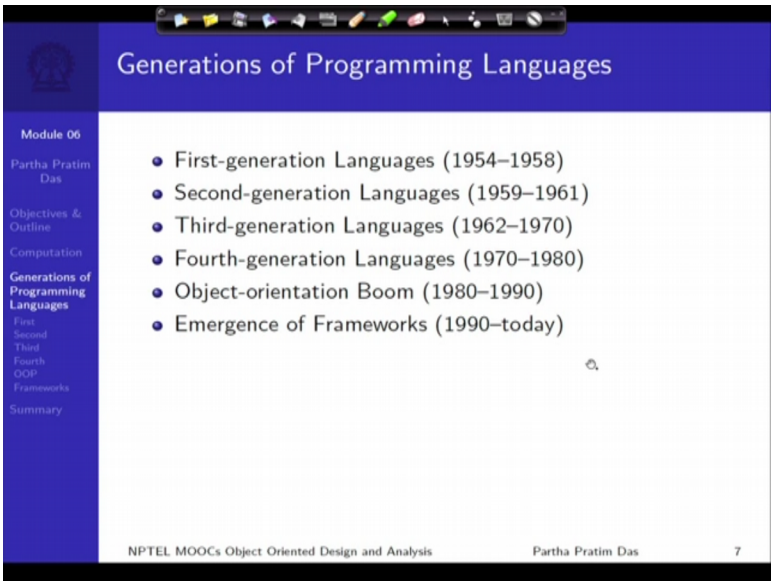


So, we can say that the program typically consists of data and algorithm or computation and though they have been several evolutions of the computing machine, typical hardware architecture or the typical schematic view of the hardware system on which programs execute continue to remain a sample von Neumann architecture where there is a central processing unit comprising an arithmetic and logic unit doing the basic mathematical.

And logical operations and along with that different control units to control the sequence of execution of instructions on CPU memory unit where temporary values, variables can be stored, there could be additionally secondary hardware storage and so on. There will be an input and there is a output divided. So, this is the basic computing model and uh am sure all of you are aware of this model, I just put it up again here.

Just to emphasize that as we evolved into object oriented paradigms our computing model continues to remain the same relatively simple model that have been evolving over the last couple of decades.

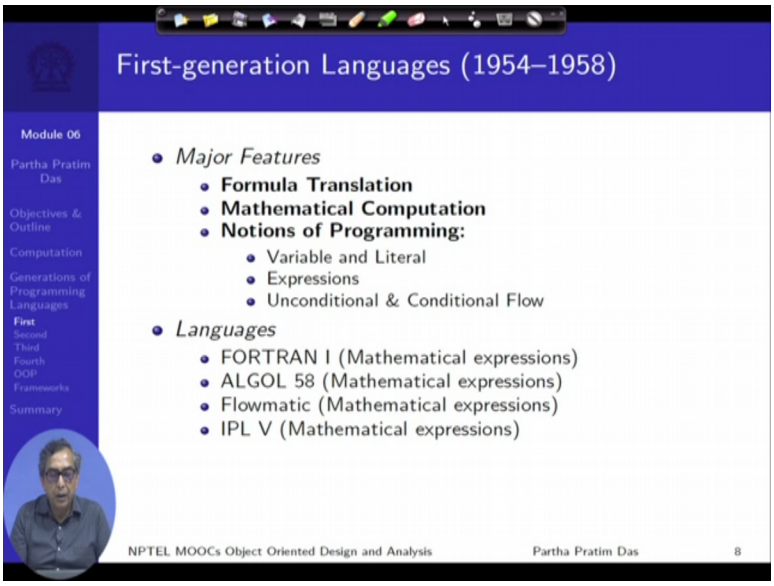
(Refer Slide Time: 05:54)



Now the programming languages all of you am sure know one or more of programming languages and you would be aware that programming languages have evolved and we are talking about high level programming languages not really binary or assembly kind of which is looks more like coded strings. We are not talking about those languages. We are taking about high-level languages.

And there have been several generations programming in high-level language started somewhere in the early 50s and the mark generation started from 1954 and kind of different generations have passed and we will just quickly take a look as one set of languages evolve from one generation to the next what has significantly happened in terms of the language and the computation model.

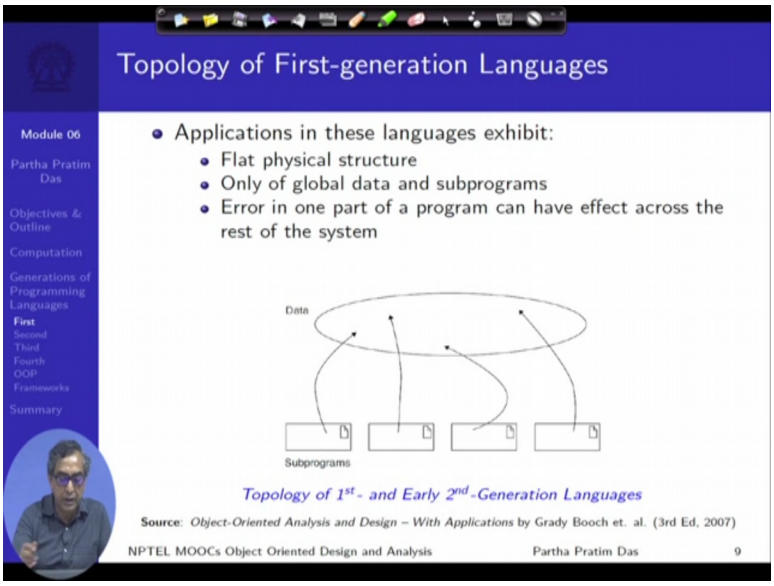
(Refer Slide Time: 06:47)



So, we start with the first generation which certainly is pioneered by Fortran the language known for formula translation. so naturally the major feature of first generation programming languages was to solve arithmetic problems, there you can call them mathematical computation but significantly they are arithmetic problems. So, they translate certain formula so if I know the value of radius of a circle.

Then I can write a program to find the circumference of that circle or the area of that circle using certain formula in Fortran kind of a language and certainly this is the beginning of the high-level language so the basic notions of programming that got introduced during this time include literals that is constants like 5, 10, 2.3 and so on. variables expressions with variables with some basic operators and a conditional and unconditional flow of control though in languages like Fortran or Ipl v these control flow were very archaic rudimentary and was kind of very cryptic in nature.

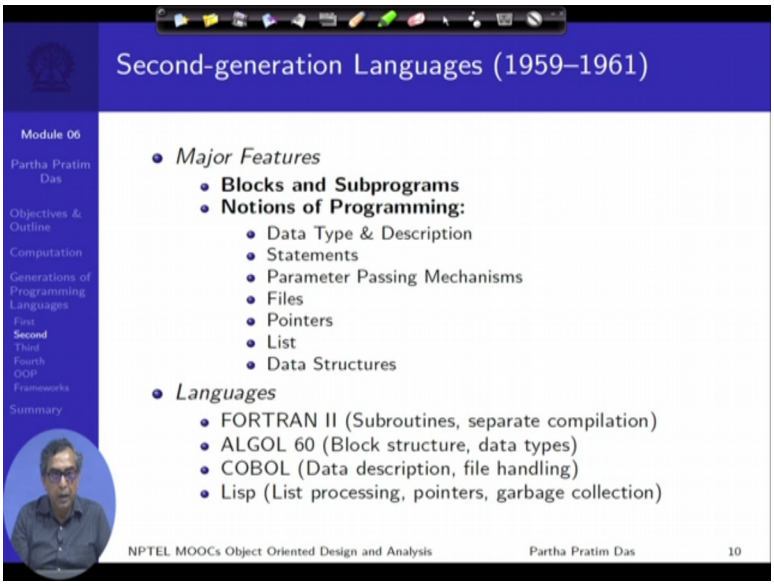
(Refer Slide Time: 08:02)



So, if we look into the topology and if in every generation we would try to look at the topology to see how the computation actually behaves between the data, this is the data and the program or the algorithm, we said that computation is basically an interaction between these 2. So, in the first generation we had in terms of the data, we had a typically flat physical structure, data used to be in a flat structure, possibly one single data only global.

And then you have number of subprograms, these subprograms are what we call these days as functions or subroutines. So, some subprogram would be adding numbers, some subprograms will be printing a value and so on. and any error in one part of the program certainly can affect across rest of the system. so, this was very rudimentary and risky and it continued from the first generation to the early second-generation programming languages.

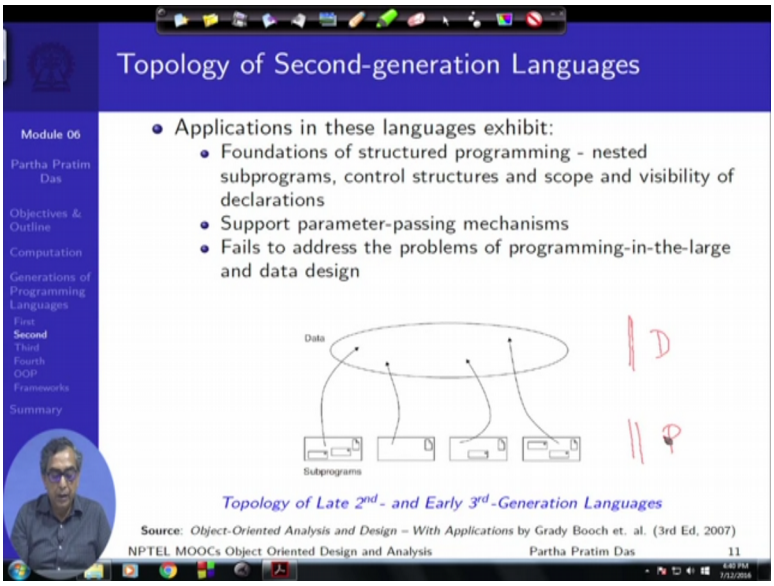
(Refer Slide Time: 09:10)



The second generation which started around the end of 1950s and was relatively a short one saw the emergence of evolution of the Fortran language we had Fortran 2 which is kind of what some of us have had the opportunity to use when we were students like you all, this was the first version of Fortran which had a number of additions to the notions of programming. It had certain well-defined data types, it had a number of different statements, parameter passing mechanisms, files a some not Fortran

But languages like ALGOL started introducing pointers that is address variables so you could create some kind of a list and specific languages like lisp was created to manipulate least kind of structures, data structures started happening. So, these are the though it's a kind of typically marked as a short generation in terms of the lifespan of the programming languages but it was a very significant generation which introduced lot of the fundamental programming concepts in the whole computing module.

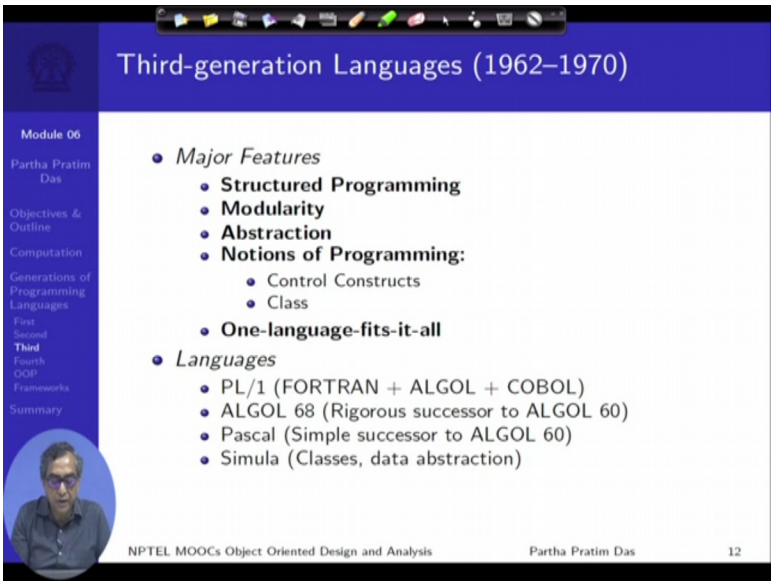
(Refer Slide Time: 10:28)



Looking at the topology again the same we have the data here, we have the program here, but now the basic foundation of structured programming is starting to come in. so earlier if you if you look into the earlier the subprograms were just like this, subprograms are just one single blocks in one single code. Now you have certain structured constructs like loop constructs, condition constructs which are started to get introduced.

And subprograms are divided in terms of those so those kind of you know structured constructs are being depicted here but they continue to work moderately on a flat pool of data supports better parameter passing mechanism but because as the whole collection of data is kind of flat, the second generation of languages fail to address the problem of programming in the large data in case of large data, things naturally became very unmanageable, unwieldy to work with.

(Refer Slide Time: 11:43)

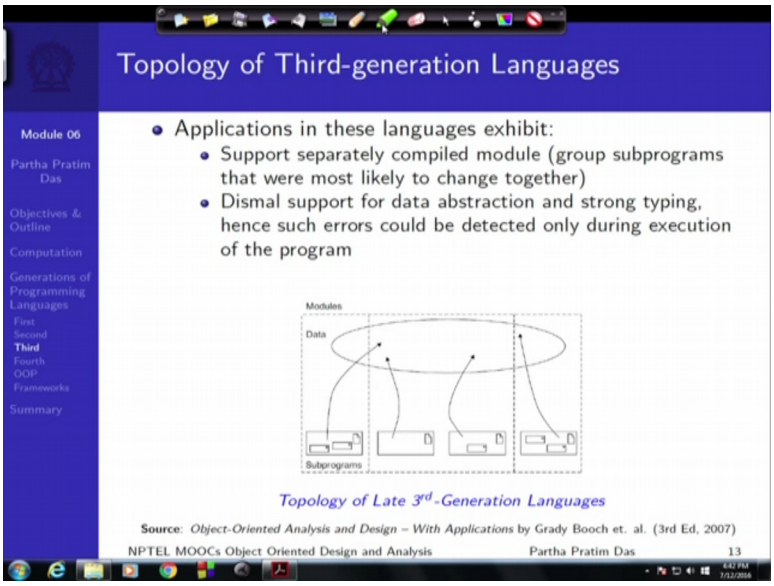


Moving on the third generation spanned relatively longer almost the whole of 60s where several new kinds of path-breaking languages starting started to happen like ALGOL had a next version which is kind of considered one of the precursor to the recent programming style with a with its strong structured programming introduction with its modularity, Pascal the first very simple

Yet robust language for programming was introduced, it was strongly typed and for several years we studied and taught Pascal to our students kind of as we do for c or python these days and this is what saw the emergence also of languages like pl/1 pl stands for programming language 1. So, kind of since as the programming started maturing more and more constructs were getting identified, class got introduced, abstraction got introduced, happened in ALGOL.

So, an idea that started doing rounds is we should have one programming language which solves all kinds of problems. So, one language fits it all kind of an idea and pl1 is one of the biggest examples of that happening which turned out to be a huge language and could you could almost do anything and everything with that language.

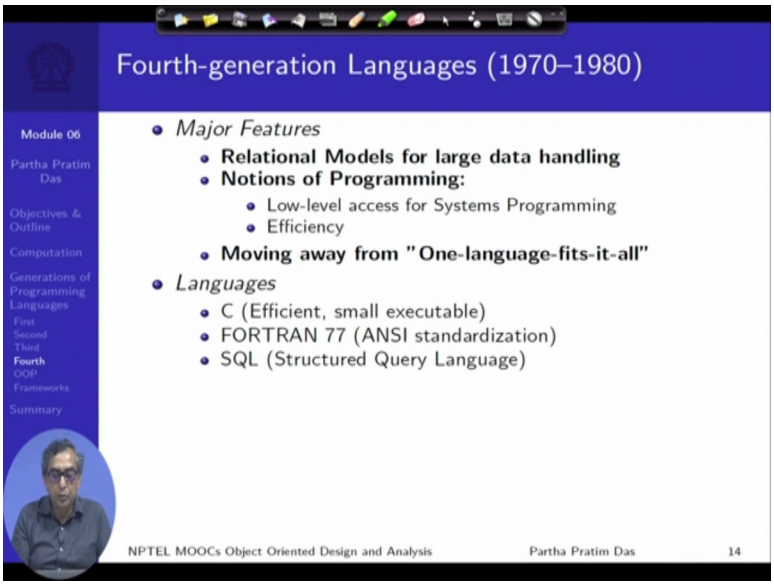
(Refer Slide Time: 13:16)



So that's with that state of affairs the certainly the topology saw some significant changes what is the data is here, the programs, subprograms are here but what you see significantly are demarcations between the subsets of subprograms which was achieved because the first time many of the languages started loving multiple source files which could be separately independently compiled, groups of programs were introduced.

But yet the support for data abstraction or strong typing Pascal as I said had a little bit of strong typing but in general strong typing and abstraction were not widely available in programming languages and therefore handling errors when data starts becoming larger and larger still remained to be a nightmare with the programmers.

(Refer Slide Time: 14:18)



The fourth generation the 70s typically saw the emergence of one of the most popular, 2 of the most popular languages so to say of today. It saw the emergence of c which was created somewhat accidentally by a team of researchers, we were trying to write an operating system from scratch, trying to write Unix and to write Unix certainly it was not fun to write it in assembly so they wanted to create a high-level language for that.

High-level languages could not be used for writing assembly, writing operating systems or writing systems, programs because high-level languages could do only high-level stuff like taking the radius and finding the area and circumference of a circle but they were not equipped to actually go to bits and registers and ports of the hardware and make changes as an operating system or as a systems programs would need to do.

So, coupled with these ideas c was born amongst the people who were developing Unix and pioneers we all know the names of gar Nening Hannon and Richie and there are several others in the team as well and it gave birth to a language which was on one side satisfying the basic requirements of the high-level language at the same time, it was providing the low-level access that is required for systems programming.

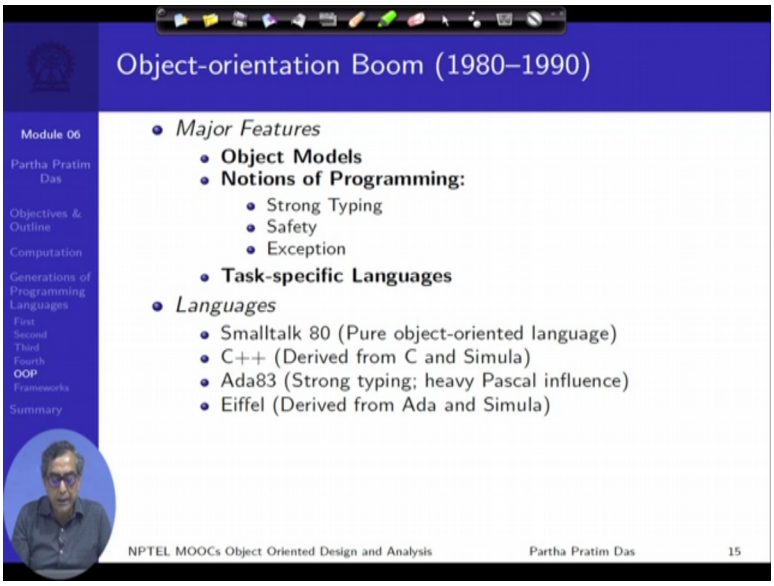
The c was also characterized has always been characterized for the efficiency of the programs that that can be written in c. the other significant development during this time is the evolution of

systems specifically dedicated for handling large volumes of data may be not doing complex operations but being able to do bulk operations of certain style and these came to be known as relational database systems or relational models for large data handling

And that gave rise to a popular language known as structured query language or sql which is still predominantly used today in several rational databases to query different large volume data sequences but c and sql remained to continue to remain in independent domains where on one you in c you are looking at more complex logic even systems programming but relatively less volume of data, in sql you are looking at managing large volumes of data but less complex logic.

So, there were certain bridge languages also which were generated in this in this generation which were created in this generation like embedded sql where you could write sql inside a c program or kind of you could call c programs routines from sql and so on and it was dominated by a very strong growth which on one side gave a strong impetus to impetus to generate a I mean designing languages which could solve more complex problems.

And other side could handle bigger volumes of data and as you can see people started realizing that one language fits it all is not going to solve all the problem. So, we see the emergence of different languages for different purpose, c for systems programming, sql for database programming and so on. Fortran 77 also gained a lot of momentum; it is a revised version or Fortran which borrowed lot of other good features from different programming languages.
(Refer Slide Time: 18:19)



The next kind of the next decade, the 80s started seeing the emergence of the boom of object orientation as you would recall from earlier generations, the basic concept of modularity, abstraction, decomposition, these are slowly getting introduced in languages like ALGOL certain part in Pascal, in pl/1, c had some of those but substantially object-oriented or strongly backed by object-based thoughts, the languages started happening in the 80s.

And saw the emergence and establishment of c plus as a dominant language which at present is called a multi-paradigm language on one side, it has strong procedural flavors and you can write very strong algorithms which extremely good efficiency in C++ at the same time, you could do very strong object based modeling in C++ as well. So, the major feature of this decade has been the boom of object models, strong typing came in in a strong way.

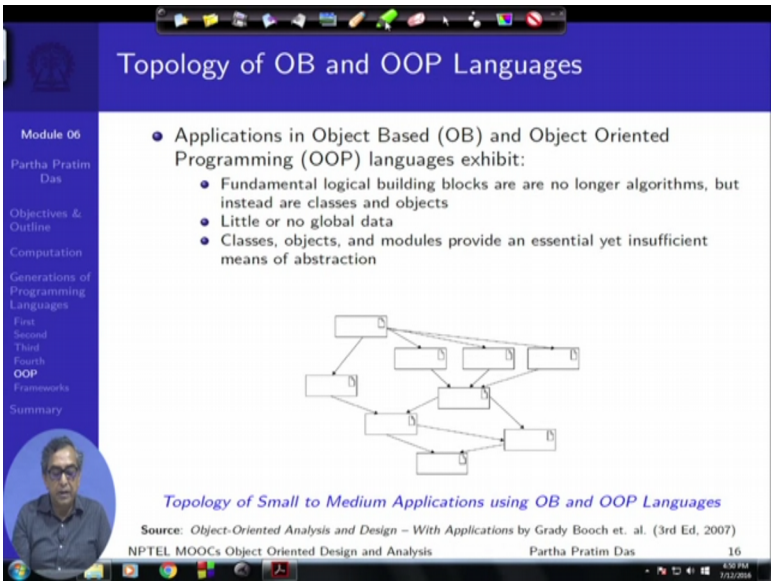
And several new concepts started coming into programming people started talking about safe programs that is programs which can which you can rely on because during this period, several embedded systems started using software as components and for example Boeing started using software components for controlling the aircraft engine. So, the correctness of the software the guarantee that the software will not fail.

The guarantee that the software is safe became critical for human survival, software was heavily put into sending rockets to space and so on. So, safety concepts of safety, concepts of exception

that is taking care of conditions or paths or you know situations in the software which has not been thoroughly thought of became came strongly in terms of different languages, several experimental languages happened.

Beautiful ones like small talk which is very pure object-oriented language unfortunately it has not been very popular in commercial terms ada has had a contribution in terms of strong typing, Eiffel a beautiful language coming from ada on similar which gives a really powerful modeling in object based terms so these are different possible but none of them have besides C++ have made really a mark in terms of large volume of programming.

(Refer Slide Time: 21:09)



Now if you quickly look into the again the emergence of the topology in terms of this object based and object-oriented programming languages, the distinction between object based and object-oriented programming languages are not very high one typical characterization that is done is a language is called object based if it does not if it supports the basic concepts of object orientation like abstraction encapsulation, modularity.

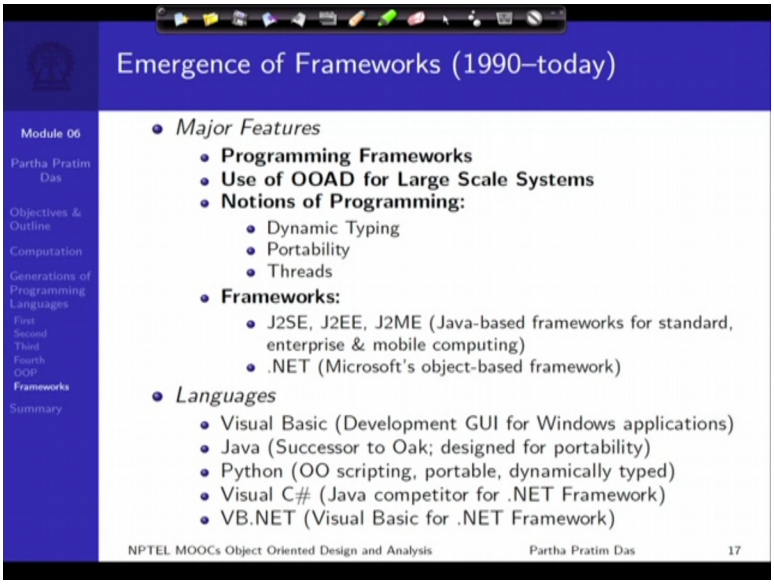
And so, on but if it fails to support inheritance specialization then its typically called object based. You will understand this is difference more as we go forward. Other languages are called object oriented languages. So, the significant difference that happens in this is the fundamental logic building blocks are no longer algorithms but they are classes and objects. So, you have had

quite a bit of discussions, we have had quite a bit of discussion in terms of classes, structure, object structure.

They become the fundamental concepts in the programming language itself. Instead of the procedural logic that decides given certain inputs what should be the output. There is very little global data, the data is more and more getting encapsulated in terms of objects, they are getting localized, getting inside the object and classes objects, module provide and essentially yet often insufficient I mean it its is not always that classes objects.

And modules can solve all issues of abstraction but a significant amount of higher abstraction got created in terms of these languages. So, you can see that the topological model has changed now. Now you do not have the data separately that has completely disappeared everything is now in terms of objects which has inside it the encapsulated data and it exchanges as we have discussed earlier also. It exchanges which message between them to decide what computation needs to be done.

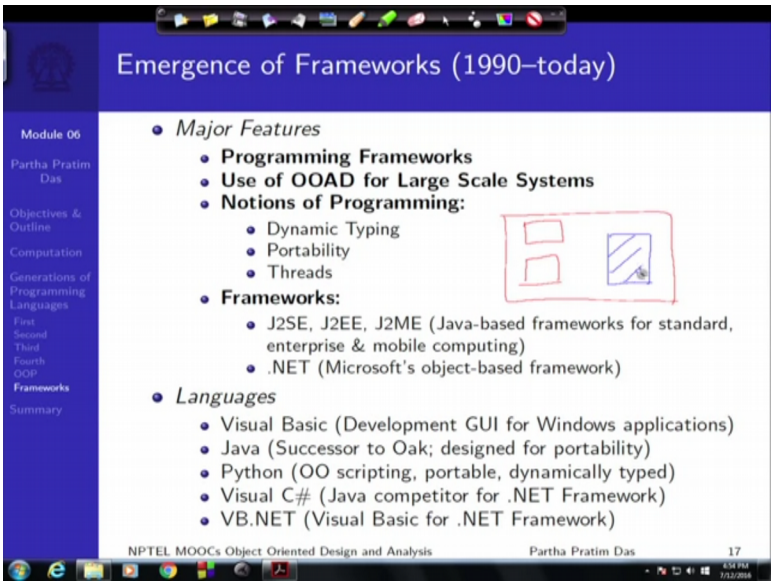
So, client object sends messages to server objects to other objects who will provide that service who has that kind of an interface and the server object will perform that operation and send back a response is kind of the typical topology that turned out for this generation of languages.
(Refer Slide Time: 23:36)



Beyond 1990s slowly the situation started becoming different the trend of designing new programming languages and empowering them with more of object oriented flavor continued but a major trend was being observed to develop programming frameworks that is earlier without a framework you have a programming language, you write a program and then you can compile it with a compiler, you get a user linker to link different sub modules of your program

and you get a get a single executable which you work which will work on the system. with frameworks, the things become somewhat different with framework what it turns out to be there is a framework so there is a certain framework which is given which means that there are some software components which are given and within then you are writing your this is your.

(Refer Slide Time: 24:40)



This is your program but the red boxes are components which are given you not only reuse them but actually them may exist even when you are executing the program all through the process of your lifecycle of variables objects and the program, this framework start helping move in multiple different ways. So, framework based programming started to become more and more popular even that is what made the use of object-oriented analysis.

And design for large-scale systems more and more possible. so, these are the major features in terms of notions of programming, a whole lot of dynamic typing notions got emphasized that is a dynamic typing is a concept where you do not specify a type for a particular variable, you do not

say in text to mean that x will always be integer but depending on what kind of value you keep on a kept keep in x whether you keep 17 then you will consider x to be integer.

You may keep 13.5 we will consider x to be a floating-point number. You can consider a string path then you will take x to be a string variable and so on. so, some concepts of dynamic typing became popular. Emergence of internet happen so portability became a major question. Portability became a major issue because it became important that you deliver software across different systems.

And certainly, what became very very important is instead of relying on large computing systems which could just work with one very powerful processor solving millions and trillions of computations per second. Instead of that it became a typical to distribute your computation in terms of different threads which started as a trend in parallel programming in larger systems and then percolated as smaller threads.

And smaller concurrent models in terms of even all the small devices that we use today, your cell phone being a big example. So typical frameworks that have emerged include the frameworks on which java that you know the most popular application programming language of today is based. So, when you program in java you typically make use of certain programming software environment like you could use j2SE, j2EE, j2ME.

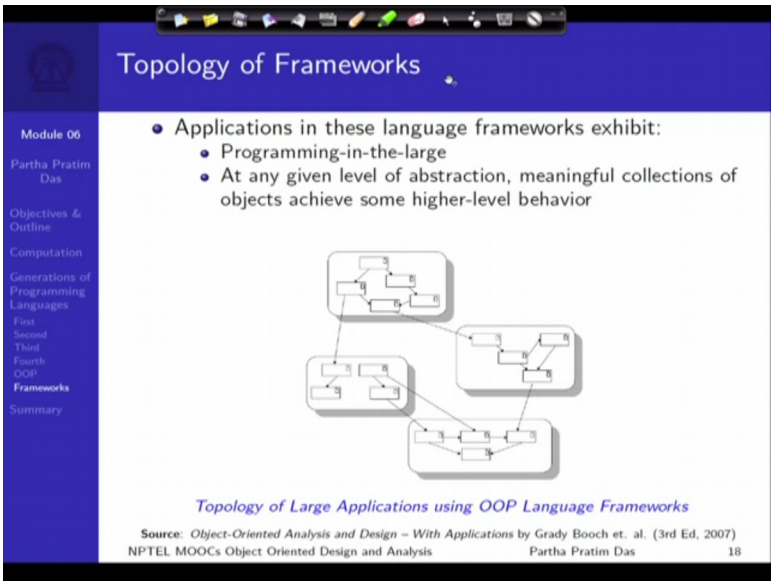
So, these are all different standard and standard environment or the enterprise environment or the mobile environment in which you do certain programming in java and these frameworks help you particularly by providing the support that you need for developing your application. For example, you want to develop and application for an android device, so using j2me it could become very easy.

You do not even physically need a device, the j2me could provide you with a model of the with a simulation of an android phone and you could write all the development on your desktop computer itself by using that model. So, frameworks started becoming really significant of dot net framework from Microsoft has even very dominantly used in variety of application

development and therefore you will find that several of the languages that happen during this time have significantly been linked to different framework.

So when we talk about visual basic we are certainly talking about GUI programming in on Microsoft or windows applications and when we talk about vb.net we are certainly talking about programming with visual basic under dot net framework, java frameworks we have talked of, even independent languages like python which is one of the excellent object-oriented programming languages of today which is highly portable very easy to learn and its supports dynamic type, in interpreter based also has its own framework to work with.

(Refer Slide Time: 29:00)

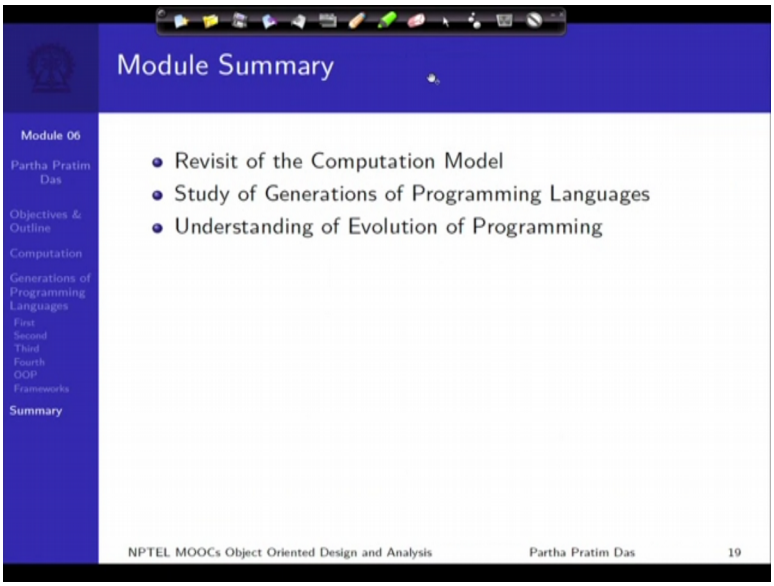


So, this is the these are the different kind of emergence that has happened and the topology for this at least started becoming more and more fragmented and distributed. So, you now have earlier in in terms of the object-oriented systems you had this kind of structure where objects were exchanging messages but they were all together now in terms of the framework they have got grouped in different subsystems.

So, there is a subsystem say if you are programming for android phone as a subsystem which is supporting so your address book, there is another subsystem which is supporting say your messages, there is another subsystem which is supporting your calls and so on. and using those abstraction you can do write a program. So, it is it is migrating beyond just the requirements of

programming languages, it's kind of becomes a framework based programming system on which you typically develop your programs.

(Refer Slide Time: 30:00)



So, to summarize in this module we have revisited the basic computation model we continue to use a different variants of the one moment model though we use it with certain client-server flavor in the whole thing. And we have taken a look at the different generations of programming languages and the current generation being the framework based programming.

And we have tried to understand the evolution of programming so that now when once we understand how to do the analysis and design of a system using object oriented methods, we can choose our right programming language or right framework in which we can go ahead and implement them.