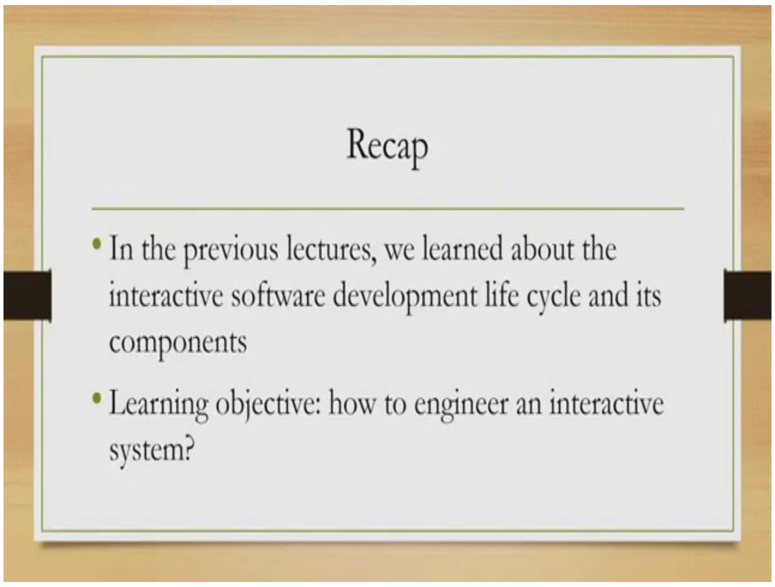
User-Centric Computing for Human-Computer Interaction

Prof. Samit Bhattacharya
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 09
Introduction to User-Centric Computing

Hello and welcome to the course User-Centric Computing for Human-Computer Interaction. This is lecture number-9. So, far what we have learned in the previous lectures let us quickly recap.

(Refer Slide Time: 00:49)

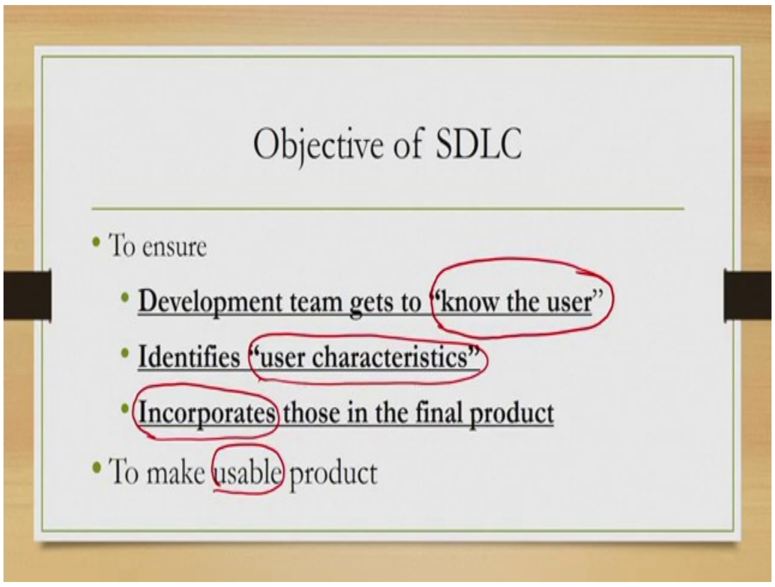


Recap

- In the previous lectures, we learned about the interactive software development life cycle and its components
- Learning objective: how to engineer an interactive system?

So, in the earlier lectures, we have seen how to engineer an interactive system; in other words how we can basically think of building a software for an interactive system in terms of software development lifecycle stages. Now, these stages are essentially a way to structure our thinking process about creation of the software.

(Refer Slide Time: 01:22)



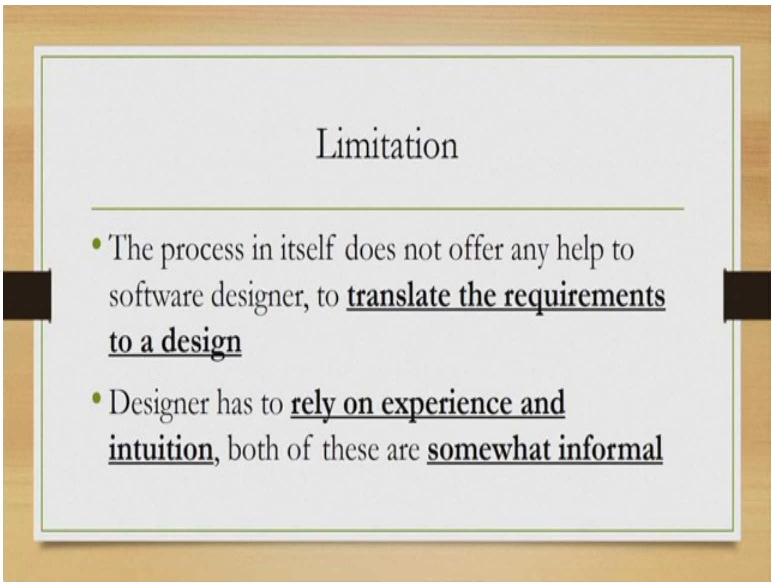
Objective of SDLC

- To ensure
  - Development team gets to "know the user"
  - Identifies "user characteristics"
  - Incorporates those in the final product
- To make usable product

What are the objectives of these stages? So, with the development of a software by following a software development lifecycle, we want to ensure that the developer get to know the user that is very important. So, first thing is know the user. Now, the stages at designed denotes as that developer gets to know the user through the conduction of field studies such as contextual inquiry and similar things.

With this knowledge of the user, they can identify user characteristics that is another requirement, another objective that we want to achieve by following the development lifecycle. And the ultimate objective of course, is to incorporate all these things the knowledge and the characteristics in the final product, so that we can make the product usable. So, this is essentially to ensure that we follow an user-centric design approach where we are incorporating the user in the design either as an active participant or a passive participant in the design process.

(Refer Slide Time: 02:36)



Limitation

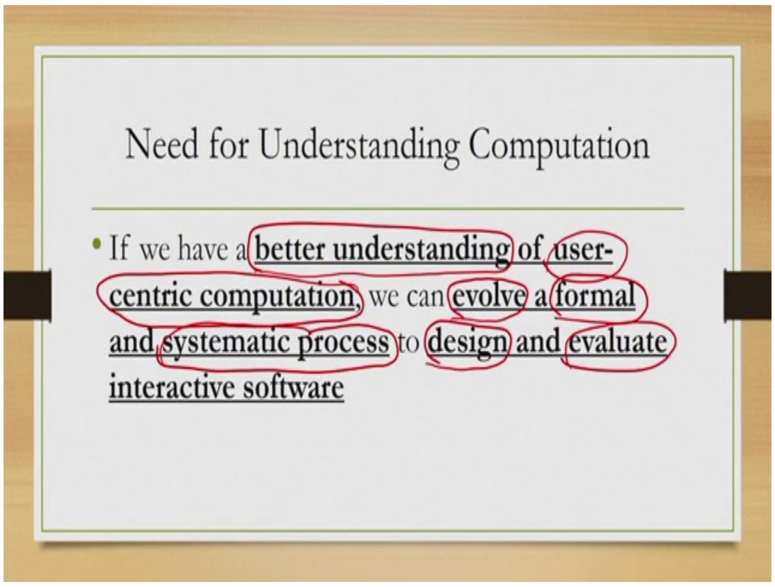
- The process in itself does not offer any help to software designer, to **translate the requirements to a design**
- Designer has to **rely on experience and intuition**, both of these are **somewhat informal**

However, the process in itself does not help the designer translate the requirements into design. So, essentially when we are talking of is the process, its primary objective is to let the designer learn about the user, let the designer identify their characteristics and then translate the knowledge into the design of the system. Now, the process in itself does not tell us how to translate the knowledge into the design.

So, essentially what the designer is left with? The designer is left with his or her intuition and past experience based on which he or she wants to or tries to come up with the best possible design. This approach of course, as you can understand, this approach is somewhat informal. So, there is no standard set of rules guidelines or anything to tell us exactly how to do this translation, how to translate the knowledge of the user to the design of the system, so that the ultimate product becomes usable.

Now, this informal nature is of course, not a very good idea if you try to understand things informally, then it leaves many things to interpretation of the person who is translating.

(Refer Slide Time: 04:03)



Need for Understanding Computation

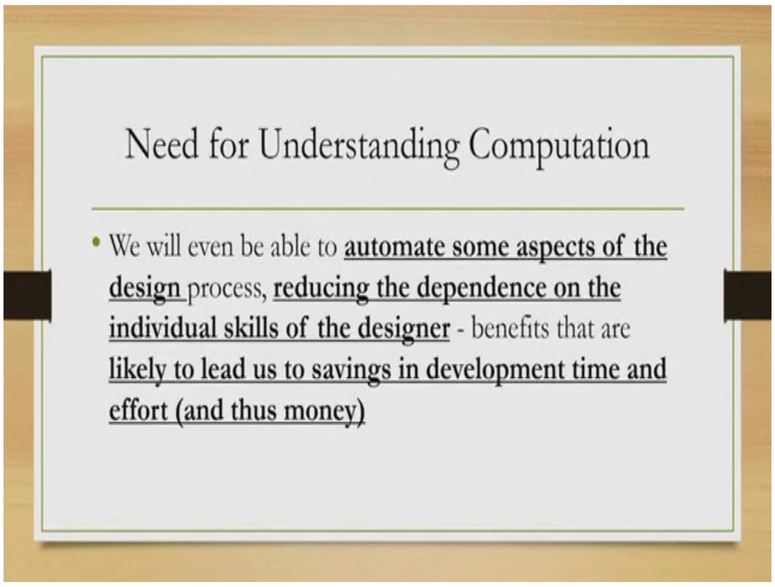
- If we have a better understanding of user-centric computation, we can evolve a formal and systematic process to design and evaluate interactive software

But it we have a better understanding of the underlying computations involved underlying computations that involved the users or in other words user-centric computing, then we can evolve a more formal and systematic process to design and evaluate interactive software. So, what we need is essentially a better understanding of the underlying computation which we are terming as user-centric computation, this is one of the major objective of this course.

If we understand that, then we can evolve a better mechanism or a more formal and systematic process to design and evaluate interactive system, so that it will help both in design as well as in evolution. So, in other words, we need to understand the computations better, so that we can understand the overall process in a more systematic manner and we can transfer this knowledge to others who wants to design such system.

Also this knowledge of or better understanding helps us in evaluating the final system. In fact, there is other advantages as well we would be able to basically automate some aspect of the design process if we understand the underlying computation.

(Refer Slide Time: 05:27)

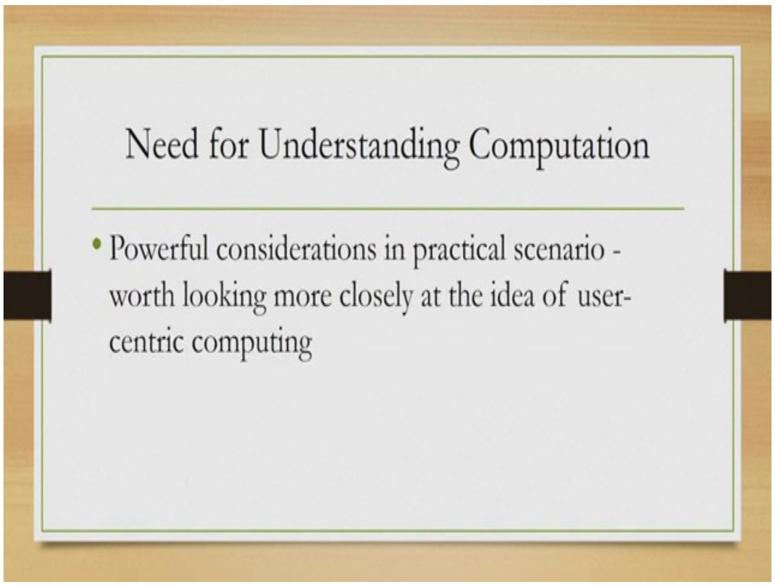


Need for Understanding Computation

- We will even be able to **automate some aspects of the design** process, **reducing the dependence on the individual skills of the designer** - benefits that are **likely to lead us to savings in development time and effort (and thus money)**

Then if we would be able to automate some aspects of the design which in turn is going to help us in reducing the overall design time, effort and consequently the cost.

(Refer Slide Time: 05:45)

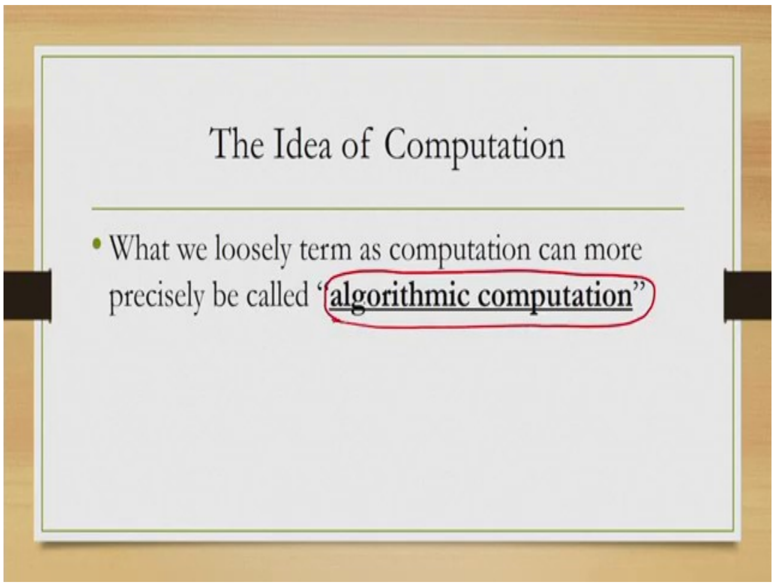


Need for Understanding Computation

- Powerful considerations in practical scenario - worth looking more closely at the idea of user-centric computing

Now, these considerations reduction of time effort and cost are of course, powerful considerations if you consider from a practical point of view. So, if we can achieve that then definitely it will help in making the product practical. Thus, if we can understand the underlying computation we can realize the product in a more practical way which makes it worth looking into the idea of user-centric computing more closely.

(Refer Slide Time: 06:11)



The Idea of Computation

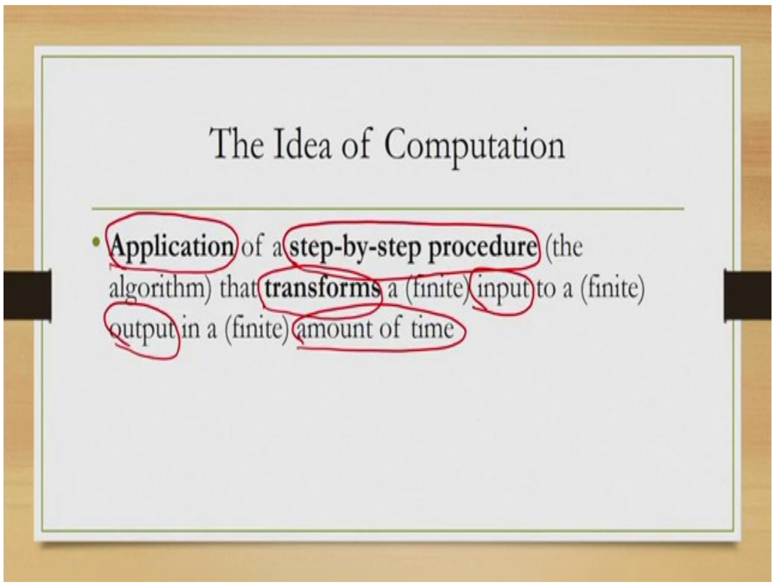
- What we loosely term as computation can more precisely be called "**algorithmic computation**"

So, what we need to understand is basically what is the computation underlying this whole process of the development of user-centric system. Before we try to understand this computational aspect of the design of user-centric system, let us first briefly recollect our knowledge of computation.

What we mean by the term computation? Most of us probably or most of you probably have learned about the idea of computation in your undergraduate courses. So, there you probably have come across this term in a different context rather than development of software systems, but here we will try to understand this term in its traditional way, and we will try to link that understanding to the development of interactive systems.

So, when we say computation and more precisely what we mean is algorithmic computation. Remember that we said that understanding computation will help us in a more formal and systematic understanding of the process. Now, formal and systematic understanding comes only when we are talking of algorithmic computation. The term algorithmic computation is a specific term which has a specific meaning. What is that meaning?

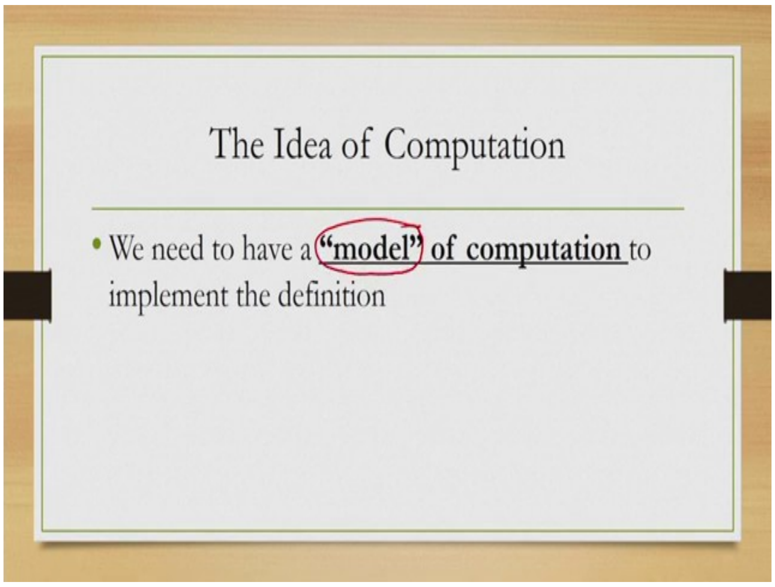
(Refer Slide Time: 07:31)


The Idea of Computation

- **Application** of a **step-by-step procedure** (the algorithm) that **transforms** a (finite) input to a (finite) output in a (finite) amount of time

It is nothing but the application of a step-by-step procedure which in other word is the algorithm that transforms a finite input to your finite output in a finite amount of time. So, essentially what we are referring to here is the application by the by the use of the term computation what we are referring to here is the application of a step-by-step procedure, which is the algorithm that transforms an input to an output in a finite time. For a better understanding, what we need is basically to know a model of computation.

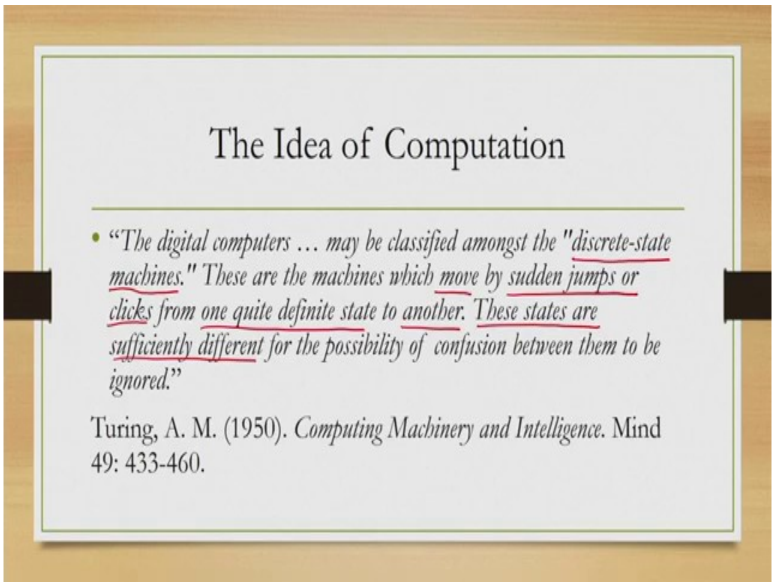
(Refer Slide Time: 08:24)


The Idea of Computation

- We need to have a **"model" of computation** to implement the definition

When we are talking of algorithmic computation, we are assuming certain things which in the previous definition which we have shown here. The computation is application of a step-by-step procedure; here we are assuming certain things which are not explicit. So, if we make those explicit then what we are talking of is a model of computation. So, in order to understand this idea of algorithmic computation what model we assume, what model we refer to, this is very important, because unless we know the model will not be able to understand the n idea of computation.

(Refer Slide Time: 09:16)



The Idea of Computation

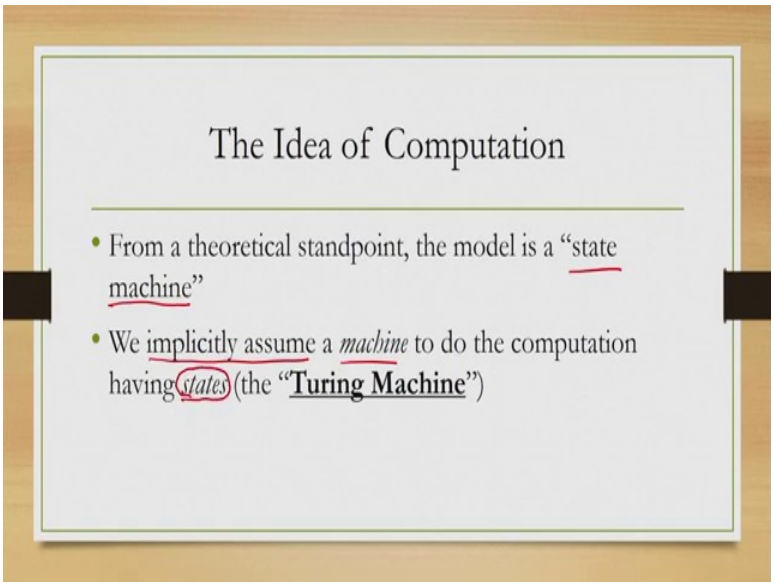
- "*The digital computers … may be classified amongst the "discrete-state machines." These are the machines which move by sudden jumps or clicks from one quite definite state to another. These states are sufficiently different for the possibility of confusion between them to be ignored.*"

Turing, A. M. (1950). *Computing Machinery and Intelligence.* Mind 49: 433-460.

So, what is the model? Let us start with the definition of digital computers proposed by the pioneer Alan Turing, way back in 1950 in the article Computing Machinery and Intelligence. According to his definition of digital computers, we can say that the digital computers may be classified amongst the discrete-state machines. This term is very important digital computers may be classified amongst the discrete-state machines. These are the machines which move by sudden jumps or clicks from one quite definite state to another.

So, the other important term is movement by sudden jumps or clicks from one quite definite state to another. These states are sufficiently different states are sufficiently different for the possibility of confusion between them to be ignored. So, there are a few important concepts hidden in this definition proposed by Turing that is discrete-state machines which are machines which move by sudden jumps or clicks from one quite definite state to another. And these states are sufficiently different for the possibility of confusion. So, you should be able to distinguish them without any confusion. And this definition is widely followed.

(Refer Slide Time: 10:48)

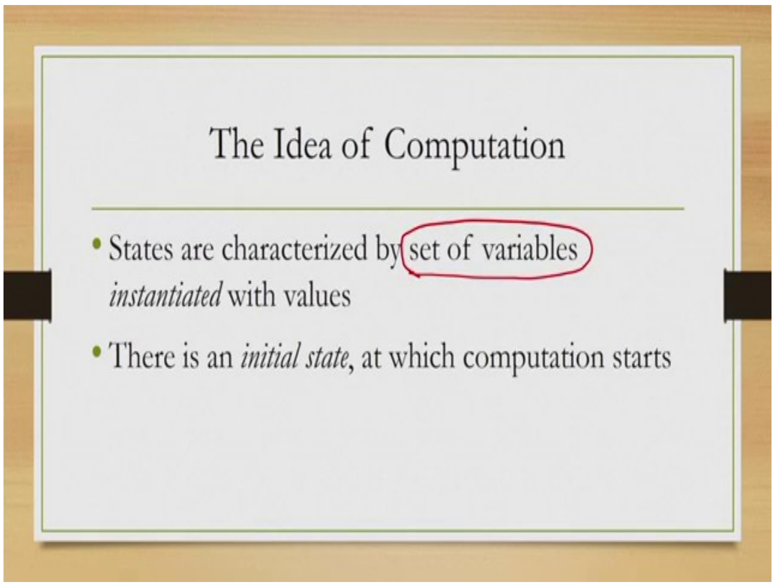


The Idea of Computation

- From a theoretical standpoint, the model is a "state machine"
- We implicitly assume a *machine* to do the computation having *states* (the "**Turing Machine**")

So, essentially from the theoretical standpoint when we talk of a model of computation we are referring to a state machine or a discrete-state machine. In other words, we implicitly assume a machine note that we are implicitly assuming a machine to do the computation which has states is called the Turing machine.

(Refer Slide Time: 11:07)



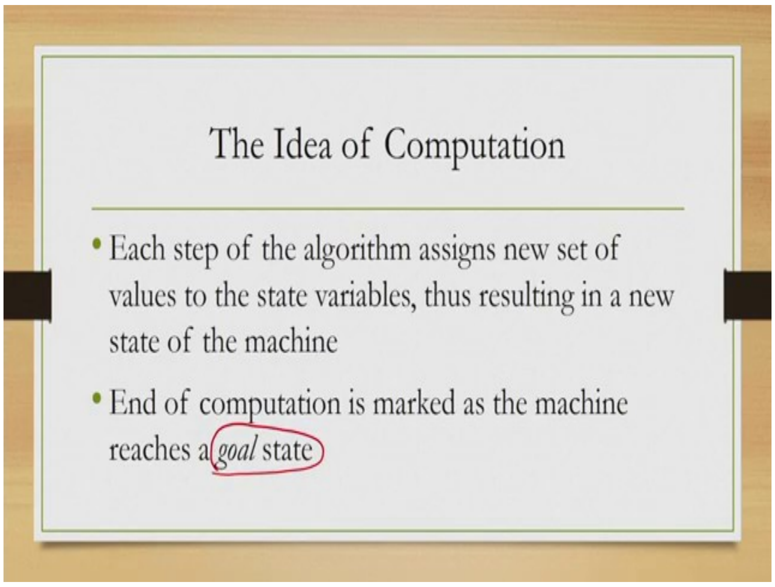
The Idea of Computation

- States are characterized by set of variables *instantiated* with values
- There is an *initial state*, at which computation starts

What are the states that is a very important idea state of a machine or discrete-state machine. So, essentially when you talk about states, what we are referring to is a set of variables. Now, the definition of state is then very simple it is simply a set of variables which defines a state. In a particular state, these variables gets some values.

So, when the variables are instantiated with values, we get physical realization of a state. And in a computation, it is a step-by-step process as you probably can recall from our earlier discussion. So, in the first step is basically there is an initial state at which the competition starts.

(Refer Slide Time: 11:55)



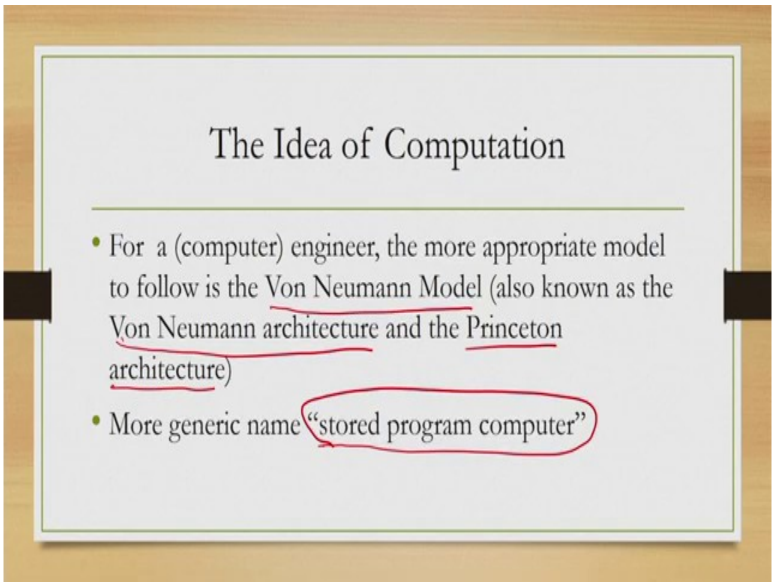
The Idea of Computation

- Each step of the algorithm assigns new set of values to the state variables, thus resulting in a new state of the machine
- End of computation is marked as the machine reaches a *goal* state

Now, in each step of the algorithm or the step-by-step procedure, the values of the state variables changes; so, each step essentially is meant to change the set of state variables, so that a new state of machine is obtained. And, when the computation ends, when we reach a final state which is often called a goal state; so, essentially what we do in a mathematical computation or algorithmic computation, we start with an initial state where the a set of values are used to instantiate the state variables.

Then we perform the steps in each step the set of values assigned to the variables changes we obtain a new state, and this process continues till we reach a final or goal state that is the idea of computation, where we are implicitly assuming that there is a discrete-state machine to implement the idea that is traditional view of computation. When we are talking of a computer engineer, who is more concerned about implementing a system practical system like a computer, the idea of computation is slightly different in its appearance.

(Refer Slide Time: 13:10)



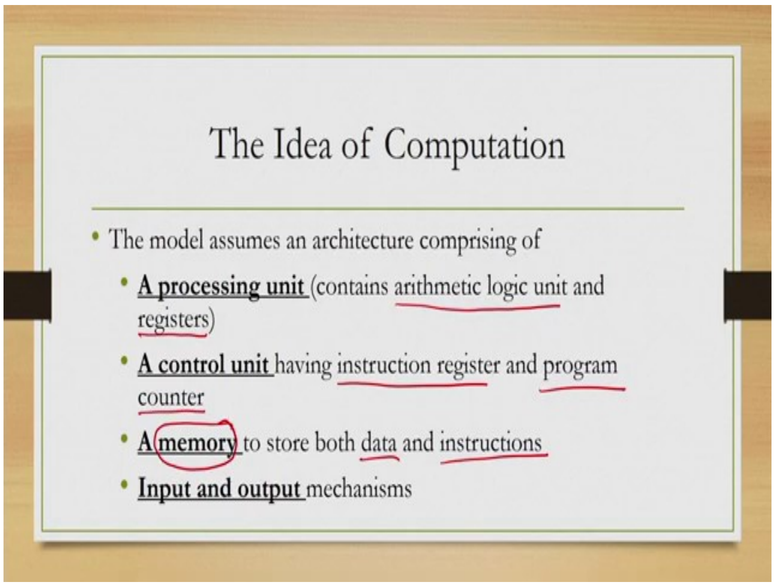
The Idea of Computation

- For a (computer) engineer, the more appropriate model to follow is the Von Neumann Model (also known as the Von Neumann architecture and the Princeton architecture)
- More generic name "stored program computer"

So, this idea is more commonly or more popularly termed as stored program computer which is a model to indicate the idea of computation. The original name was Von Neumann model or Von Neumann architecture also sometimes called Princeton architecture.

However, there are some there is some debate, but we may ignore that debate on what is the right term to be used. More generic name will refer to that is called the stored program computer. So, this is the model of computation which and a computer engineer assumes to implement the idea of computation.

(Refer Slide Time: 13:59)



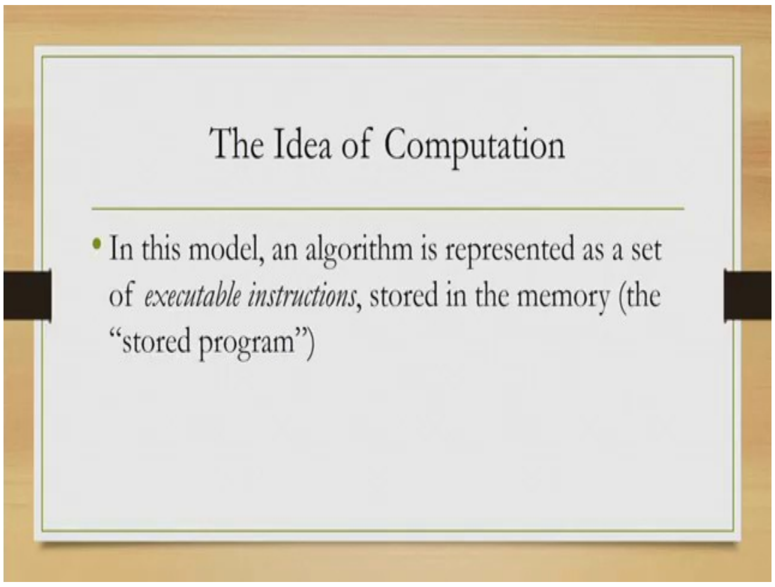
The Idea of Computation

- The model assumes an architecture comprising of
  - **A processing unit** (contains arithmetic logic unit and registers)
  - **A control unit** having instruction register and program counter
  - **A memory** to store both data and instructions
  - **Input and output** mechanisms

So, there are basically four things that are assumed as part of the model. First a processing unit which contains the ALU or arithmetic logic unit and the various registers; then a control unit which consists of instruction register and program counter. A memory both, both RAM and hard disk or both primary and secondary memory which contains both data and instructions, and finally, some input and output mechanism.

So, the four things that a computer engineer assumes as a model of computation are to repeat a processing unit, a control unit, a memory and the input and output mechanism. So, here what is the algorithm that is basically the set of instructions that are stored in the memory.

(Refer Slide Time: 14:50)

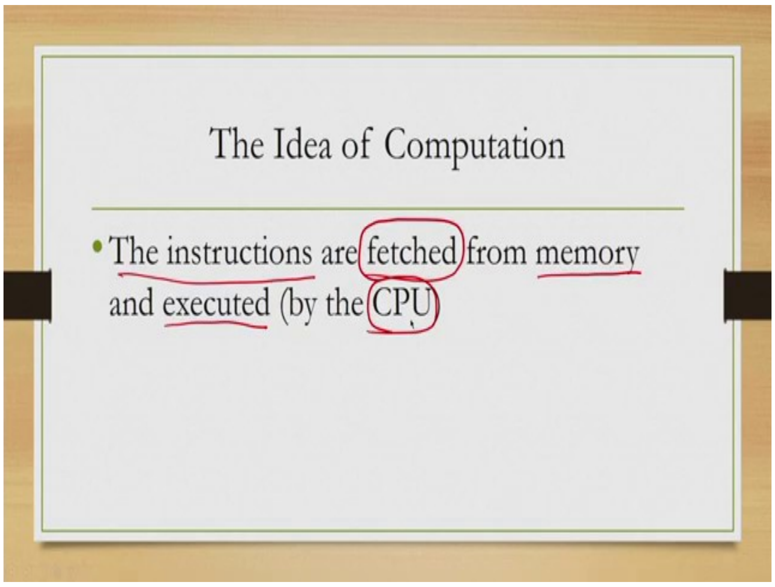


The Idea of Computation

- In this model, an algorithm is represented as a set of *executable instructions*, stored in the memory (the "stored program")

And, that is what the term stored program refers to in the term stored program computer.

(Refer Slide Time: 14:55)

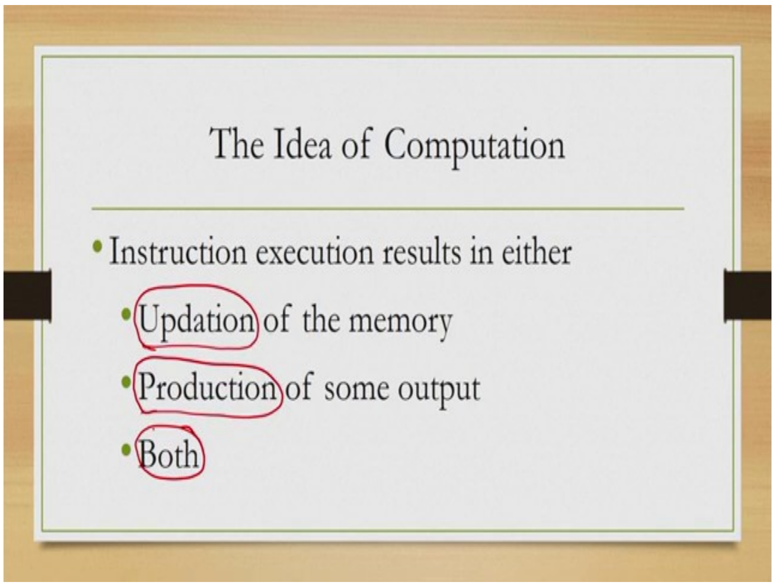


The Idea of Computation

- The instructions are fetched from memory and executed (by the CPU)

And, it is assumed or that is the idea of implementing this model of computation that these instructions are fetched from memory and executed by the CPU. So, we have a memory where the instructions are stored they are fetched from that memory, and then executed by the processing unit or CPU by following certain controls.

(Refer Slide Time: 15:30)



The Idea of Computation

- Instruction execution results in either
  - Updation of the memory
  - Production of some output
  - Both

Now, what is the result of the execution? Execution of an instruction can result in either updation of the memory main primary or secondary, production of some output to the user or both. So, I can execute an instruction to update both memory as well as produce some output.

So, earlier we have seen one model of computation that is the discrete-state machine as explained by Turing. Now, we have seen another model which is more popular among the computer engineers which is the stored program computer or the Von Neumann architecture or Princeton architecture are they related that is of course, one important question or are they totally different.

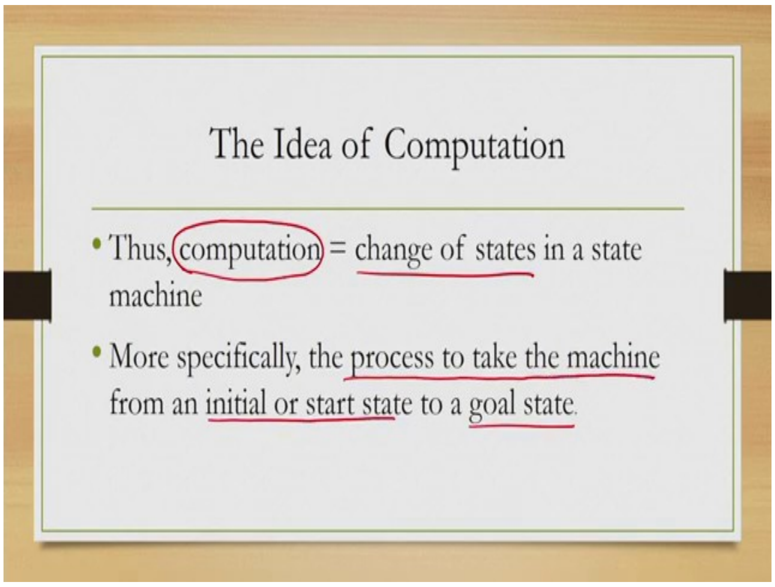
So, when a traditional computer science person thinks of an algorithm and when a computer engineer thinks of an algorithm from a theoretical point we are talking of discrete-state machine, from a from an engineering or practical point of view we are talking of a different model that is the stored program computer. Are they equivalent? Clearly they are equivalent stored program computer can also be thought of as a state machine. How, remember that a state is defined as a set of state variables.

Now, in a stored program, we can define a set of state variables in the same way. For example, the registers the content of the memory, the content of the program counter, the content of instruction register these all these we can talk of as state variables. So, whenever the content of registers or the program counter or the memory gets a specific set of values that is one state, when these values changes we get another state.

And in the execution of any software, our objective is the same. We start with an initial state which is defined by the set of values that are there in those registers or program counters or memory, and then the stored program gets executed one by one after being fetched from main memory. Then during execution as we have already mentioned the memory content may be updated leading to a different state. And this process continues till we get a final state where a specific set of loser they are assigned to the state variables and there may be some output.

So, conceptually idea of computation in a discrete-state machine and an idea of software execution in a computer are same both referred to a model of computers and which it is a discrete-state machine or equivalent to a discrete-state machine, and both of which refers to a step-by-step process or algorithm. These algorithms are used to change the state from one initial state to the goal state and that is the similarity between the two. So, essentially when we are talking of building a software, we are talking of basically a mathematical computation.

(Refer Slide Time: 18:37)

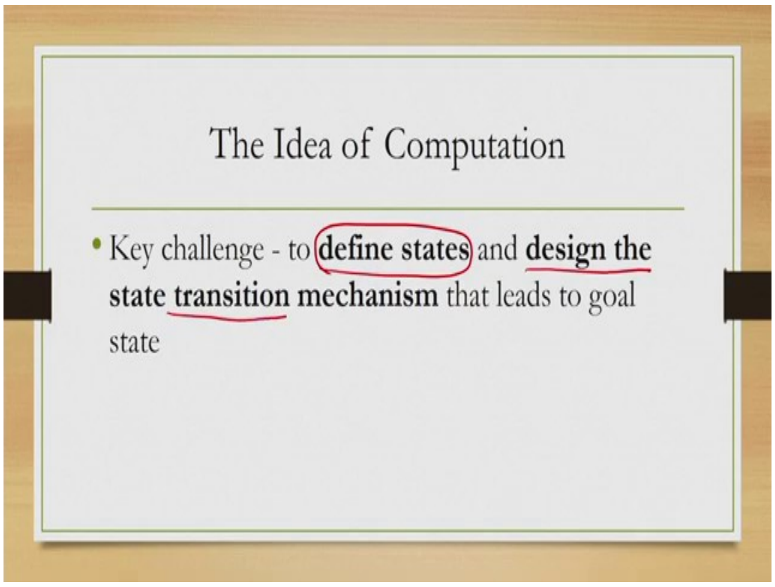


The Idea of Computation

- Thus, computation = change of states in a state machine
- More specifically, the process to take the machine from an initial or start state to a goal state.

So, the simple take away from this discussion is that we can equate computation to the change of states in a state machine, or more specifically the process of taking the machine from the initial or start state to a goal state. So, computation is equivalent to the change of state. And this change of state starts with an initial state and the process that takes the machine from the initial state or start state to the goal state is computation.

(Refer Slide Time: 19:10)



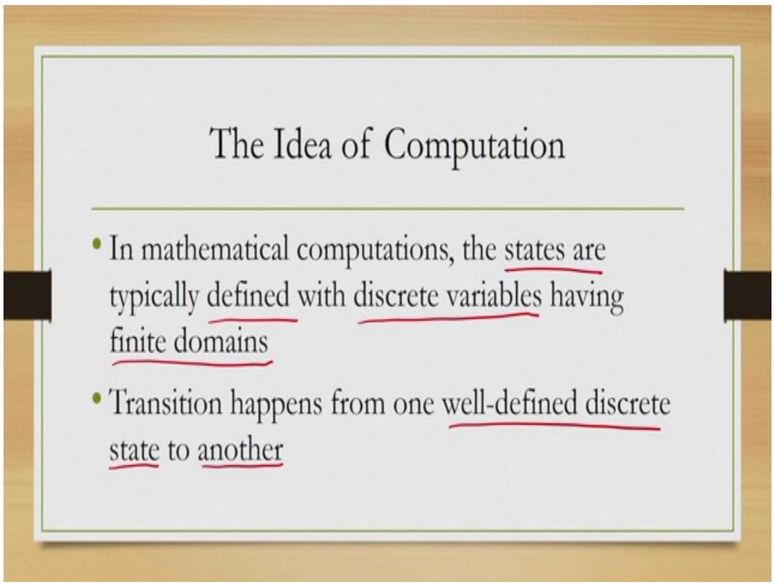
The Idea of Computation

- Key challenge - to **define states** and **design the state transition mechanism** that leads to goal state

So, what is the key challenge here? There are two challenges, one is to define the states, and the other one is to design the transitions. So, essentially what should be the set of variables that define a state and how we can change the values of the variables, how is perform a transition from one state to another that leads to the goal state. Now, these challenges there in the idea of computation.

Now, let us try to understand the key difference between mathematical computation and when we are talking of computation in interactive systems, essentially computation in software which is part of the interactive system.

(Refer Slide Time: 19:49)

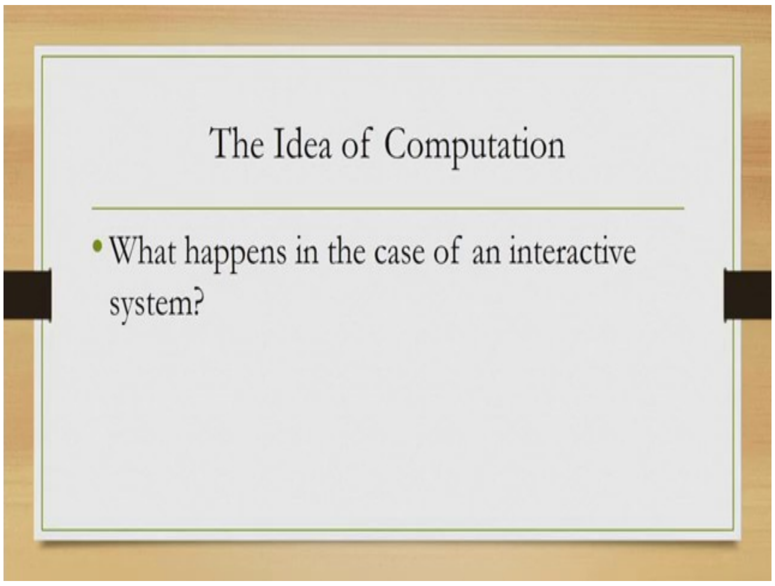


The Idea of Computation

- In mathematical computations, the states are typically defined with discrete variables having finite domains
- Transition happens from one well-defined discrete state to another

So, in mathematical computation the states are typically defined with discrete variables having finite domains, and the transition happens from one well defined discrete-state to another, so that is the traditional definition of computation as we have already discussed.

(Refer Slide Time: 20:12)

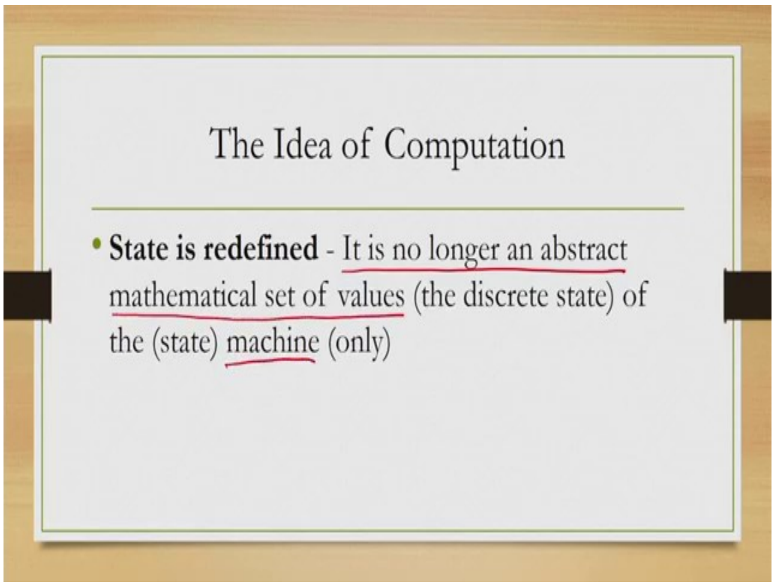


The Idea of Computation

- What happens in the case of an interactive system?

What happens in the case of interactive systems, can we still go by this definition that a state is a set of discrete variables which are well defined and that is the difference between a purely mathematical computation or the idea of computation in the context of interactive systems. So, when you are talking of computation in interactive software, it is not possible to simply go by the traditional definition.

(Refer Slide Time: 20:41)

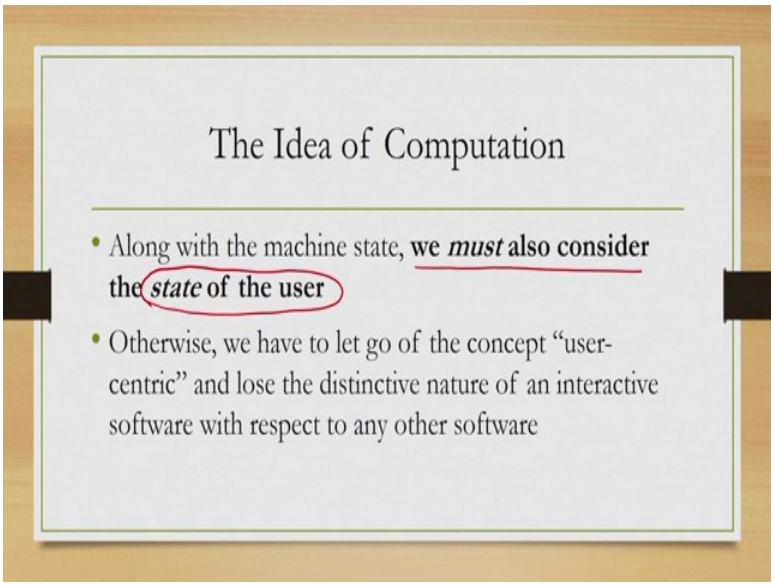


The Idea of Computation

- **State is redefined** - It is no longer an abstract mathematical set of values (the discrete state) of the (state) machine (only)

Because here what we talk of when we talk of state, it is no longer an abstract mathematical set of values which characterizes the machine, it is no longer the case.

(Refer Slide Time: 20:51)



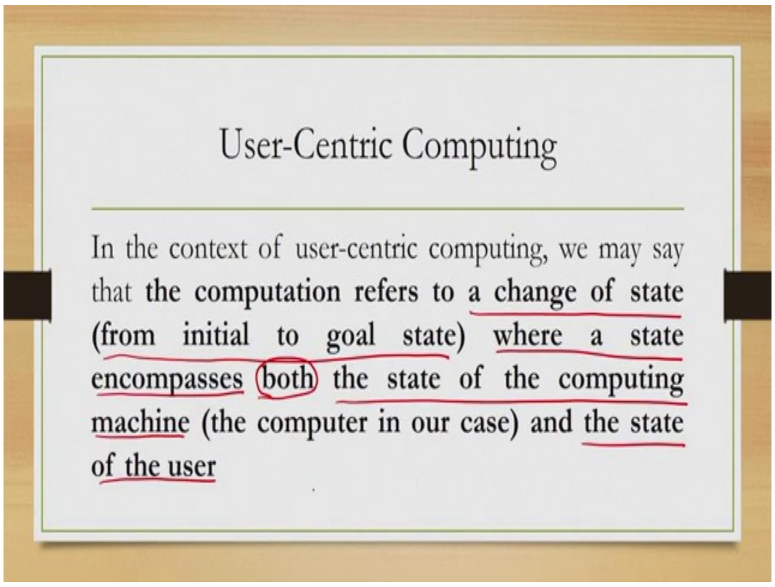
The Idea of Computation

- Along with the machine state, **we *must* also consider the *state* of the user**
- Otherwise, we have to let go of the concept "user-centric" and lose the distinctive nature of an interactive software with respect to any other software

Why, because here along with the machine we also have to think of the user the human. So, we must also consider the state of the user. Now, state of the user is something which is not there in purely mathematical and abstract computation.

But if we do not consider it, then definitely our idea of interactive system which is supposed to be usable will not work, because the usability comes only when we consider the human side of the interaction the users. And if we do not consider the users in the definition of states, then we cannot talk of user-centric systems. So, therefore, we need to redefine the idea of computation in the context of user-centric system.

(Refer Slide Time: 21:31)



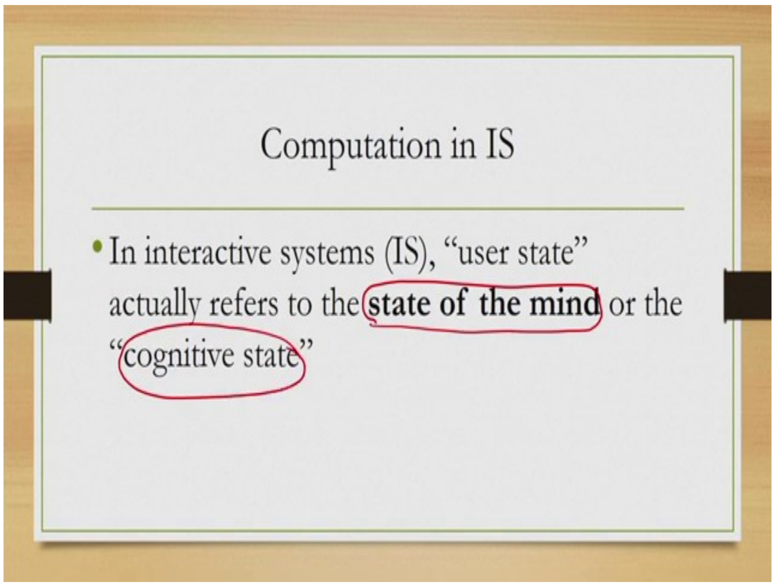
User-Centric Computing

In the context of user-centric computing, we may say that **the computation refers to a change of state (from initial to goal state) where a state encompasses both the state of the computing machine (the computer in our case) and the state of the user**

What is this definition? So, essentially in this case when we talk of computation we refer to a change of state the same way we have said earlier from initial state to goal state, again in the same way we talked of earlier, but with a twist that now here a state encompasses both this is important both the state of the computing machine and the state of the user.

So, here competition essentially is the same thing that is taking the system from the initial state to the goal state where the system state essentially refers to or combines both the state of the machine the computer and the state of the users together.

(Refer Slide Time: 22:31)



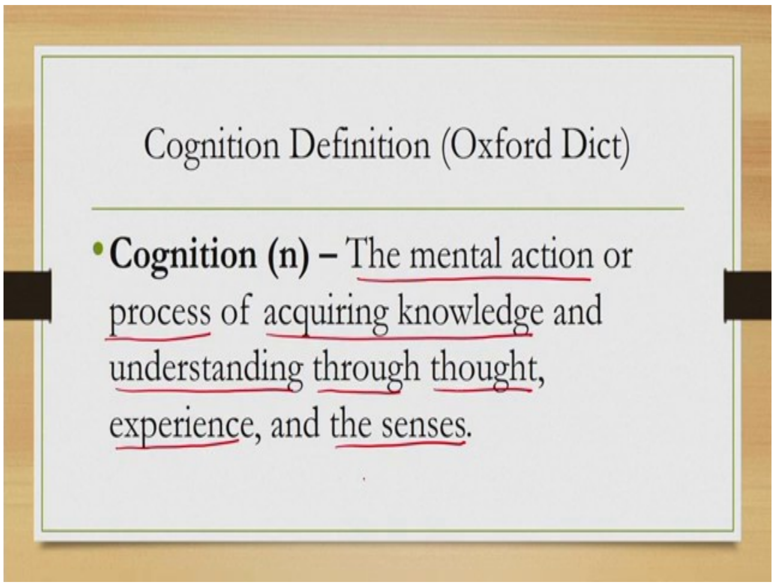
Computation in IS

- In interactive systems (IS), "user state" actually refers to the **state of the mind** or the "cognitive state"

And when we talk of user state, state of the user, what we refer to it is basically the state of the mind or sometimes more popularly it is called the state of cognition or cognitive state. So, that is important that the idea of computation in the context of interactive system software is redefined.

Now, we are referring to computation as a step-by-step procedure that takes the system from one initial state to the goal state where the system is defined in terms of its states which combines the state of the computer and the state of the user. The user state essentially refers to the state of mind, state of the mind of the user or the cognitive state.

(Refer Slide Time: 23:20)

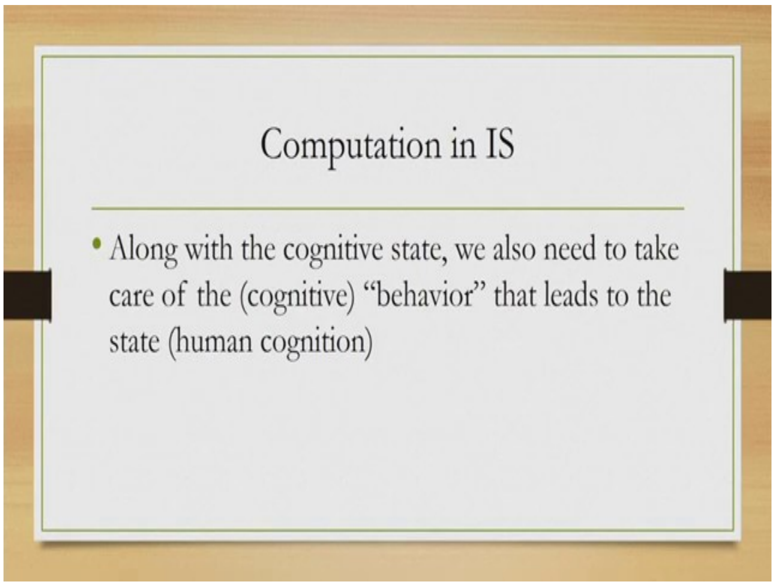


Cognition Definition (Oxford Dict)

- **Cognition (n)** – The mental action or process of acquiring knowledge and understanding through thought, experience, and the senses.

So, what is this cognitive state or what is cognition? According to Oxford English Dictionary, cognition is the mental action or process of acquiring knowledge and understanding through the use of thought experience and the senses. So, essentially cognition refers to a process of acquiring some knowledge and understanding the knowledge through the use of thought, our experience and our senses. So, it is a process.

(Refer Slide Time: 23:51)



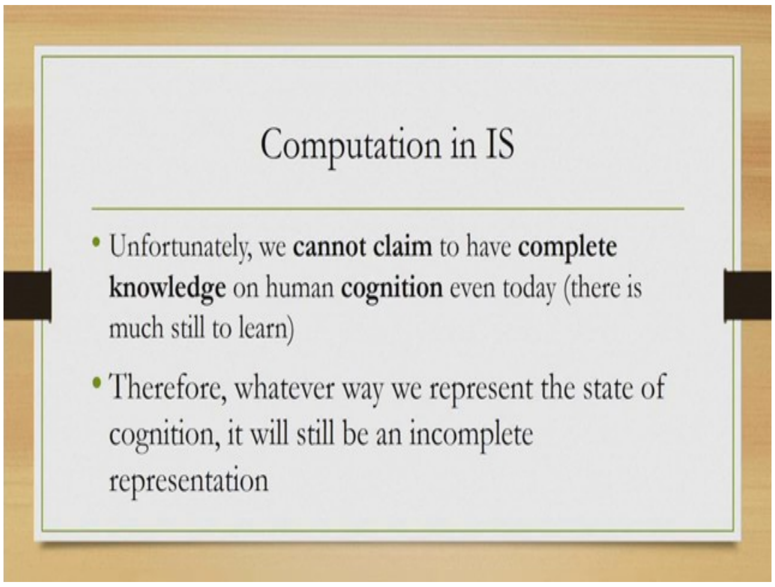
Computation in IS

- Along with the cognitive state, we also need to take care of the (cognitive) "behavior" that leads to the state (human cognition)

Now, during this process at any instant we can define the state of mind or cognitive state. Now, this cognitive state essentially reflects or represents our behavioral aspects how we perform the process how we behave to perform the process that leads to the particular state of cognition or cognitive state. So, essentially when we are talking of state we are talking of capturing the behavior of the user.

How easy it is to capture cognitive states? So, earlier when we are talking of purely mathematical computations we talked of a state as a set of variables, now it is relatively easy to define the set of variables for an abstract machine because everything is in our hand, we are free to define our set of variables. Of course, it is not as easy as I am saying, but it is relatively easy we service defining a set of variables for cognitive state because for the simple reason that we do not know how cognition works, we still do not have complete knowledge of human cognition.

(Refer Slide Time: 25:00)



Computation in IS

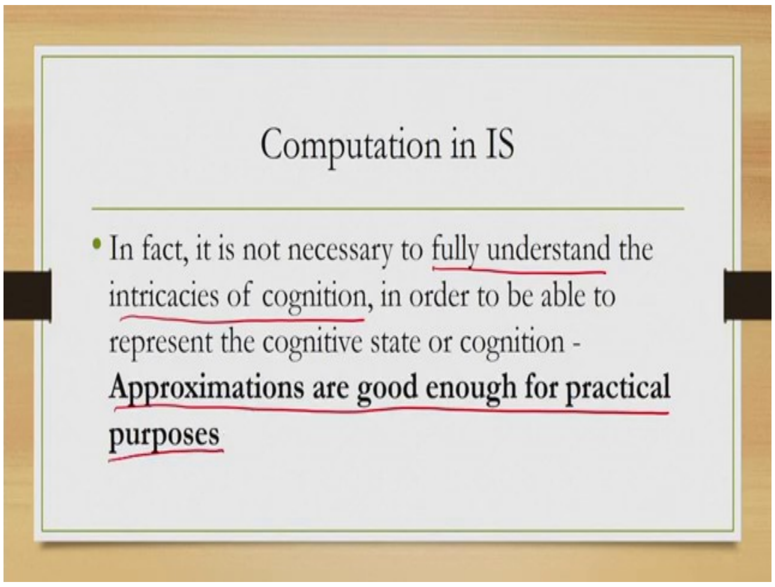
- Unfortunately, we **cannot claim** to have **complete knowledge** on human **cognition** even today (there is much still to learn)
- Therefore, whatever way we represent the state of cognition, it will still be an incomplete representation

So, essentially whatever propose whatever we say that ok, these set of variables refer to the state of cognition that is still based on some incomplete knowledge. So, we can at best say that it is an approximate representation of cognition rather than an actual representation. Unlike in the case of discrete machines where we can actually define our set of variables because the machine is defined by us.

So, defining cognitive state is not an easy thing. And when we are talking of interactive systems where we need to define cognitive state as well as system state combine them together to define or understand cognition, clearly that is adding one more dimension to the to our understanding of computation which is a difficult thing to do.

One good thing about this idea of cognitive state is that even if we do not have the complete knowledge, still we can represent cognition in some approximate way.

(Refer Slide Time: 26:02)



Computation in IS

- In fact, it is not necessary to fully understand the intricacies of cognition, in order to be able to represent the cognitive state or cognition - **Approximations are good enough for practical purposes**

It is not necessary to fully understand the idea of and the intricacies of cognitions, we can still represent it in some approximate way, and these approximations are good enough for practical purpose. So, if we want to implement the idea of cognitive state cognition in building interactive systems softwares, we can still work with approximate representation of cognition which we will see in our subsequent lecture.

So, just to recollect what we have learned so far. So, first of all we trying to understand the idea of user-centric computation because earlier we have seen that the user-centric software can be developed following an iterative lifecycle, we have discussed the stages of the lifecycle, but this knowledge although is helpful in structuring our thought process about the development, still does not give us a formal way of understanding the design.

So, through this process, we may get to know the user and their characteristics, but how to translate that knowledge to design is still informal, still dependent on our experience and intuition by our, I mean the designers experience and intuition. If we understand the computational aspects of the software, then that informality may be reduced we may get a more formal and systematic understanding of how things work, how this design can be achieved by translating the requirements into proper design.

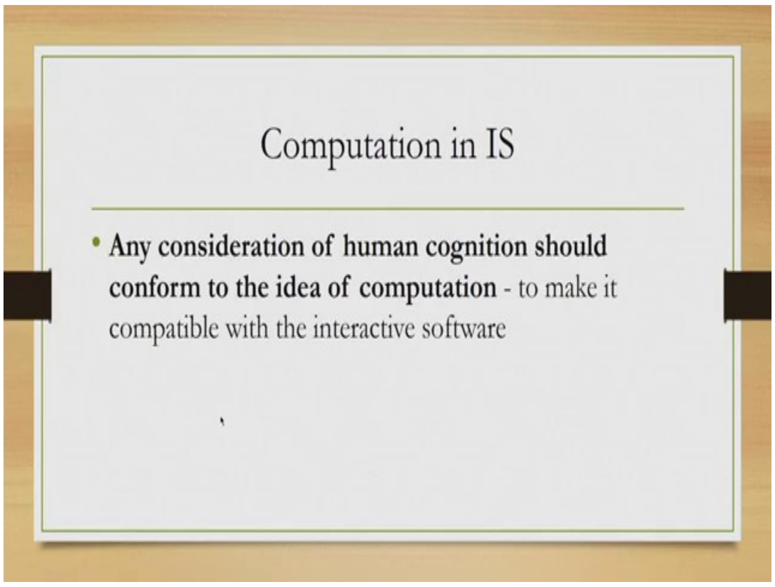
In order to understand this computational aspects of the software, interactive software, we started by discussing the traditional notion of computation which is essentially referring to mathematical computations. This idea is based on the concept of a discrete-state model of computation proposed by Turing. And in this model the computation takes place by changing an initial state to a goal state through a step-by-step procedure or the algorithm.

This model of computation is not relevant for interactive systems for interactive softwares, where when we are talking of the state of the system we are referring to both the state of the computer or the machine, and the user - the human. Now user state is very difficult to define because it is the state of cognition or cognitive state and we have inadequate knowledge or understanding of human cognition till today. But fortunately we can still work with this incomplete knowledge; we can still come up with approximate definition of human cognitive state for all practical purposes.

The next important thing that we should try to understand is that we talked of human cognition, now earlier we have seen that software we are thinking in terms of computation because the stored program computer is essentially equivalent to this notion of mathematical computation. Then we tried to combine the computer with the human through the merging of the state variables of computer and the state variables of users mind or cognition.

Now, philosophically at a very philosophical level, before we go for this margin, we should ask whether cognition can be thought of as computation that is very important, unless we can think of cognition as a mathematical computation. Technically we should not be merging the two distinct or different entities of a cognitive state and cognitive process to that of system state and mathematical computation.

(Refer Slide Time: 30:22)

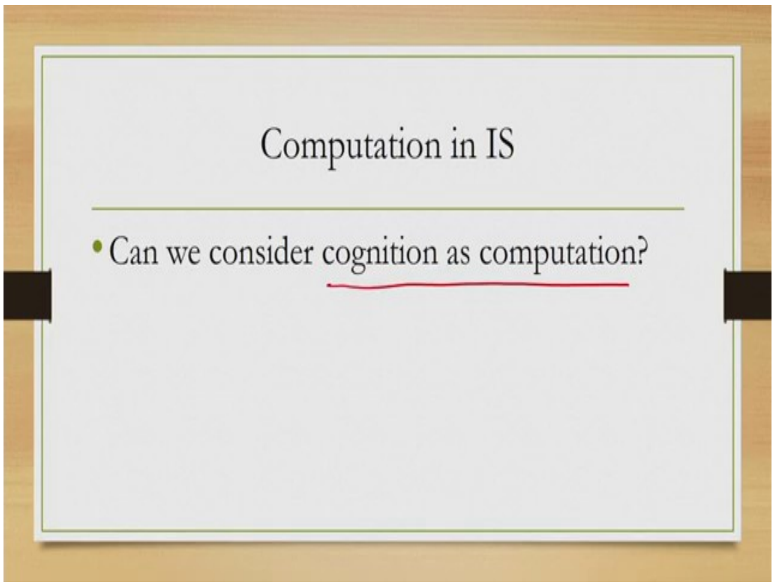


Computation in IS

- **Any consideration of human cognition should conform to the idea of computation** - to make it compatible with the interactive software

So, the question is whether human cognition conforms to the idea of computation?

(Refer Slide Time: 30:26)

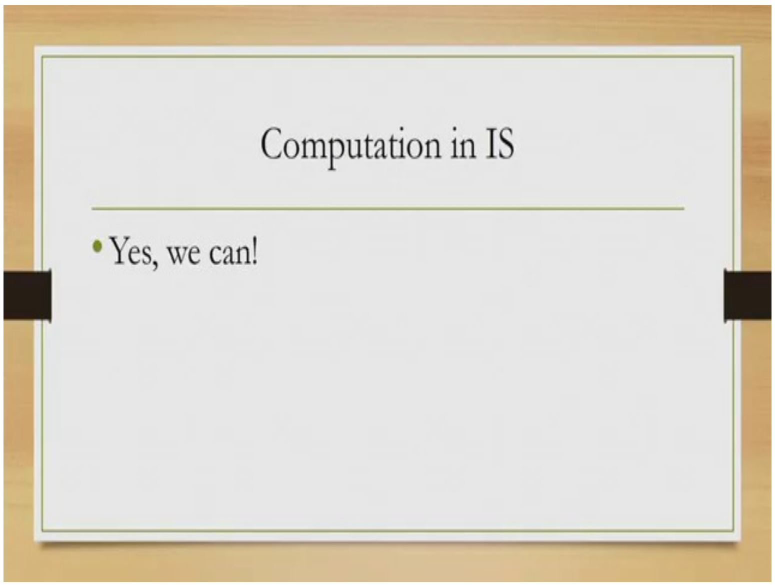


Computation in IS

- Can we consider cognition as computation?

Or can we consider cognition as computation? This is as I said is very important unless we can consider cognition as computational, we cannot merge the idea of cognition to the idea of mathematical computation. So, we cannot get this understanding of interactive system as a process of computation.

(Refer Slide Time: 29:53)

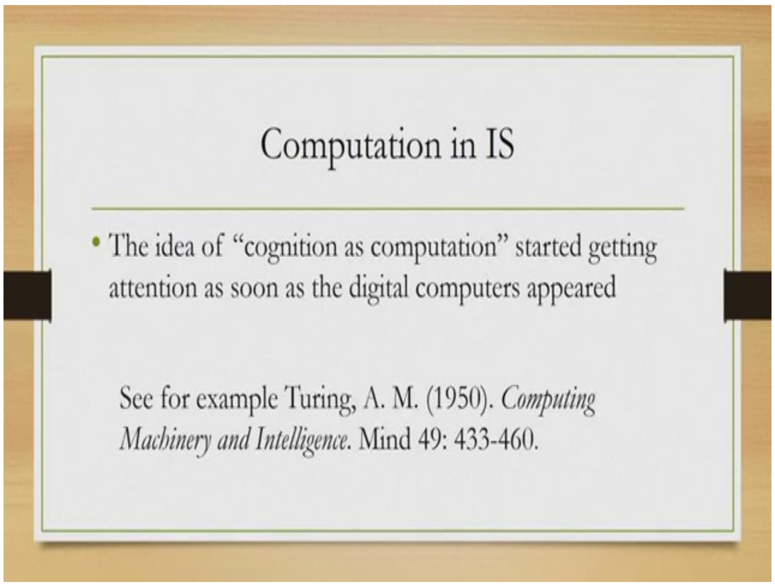


Computation in IS

- Yes, we can!

Fortunately, we can think of cognition as computation

(Refer Slide Time: 30:56)



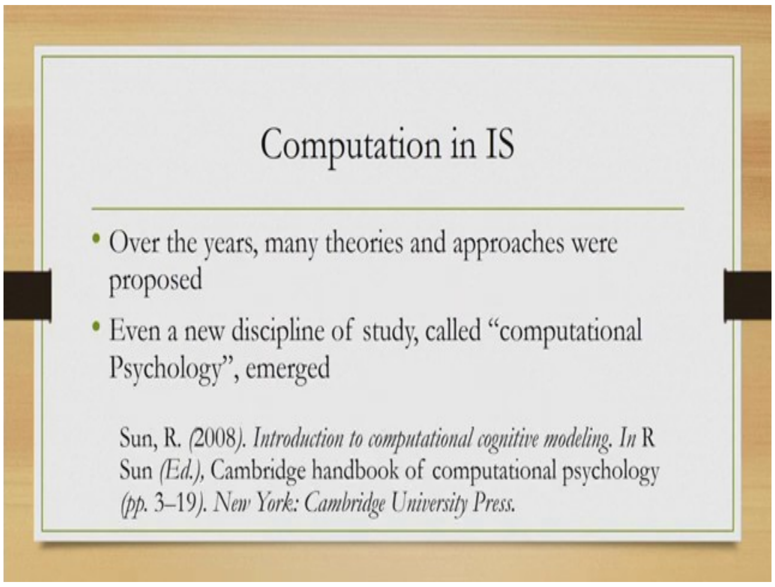
Computation in IS

- The idea of "cognition as computation" started getting attention as soon as the digital computers appeared

See for example Turing, A. M. (1950). *Computing Machinery and Intelligence*. Mind 49: 433-460.

This idea, that cognition is essentially nothing but computation or step-by-step application of a process or algorithm which changes states. Actually this idea was very old it appeared as soon as the idea of digital computers appeared which you can get if you follow this article by Turing Computing Machinery and Intelligence which we have referred to earlier. So, there this idea was mentioned already, so that is 70 years ago.

(Refer Slide Time: 31:34)



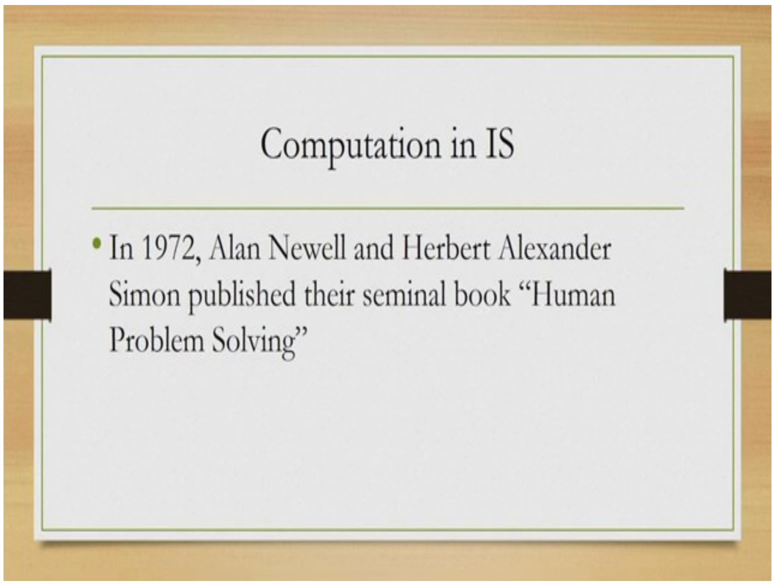
Computation in IS

- Over the years, many theories and approaches were proposed
- Even a new discipline of study, called "computational Psychology", emerged

Sun, R. (2008). *Introduction to computational cognitive modeling. In* R Sun *(Ed.),* Cambridge handbook of computational psychology *(pp. 3–19). New York: Cambridge University Press.*

And over the years this idea further matured many theories and approaches appeared proposed and a new discipline also appeared which is called computational psychology. More about this discipline or the historical evolution, you can find in this article Introduction to Computational Cognitive Modeling which was published in this handbook of computational psychology.

(Refer Slide Time: 32:17)



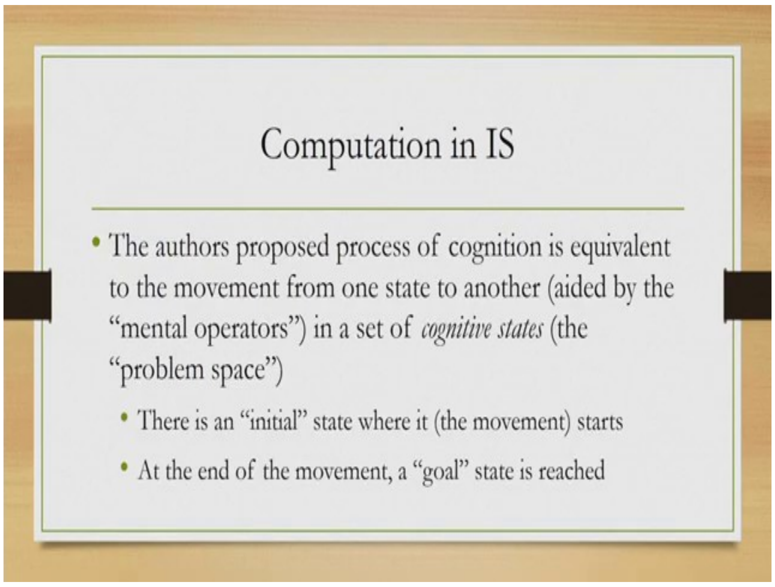
Computation in IS

- In 1972, Alan Newell and Herbert Alexander Simon published their seminal book "Human Problem Solving"

However, the landmark or a major development in this evolution of this thought process that cognition is computation came in 1972 with the publication of the book Human

Problem Solving by Alan Newell and Herbert Alexander Simon. So, this is a very seminal book.

(Refer Slide Time: 32:28)



Computation in IS

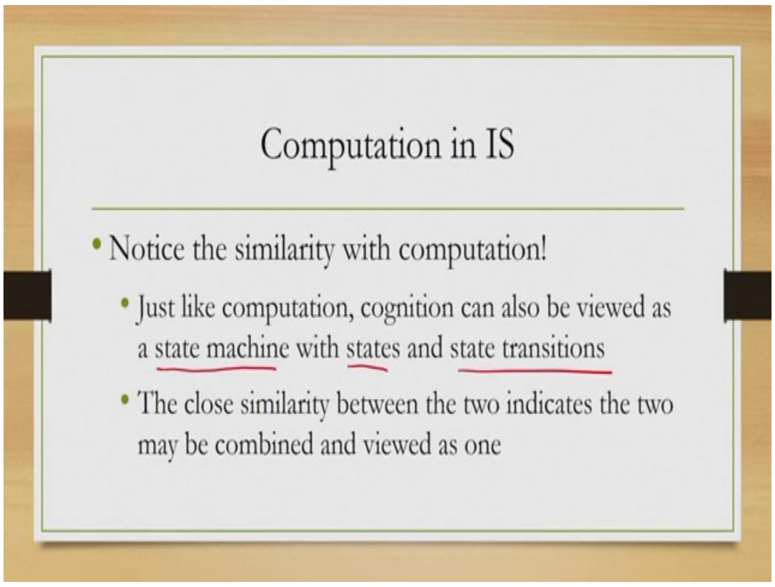
- The authors proposed process of cognition is equivalent to the movement from one state to another (aided by the "mental operators") in a set of *cognitive states* (the "problem space")
  - There is an "initial" state where it (the movement) starts
  - At the end of the movement, a "goal" state is reached

In this book what the authors proposed is that the process of cognition is essentially the transition of or the movement from one state to another state in a space of states or set of cognitive states called problem space. So, the states are mental states which was referred to as mental operators. So, the state transitions take place with the use of mental operators.

So, essentially what the author said is that we can think of the process of cognition as transition between mental states aided by mental operators in a large set of such mental states which is known as which they called as problem space. So, the key takeaways from this theory is that there is always an initial state from the initial state this state transition starts, then this mental operators are applied on the states.

So, this state changes to another state till it reaches the goal state. Clearly, these two are similar; the idea of state in a mathematical computation and the idea of state in this model by or problem solving mechanism proposed by Newell and Simon.

(Refer Slide Time: 33:48)



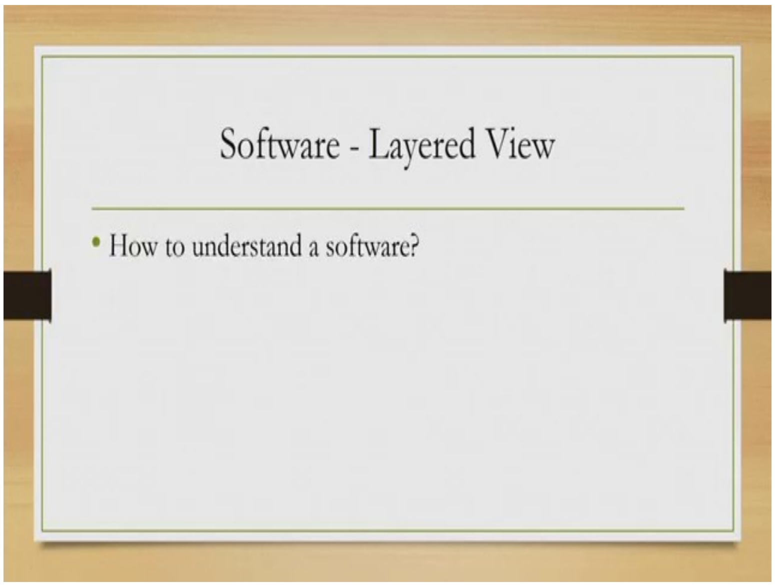
Computation in IS

- Notice the similarity with computation!
  - Just like computation, cognition can also be viewed as a state machine with states and state transitions
  - The close similarity between the two indicates the two may be combined and viewed as one

Here just like computation we can see that cognition is also viewed as a state machine with states and state transitions. States are defined in terms of a set of variables. So, both are considered to be similar. And since they are similar, then we can actually combine the cognitive process and the computation process together to build an understanding of computation in the context of interactive systems.

So, far we are trying to understand the idea of computation. And we have already referred to the fact that here our objective is to understand that and user-centric software that defines user-centric system from the point of view of an user essentially is nothing but a computation; computation that changes states. So, it starts with a set of register or memory values there is a stored program which is executed step-by-step which changes these values till you reach a goal state where the desired state is arrived at that is one way of understanding.

(Refer Slide Time: 34:57)

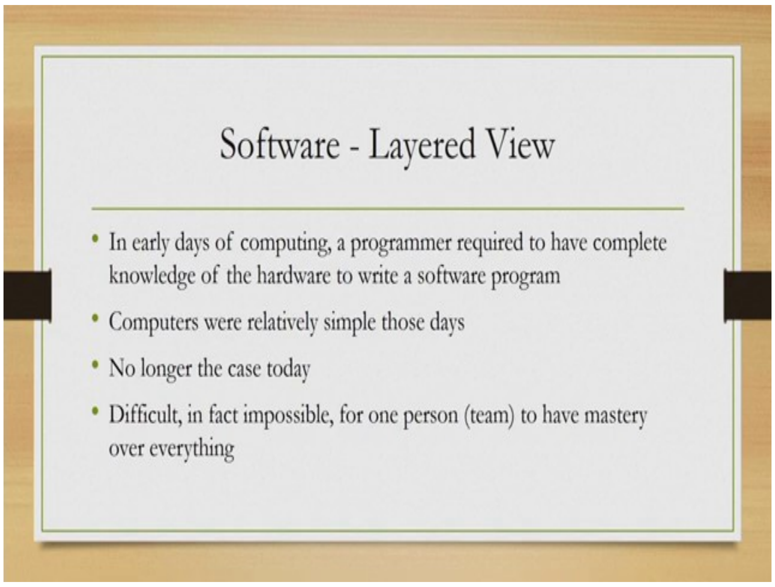


Software - Layered View

- How to understand a software?

Now, based on this knowledge, we can try to have a simpler understanding. So, how we can understand a software in a very simple way that is another thing that we should learn.

(Refer Slide Time: 35:21)

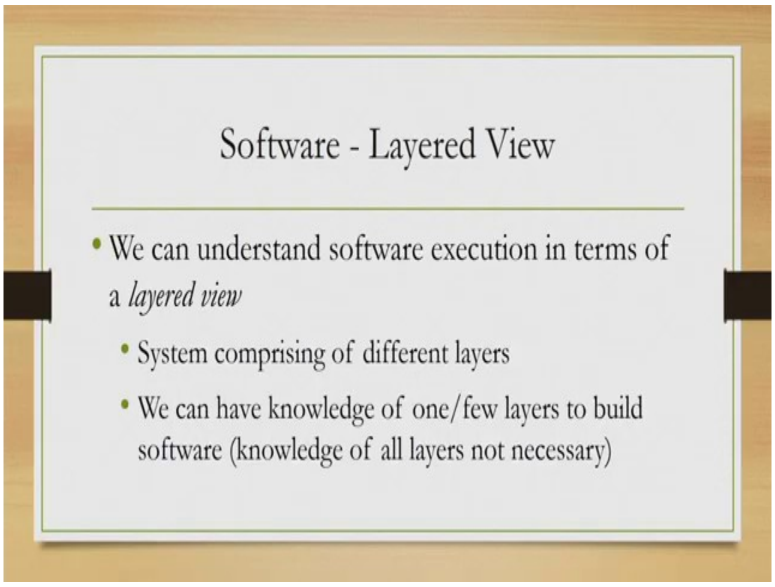


Software - Layered View

- In early days of computing, a programmer required to have complete knowledge of the hardware to write a software program
- Computers were relatively simple those days
- No longer the case today
- Difficult, in fact impossible, for one person (team) to have mastery over everything

In fact, if you know or if you have read the history of computers, you may be aware of that in the early days of computers a programmer needed to know the entire hardware to build a software. And people who are able to do that because in those systems, the number of functionalities were relatively less, the number of components were relatively less, and those are simpler, but that approach is obviously not applicable today when a computer has become very, very complex. It is not possible for a person to know the hardware of a computer in order to build a software for it. So, essentially what approach is adopted nowadays is a layered view of a system.

(Refer Slide Time: 36:13)

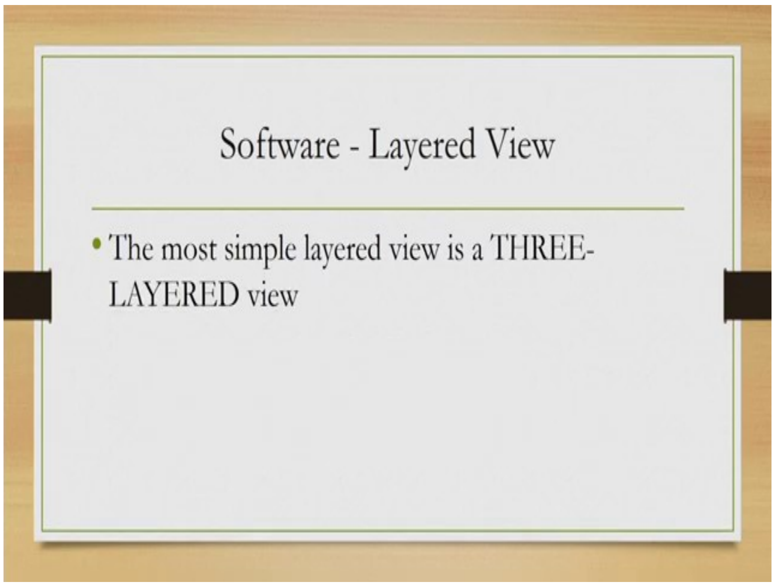


Software - Layered View

- We can understand software execution in terms of a *layered view*
  - System comprising of different layers
  - We can have knowledge of one/few layers to build software (knowledge of all layers not necessary)

In this view, typically a system is represented as consisting of different layers. And a particular programmer or a particular developer may concentrate on one layer based on his or her or their team's expertise. And it is sufficient to know about that particular layer only and other layers the knowledge of other layers may not be required, and there will be interfaces to interact with other layers. So, well-defined standard interfaces will be provided rather than the complete technical knowledge of the layers.

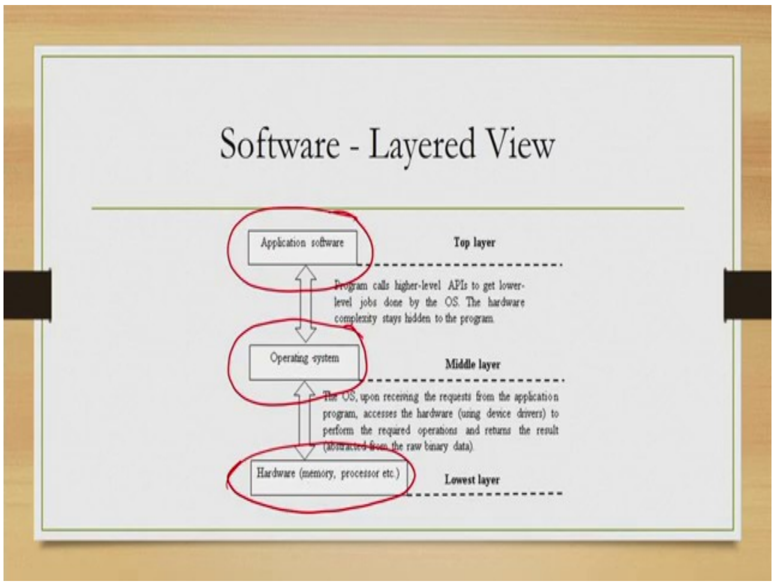
(Refer Slide Time: 36:46)



Software - Layered View

- The most simple layered view is a THREE-LAYERED view

(Refer Slide Time: 36:53)



Software - Layered View

Program calls higher-level APIs to get lower-level jobs done by the OS. The hardware complexity stays hidden to the program.

The OS, upon receiving the requests from the application program, accesses the hardware (using device drivers) to perform the required operations and returns the result (abstracted from the raw binary data).

And the simplest of this layered view can be the three layered view of a system. Have a look at this three layered view. So, in this view, we can visualize a system as consisting of three layers. At the top is the application software. These are the programs that the user interfaces with; below that is the operating system which interfaces between the hardware and the application software; and at the bottom is the hardware.

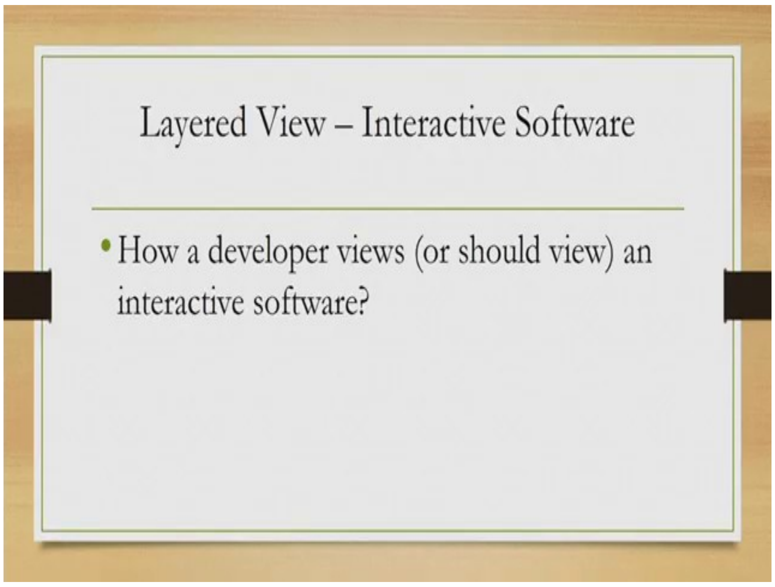
Now, when somebody is trying to build an application software, does he need to know about the operating system or the hardware layers? It is not necessary because the operating system provides APIs through which the application software can access the hardware. So, only knowledge of those APIs would be sufficient rather than a complete knowledge of how the APIs work, how the operating system works how it manages various resources.

Similarly, when somebody working at this operating system layer, he or she did not know about the application software layer what applications are going to be developed in future whether to provide support for all of those, instead a standard set of APIs will be developed which will be made known to all the developers. So, whoever wants to develop application software can make use of the API. So, the operating system developer did not know about the application software.

And the hardware developer or hardware designer whoever is building the hardware did not know about how to create an operating system or application software, they can simply provide again some standard interfaces, and some standards through with the operating system developers will interact with the hardware. So, exactly which voltage is there at a particular instant to define a particular binary value, and all these things all these details or how the signals are propagating, all these things did not be the concern of operating system developer.

So, here then what we are trying to achieve is basically separating out segregating different tasks for different group of people, so that this entire thing becomes useful to a user. Now, let us come back to the idea of interactive systems. So, if I talk of an interactive system clearly that is at the application layer, it has nothing to do with the ways because user interacts with it.

(Refer Slide Time: 39:45)

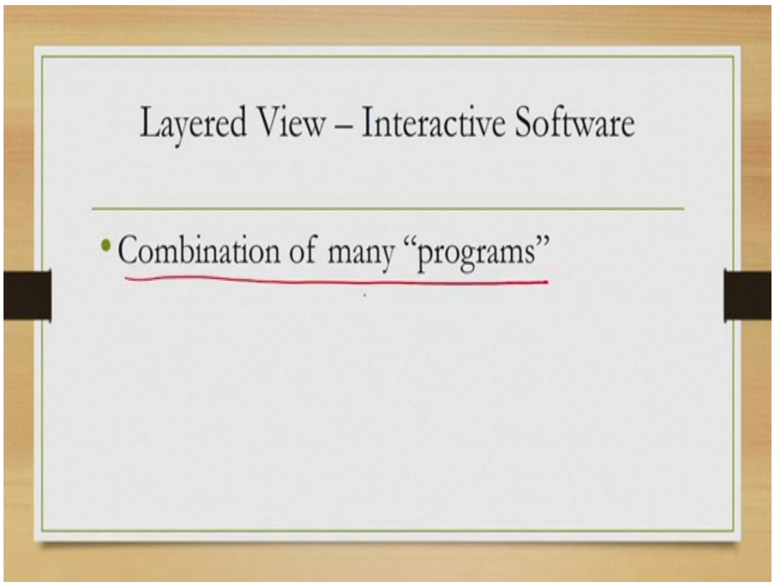


Layered View – Interactive Software

- How a developer views (or should view) an interactive software?

Now, to the developer of an interactive system how this application layer looks, let us try to understand that.

(Refer Slide Time: 39:55)

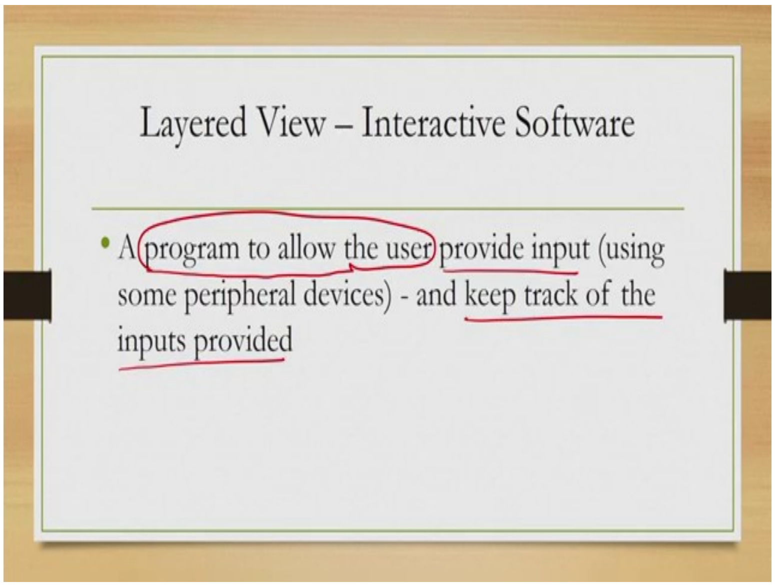


Layered View – Interactive Software

- Combination of many "programs"

So, a developer how a developer builds an interactive system software? To developer this software is essentially a combination of many programs, so not one, but a combination of multiple programs. How many programs are there?

(Refer Slide Time: 40:11)

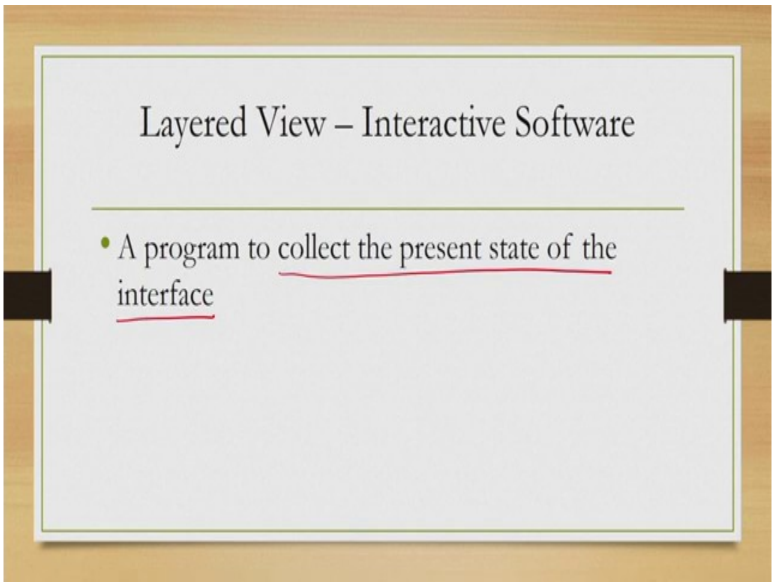


Layered View – Interactive Software

- A program to allow the user provide input (using some peripheral devices) - and keep track of the inputs provided

At a very broad level we can think of a few 3 to 4 such programs. First of all there is a program that should allow the user to provide input. To an application programmer there should be a program to allow the user to provide input. The input can be provided using some peripheral devices; it may also be necessary to keep track of the inputs provided. So, the program should allow the user to provide input as well as keep track of what input is coming from the user, so that based on that input some action can be taken.

(Refer Slide Time: 40:46)

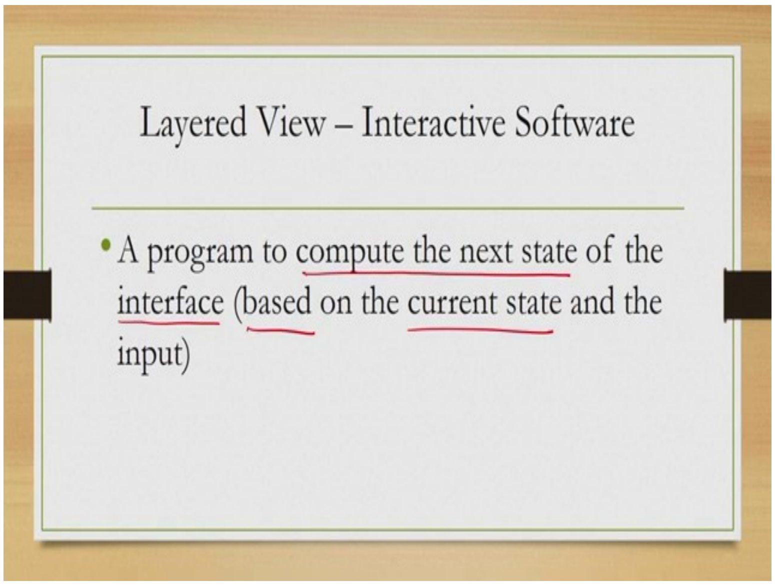


Layered View – Interactive Software

- A program to collect the present state of the interface

Next to an application programmer to the programmer of an interactive software, there is also a need to have a program that collects the legend state of interface. So, I have provided input and certain changes took place. So, what is the current state of the interface, there may be a need of that program.

(Refer Slide Time: 41:11)

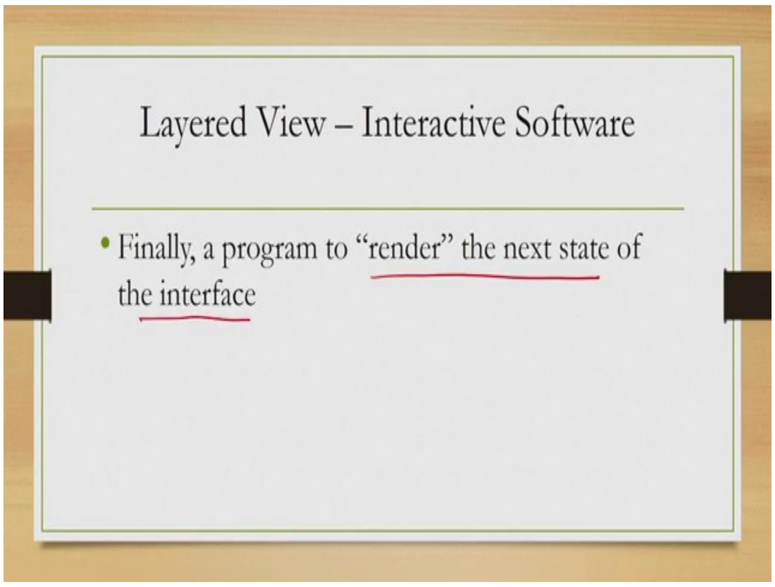


Layered View – Interactive Software

- A program to compute the next state of the interface (based on the current state and the input)

And based on the current state, since we have already seen software as a computation execution of a software as equivalent to computation, so a programmer may need to have a program to compute the next state based on the current state. So, essentially the program to implement the state transition mechanism.

(Refer Slide Time: 41:39)



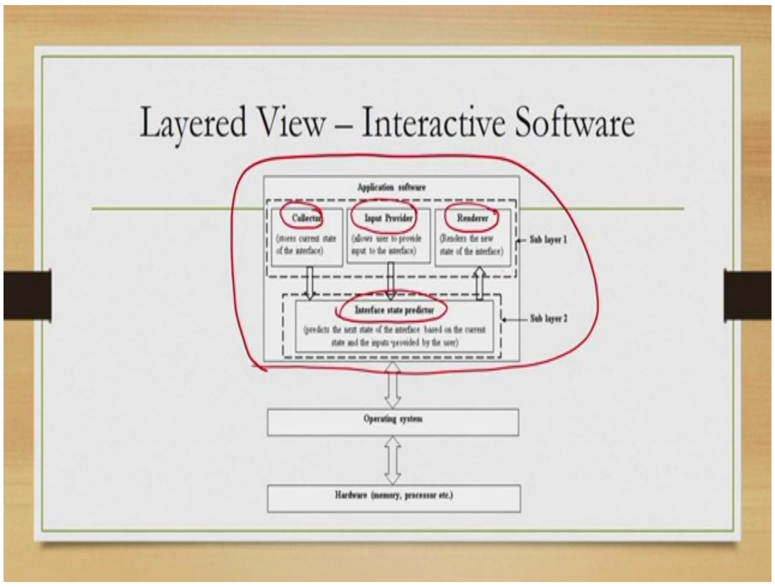
Layered View – Interactive Software

- Finally, a program to "render" the next state of the interface

And finally, a program to render the state or to give the output to the user; now, so there are roughly we can think that there are these four states that comes to him comes to the mind of a interactive developer. First of all there should be a program that allows a user to provide input, because interactive system means users are supposed to interact with the system. So, the program should let the user interact and at the same time the program should capture the input the program should keep track of the input, so that it can be used for further action.

Then as we have already seen it is computation. So, we need a way to capture the current state. So, how to implement this definition of state and how to capture those state variables, one program is required. Thirdly we require a program to implement the state transition mechanism going from current state to the next state. And finally, we need a program to render this next state to the user in a understandable form. So, based on this knowledge; so these four programs that are apparently required to implement an interactive system as perceived by a software developer.

(Refer Slide Time: 43:19)



Layered View – Interactive Software

Application software
Collector
(stores current state of the interface)
Input Provider
(allows user to provide input to the interface)
Renderer
(Renders the new state of the interface)
Sub layer 1
Interface state predictor
(predicts the next state of the interface based on the current state and the inputs provided by the user)
Sub layer 2
Operating system
Hardware (memory, processor etc.)

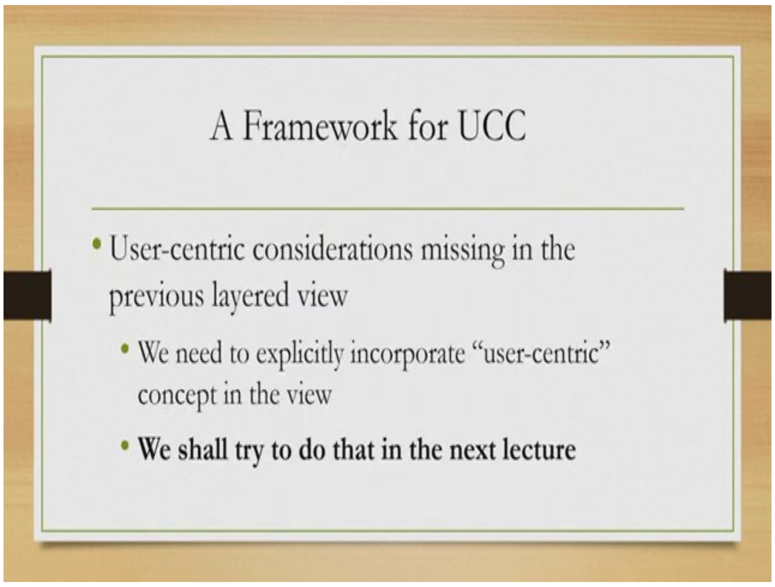
We can come up with a layered view of the interactive software. Let us have a look at this view and compare it with the earlier view. So, earlier we had application layer, operating system and hardware. Now, this application layer is further divided as you can see here we have introduced new sub layers. So, we have broadly two sub layers; sub layer 1 and sub layer 2. In the first sub layer, we have three programs collector, input provider and renderer. And in sub layer 2, we have interface state predictor.

So, the job of the collector is as the name suggests it is essentially to capture the current state of the interface. Input provider again as the name suggests is to allow the user provide input to the interface, and the renderer as the name suggests is to render the new state. And in the second sub layer, sub layer 2, we have this interface that predictor which essentially predicts the next state or implements the state transition mechanism.

So, those four programs are mapped to these four components. Three of those components are organized in the form of sub layer 1; one component is placed in sub layer 2. So, this collector essentially captures the current state; input provider is used to provide input or let the user interact with the system. And based on this input and current state the state transition is implemented as interprets state predictor which predicts the next state. And this state information with the set of values for the variables passed on to the renderer which renders the new state of the interface.

Does this layered view actually sufficient to capture the essence of interactive system? As you can see we talked of four components organized into two sub layers, but none of these components clearly capture the user-centric miss.

(Refer Slide Time: 45:47)



A Framework for UCC

- User-centric considerations missing in the previous layered view
  - We need to explicitly incorporate "user-centric" concept in the view
  - **We shall try to do that in the next lecture**

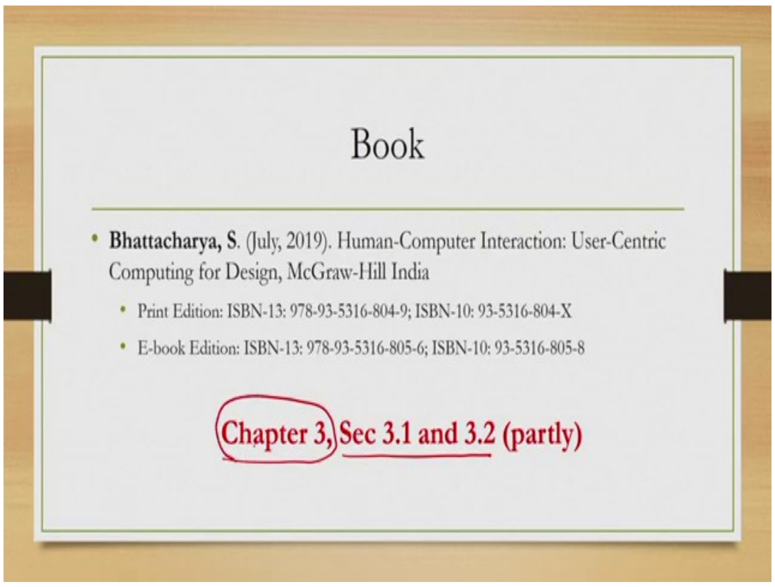
So, what is missing is that the user-centric considerations are not explicitly incorporated in the layered view of the software. So, what we need is some other components which explicitly capture the user-centric concerns, so that we can talk of this layered view as representing a user-centric software, so that modification that refinement we will discuss in the next lecture. So, we have reached the end of the lecture.

So, let us just summarize what we have learned. We started with the basic definition of computation or mathematical computation, then we have equated this idea with the idea of a computer engineer who thinks of computer as a stored program device. And the way this device or machine works is similar to the way a discrete-state machine works which is the traditional model of computation. So, essentially both the concepts are similar. So, a software the way it works can be equated to the working of a discrete-state machine.

Then we have introduced the idea of cognition cognitive state and discussed that how the idea of state in the context of interactive system is different from that of a purely abstract state machine. Here the state refers to the combination of both state of a computer as well as state of the user. And this user state is essentially the cognitive state which it has been shown works in the same way as a state in a discrete-state machine. So, the process of human cognition can be equated to the process of computation. So, essentially this combination is justifiable is logical and we can combine these two concepts to define a combined state of the system.

And finally, based on this knowledge, we can further simplify our understanding of a software in terms of a layered view. So, in this layered view, we can view different components that are typically are supposed to be part of an interactive system in the form of layers sub layers. But whatever view we have so far discussed does not explicitly incorporate the various user-centric concerns, and we need to modify it we need to come up with a better view which we will discuss in the next lecture.

(Refer Slide Time: 48:22)



Book

- **Bhattacharya, S.** (July, 2019). Human-Computer Interaction: User-Centric Computing for Design, McGraw-Hill India
  - Print Edition: ISBN-13: 978-93-5316-804-9; ISBN-10: 93-5316-804-X
  - E-book Edition: ISBN-13: 978-93-5316-805-6; ISBN-10: 93-5316-805-8

Chapter 3, Sec 3.1 and 3.2 (partly)

So, whatever we have discussed today are taken from a part of section 3.2 and section 3.1 from chapter 3 of this book. So, you will get all this material from this chapter.

Thank you and goodbye.