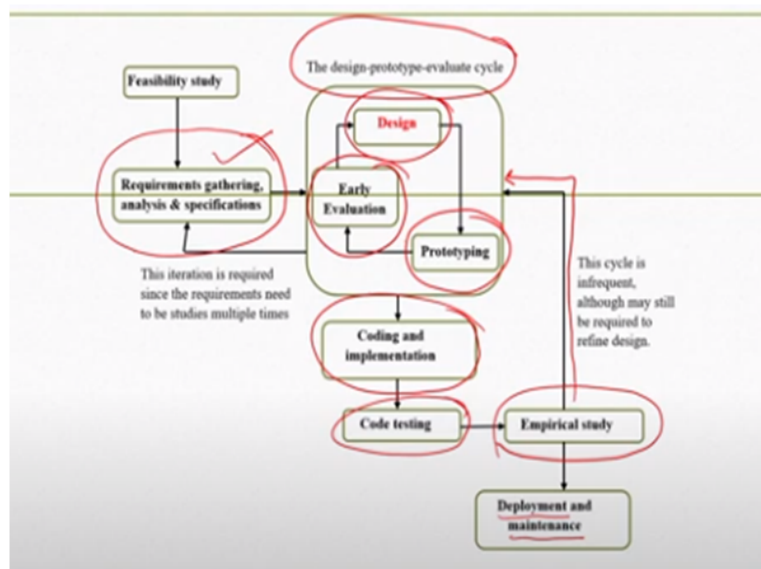


Design & Implementation of Human – Computer Interfaces
Prof. Dr. Samit Bhattacharya
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Module No # 05
Lecture No # 21
Data Flow Diagram

Hello and welcome to the NPTEL MOOCS course on design and implementation of human computer interfaces, lecture number 19.

(Refer Slide Time: 00:52)



So before we start we will quickly go through the stages of the interactive system development life cycle, and then we will identify the stage where we are going to have the material for today's lecture. So we have a development life cycle for interactive systems. We have several stages, the requirement gathering stage is the first stage which we discussed. This is followed by a cycle consisting of 3 stages; this is called the design prototype evaluate; cycle.

Where we have the design stage, the prototype stage, the evaluation stage and these 3 stages work in a cycle till we arrive at a stable design; this cycle is primarily meant for design of interface. Once we get an interface design ready we then start working on design of the system, where our main focus is the modules of the system and how they interact with each other; so that is the second type of design.

So in this interactive system development life cycle we have 2 design concepts one is design of the interface primarily from the point of view of usability; and the other one is design of the system primarily from the point of view of code management. Now once we are ready with a system design then we go for the next stage that is coding or implementation of the design; this is followed by testing of the code so at the end what we get we get an executable system.

Once the system is ready and executable we go for another stage called empirical study in this; stage we test for usability of the end product. Now empirical study is different than; code testing and we shall have detailed discussion on these stages in later lectures. There may be a cycle from empirical study to the design prototype evaluate; cycle and subsequent stages. If we find any usability issues during the empirical study then we may need to revisit our design, design of the interface as well as system; and then revise our implementation and further test our code.

So that cycle may be there however our objective should be to minimize this cycle at maximum one or 2 times more than that means it will result in huge cost and time overrun. So once we have a system which is executable and usable after going through all these stages we go to the deployment stage and maintenance stage. So these are the stages of an interactive system development life cycle which we are discussing.

We have already covered the requirement gathering stage. We have covered in detail the design prototype evaluate; cycle for interface design. Currently we are discussing the second type of design that is design of the system.

(Refer Slide Time: 04:22)

Recap

- We learned basics of system design
- Today we shall learn about one “language” for expressing system (code) design
 - DFD

So in the earlier lectures we have learnt the basics of system design, what are those basic concepts that let us quickly revisit. We will first revisit those concepts and then we shall start our discussion for this lecture that is one language for expressing the design of the system. So we earlier discussed certain concepts including language for expressing our design and in today's lecture we are going to cover one such language in detail.

And that language is DFD or data flow diagram in earlier lectures we have given some idea of DFD in terms of one example. However, today we are going to discuss in detail the different conventions, notations and symbols used in DFD and how it can be utilized to express a design idea.

(Refer Slide Time: 05:35)

Background

Before we start our discussion on DFD let us quickly recap what we have learned earlier about system design.

(Refer Slide Time: 05:44)

Design Concerns

- Two issues
 - Where to start?
 - How to represent (design language)?

So whenever we are concerned about design any design not necessarily system design then there are primarily 2 issues. Where to start and how to represent the design in other words what language to use to represent the design.

(Refer Slide Time: 06:08)

Where to Start?

- SRS – Software Specification Language
- Two phases
 - Preliminary (high-level) design
 - Detailed design (also called module-specification document)

In the context of system design when we try to address the question where to start the design process we said that the starting point can be the SRS document or software requirement specification document. Now SRS is a language which is used to specify or express the requirements for the system and SRS can give us the starting point for our system design. With that starting we can go ahead and ideate the design however there are 2 phases in the system design process.

First we go for preliminary or high level design; in high level design our primary concerns are design of modules and sub modules; and how they interact with each other. Then in the second phase we go for detailed design of individual modules and their components; and this detailed design document which we get after this process is over is also called module specification document. So there are 2 phases, high level design and detailed design.

(Refer Slide Time: 07:40)

High-Level Design

- Identification of modules
- Control relationships between modules
- Definition of interfaces between modules

Earlier we discussed both in brief what happens in high level design; we identify the modules at a very high level. The control relationships between the modules that also we identify, and the definition of interfaces between modules, primarily these 3 things we focus on in the high level design phase. Now when we are trying to design a system different approaches can be taken broadly there are 2 such approaches.

(Refer Slide Time: 08:25)

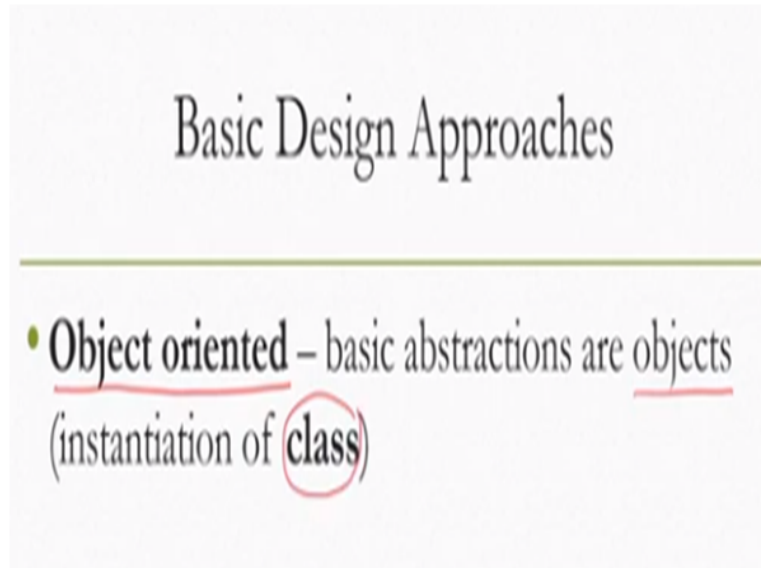
Basic Design Approaches

- Function oriented – basic abstractions are
functions

As we have seen earlier one is the function oriented design approach so when we are trying to design a system using a function oriented approach; the basic abstraction that we consider are the

functions or procedures. So any problem is considered to be broken down into a set of functions which are interacting with each other.

(Refer Slide Time: 08:53)

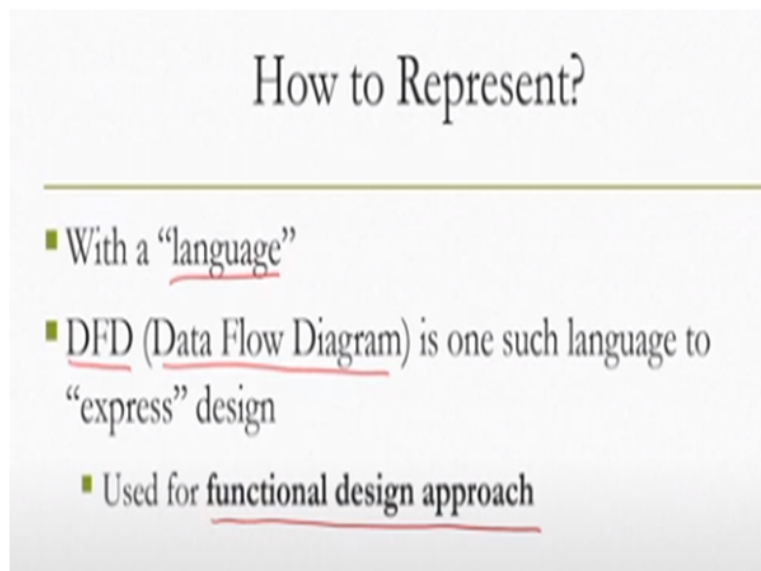


Basic Design Approaches

- Object oriented – basic abstractions are objects
(instantiation of class)

There is another broad approach that is called object oriented design approach; in this the basic abstraction that we consider is object. So there are objects and a system is designed considering the objects and their interaction with each other. Now the objects are instantiation of a concept called class; so we will discuss more about it later.

(Refer Slide Time: 09:27)



How to Represent?

- With a “language”
- DFD (Data Flow Diagram) is one such language to “express” design
 - Used for functional design approach

Now let us come back to the second question that we encounter while we are trying to design a system. That is once we have come up with some design idea how do we express it how do we represent it? So, that it is understandable to others those; who are not part of the design process. In order to do that we definitely need a language. Anything we need to express requires a language. The same is true here. In order to express our design idea we need a language.

One such language is DFD or data flow diagram which we can use to express our design and this language we are going to learn in this lecture. One thing we should keep in mind is that DFD is one graphical language that means; it uses graphical notations to express the design and it is primarily useful for functional design approaches. So when we are trying to design something using the function oriented approach then expressing that design idea will be easier if we use DFD

(Refer Slide Time: 10:45)



With that basic idea let us now understand and learn what is DFD, what are the conventions used in DFD; and how we can use it to express our function oriented design ideas.

(Refer Slide Time: 11:07)

DFD

- Represents “flow of data” through a process or a system
- Focus on data “movement” between external entities and processes, and between processes and data stores

As the name suggests DFD essentially represents flow of data through a process or a system; DFD stands for as we have seen a data flow diagram; that means it represents flow of data through a process or a system. So the focus is on data movement between external entities and processes and between processes and data stores. So when we are trying to use DFD to express our design ideas the focus is primarily on movement of data between external entities and processes as well as between processes and data stores.

So there are 3 concepts involved in this statement: one is external entities, one is process, one is data store. Let us see what; are these concepts and how we can use DFD to express our design using these concepts.

(Refer Slide Time: 12:22)

Why DFD?

- Provides overview of
 - What data processed by a system
 - Transformations that are performed
 - Which data are stored
 - What results are produced and where they go
- Graphical nature makes it a good communication tool between
 - User and (system) designer
 - Designer and developer

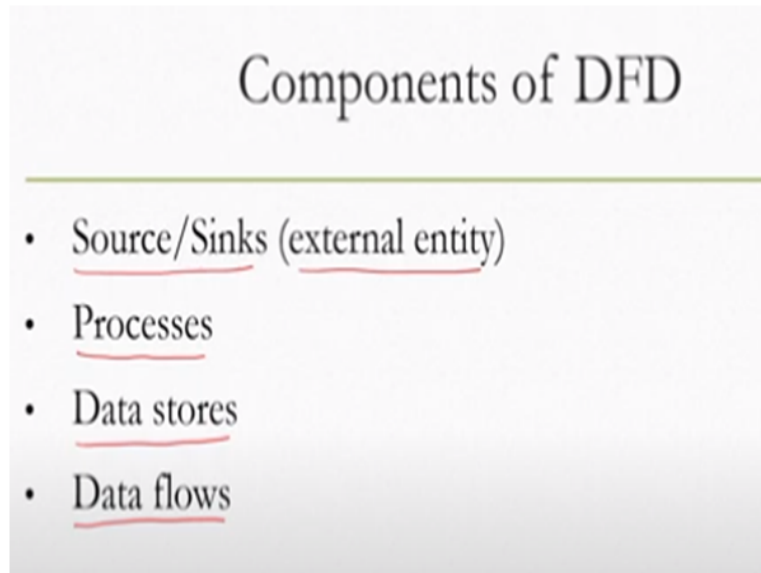
Before we learn about DFD, it may be pertinent to ask the question why we should use DFD. So there are several reasons DFD provides an overview of the data that are processed by a system; so different types of data that a system processes. It also provides an overview of the transformations that are performed on this data by some processes. It also gives us some idea of which data is stored and which is not; stored.

And from DFD we can get the information about what results are produced and where they go so there are several reasons which makes DFD a good choice for expressing a system design. That is not all there are other reasons as well, because DFD is a graphical language that means it uses graphical notions; notations to express the design idea. Now this graphical nature makes DFD a good communication tool between the user and the designer, as well as the designer and the developer.

Now here the term user need not be confused with the end user rather it can be considered to be the clients or customers of the product. They may like to see the design and to express the design to them. DFD can be good because it is graphical and does not require too much technical knowledge. Similarly the designers can find it easy to express their design ideas to the development team who will be ultimately responsible for implementing the design.

Because it is a graphical language development team need not spend too much time understanding the design so there are several reasons including; easy communication between different stakeholders.

(Refer Slide Time: 14:56)



So what are the components of a typical DFD diagram? Earlier we mentioned that DFD focuses on data flow between external entities processes and data stores, so these are the components. So when we talk of components of DFD we talk of source or sink of the data which are typically external entities; processes which as the name suggests, process the data and produce; output and data stores which store the data.

So these 3 are the primary components of any data flow diagram along with that of course we should not forget that since it is depicting data flow so flow of data itself is a component and how do we depict it is an important issue in constructing DFD.

(Refer Slide Time: 16:06)

Component Representation

Symbol	Symbol 1 (Gane & Sarson)	Symbol 2 (DeMarco & Yourdan)
External entity	NAME	NAME
Process	NAME	NAME
Data store	D1 NAME	D1 NAME
Data flow	Name	Name

Let us have a look at how these different entities or different components of a DFD are represented. In fact there are broadly 2 notations which are used, any one you can, use but try not to mix them one is by Gane and Sarson, other one is by DeMarco and Youdan. So when you are using DFD notations you should stick to any one particular notation rather than mixing up notations from these 2 different sets of notations.

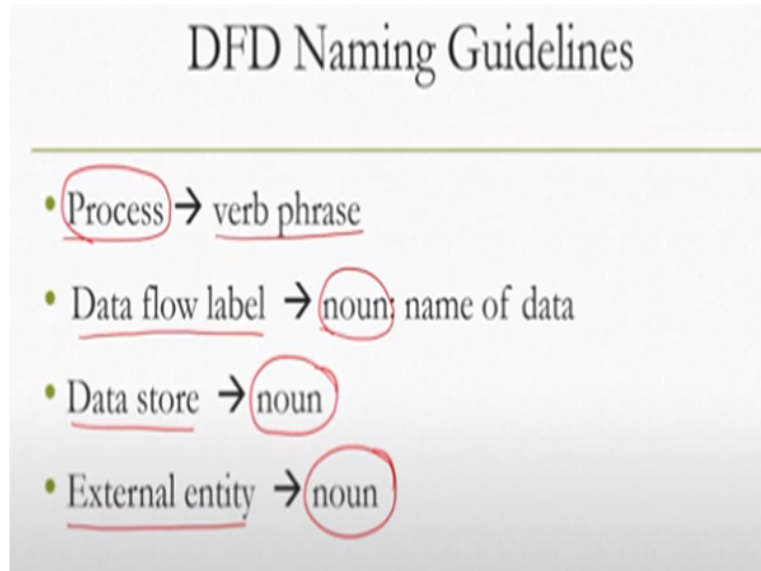
So when we are trying to represent an external entity, according to symbol 1 we should use a rectangle with a name of the entity inside it, in fact this symbol is the same in the other set of symbols with the same rectangular symbol with name. For; process the other component in symbol 1, we have a special symbol, which is a rectangle with the corners, which is somewhat circular then there is this line and then the name of the process.

Whereas in symbol 2 we have a different symbol with the name inside it which is completely different from what is used in symbol 1. For the data store we have an open ended rectangle with 1 notation and the name of the data store inscribed inside. Now the notation is within a square like shape; whereas in symbol 2 the same data store is represented again with an open ended rectangle, but here the square like part is missing.

Instead everything including the identifier of the data store as well as the name both are inscribed inside for data flow. The other component notation used is the same in both the symbol sets; an arrow with a label which indicates the name of the data flow; in both the cases the notation used

is the same. So when we are trying to express some design using DFD we should stick to either symbol 1 or symbol 2, we should not mix notations from these 2 symbols.

(Refer Slide Time: 19:06)



Now while using the names or while trying to assign some names to the different components generally some conventions can be followed. So when you are trying to give a name to a process try to give it in the form of a verb phrase. When a data flow level is to be given try to use a noun as the name for a data store, also try to use a noun as the name. And for external entities again a noun should be used as the name of the entity.

So out of the 4 components for processes we should use verb on verb phrase; to name it for the other 3 namely data flow, external entity and data store we should stick to use of nouns for naming those components.

(Refer Slide Time: 20:23)

External Entity

- People or organizations that send data into or receive data from system
- They either supply or receive data
 - Source – supplies data to system
 - Sink – receives data from system

Now let us slowly try to understand what are these components and what they signify; let us begin with an external entity; what are these things; people or organizations that send data into or receive data from the system. So you are designing a system, now there may be people or organizations who; send data to the system or receive data from the system; such people or organizations are generally considered to be external entity; for the system.

So their job is to either; supply data to the system or receive data from the system, so either supply data or consume the data. So if an external entity is a source then as the name suggests they supplies; data to the system; in contrast if the external entity is a sink then that means it receives data from the system. So; external entities are people or organizations that are receiving or sending data to or from the system. When they receive data they are called sinks. When they send data to the system they are called sources.

(Refer Slide Time: 22:03)

Process

- Process – series of actions that transform data
- Notation
 - Straight line with incoming arrows → input data flows
 - Straight lines with outgoing arrows → output data flows
 - Labels assigned to data flow



The next component is the process. What is a process? As the name suggests it processes something; so we can say that a process is a series of actions that transform data. So when we say a process transforms data that means it acts on some data that is coming from somewhere and it produces some data; that is going to; somewhere. So typically the notation that is used or that straight line with incoming arrows, in coming to the process notation indicates input data flow, which is quite obvious.

Straight lines with outgoing arrows indicate output data flow, so if we are using a process symbol like this using the symbol 2 notation then, this is the input data flow; and this will be the output data flow. And of course as we have seen the data flow should be labeled to indicate what kind of data is being moved, so there should be some level on the arrows.

(Refer Slide Time: 23:31)

Data Store

- Represents permanent data used by system
 - Data can be written into data store – denoted by an incoming arrow
 - Data can be read from a data store – denoted by an outgoing arrow

The third component is data store, now data stores represent permanent data used by the system. So some data can be transferred in nature and they get destroyed whereas some data remains for use in subsequent iteration of the processes. And data stores are essentially representing those permanent data. Now data can be written into a data store in that case it will be denoted by an incoming arrow or data can be read from a data store.

So in that case it will be denoted by an outgoing arrow; it is a simple convention, if data is written into a data store indicate that by using an arrow that is going to the data store. And if data is being read from the data store then that is represented by an arrow, with the arrowhead outgoing from the data store.

(Refer Slide Time: 24:49)

Data Flow

- Data flow - depicts actual flow of data between elements
 - Connects processes, external entities and data stores

Finally we have data flow the other component of a DFD, a data flow depicts actual flow of data between different elements; now here by elements we refer to the other components namely external entity, data store and process. So data flow connects these elements processes external entities and data stores; without data flow of course, we will not be able to show the connection between them.

(Refer Slide Time: 25:24)

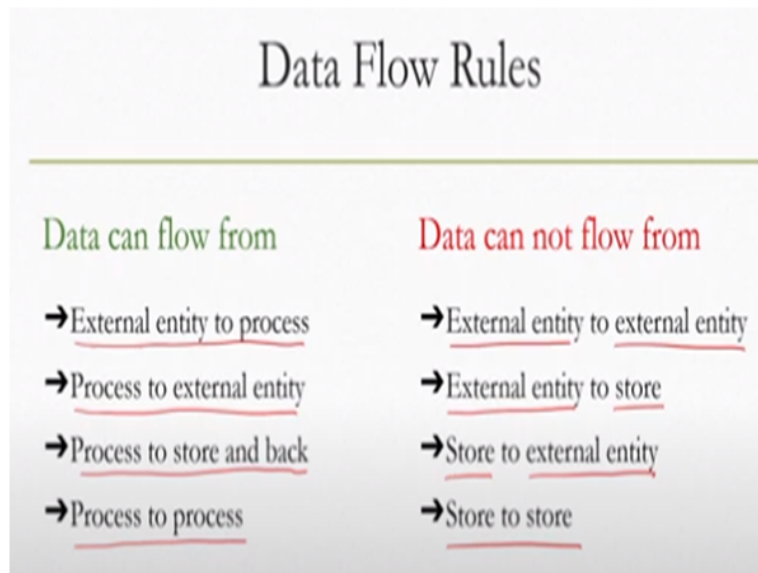
Data Flow

- Notation – an arrow with label
 - Generally unidirectional
 - If same data flows in both directions, double-headed arrow can be used

And the notation as we have seen earlier is simple; an arrow with a label indicating the type of data that is being represented. Now when we are drawing data flow or representing data flow generally the arrow is unidirectional however; if the same data flows between 2 elements in both

directions then we can use a double headed arrow to represent the same rather than using 2 arrows in opposite directions. So that will be a shorthand notation for indicating movement of data in both the directions.

(Refer Slide Time: 26:19)

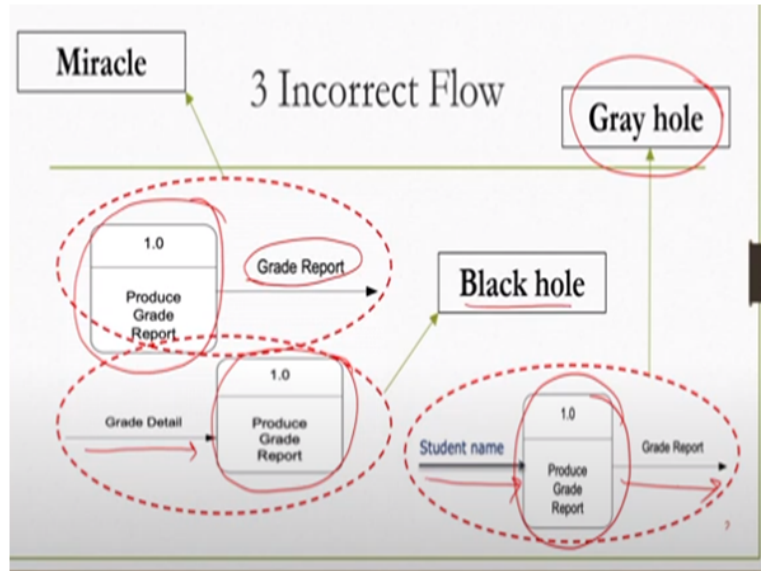


So when we are trying to create a data flow diagram there are some rules that we should follow while trying to create it. We should remember where data can flow, and where data cannot flow. Data can always flow from external entity to process; from process to external entity; from process to store data store, and back, and between the processes from one process to another process.

So these are valid directions for data flow in any DFD from external entity to process, and vice versa from process to external entity; similarly from process to data store, and from data store to process, and also between 2 processes. However we should also remember invalid data flows data can never flow from an external entity to another external entity; an external entity to a data store directly by passing a process; from a data store to an external entity again bypassing the process; and between 2 data stores.

So data can never flow from an external entity to a data store, and vice versa, or between 2 data stores, or between 2 external entities. So these are the rules that we should remember while trying to create data flow diagrams with valid data flow.

(Refer Slide Time: 28:17)



Let us see some examples of incorrect depiction of data flow, suppose you have created one process and it produces some data, say some report in this case. So the notations are fine, the process is represented with suitable notation, data flow represented with suitable notation. However note that here there is no input data, in order to make the output the process should receive some input data which it is going to work on.

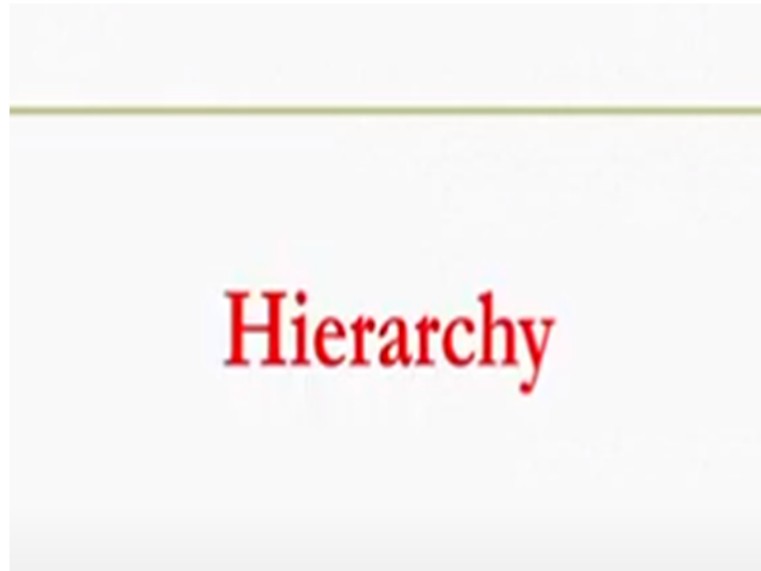
If a process is mentioned without any input data but it is producing some output, then that is an incorrect way of depicting DFD; and this is generally termed as a miracle DFD. Similarly suppose there is a process which only takes an input but does not produce anything, that means whatever data goes into the process vanishes this situation again is something which is incorrect and generally it is called a black hole situation.

Sometimes it may happen that there is some input data, there is a process, and there is some output data. But we forgot to mention where this output data goes if we simply left it; leave it hanging like this then that is also incorrect. Output data must go somewhere, and input data must come from somewhere so this is a situation which is called a gray hole situation; where we forgot to mention the source or the sink of data.

So these are some of the 3 incorrect ways of depicting DFD's. It may be remembered that there may be some other situations where incorrect depiction of DFD happens; but these are common ones. So while creating DFD you should always keep in mind that you must specify an output

data flow when you are dealing with a process with an input data flow; or you must specify an input data flow when you are dealing with a process with an output data flow. And you must specify source and sync of input and output data flows for any process.

(Refer Slide Time: 31:10)



One important thing in any system design is to create the design in a hierarchical manner, recollect our discussion from the previous lecture where we mentioned. That; in order to manage the design and maintain the design it is preferable to break it down into smaller problems or smaller systems. In effect we are creating a hierarchy of processes or modules to represent the whole design. Our language should support that hierarchical design idea and DFD itself is well equipped to support a hierarchical design.

(Refer Slide Time: 31:59)

Representing Complex Designs

- Real-life systems can be very big
 - Involving large number of processes and data stores
- Representing such systems with DFD difficult
 - Very complex diagram
 - Difficult to understand
 - Difficult to modify

So when we are dealing with complex designs generally systems which are very big real life systems such systems involve a large number of processes and data stores. And it is difficult to manage them unless we do something, if we simply try to create a DFD with this large number of processes and data stores. Then the resulting DFD will be very complex, it will be very difficult to understand and it will be even more difficult to modify.

So if we find some problem and we want to modify then; large portion of the DFD probably needs to be modified in order to get the refined and revised design. So these are 3 important problems when we are dealing with large systems involving a big number of processes and data stores.

(Refer Slide Time: 33:02)

Representing Complex Designs

- Way out – decomposition
- Create hierarchy of “levels”

To manage such big systems, what we can do is we can go for the idea of decomposition of the design. This indicates that we create a hierarchy of levels of design.

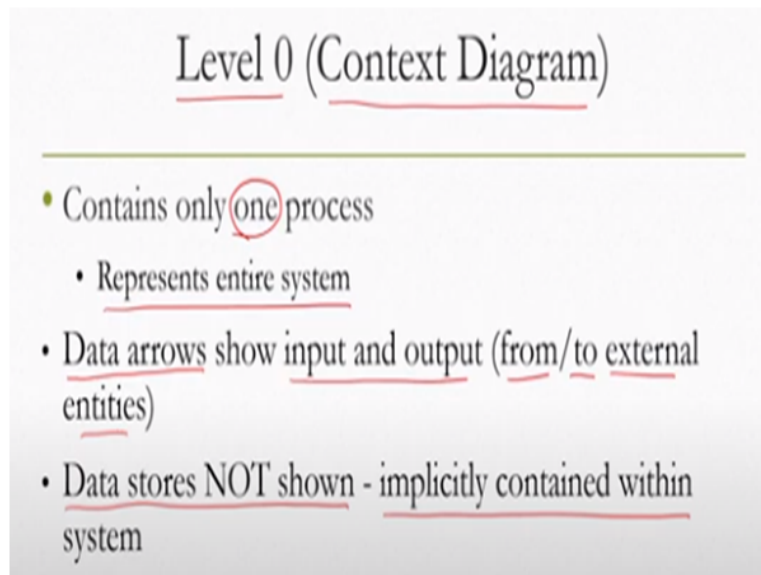
(Refer Slide Time: 33:24)

DFD Levels

- At least 3
 - Level 0 – context diagram
 - Level 1 – overview diagram
 - Level 2 – detailed diagram
- There can be further levels, if required

Generally it has been found that at least 3 levels are required to represent complex systems, more levels may be necessary but lesser levels generally lead to incomprehensible and difficult to manage designs expressed in dft. So we have level 0 which is generally called a context diagram, level 1 generally called overview diagram, and level 2 which is a detailed diagram. Now we can go for subsequent levels also which will be; again detailed diagrams but at least these 3 levels are required.

(Refer Slide Time: 34:07)

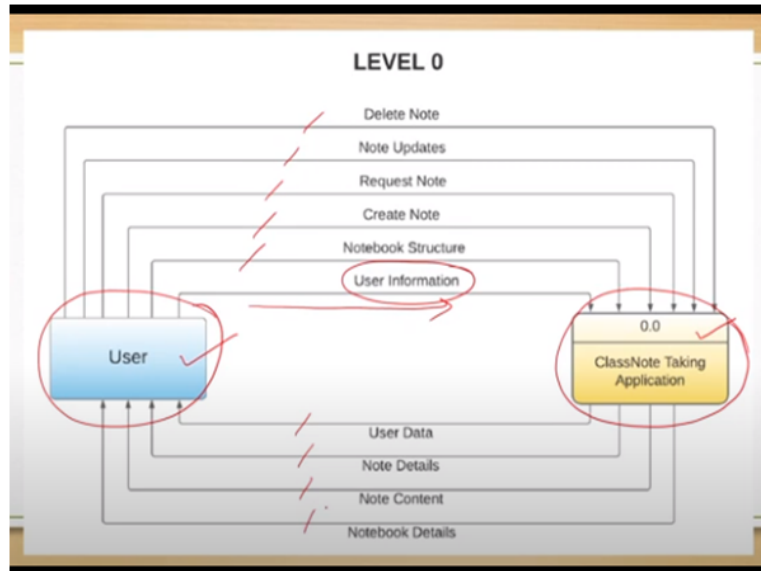


Let us try to understand these levels, and how we can create these levels. Let us start with level 0 which is called a context diagram. Now this level is a special level here; there is only one process, so the entire system is represented as a single process; with the corresponding process notation. Data arrows show input and output from an external entity or external entities or 2 external entities.

So at this level we have data arrows that depict data flow from external entity to process, and vice versa. At this level there is no data store shown because we are using a single process to represent the entire system; everything is content within that system, so separately no data store is shown. It is assumed that it is implicitly content or these data stores are implicitly contained within the system itself.

So at level 0 or context diagram level we have one process representing the whole system, all external entities, data flow between external entities and processes and the process and no data stores are shown at this level.

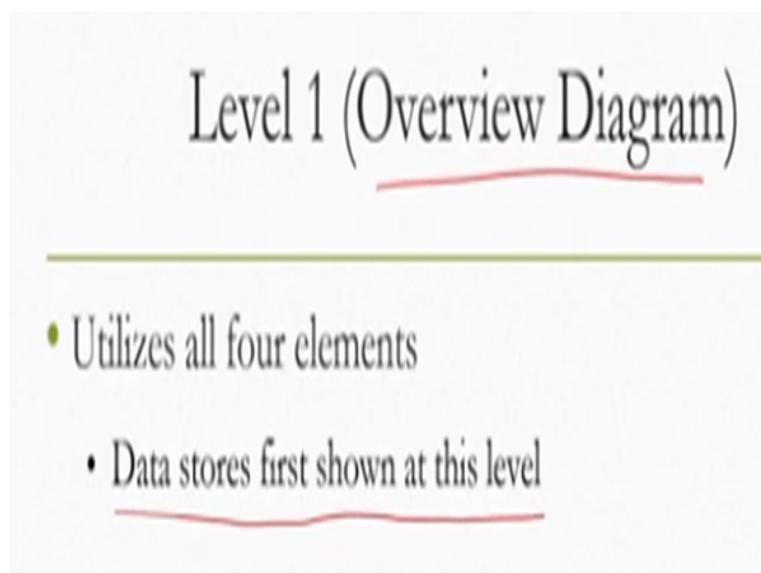
(Refer Slide Time: 35:38)



Let us see one example: suppose we are asked to create an application, a class note taking application so this whole process, the whole system is represented as this context level process class note taking application. Now there is one external entity user for this application, and this data flow is shown with an arrow and levels. So between the external entity and the process all data flow is shown with the kind of data that is being flown.

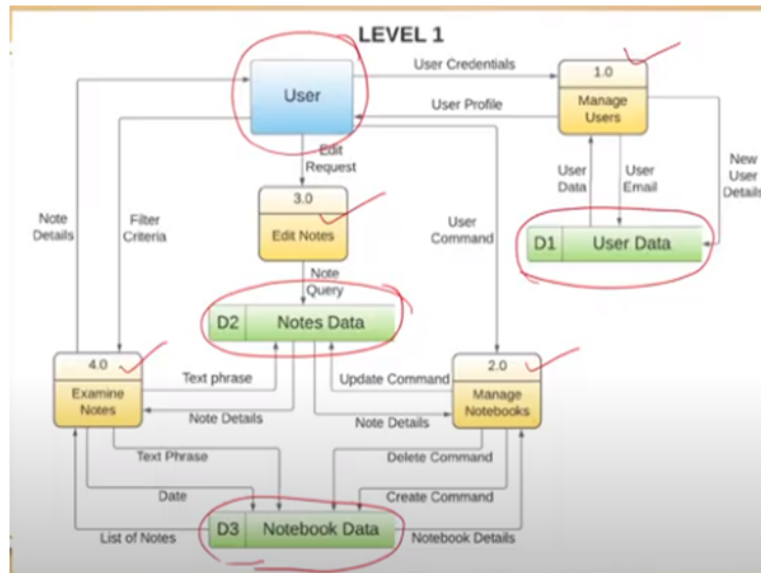
So data flow arrows with levels are drawn at this level note that here we have one process, here an external entity, here the data flows all the data flow arrows to the process and from the process but no data store explicitly shown.

(Refer Slide Time: 36:57)



Then comes level 1 or overview diagram level 1 is also called overview diagram, here all the 4 elements are utilized, that is process, data store, data flow and external entity. Data stores are first shown at this level, so if the system has one or more data stores those are shown at this level for the first time.

(Refer Slide Time: 37:29)



So if we want to draw a level one or overview diagram for the class note taking application it may look something like this, of course there can be different designs, so this is one possibility. So we have this external entity user shown here again there are 4 processes shown earlier all these 4 processes together were shown as a single process. Now we have process 1 manage users, process 2 manage notebook, process 3 edit notes and process 4 examine notes.

Similarly there are 3 data stores shown data store 1 user data, data store 2 notes data, data store 3 notebook data and it also shows data flow between the processes external entities and data stores so this is level one diagram. So when we are creating level one diagrams we should remember that we can use all 4 components of a DFD at this level for the first time and we should show external entity data stores processes and all data flow between all these 3 elements at this level.

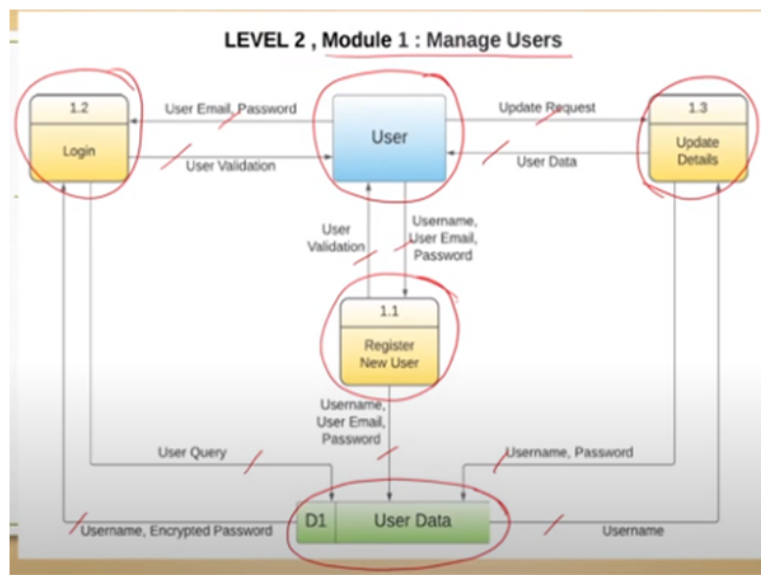
(Refer Slide Time: 38:59)

Level 2 (Detailed Diagram)

- Detailed representation of level 1 processes
- Like level 1, utilizes all four elements

So that is level one then we come to level 2 which is also called a detailed design diagram, here what we do we go for detailed representation of the processes that are depicted in level 1. So all level 1 processes are further decomposed and shown in level 2 if required. Now the nature of this level is similar to level 1 so here all 4 elements we can use to; so the detailed design of each of the processes of level 1.

(Refer Slide Time: 39:49)

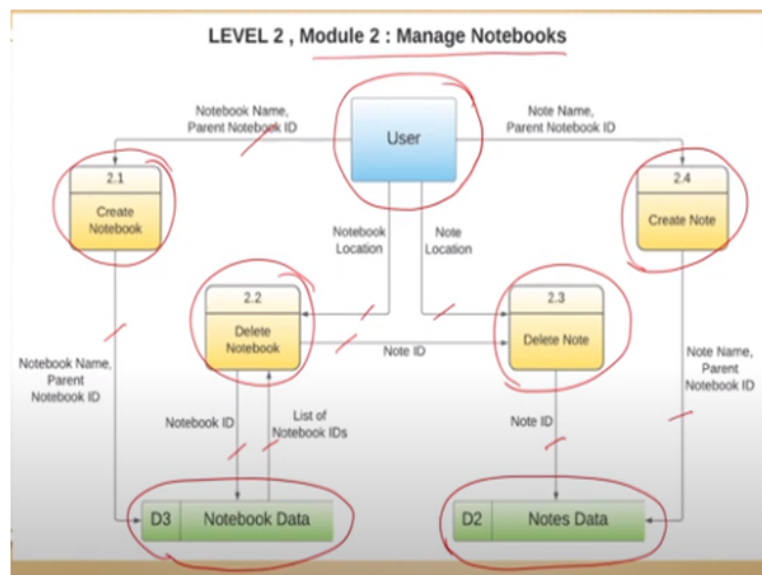


Let us continue with the same example, so in level 1 of the class note taking application we have seen 4 processes we can call them; modules. So let us see level 2 diagram; for module 1 that is manage users; so here we can see that there are 3 processes which are forming part of this

manage users module. Process 1 is register; new user, process 2 is login, process 3 is to update user details. It makes use of the data store user, data we can use external entity notation here also and all data flow between the processes.

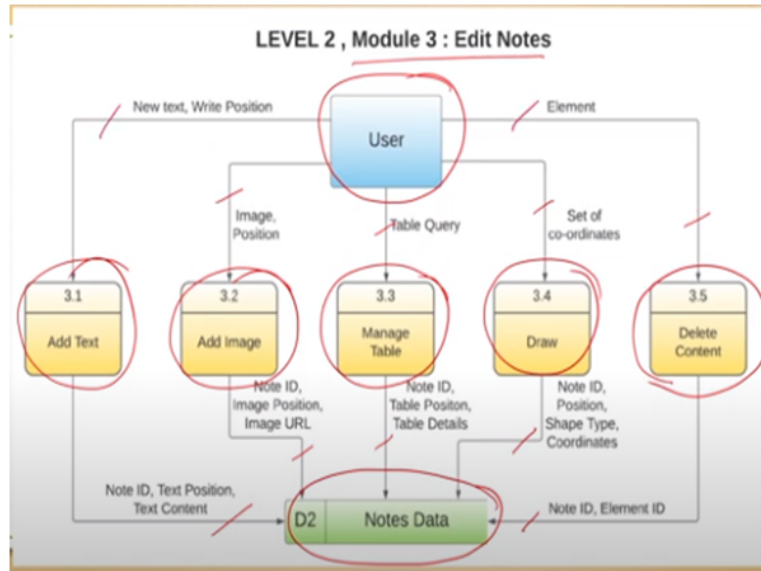
And the external entities are shown with appropriate notation at this level, now this level is only depicting the detailed design for individual processes or modules of level 1, so this is module 1 manage users.

(Refer Slide Time: 41:12)



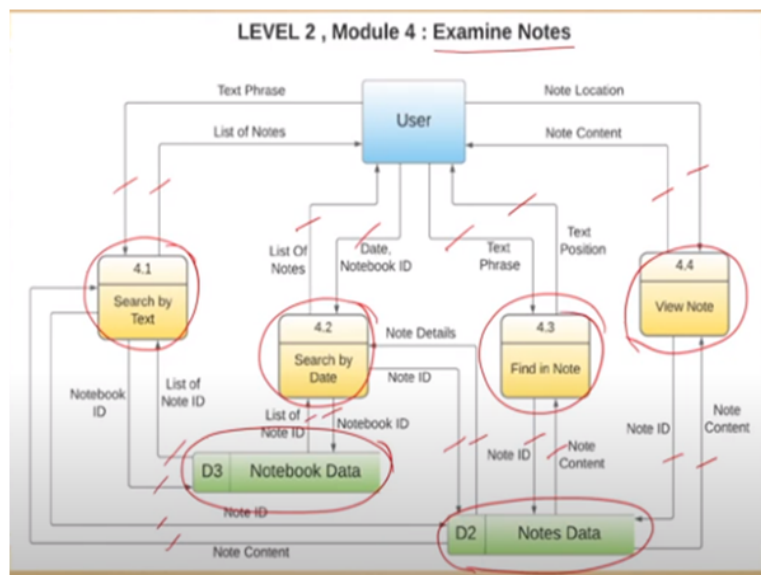
Let us see module 2 manage notebook; it contains 4 processes: create notebook, delete notebook, delete note, and create note; it makes use of the data stores notes data and notebook data which we have seen at level 1. And also it uses external entities as before and shows all data flow along with direction and level between the processes the data stores as well as the external entities.

(Refer Slide Time: 42:05)



Module 3 was edit notes which contains 5 processes: add text, add image, manage table, draw, and delete content; makes use of the data store notes data external entity is present here also. And all data flow between processes and the data stores, as well as between external entities and the processes are shown in this detailed diagram.

(Refer Slide Time: 42:48)



There was a fourth process or module examining notes in level 1; which can be further broken down into 4 processes: search by text, search by date, find in note, and view note; makes use of the 2 data stores notebook data, and notes data. And all the data flow between the process and the

data stores are depicted like before; similarly all the data flow between the external entity and the processes are depicted as before.

Now as you can see if we do not have the decomposition of how complex the single DFD would be; suppose instead of level 0, level 1, level 2 this hierarchical way of showing the design we wanted to show it at a single level. Now as you may recollect the system consists of 4 top level modules each module on an average has between 3 to 5 modules submodules or processes. So together there will be between 12 to 20 processes to represent this particular class note taking application design.

Now between 20 processes if we are trying to depict data flow then there will be lots and lots of such arrows going around; then it will become a really messy picture, really messy diagram. Where; there will be lots of overlapping arrows, lots of crossing arrows and it will be very difficult to comprehend the direction of data flow, the type of data flow, because there will be a large number of levels. So it will be very difficult to read or comprehend the levels from this messy picture. So the hierarchical way of depiction gives us a nicer and cleaner mechanism to represent complex designs.

(Refer Slide Time: 45:32)

Creating Good DFD!

- Use meaningful names for data flows, processes and data stores.
- Use top down development starting from context diagram and successively levelling DFD
- Stop decomposition if levels become trivial (no significant change)
- Remember - only previously stored data can be read
- Remember - data stores cannot create new data

Hierarchy is of course one way of creating a good nice clean design, now there are some other conventions you should follow while creating a good DFD. One is try to use meaningful names for data flows, processes and data stores. This will be helpful in understanding and remembering

these components. Use top down development starting from context diagram and successively creating the DFD levels.

So do not start from bottom up, that is first try to understand all the processes and then try to create the hierarchy; should not follow that approach instead you should start from the top or context level and then successively expand in subsequent levels the design. Stop decomposition if levels become trivial that means no further significant changes are depicted in the levels, then you should not go for further decomposition.

Always keep in mind that only previously stored data can be read and data stores cannot create new data; so unless there is a process new data cannot be created, and a process can read only data that is already available. These 2 may seem simple but there are actually cases where people make mistakes because they do not keep these things in mind. So these are some of the things that you should keep in mind while trying to create a good DFD, now in DFD we mentioned that we have data stores.

However; DFD itself does not tell us how to represent data that are part of data stores. So we have a notation to represent data stores but that notation does not reveal the internal structure of the data. It is often useful to know the internal structure and for that we can go for another graphical notation that is called ER diagram or entity relationship diagram.