Software Engineering
CS-303

Software Process & Process Models

Lecture # 7,8
04, 06 Feb

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

- DFD
- Context Level DFD

Data Flow Diagrams

- DFDs are graphic representation of the **flow of data** or information into and out of a system
  - ***what does the system do to the data?***

- A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system.
- The DFD also provides information about the outputs and inputs of each entity and the process itself.
- A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart

Why DFD?

- DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer.
- DFD has often been used due to the following reasons:
  - Logical information flow of the system
  - Determination of physical system construction requirements
  - Simplicity of notation

DFD Symbols

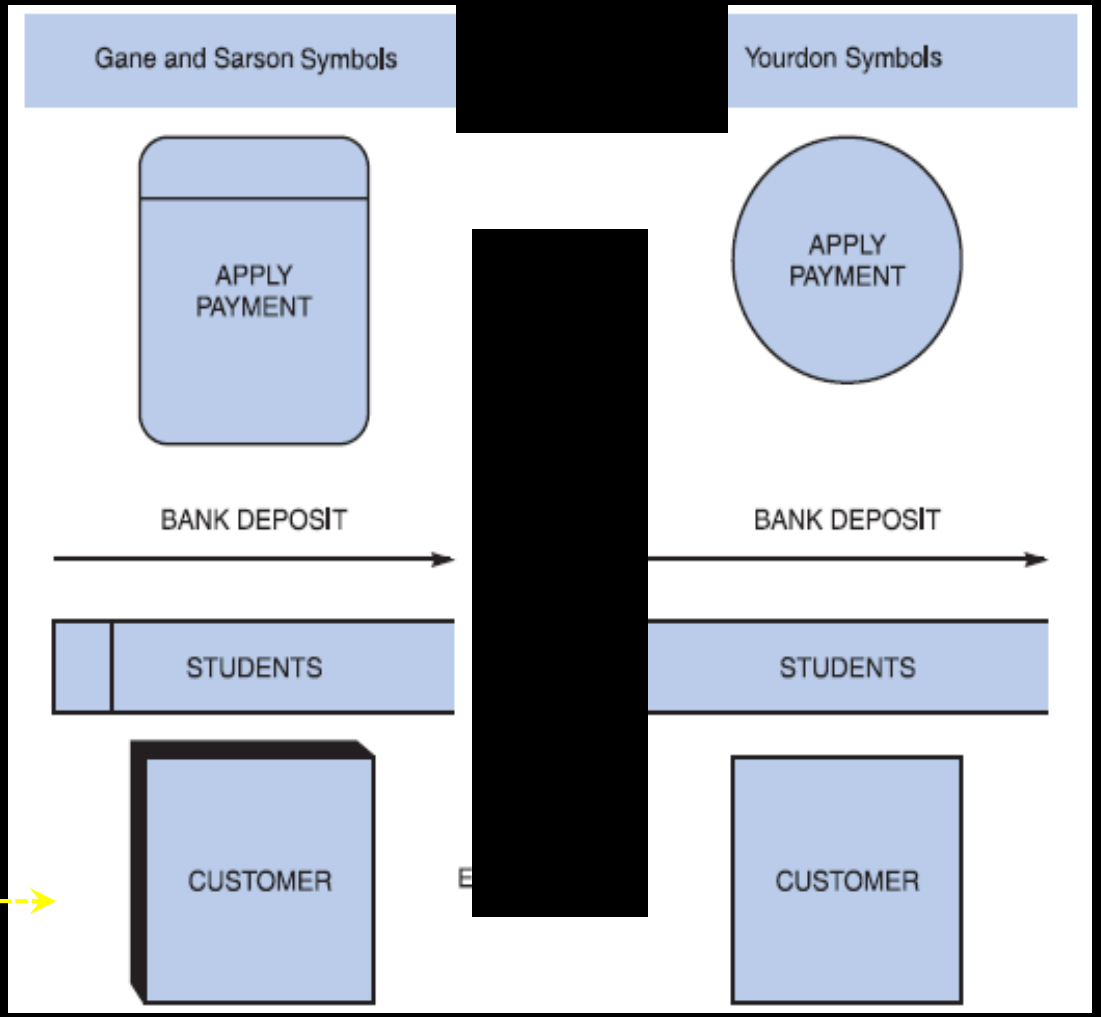
DFDs use ***four basic symbols*** that represent

Processes

Data flows

Data stores

External Entities

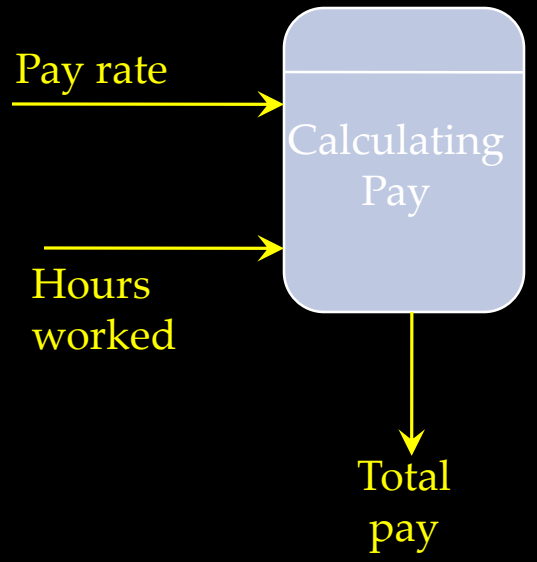


Data Flow Diagrams

Process Symbol

- A process receives ***input data*** and produces ***output*** .

- For instance, the process for ***calculating pay*** uses two inputs (pay rate and hours worked) to produce one output (total pay).
  - Processes can be very simple or quite complex.
  - Note: Processing details are not shown in a DFD.



Process Symbol

- A process symbol can be referred to as a **black box**, because the inputs, outputs, and general functions of the process are known, but the ***underlying details*** and ***logic*** of the process are ***hidden***.

- By showing processes as black boxes, an analyst can create DFDs that show how the ***system functions***, but ***avoid unnecessary detail***.

- When the analyst wishes to show additional levels of detail, he/she can zoom in on a process symbol and create a more in-depth DFD (leve-1 and level-2) that shows the process's internal workings.

Data Flow Diagrams

Data Flow Symbol 

- A data flow is a path for data to move from one part of the information system to another.

- A data flow in a DFD represents one or more ***data items***.

- In previous example we saw , the process for ***calculating pay*** uses two inputs (pay rate and hours worked) to produce one output (total pay).

Data Flow Symbol

- The symbol for a data flow **is a line** with a arrowhead.
- **Data flow name** appears above, below, or alongside the line.
- A **data flow name** consists of a singular noun and an adjective, if needed.

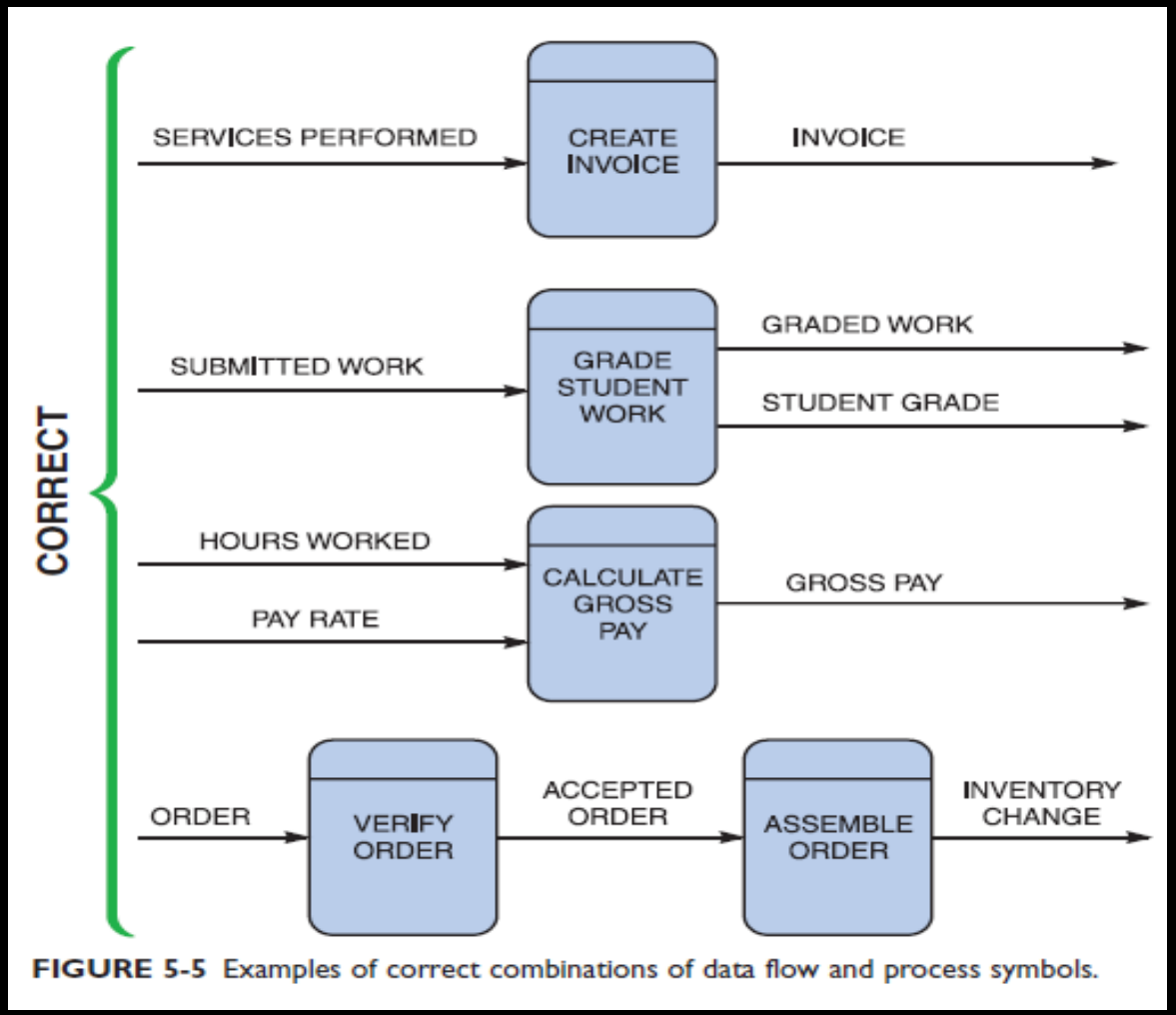


FIGURE 5-5 Examples of correct combinations of data flow and process symbols.

Data Flow Diagrams

Data Store Symbol

- A ***data store*** is used in a DFD to represent data that the system stores because one or more processes need to use the data at a later time.

- For instance, ***teachers*** need to store student scores on tests and assignments during the semester so they can assign final grades at the end of the term.

- Similarly, a ***company*** stores employee salary and deduction data during the year in order to print W-2 forms with total earnings and deductions at the end of the year.

- A DFD ***does not show*** the detailed contents of a data store.

Data Store Symbol

Examples of **data store** names are

DAILY PAYMENTS

ACCOUNTS RECEIVABLE

PATIENTS

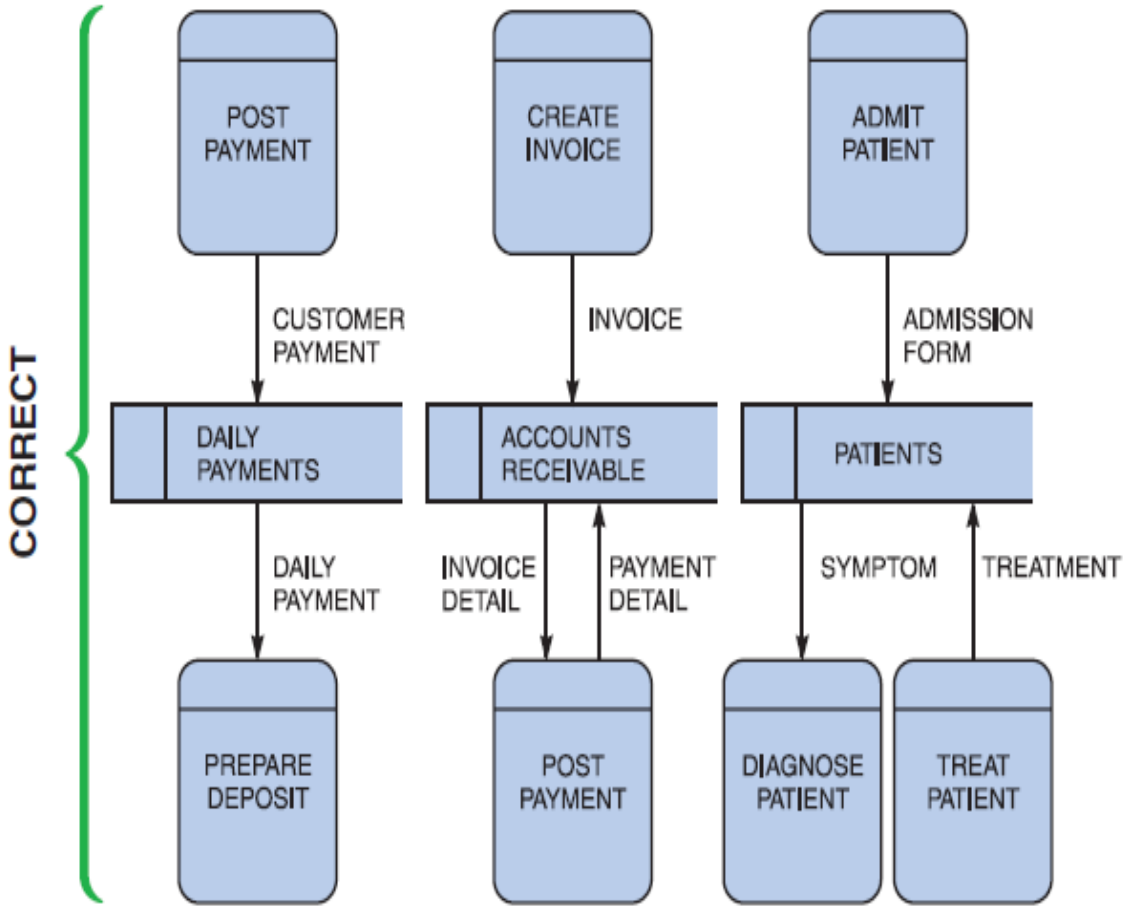


FIGURE 5-7 Examples of correct uses of data store symbols in a data flow diagram.

Data Flow Diagrams

External Entity Symbol

- The symbol for an ***entity*** is a ***rectangle.*** The name of the entity appears inside the symbol.
- DFD shows only ***external entities*** that ***provide data*** to the system or ***receive output*** from the system.
- DFD shows the ***boundaries*** of the system and how the system interfaces with the outside world.
- DFD entities also are called ***terminators***, because they are ***data origins*** or ***final destinations***.

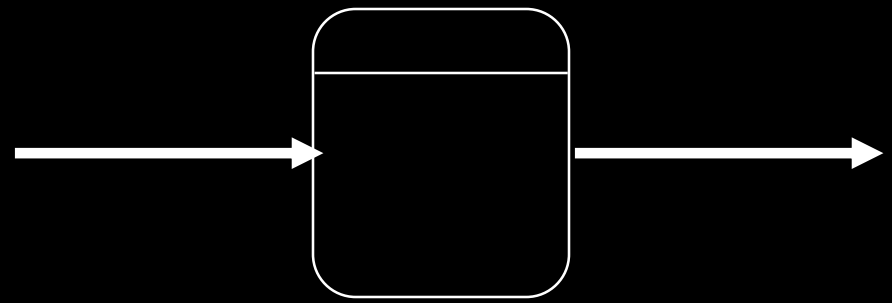
External Entity Symbol

- Systems analysts call an entity that supplies data to the system ***a source***, and an entity that receives data from the system **a sink**.

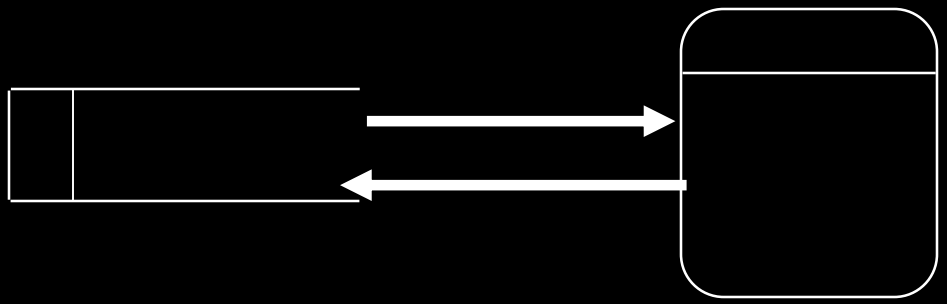
- An entity name is the singular form of a
  - department, outside organization,
  - other information system, or person.

- An external entity can be a source or a sink or both, but each entity must be connected to a process ***by a data flow***.

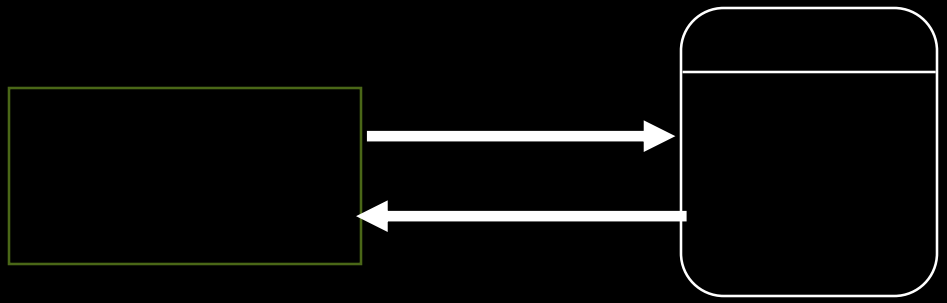
Rules for Drawing DFD's



A **minimum** of one data flow in and one data flow out of a process



A datastore **must** be connected to a process (either in, out, or both)

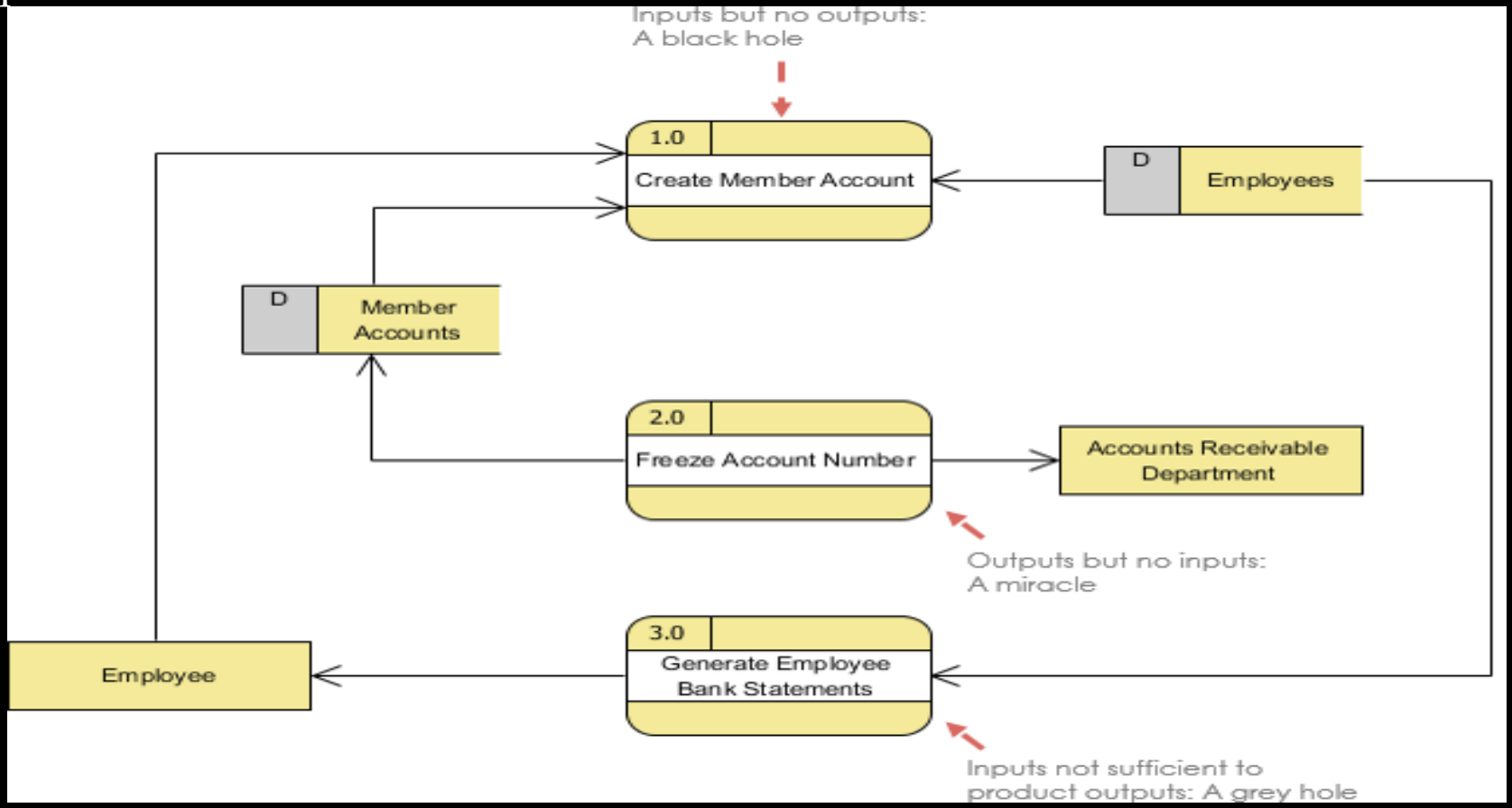


An external entity **must** be connected to a process (either in, out, or both)

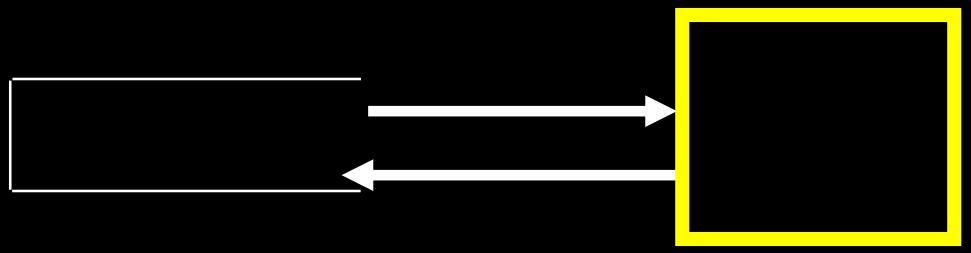
A single data flow **must** only flow one way

DFD: Common Mistakes

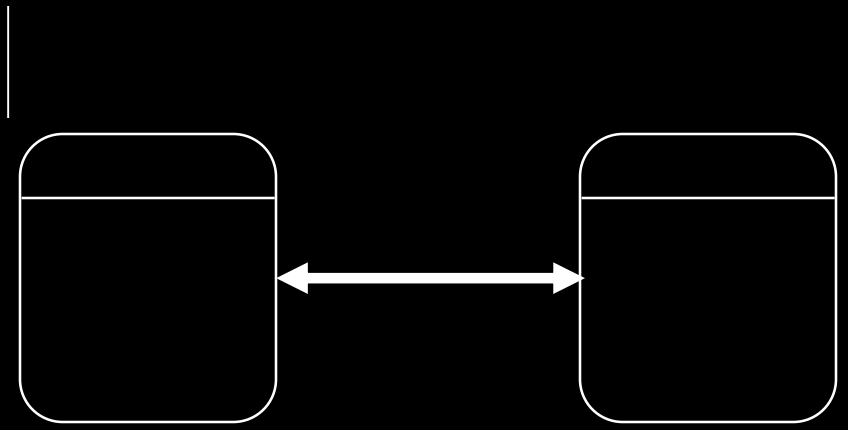
Black holes - A processing step may have input flows but no output flows.
Miracles - A processing step may have output flows but no input flows.
Grey holes - A processing step may have outputs that are greater than the sum of its inputs.



DFD: Common Mistakes



- Data store is hooked to external entity. This means external entity can read and write to your data file without auditing!!



- The data flow goes in two directions at once. Two or more arrows should be used to show the flow to and from each process.

DFD Levels: Context Diagram

- First we can start to draw a general overview. This general overview is called a "Context Diagram".
- A context diagram is a data flow diagram that only shows the top level, otherwise known as Level 0. At this level, there is only one visible process node that represents the functions of a complete system in regards to how it interacts with external entities. Some of the benefits of a Context Diagram are:
  - Shows the overview of the boundaries of a system
  - No technical knowledge is required to understand with the simple notation
  - Simple to draw,
- A Context Diagram shows three things:
  - all external entities
  - a single process labeled "0"that represents the entire system
  - the major information flows between the external entities and the system.

Data Flow Diagram Tips and Cautions

1. Process labels should be verb phrases; data stores are represented by nouns
2. A data store must be associated with at least a process
3. An external entity must be associated with at least a process
4. Don't let it get too complex; normally 5 - 7 average people can manage processes
5. Datastores should not be connected to an external entity, otherwise, it would mean that you're giving an external entity direct access to your data files
6. Data flows should not exist between 2 external entities without going through a process

Building a DFD

- It would be impossible to understand all of the data flows, and to identify all of the 'external entities' relating to our information system in one pass, so we tend to draw DFD's incrementally.
- We tend to start at the context level, break processes down to Level 0, and then start to consider where data enters and exits our information system, where it is stored, and how a process converts it from one form to another. We are interested here in the movement of *data* and its conversion.

DFD Example: Bus Garage Repairs

- Buses come to a garage for repairs.
- A mechanic and helper perform the repair, record the reason for the repair and record the total cost of all parts used on a Shop Repair Order.
- Information on labor, parts and repair outcome is used for billing by the Accounting Department, parts monitoring by the inventory management computer system and a performance review by the supervisor.

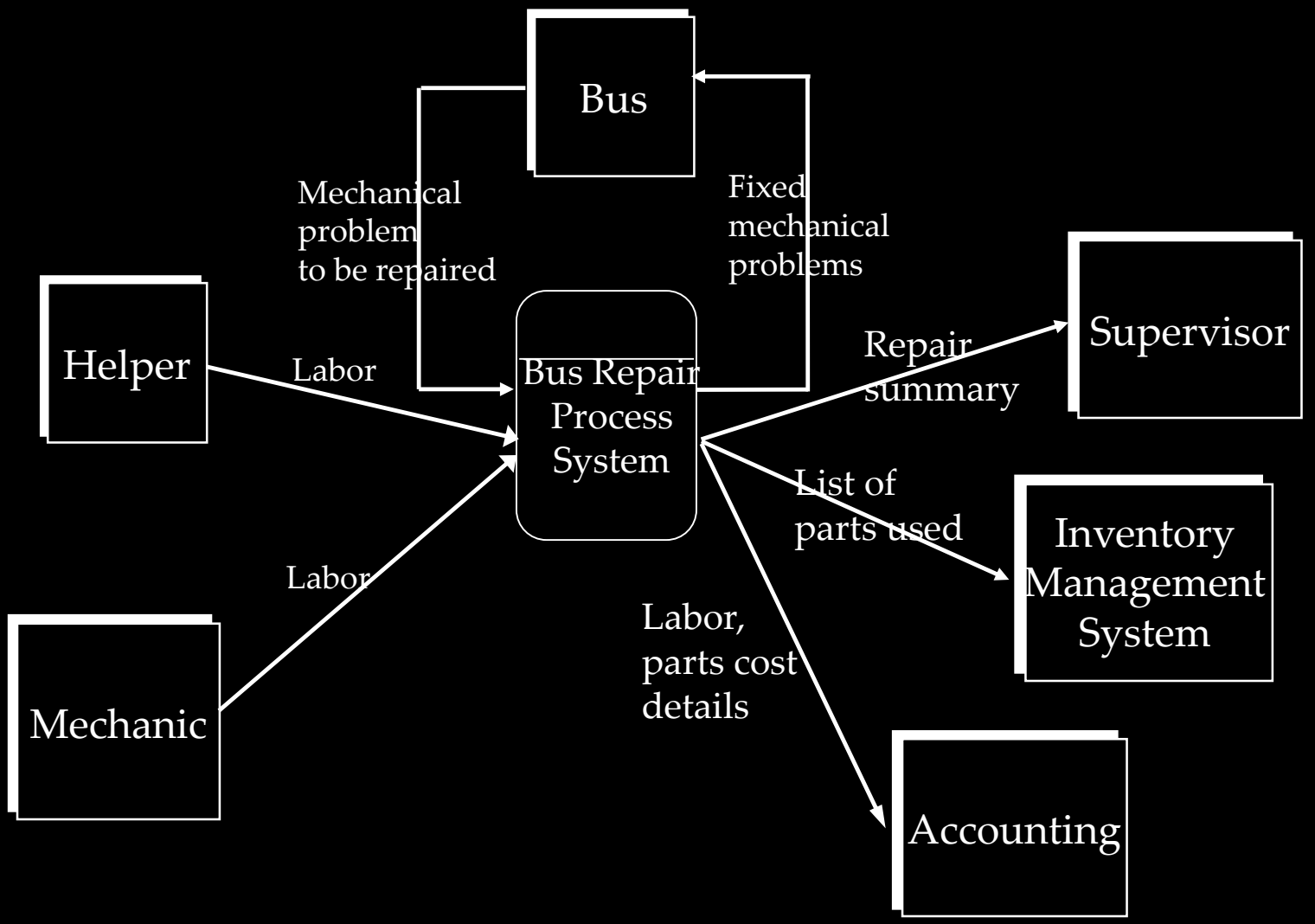
DFD Example: Bus Garage Repairs (cont'd)

- *External Entities*: Bus, Mechanic, Helper, Supervisor, Inventory Management System, Accounting Department, etc.
- *Key process* ("the system"): performing repairs and storing information related to repairs
- *Processes*:
  - Record Bus ID and reason for repair
  - Determine parts needed
  - Perform repair
  - Calculate parts extended and total cost
  - Record labor hours, cost

DFD Example: Bus Garage Repairs (cont'd)

- ***Data stores:***
  - **Personnel file**
  - **Repairs file**
  - **Parts list**
- ***Data flows:***
  - **Repair order**
  - **Bus record**
  - **Parts record**
  - **Employee timecard**
  - **Invoices**

Bus Garage Context Diagram



Building a DFD

- A DFD is NOT a flowchart. It simply shows how and where data itself progresses through our system.

Today's Outline

- Software process models
  - Water Fall Model
  - Evolutionary Development
  - Component base Software Engineering / Reuse-oriented development
- Process iteration
  - Incremental Model
  - Spiral Model
- Process activities

The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Plan-driven and Agile Processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Software Process Model

- ***Software Process*** : is coherent sets of activities for specifying, designing, implementing and testing software systems.
- ***A software process model*** is an abstract representation of a software process.
- *It is a description of the sequence of activities carried out in an SE project, and the relative order of these activities.* It presents a description of a process from some particular perspective.

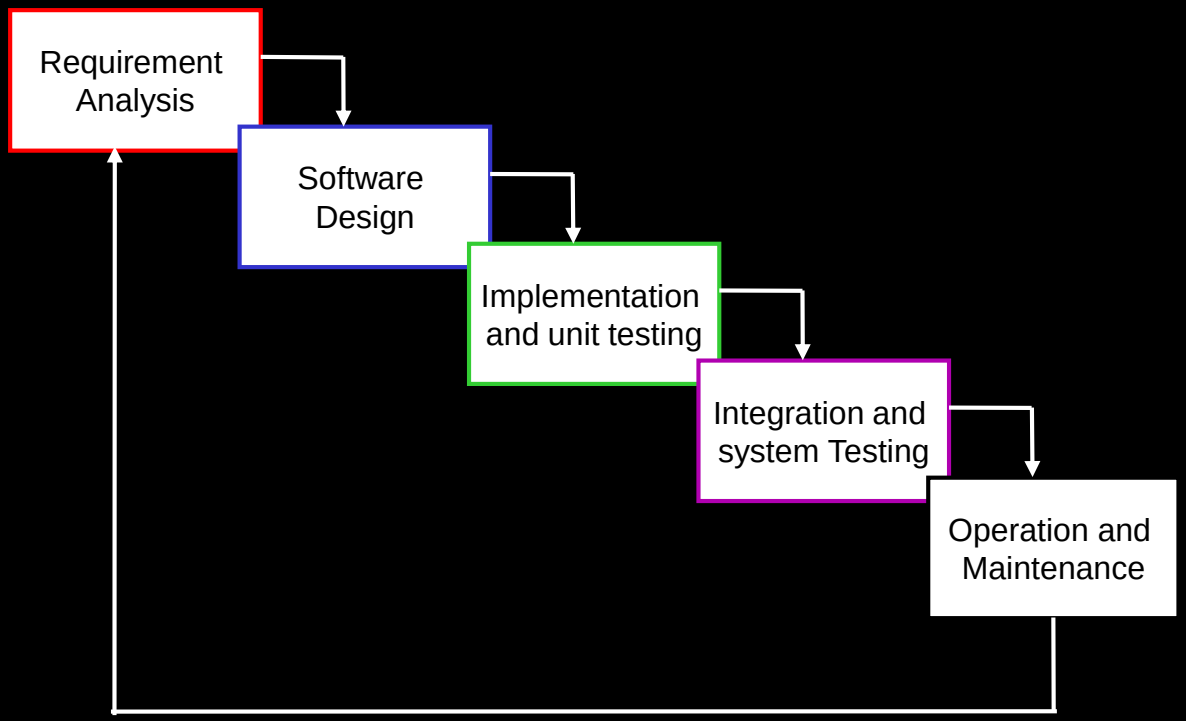
Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development. It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.
- Incremental development
  - Specification, development and validation are interleaved.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

Waterfall

Each box represents a **set of tasks** that results in the production of each or more work products.

Each new phase begins when the **work products** of the previous phase as completed, frozen and signed off.



Waterfall (when to use)

- Well suited for projects where requirements can be understood easily and technology decisions are easy
- Has been used widely
- For standard/familiar type of projects it still may be the most optimum.

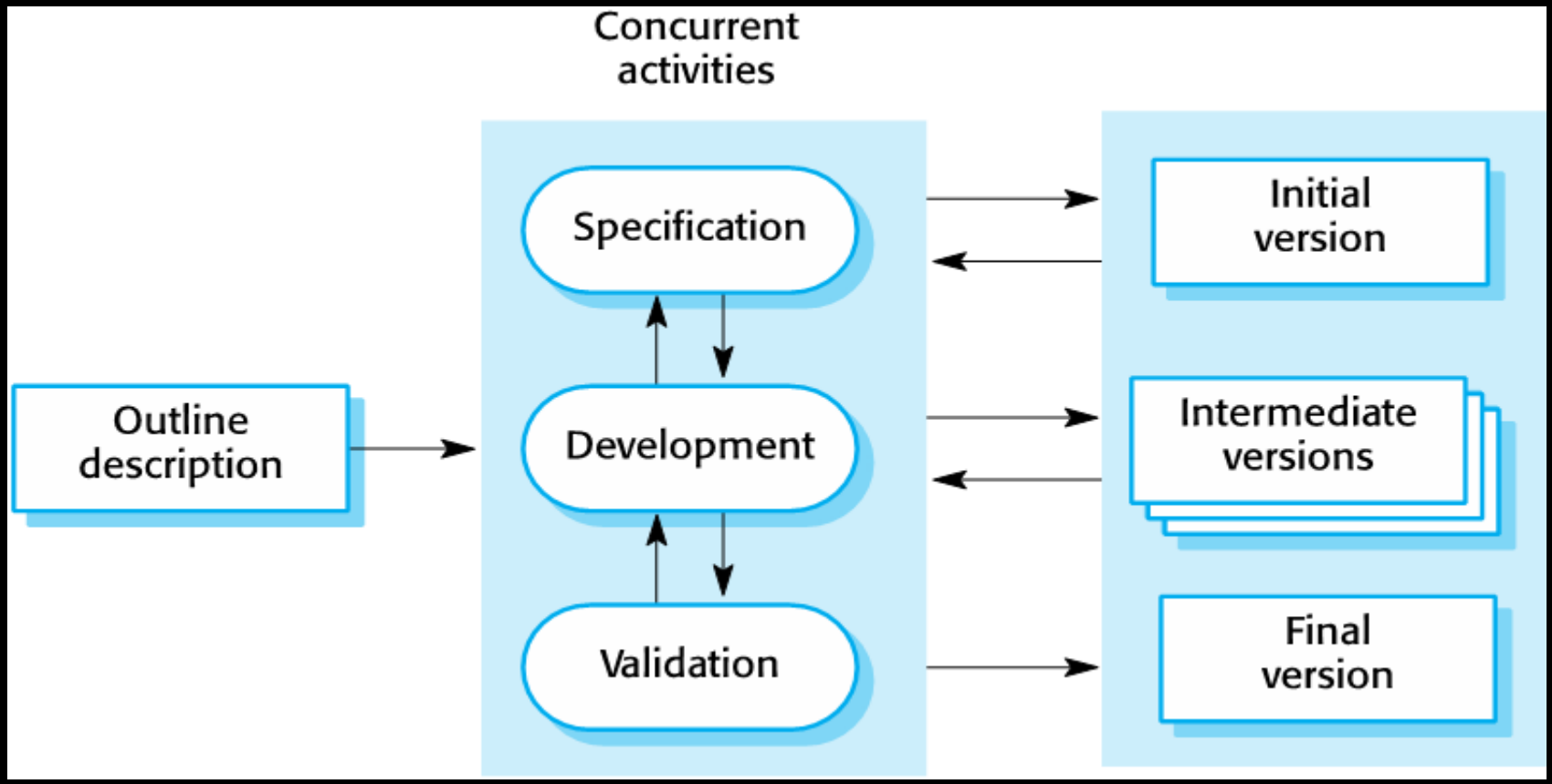
Waterfall model problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Disadvantages of Waterfall Model

- Unable to work where high level of uncertainty is involved
- Requirements need to be stable hence making model rigid
- Unrealistic to state all requirements at the beginning
- Does not handle concurrent events – development teams are delayed waiting for others
- Difficult and expensive to change decisions
- The user is involved only in the beginning phase of requirement gathering and than during acceptance phase

Iterative Development



Iterative Development Benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Iterative development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

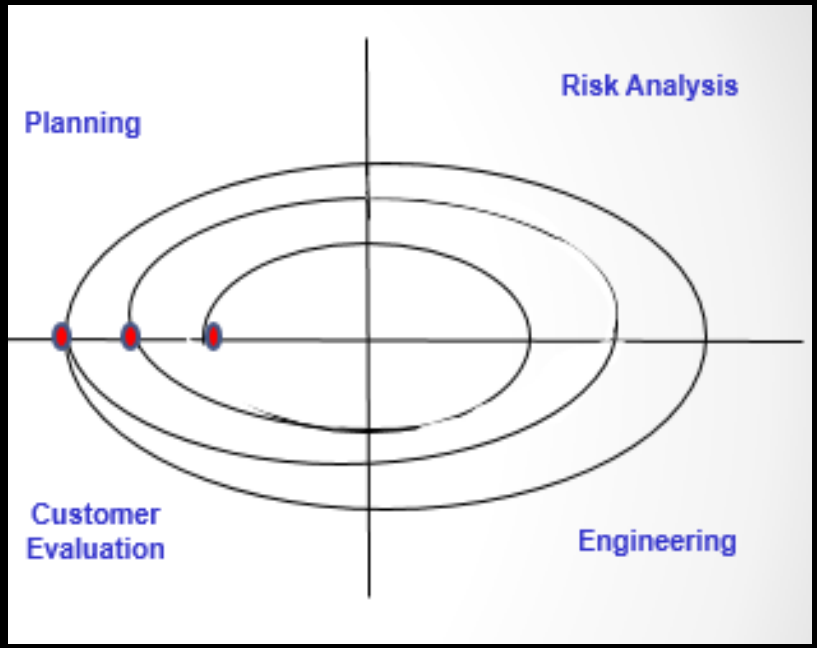
Evolutionary Models: Spiral

- Spiral is primary **Risk Driven** approach. Spiral model consist of different **cycle. In each cycle we try to address some risk elements.**

- Planning: Determines objectives, alternatives and constraints.
- Risk Analysis: Analysis of alternatives as well as an identification and/or resolution of risks.
- Engineering: Development of the next level of product
- Customer evaluation: Assessment of the results of engineering

Spiral

- With each iteration around the spiral progressively more complete versions of the software are built.
- Spiral model enables the developer, and the customer, to understand and ***react to risk*** at each evolutionary level.
- Each loop around the spiral implies that project costs and schedules may be modified.



- This creates problems in fixed-price project.

That is all