

Administrative Details

- HW #7, Project Assignment #1 due
- HW #7 solution set, Project Assignment 3 solution set handout
- Project Assignment #2, '98 Midterm #2 handout
- Midterm #2 Thursday 10/4 during regularly scheduled lab
- Web: FrontPage2000 books, project team server space
- Wednesday review session: read the Shelly, Cashman and Rosenblatt material on data flow diagrams!
- I will be out of town Friday 11/5 - Wednesday 11/10

Systems Analysis and Design

Course so far has emphasized:

- *Logical design* of databases (E-R Diagrams and Relational Database Models)
- *Implementation* of databases (Access tables, forms, queries, reports and macros)

However, information systems development must address broader *organizational* issues as well:

- What is the *justification* for the type of application to be developed?
- What actual or desired *processes* should the application perform?
- How will we *verify* that the application performs as designed?

The Systems Development Lifecycle (SDLC) methodology allows a project team to successfully build an application uniquely suited to the organization's needs

SDLC as Part of the IT Project Lifecycle

While every IT project requires *Initiation, Feasibility Analysis* and *Project Planning*, and concludes with a *Project Termination*, there are certain steps that are specific to *systems development*, which is the focus of this course. Thus, the SDLC is contained within the generic IT project lifecycle.

(Figure 8.1, p. 115)

Systems Development Lifecycle Steps

The SDLC is generally presented as an iterative sequence of five steps:

- Systems Planning
- Systems Analysis
- Systems Design
- Systems Implementation
- Systems Operation and Support,

(Figure 1.15, Shelly, Cashman and Rosenblatt)

each culminating in a deliverable, either a written document or a piece of software (or both)

SDLC allows organizations to incorporate new requirements, technology and human resources to IT development

SDLC Details

- *Systems Planning*
 - Project definition
 - Feasibility study/alternatives analysis
 - Project scope, deliverables
 - Standards, techniques, methods
 - Task assessment, skill assessment, preliminary time estimation
 - Yields *Preliminary Investigation Report*
- *Systems Analysis*
 - Analysis of existing hardware/software
 - User requirements analysis
 - Logical systems design:
 - Conceptual data model (Entity-Relationship Diagram)
 - Conceptual process model (Data Flow Diagram)
 - Functional application description
 - Yields *Systems Requirements Document*

SDLC Details (cont'd)

- *Systems Design*
 - Relational database model and data dictionary (sometimes included in Systems Analysis phase)
 - Detailed description of application inputs and outputs
 - Detailed conceptual design of forms, reports, application programs and other application components
 - Yields *System Design Specification*
- *Systems Implementation*
 - Application development and/or installation
 - Testing and Evaluation
 - Yields *Functional Information System*
- *Systems Operation and Support*
 - Maintenance
 - Revisions
 - Yields *Operating Information System*

How Is the SDLC Used In This Class?

Clearly, the SDLC requires significant time, human resources and technical resources to perform well.

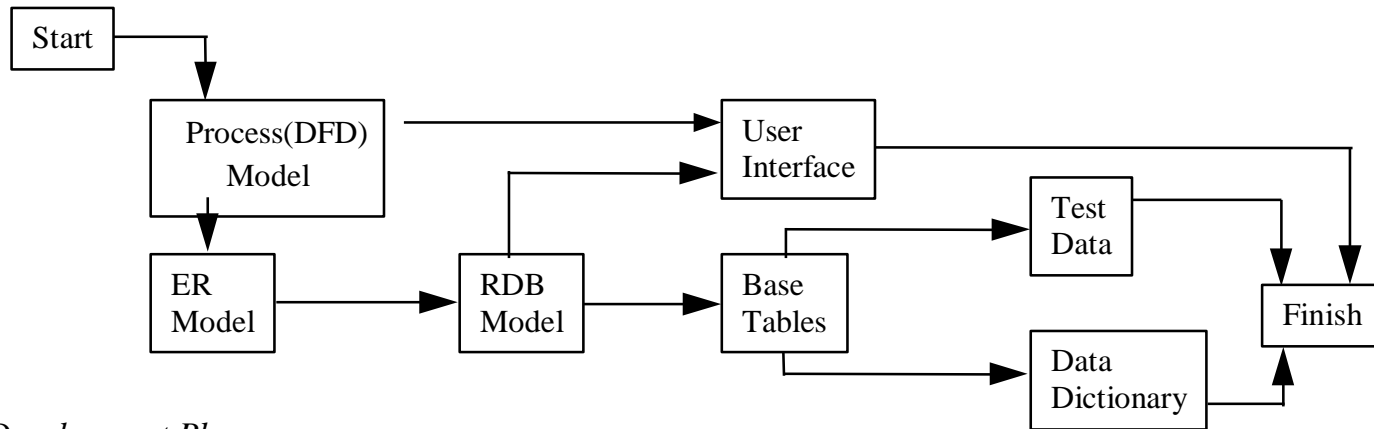
For the class project, your team must implement the SDLC steps as follows:

- *Systems Planning* (project requirements summary, project team description, preliminary work schedule, service area demographic analysis)
- *Systems Analysis* (E-R diagram, Data Flow Diagrams, high-level functional description)
- *Systems Design* (RDBM, Data Dictionary, identification/description of database objects, Web site map)
- *Systems Implementation* (Access database, Web site content, elementary test plan)

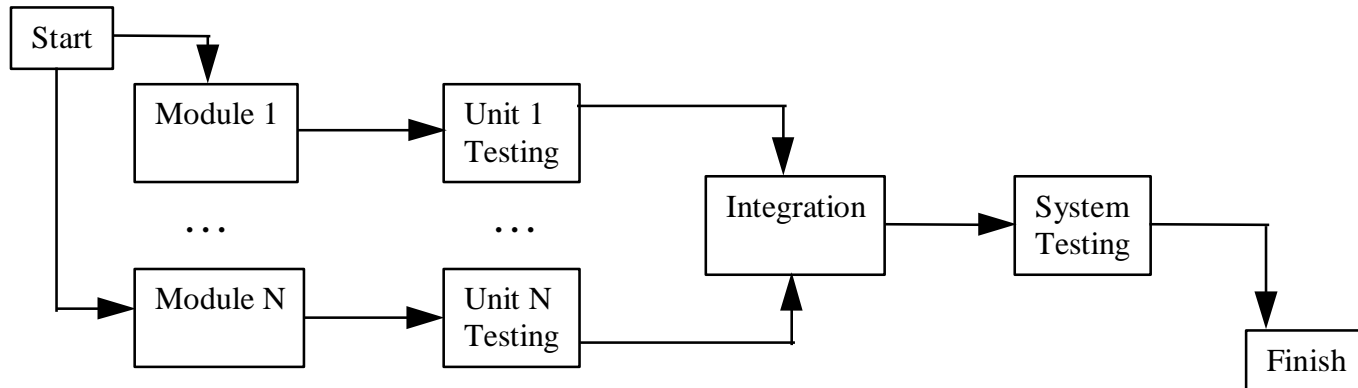
In addition, a White Paper will address practical systems implementation and operations/support issues.

PERT Chart for Abbreviated Systems Analysis and Design Life Cycle

Systems Analysis Phase



Development Phase



Business Analysis with Data Flow Diagrams

We initially claimed that entity-relationship diagrams could encode many business rules.

But many elements of business *processes* cannot be represented by E-R diagrams:

- *Actors* (individuals, organizations) which generate input data and/or receive output data;
- *Data transformations* using specific business rules (which we said required application-level code);
- *Data storage* or *display* that may or may not correspond to entities (e.g. views, reports, temporary tables, screens);
- *Data flows* between actors, processes or data stores over time, and
- Manual versus automated *processing*.

Data Flow Diagrams (DFDs) are the link between initial business analysis, E-R diagrams and relational database models

Practical Significance of DFDs

- Many older information systems (*legacy systems*):
 - Are mainframe-based (inaccessible to most non-IT users);
 - Use non-relational (hierarchical) databases, and
 - Require second-generation programming languages (e.g. COBOL) to perform data input, queries, reporting.

Thus, many IT professionals focus on *data flows* and *physical implementation*, not data relationships or RDBM design principles.

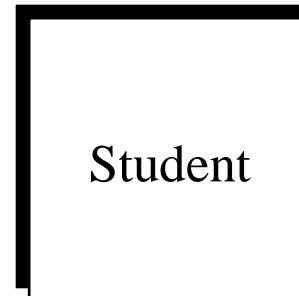
- Many business processes are so complex that E-R design alone will not give analysts insight into their businesses

DFDs allow the analyst to determine: *What* does the organization do? *How* does the organization do it?

Creating DFDs

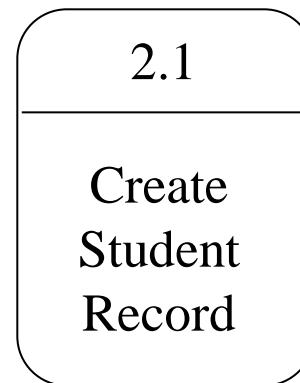
Define *Entities*

- External entities represent persons, processes or machines which produce data to be used by the system or receive data that is output by the system
- Examples: Student, Customer, Client



Define *Processes*

- Processes are discrete actions that transform input data to output data
- Examples: Create Student Record, Calculate Purchase Cost, Register Client



Creating DFDs (cont'd)

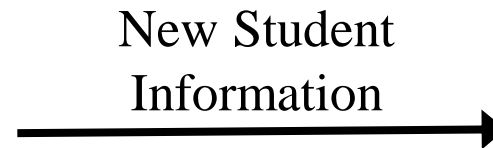
Define *Data Stores*

- Data stores are temporary or permanent repositories of information that are inputs to or outputs of processes
- Examples: Student Master, Client List



Define *Data Flows*

- Data flows represent the transfer of data over time from one “place” (entity, process, data store) to another
- Examples: New Student Information (from Student, to Student Master)



Creating DFDs (cont'd)

Define the *System*

- A system is the collection of all business processes which perform tasks or produce outputs we care about. It is “what happens.”
- The system is a single process, connected to external entities
- Represented in a “Context Diagram”

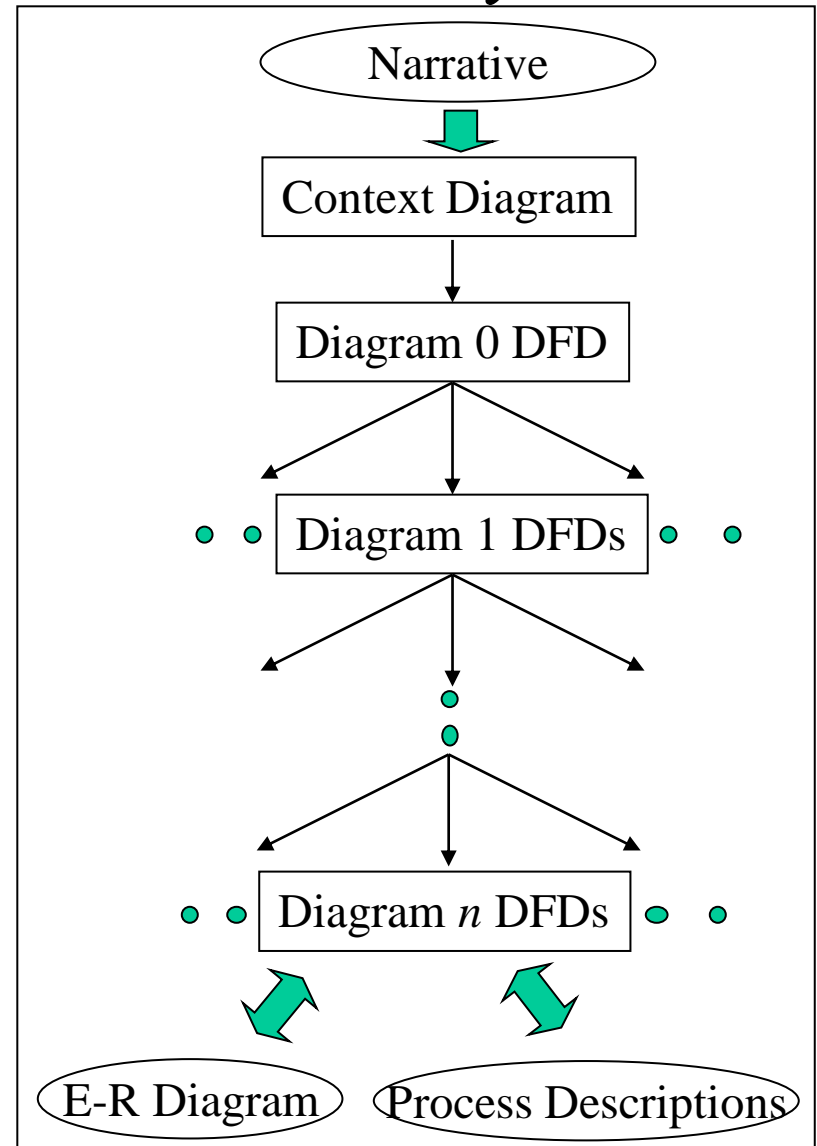
(Figure 4.13,
Shelly,
Cashman and
Rosenblatt)

Define *Subsystems*

- A subsystem gives a more detailed view individual processes contained in the context diagram
- Includes data stores, more elementary processes

DFDs Created by Top-Down Analysis

- Create a *narrative*: description of system
- Create a *Context Diagram* that contains a single process (“the system”) and all entities which share data with the system
- *Explode* the “parent” context diagram to produce a Diagram 0 (“child”) DFD
- Create Diagram 1, 2, ..., n DFDs that represent “explosions” of Diagram 0, 1, ..., $n-1$ DFDs until a diagram has only “primitive” processes
- Create *process descriptions* to be implemented by application programs: queries, macros, reports, programming languages

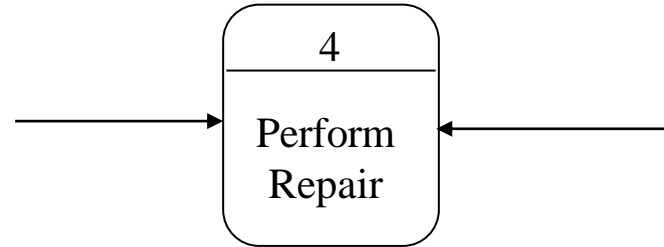


Where to Begin Creating DFDs

- Start with the data flow *from* an external entity and work forwards
- Start with the data flow *to* an external entity and work backwards
- Examine the data flows into or out of a *data store*
- Examine data flows, entity connections and data stores associated with a particular *process*
- Note *fuzzy, ill-defined* areas of the system for further clarification

What to Avoid in DFDs

Processes with no outputs or no inputs



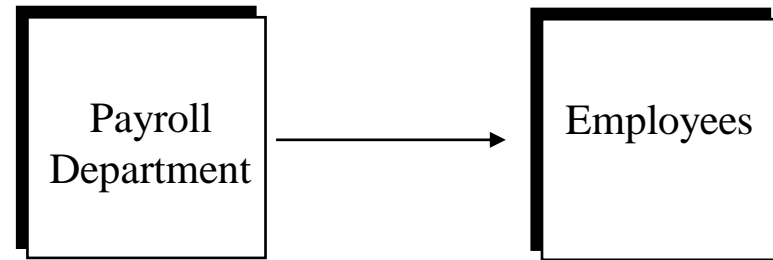
Processes whose inputs are obviously inadequate to yield outputs

Connecting data stores directly to each other



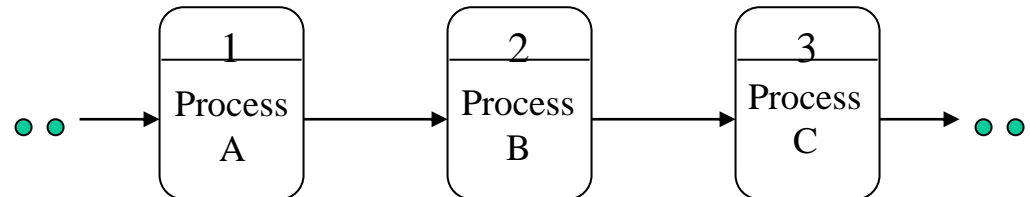
Having data flows terminate at data stores

Connecting entities to anything other than processes



Making the data flow diagram too cluttered (e.g. ≥ 9 processes)

Many processes with a single input and output (*linear flow*)



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.

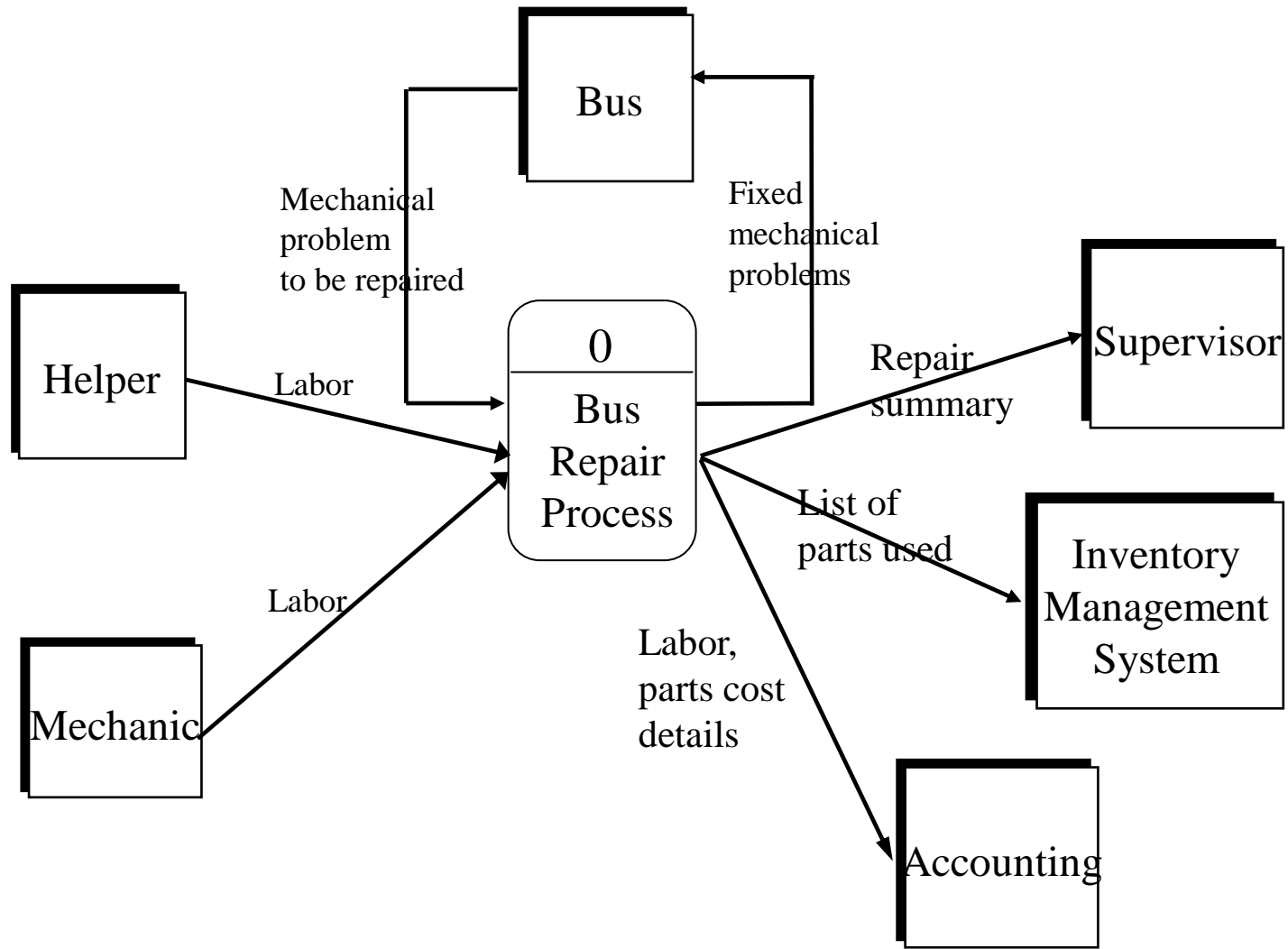
- *Key process* (“the system”): performing repairs and storing information related to repairs
- *External Entities*: Bus, Mechanic, Helper, Supervisor, Inventory Management System, Accounting Department, etc.
- *Processes*:
 - Record Bus ID and reason for repair
 - Determine parts needed
 - Perform repair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

DFD Example: Bus Garage Repairs

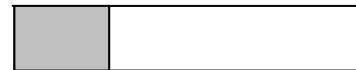
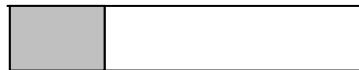
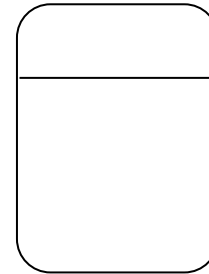
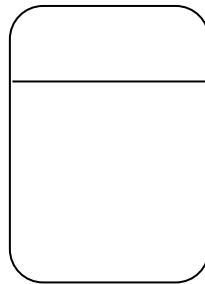
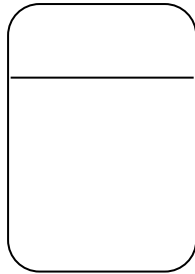
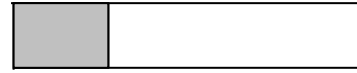
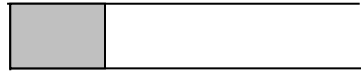
(cont'd)

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

Bus Garage Context Diagram



Bus Garage Diagram 0 DFD



Bus Garage Diagram 0 DFD (cont'd)

