

Waterfall



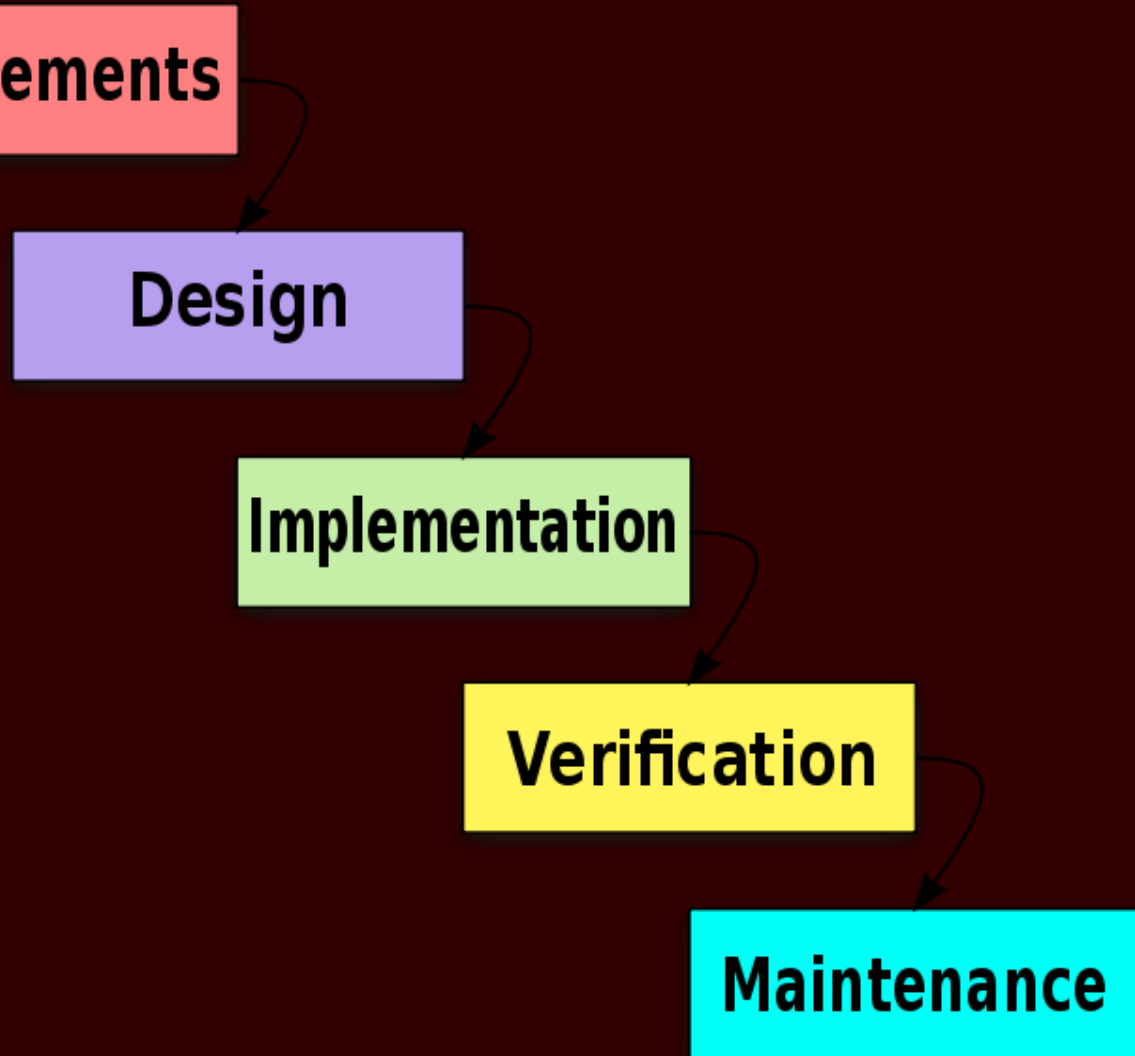
Requirements

Design

Implementation

Verification

Maintenance



Waterfall

- A sequence of stages in which the output of each stage becomes the input for the next.
- In the waterfall model, it is possible to rework earlier stages in the light of experience gained at a later stage. Each stage is signed off and the next stage is proceeded with. However the end user is rarely involved in the development stage, even though they may well be involved in signing off.
- It is therefore critical that the analysts and the programmers understand the end-users' requirements.
- This can be quite difficult with the waterfall model.

Waterfall Benefits

- Misunderstandings are detected at early stages
- Identifies systems requirements long before programming begins
- The user will notice any missing functions, incomplete or inconsistent requirements.
- Minimizes changes to requirements as project progresses.
- Can be built quickly to demonstrate systems
- It can be used for training before the system is finished

Waterfall Shortcoming

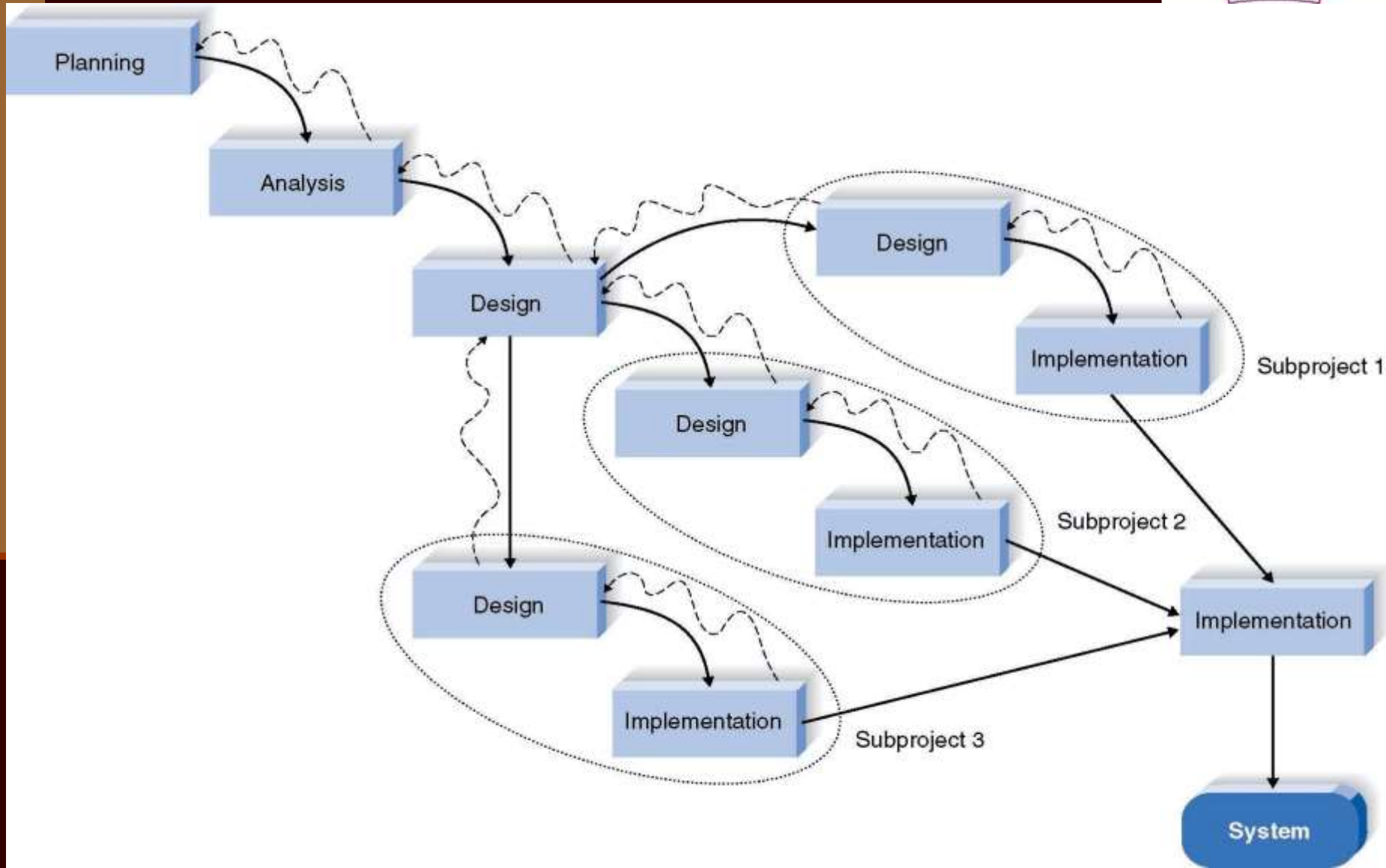
- Design must be specified on paper before programming begins
- Long time between system proposal and delivery of new system
- The waterfall model has disadvantages, which can be overcome using Prototyping, in which a model of the system is developed in partnership with the end-user.
- The features are worked out with the end user using a prototype, and the end user can have a considerable input into the development of a project.

Rapid Application Development (RAD)

Utilizes prototyping to delay producing system design until after user requirements are clear

- Phased development
 - A series of versions developed sequentially
- Prototyping
 - System prototyping
- Throw-away prototyping
 - Design prototyping

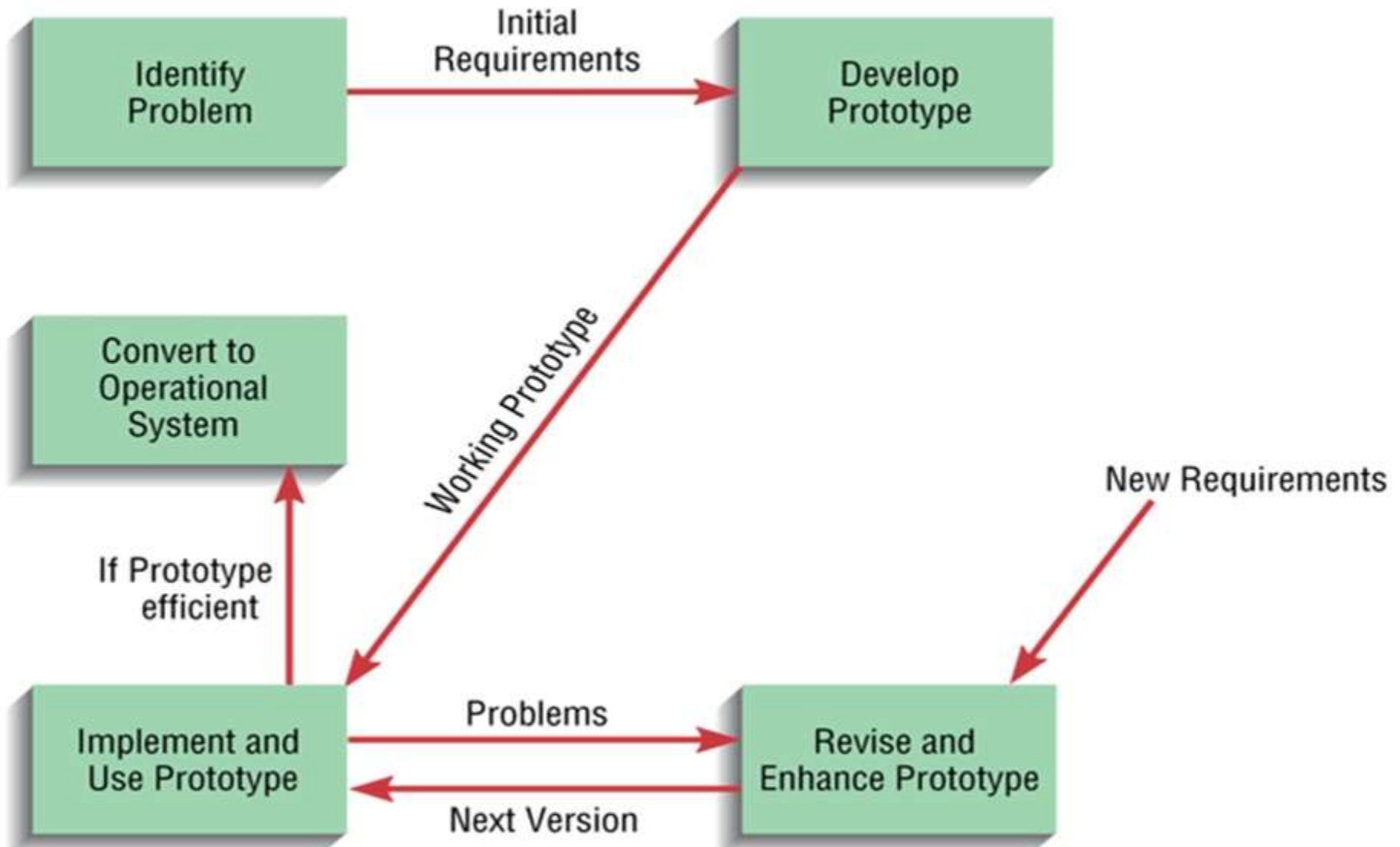
Phased Development



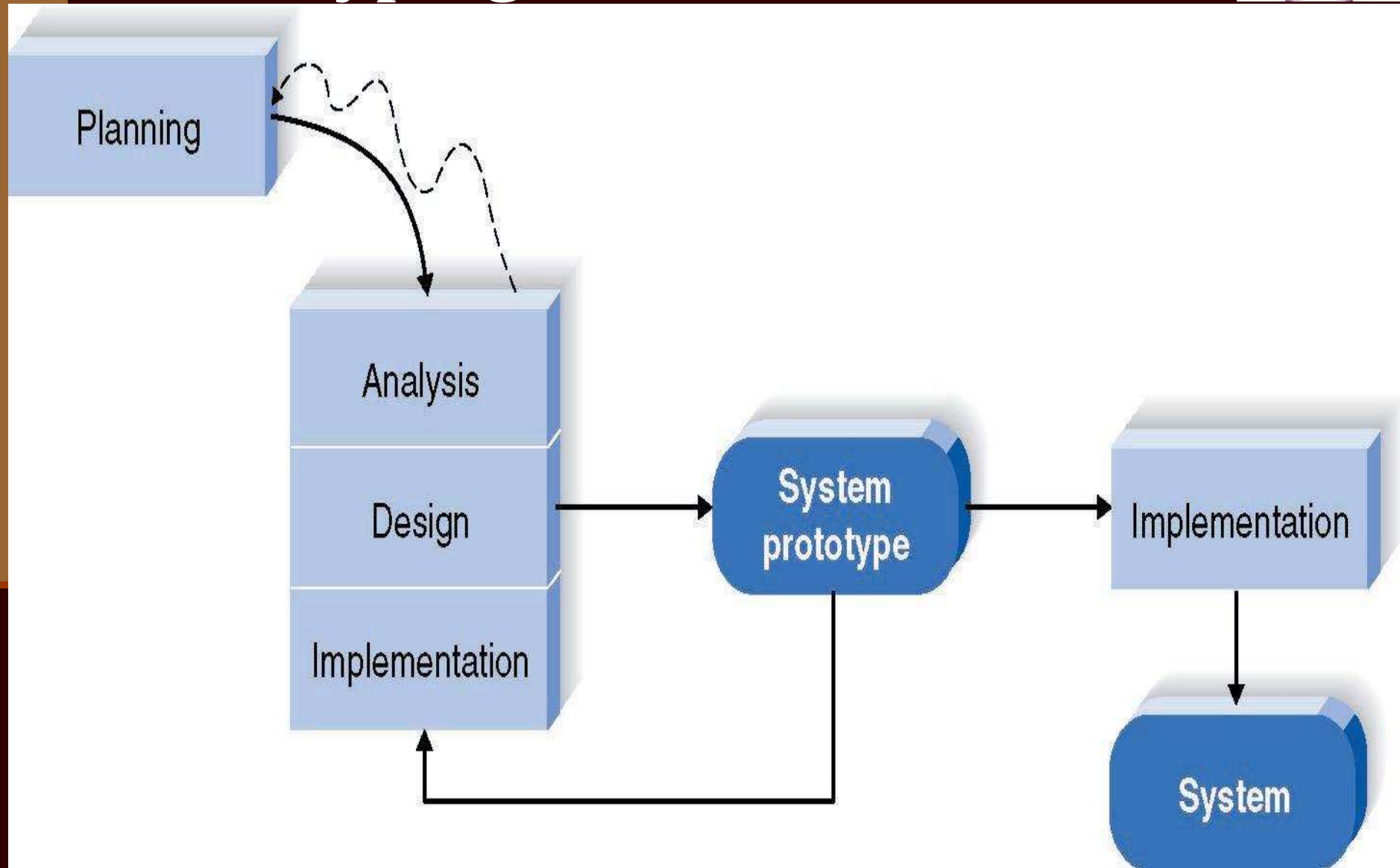
Prototyping

- A small-scale, incomplete, but working sample of a desired system.
- Working model of proposed system
- Building a scaled-down working version of the system
- Advantages:
 - Users are involved in design
 - Captures requirements in concrete form

Prototyping



Prototyping



Prototyping

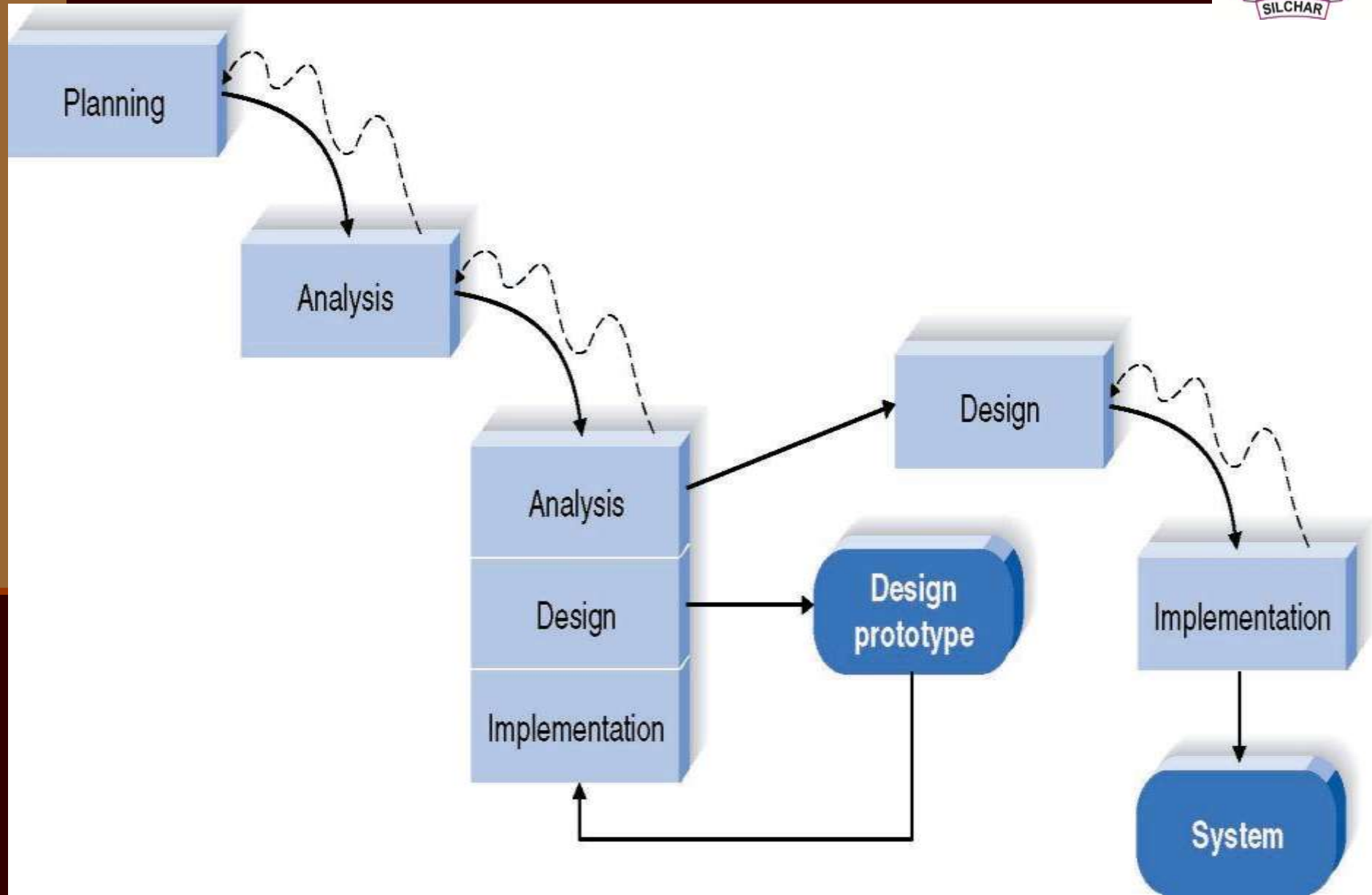
Benefits

- Users interact with prototype very quickly
- Users can identify needed changes and refine real requirements

Shortcoming

- Tendency to do superficial analysis
- Initial design decisions may be poor

Throwaway Prototyping



Throwaway Prototyping

Benefits

- Risks are minimized
- Important issues are understood before the real system is built

Shortcoming

- May take longer than prototyping

Joint Application Design (JAD)



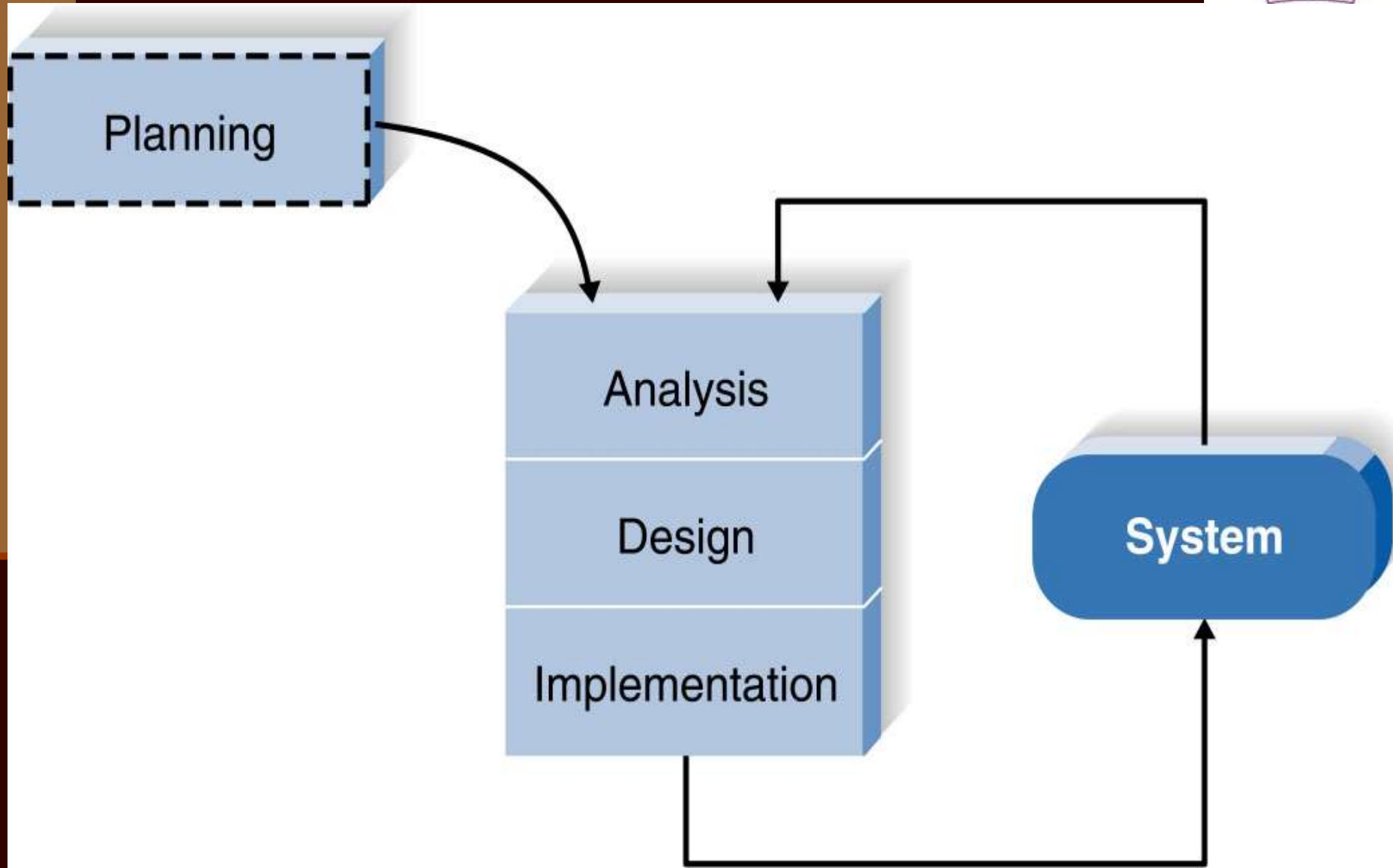
- Users, Managers and Analysts work together for several days
- System requirements are reviewed
- Structured meetings

Agile method



- The integration of various approaches of systems analysis and design for applications as deemed appropriate to the problem being solved and the system being developed.

Agile



Agile

Benefits

- Fast delivery of results
- Works well in projects with undefined or changing requirements

Shortcoming

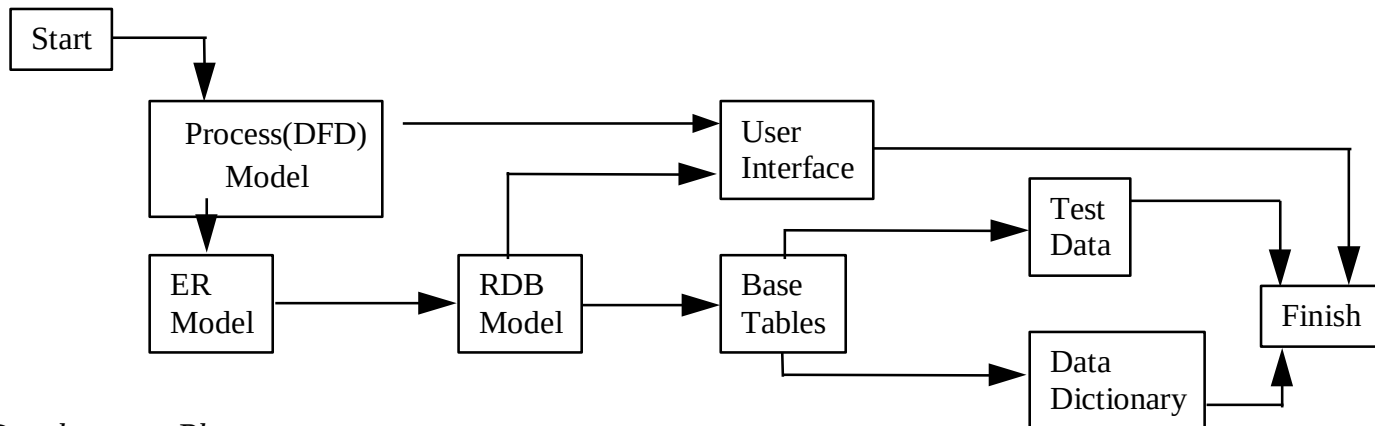
- Requires discipline
- Works best in small projects
- Requires much user input

Selecting the Appropriate Methodology

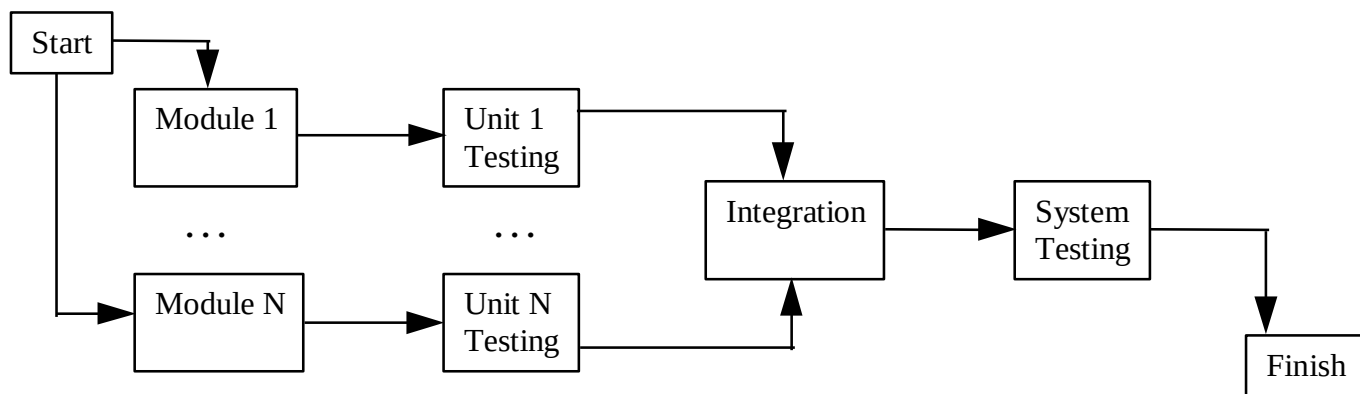
- Clear user requirements
- Familiarity with technology
- Complexity of system
- Reliability of system
- Time schedule
- Schedule visibility

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.

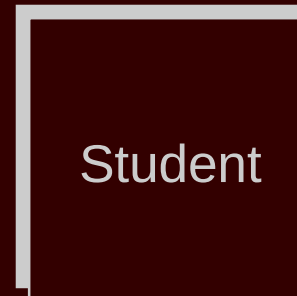
- Many business processes are so complex that E-R design alone will not give analysts insight into their businesses

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

Creating DFDs

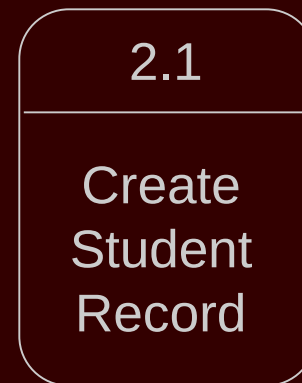
Define *Entities*

- 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)

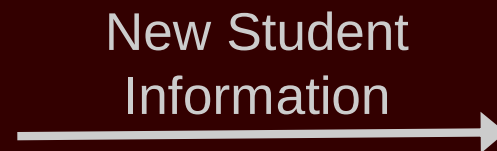
Define Data Stores

- 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)



(Templates are posted in l:\academic\90728\DFDSymbols.ppt)

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”

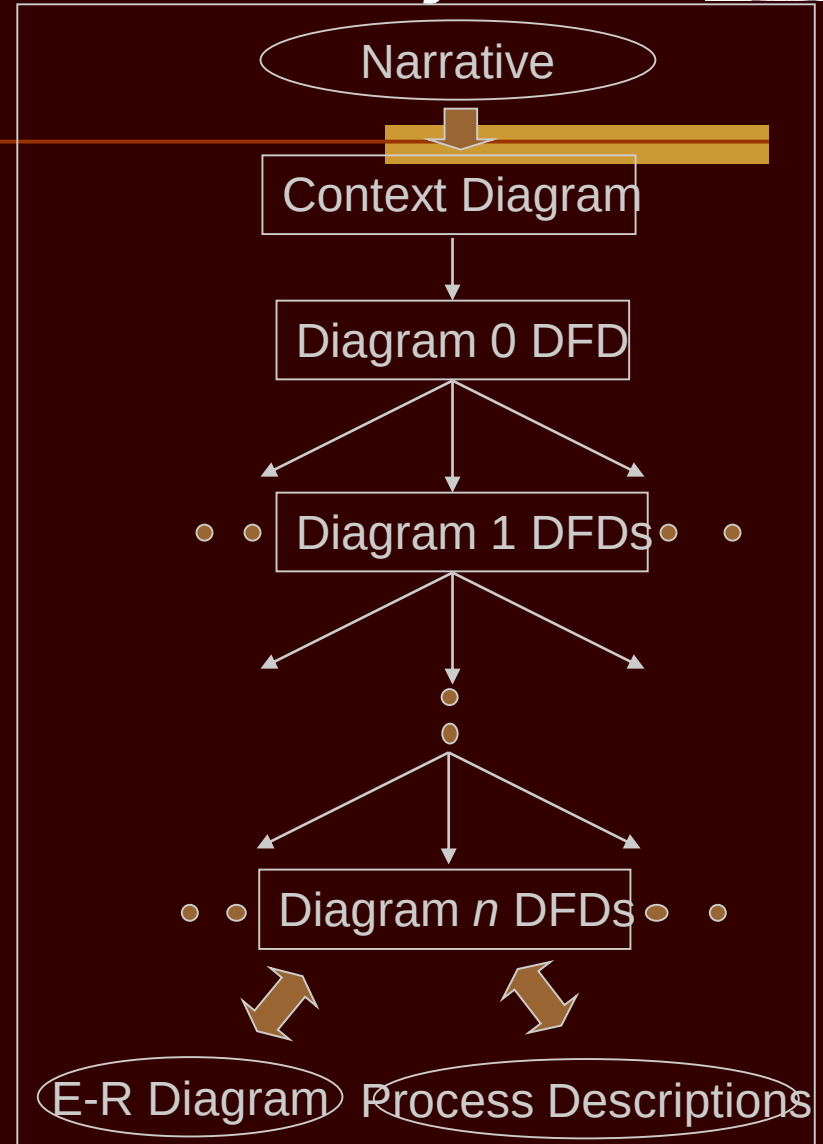
(Figure 4.13,
Shelly, Cashman
and Rosenblatt)

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

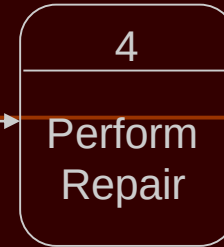


Where to Begin Creating DFDs

- 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

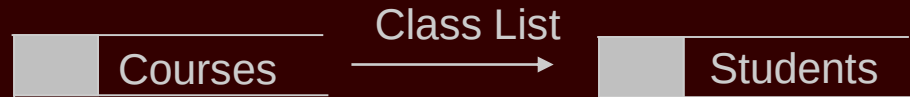
What to Avoid in DFDs

Processes with no outputs or no inputs



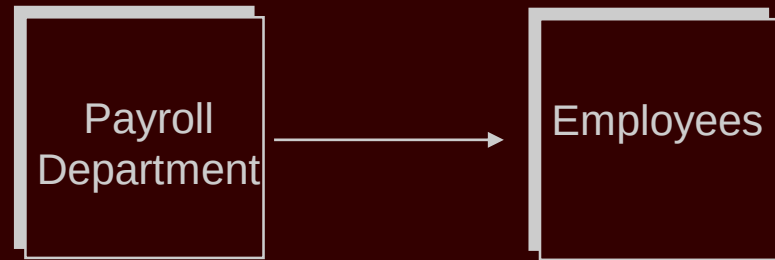
Processes whose inputs are obviously inadequate to yield outputs

Connecting data stores directly to each other



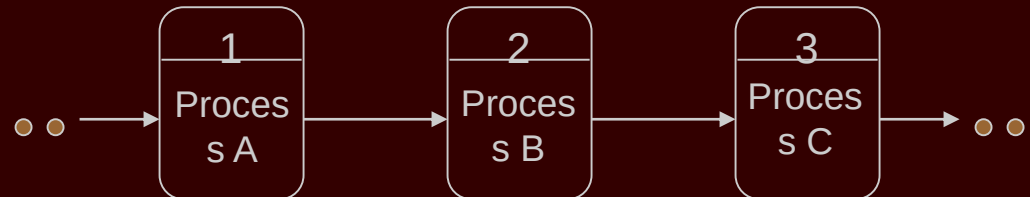
Having data flows terminate at data stores

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

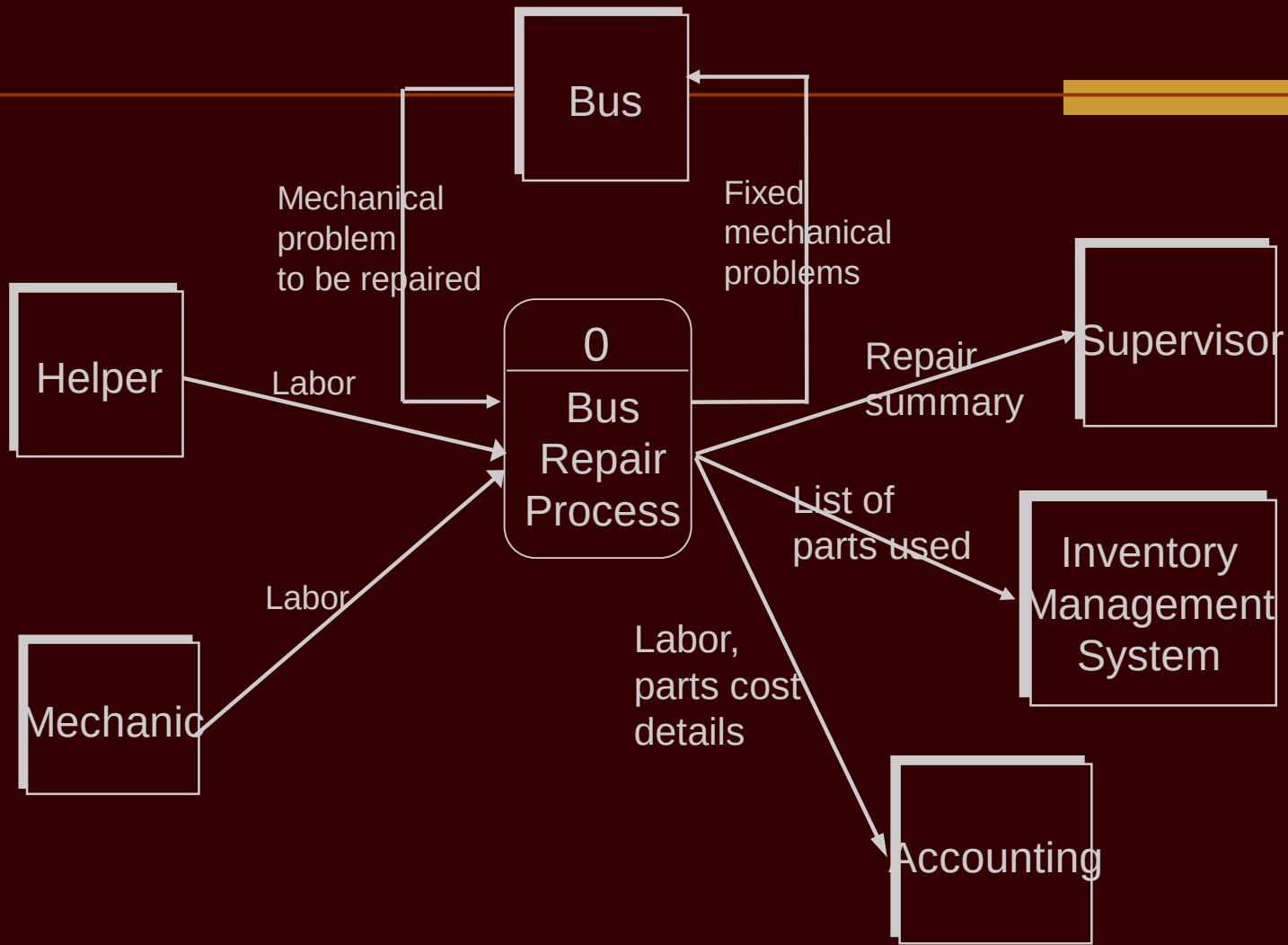
Buses come to a garage for repairs.

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.

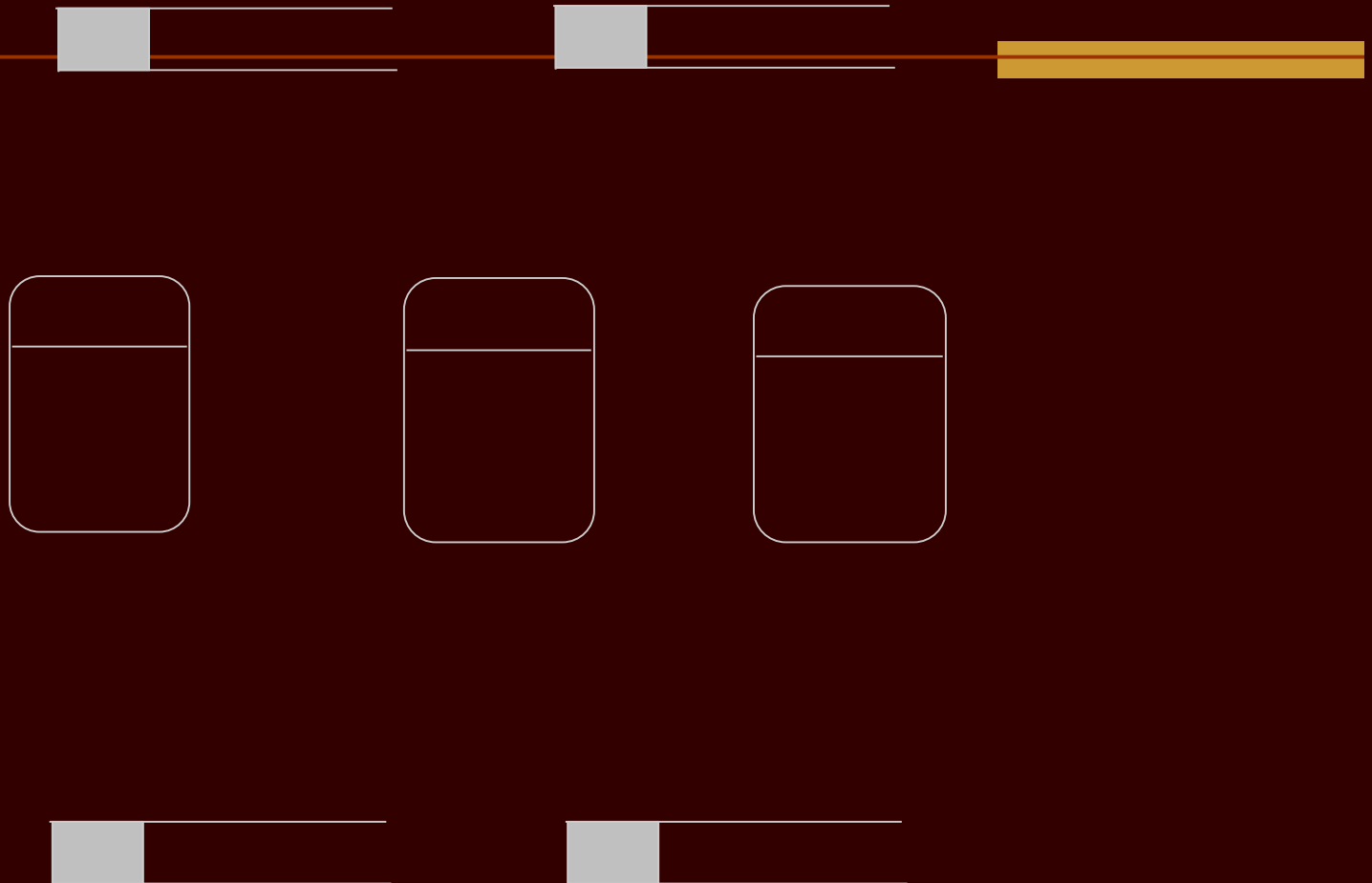
Information on labor, parts and repair outcome is used for billing by the Accounting Department, parts monitoring by the inventory management computer system and a performance review by the supervisor.

- *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

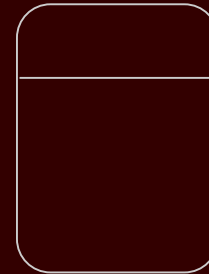
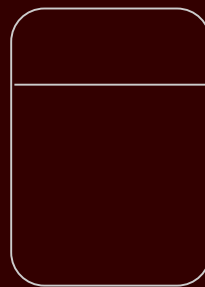
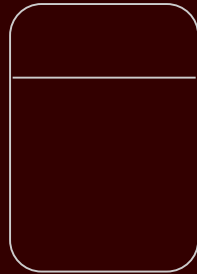
Bus Garage Context Diagram

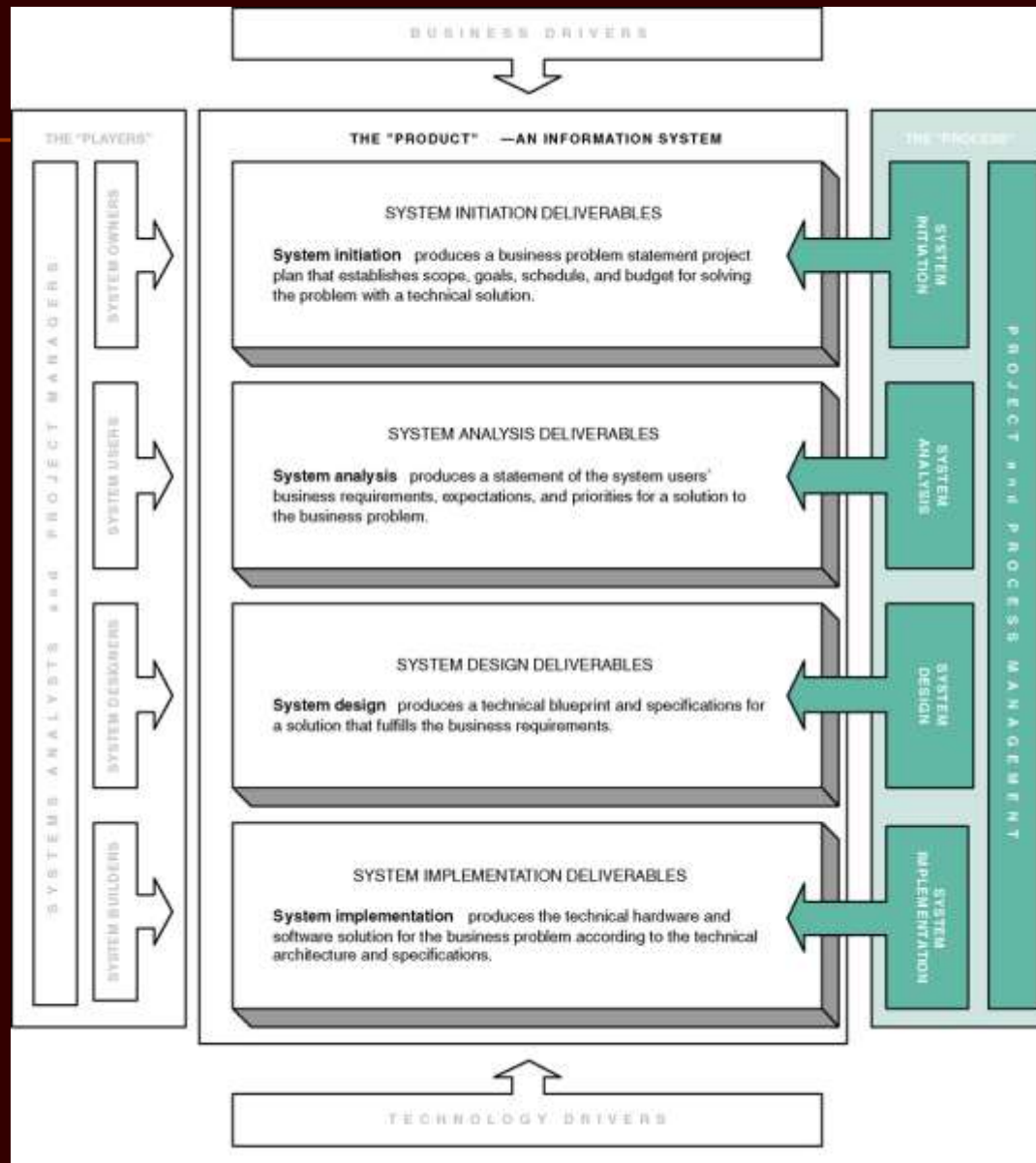


Bus Garage Diagram 0 DFD

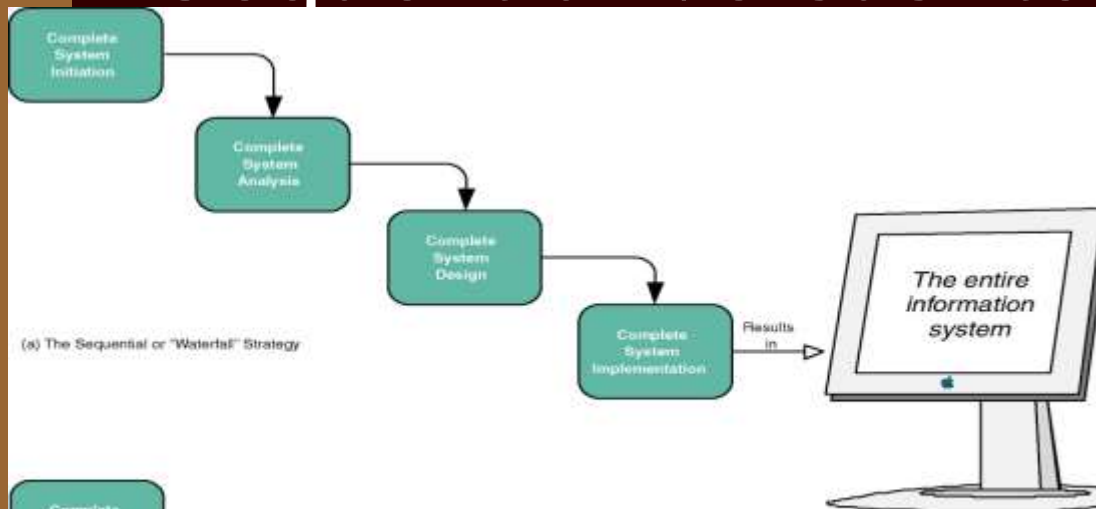


Bus Garage Diagram 0 DFD (cont'd)

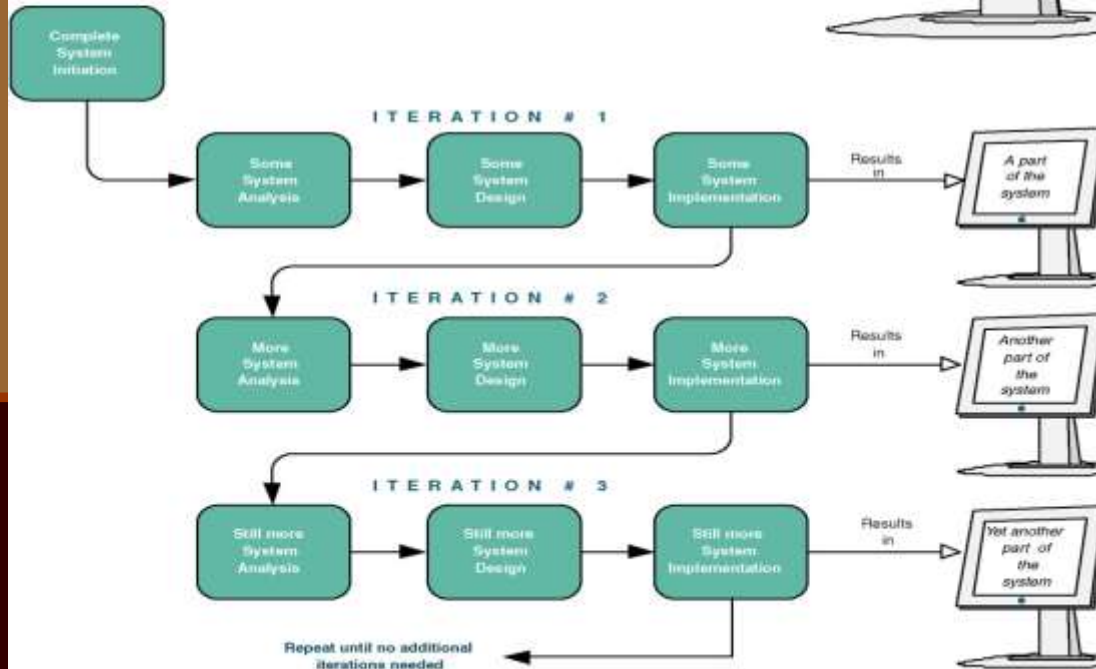




Sequential versus Iterative



Not good strategy for most modern systems development



Iterative (incremental) approach is more popular strategy