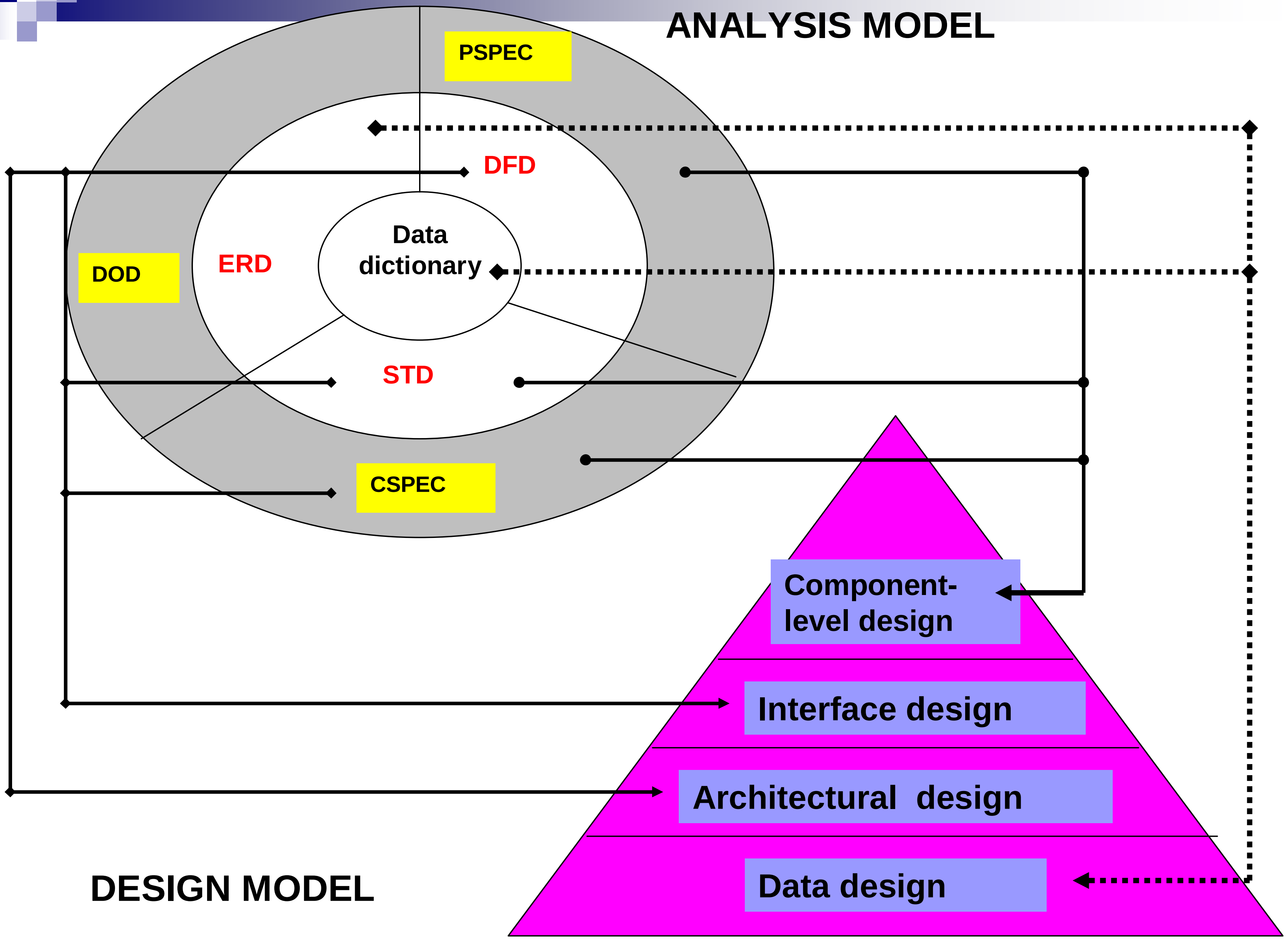
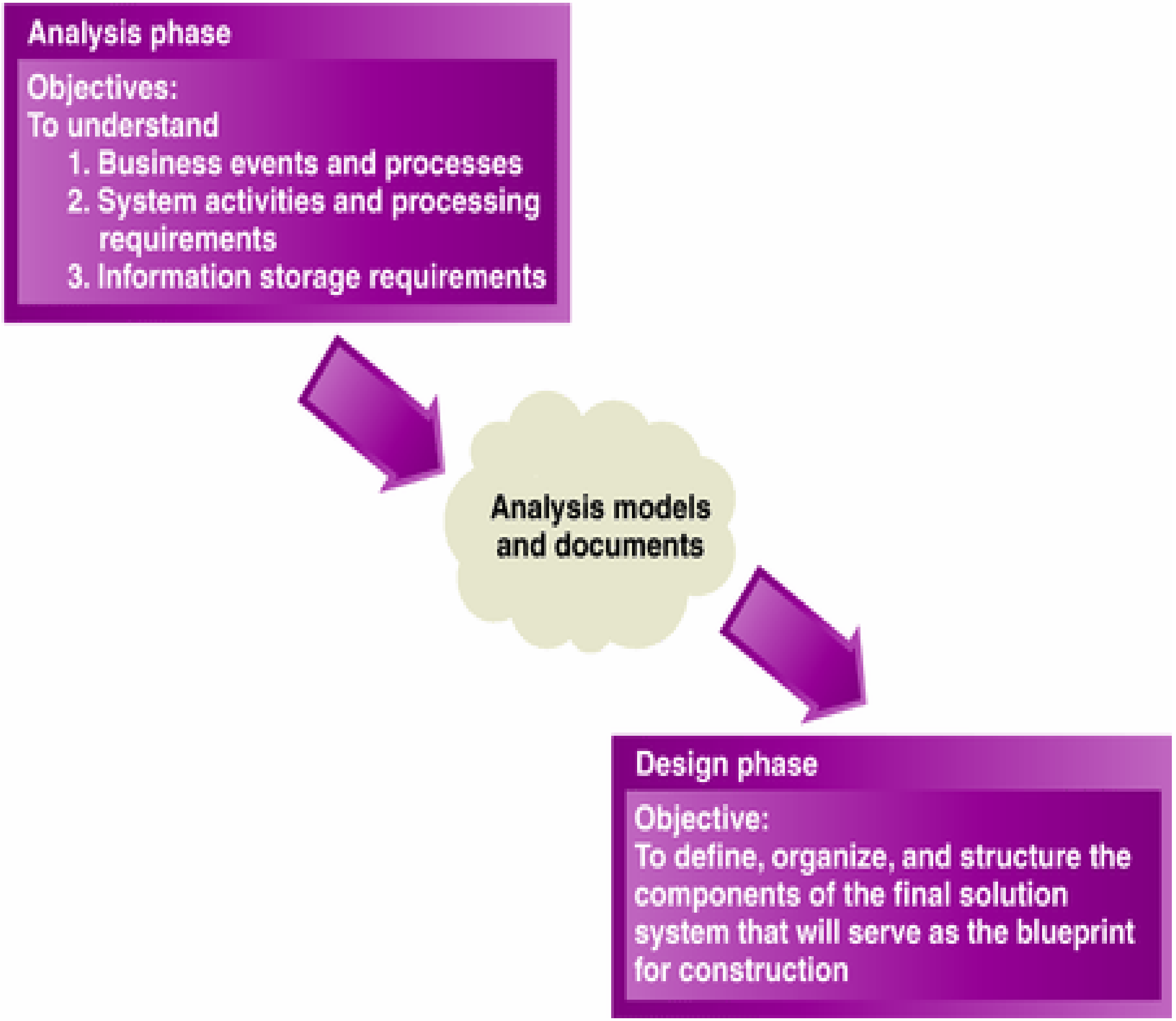
Software Engineering
Design Concepts & Principles

Transition From Analysis To Design

ANALYSIS MODEL



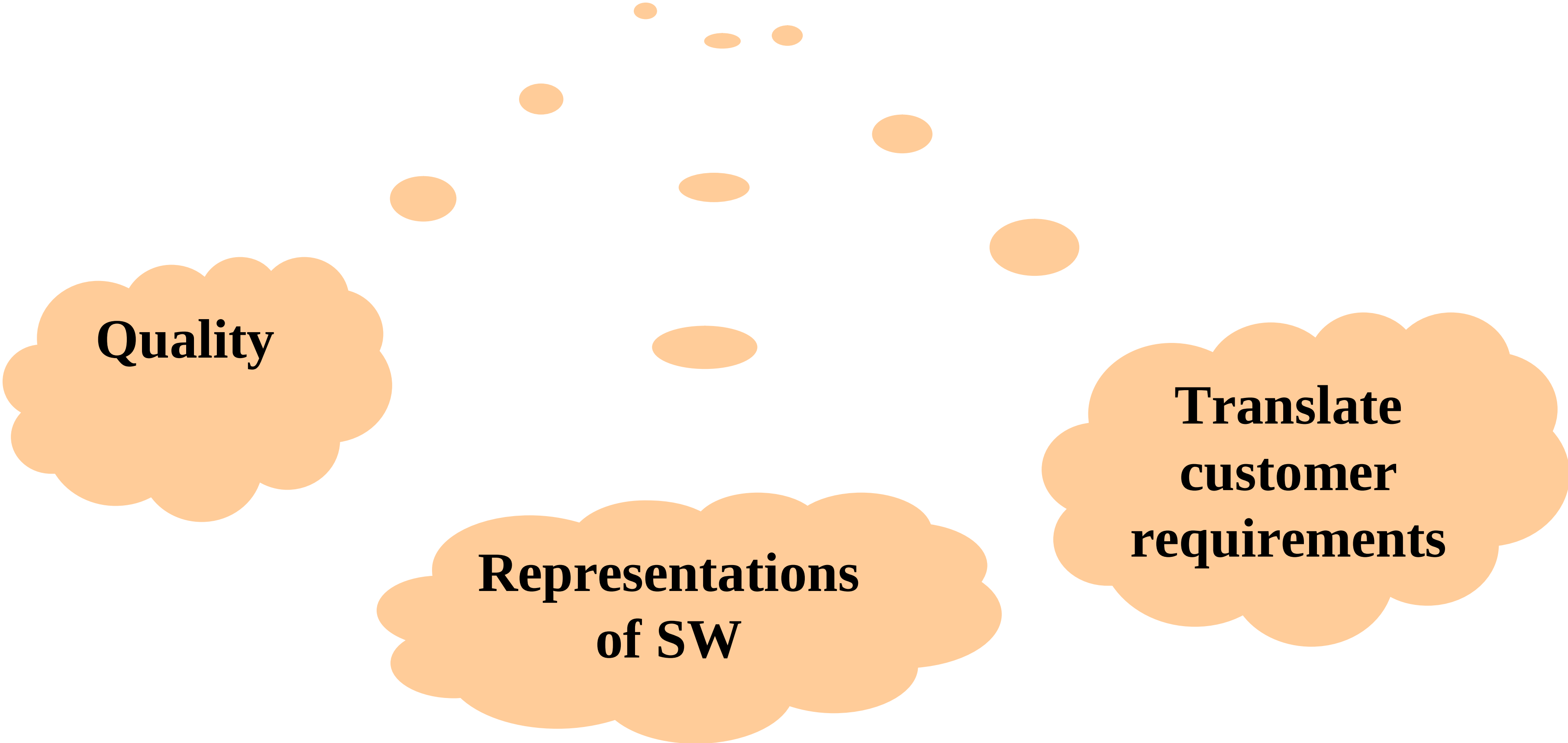
DESIGN MODEL


Analysis phase
Objectives:
To understand
1. Business events and processes
2. System activities and processing requirements
3. Information storage requirements
Analysis models and documents
Design phase
Objective:
To define, organize, and structure the components of the final solution system that will serve as the blueprint for construction

Design Model

Data design	Transforms information domain model into data structures
Architectural design	- Defines relationship between structural elements of the SW, - Represents the framework of computer based systems.
Interface design	- How S/w communicates within itself, with systems that interoperate with it, with humans - A flow of information & type of behavior
Component-level design	Transforms structural elements of the S/w architecture into a procedural description of SW components

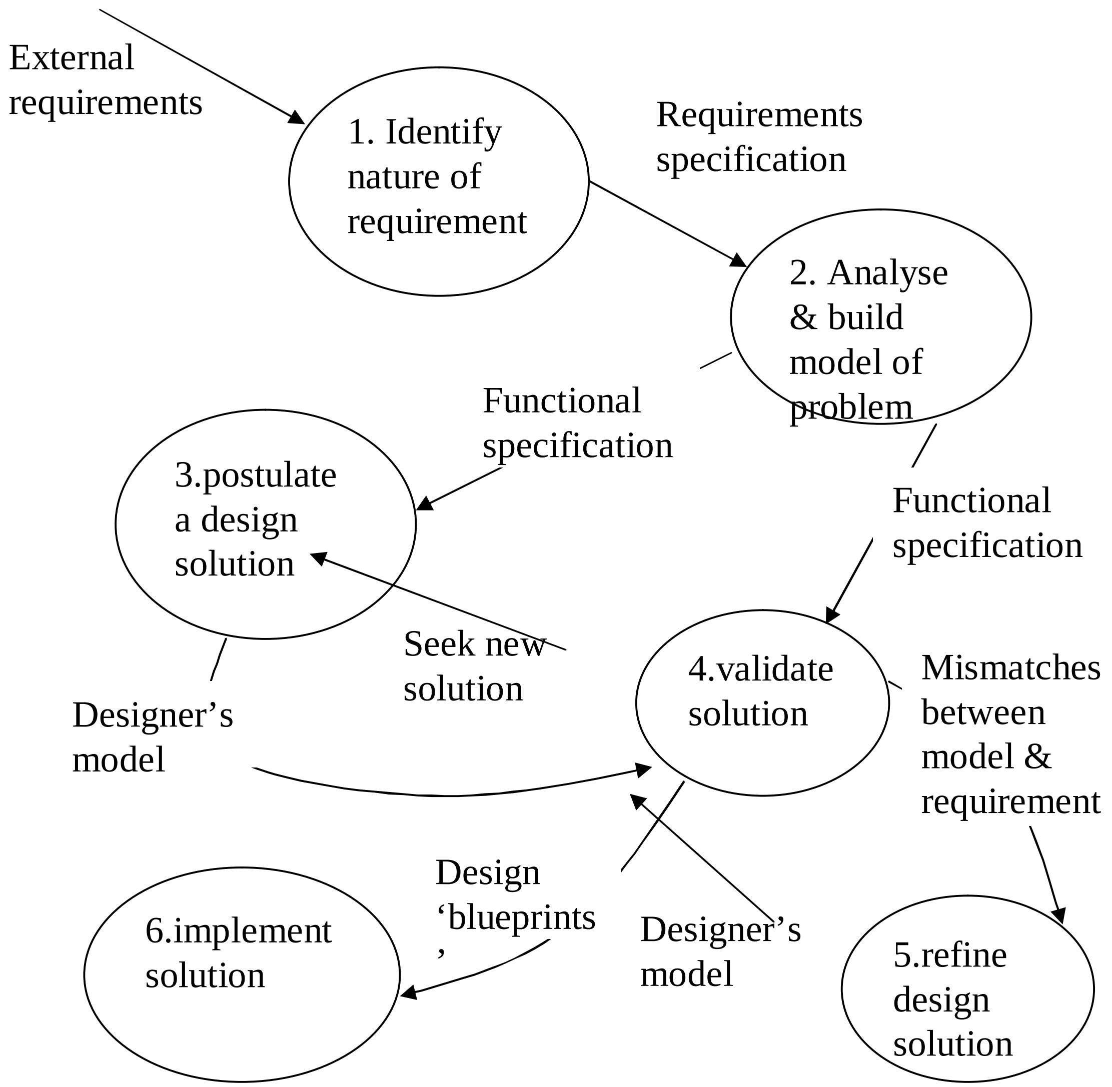
Importance of Software Design



Design Process

- SW design is an iterative process
- Requirements are translated ➔
  "blueprint" for constructing the SW

A Model of the Design Process



Design Guidelines

- A design should exhibit an architectural structure
- A design should be modular
- A design should contain distinct representations of data, architecture, interfaces, & components
- A design should lead to data structures that are appropriate for the objects to be implemented

Design Guidelines

- A design should lead to components that exhibit independent functional characteristics
- A design should lead to interfaces that reduce the complexity of connections between modules & external environment
- A design should be derived using a repeatable method , driven by information obtained during SW requirement analysis

Design Principles

- *The design process should not suffer from ‘tunnel vision’*
- *The design should be traceable to the analysis model*
- *The design should not reinvent the wheel*
- *The design should ‘minimize the intellectual distance” between the SW and the problem as it exists in the real world*
- *The design should exhibit uniformity & integration*

Design Principles …cont.

- *The design should be structured to accommodate change*
- *The design should be structured to degrade gently, even when aberrant data, events, or operating conditions are encountered*
- *Design is not coding, coding is not design*
- *The design should be assessed for Quality as it is being created, not after the fact*
- *The design should be reviewed to minimize conceptual (semantic) errors*

Fundamental SW Design Concepts

Abstraction – data, procedure, control

Information hiding– controlled interfaces

Refinement – elaboration of detail for all abstractions

Procedure – algorithms that achieve function

Modularity – compartmentalization of data & function

Architecture – overall structure of the SW

Design Concepts

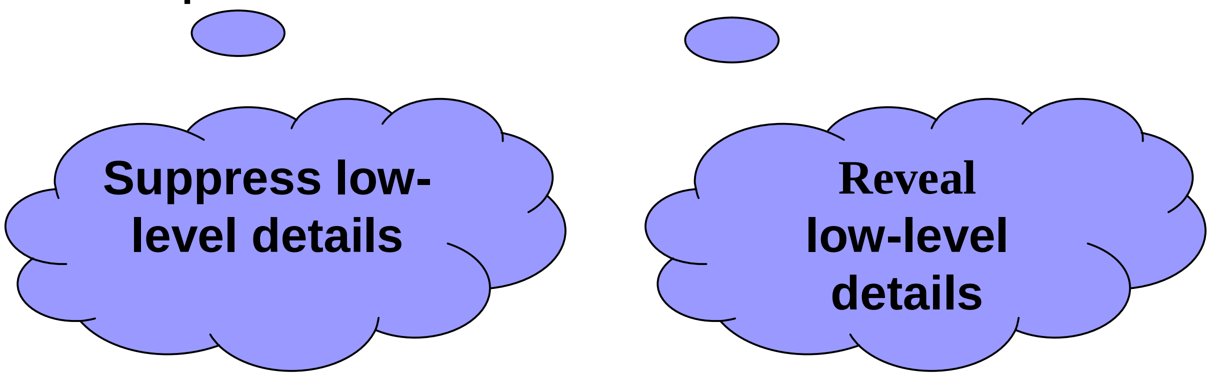
1. <u>Abstraction</u>

- concentrate on the essential features and ignore details that are not relevant
  - Procedural abstraction = a named sequence of instructions that has a specific & limited function
  - Data abstraction = a named collection of data that describes a data object
  - Control abstraction = implies a program control mechanisms without specify internal details

Design Concepts…cont

2. Refinement

- stepwise refinement = top down strategy
- refine levels of procedural detail
- develop hierarchy by decompose a procedural abstraction in a stepwise fashion until programming languages are reached
- similar to the process of refinement & partitioning in requirement analysis
- abstraction & refinement are complementary concepts



Design Concepts…cont

3. <u>Modularity</u>

- system is decomposed into a number of modules
- software architecture and design patterns embody modularity
- 5 criteria to evaluate a design method with respect to its ability to define effective modular system [meyer,88]

  i. modular decomposability

     - provides a systematic approach for decomposing the problem into subproblems

  ii. modular composability

     - enables existing(reusable) design components to be assembled into a new system

o

Design Concepts…cont

- 5 criteria to evaluate a design method with respect to its ability to define effective modular system [meyer,88]

iii. modular understandability

- *module can be understood as a standalone unit (no need to refer to other modules)*

iv. modular continuity

- *small changes to the system requirements result in changes to individual modules*

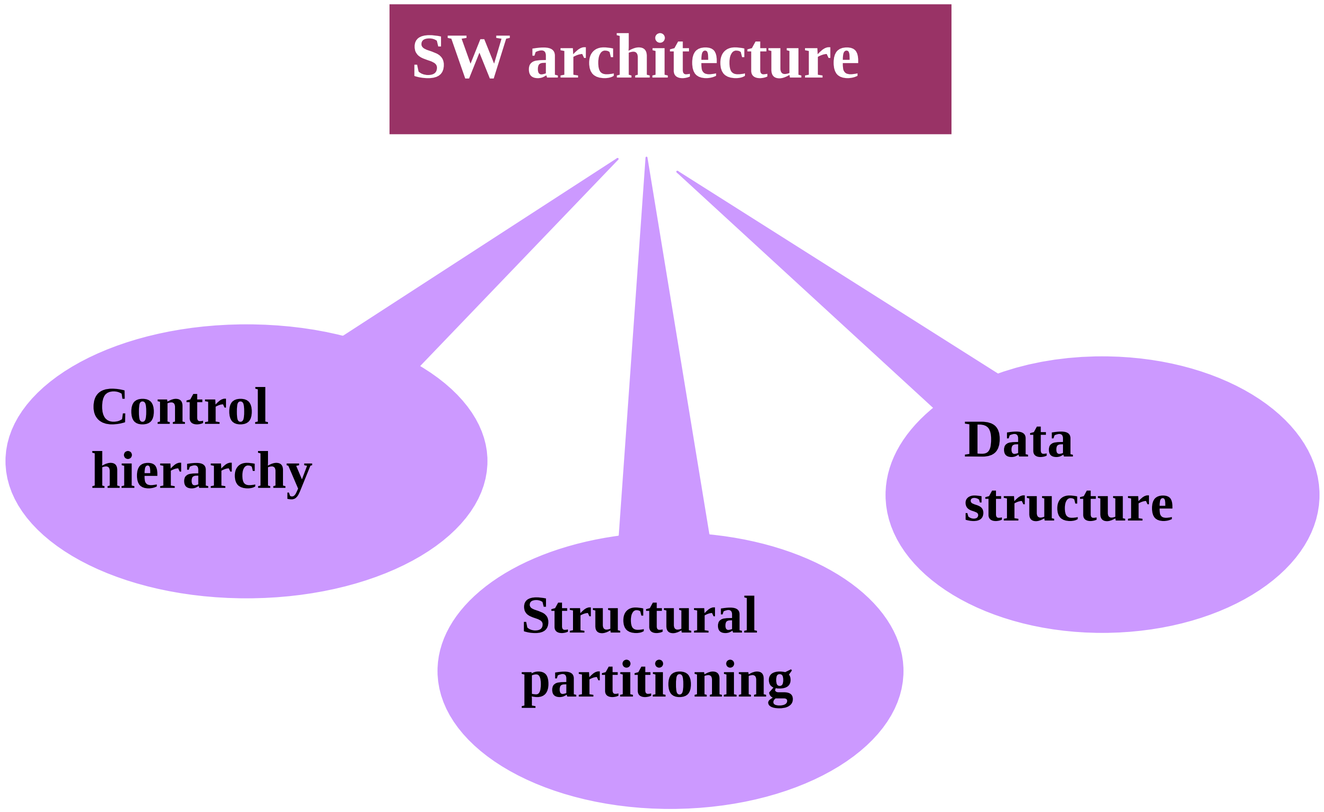
v. modular protection

- u*nexpected condition occurs within a module & its effects are constrained within that module*

4. Architecture

- is the structure and organization of program components (modules)
- 5 different types of models are used to represent the architectural design :
  - **i.** **structural models** : represent architecture as an organized collection of program components
  - **ii.** **framework models** : identify repeatable architectural design framework that similar to the types of applications
  - **iii.** **dynamic models** : behavioral aspects of the program architecture
  - **iv.** **process models** : design of the business or technical process of a system
  - **v.** **functional models** : functional hierarchy of a system

Design Concepts…cont



SW Architecture- Control Hierarchy

- also called program structure
  - represent the organization of program components
  - *depth & width = no. of levels of control & overall span of control*
  - *fan-out = no. of. Modules that are directly controlled by another module*
  - *fan-in = how many modules directly control a given module*
  - *super ordinate = module that control another module*
  - *subordinate = module controlled by another*

- **<u>horizontal partitioning</u>**
  - *defines separate branches of the modular hierarchy for each major program function*
  - *defines 3 partition = Input, Process, Output*
  - ***benefits** :*
    - *easy to test SW*
    - *easy to maintain SW*
    - *propagation of fewer side effects*
    - *easy to extend SW*
  - **drawback**: causes more data to be passed across module interfaces, complicate overall control of program flow

- **<u>vertical partitioning (factoring)</u>**
  - control (decision making) & work should be distributed Top-Down in program structure
  - **top level modules** – do control function, low level modules – do the works(input, computation, output tasks)
  - a change in control module will effect the subordinates.
  - a change in worker module will less likely effect others.

- representation of the logical relationship among individual elements of data.
- the organization, methods of access, degree of associatively & processing alternatives for information.

5. Procedure
   - focuses on the processing details of each module
   - a precise specification of processing

6. <u>Information Hiding</u>
   - modules should be specified & designed so that information (procedure & data) contained within a module in inaccessible to other modules that have no need for such information

Effective Modular Design

- Concept of functional independence is a direct outgrowth of :
  - modularity
  - Information hiding

- Functional independence measured by 2 criteria :
  - cohesion
  - coupling

Terminology to describe interactions between modules

Effective Modular Design…cont

- A piece of SW divided into modules :
  - There is a minimum interactions between modules [low coupling] AND
  - High degree of interaction within a module [high cohesion]

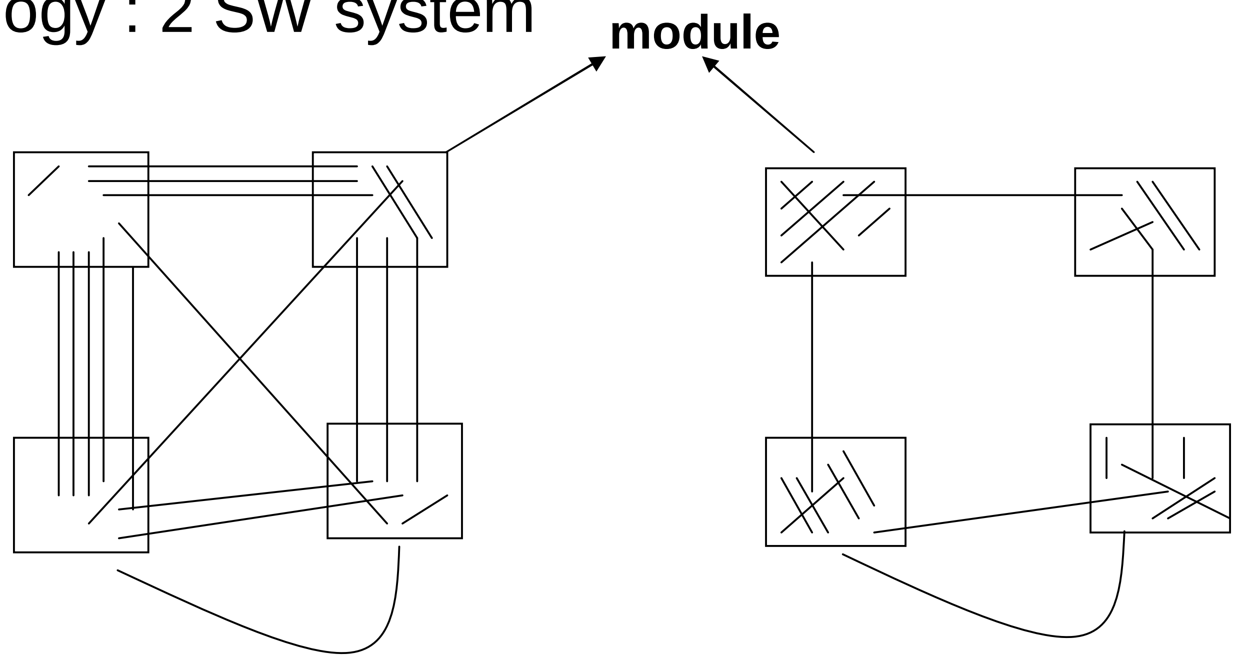
achieved

↓

Individual module can be :
DESIGNED, CODED,
TESTED OR CHANGED

Effective Modular Design…cont

- analogy : 2 SW system



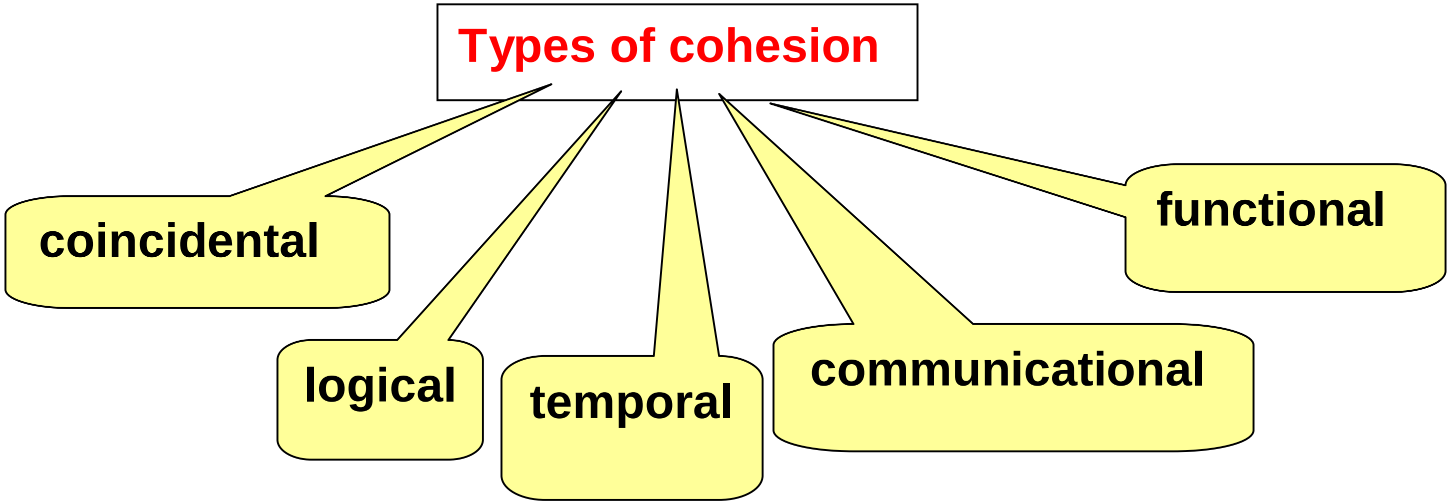
- many interactions between modules
- few within modules
- strong coupling , weak cohesion

- Few interactions between modules
- many within modules
- weak coupling , strong cohesion

Effective Modular Design…cont

COHESION

- a cohesive module perform a single task.
- Describes the nature of interactions within a SW module
- Various types of cohesion :
  - LOW cohesion(undesirable) → HIGH cohesion (desirable)



Effective Modular Design…cont

- *Coincidental* :
  - grouped into modules in a haphazard way
  - no relationship between component
- *Logical* :
  - Perform a set of logically similar functions
  - Example : function => output anything
    - Perform output operations :
      - Output text to screen
      - Output line to printer
      - Output record to file

Design Concepts…cont

- ***Temporal :***
  - Performs a set of functions whose only relationship is that they have to be carried out at the same time
  - Example : set of initialization operations
    - Clear screen
    - Open file
    - Initialize total

Design Concepts…cont

- ***<u>Communicational :</u>***
  - ✓ Acting on common data are grouped together
  - ✓ Example :

    i) displays and logs temperature

    ii) formats and prints total price

- ***<u>Functional :</u>***
  - Optimal type of cohesion
  - Performs single, well-defined action
  - Example :

    i) calculate average

    ii) print result

Effective Modular Design…cont

COUPLING

One module making a procedure call on another

Important aspect of interaction between modules ➔ SIZE

Coupling

- If modules share common data, ➔ it should be minimized
- Few parameters should be passed between modules in procedure calls [*recommended ➔2 – 4 parameters*]
- Types of coupling, from strongly coupled (least desirable) →weakly coupled (most desirable) :
  1. Content coupling
  2. Common coupling
  3. External coupling
  4. Control coupling
  5. Stamp coupling
  6. Data coupling

Types of coupling

1. Content coupling (should be avoided)

- *module directly affects the working of another module*
- *occurs when a module changes another module's data or when control is passed from 1 module to the middle of another (as in a jump)*

Types of coupling…cont.

2. <u>Common coupling</u>

- *2 modules have shared data*
- *occurs when a number of modules reference a global data area*

Types of coupling…cont.

3. External coupling

- *Modules communicate through an external medium (such as file)§*

Types of coupling…cont.

4. <u>Control coupling</u>

- *1 module directs the execution of another module by passing the necessary control information*
- *accomplished by means of flags that set by one module and reacted upon by the dependent module*

Types of coupling…cont.

5. Stamp coupling

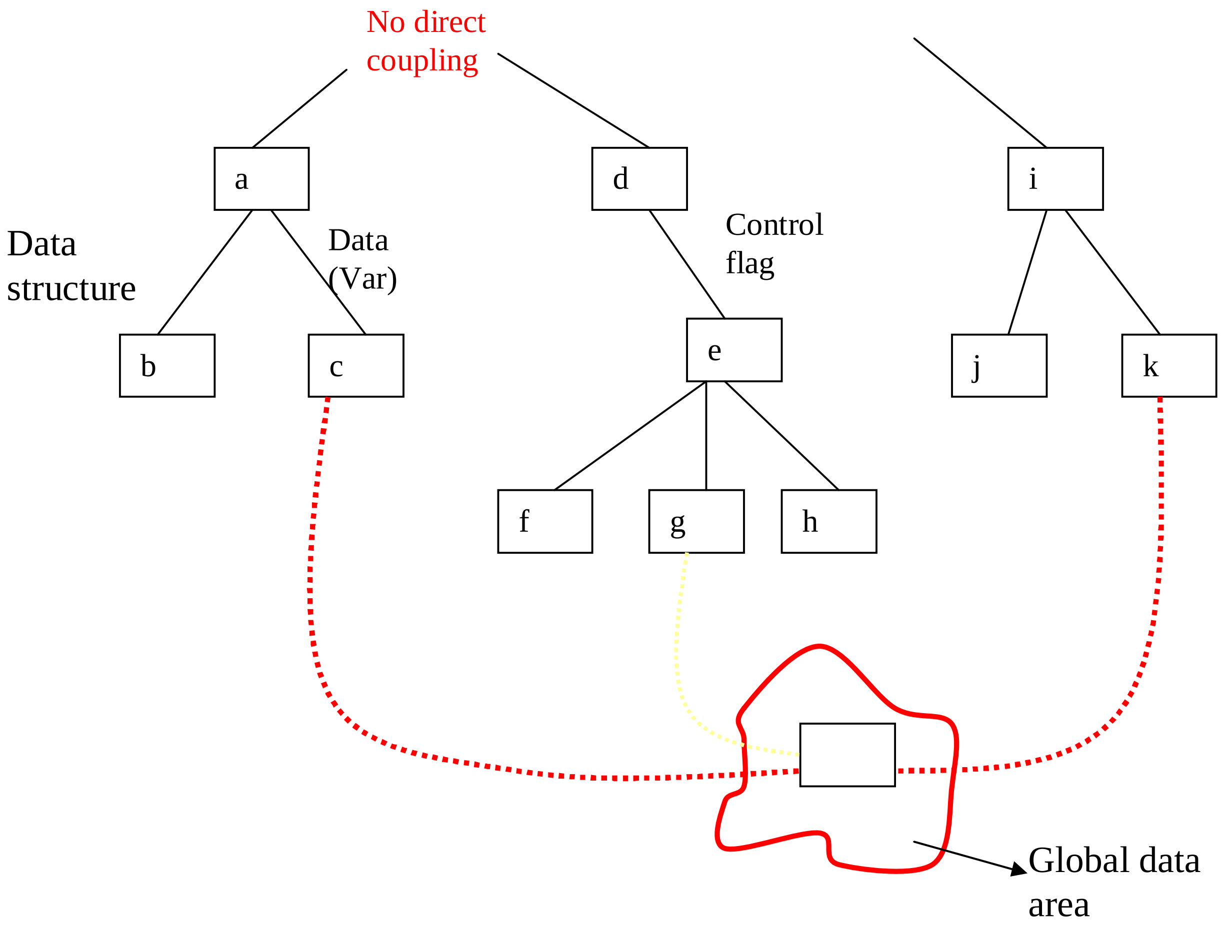
- *Occurs when complete data structures are passed from 1 module to another*
- *The precise format of the data structures is a common property of those modules*

Types of coupling…cont.

6. <u>Data coupling (low coupling)</u>

- *Only simple data is passed between modules*
- *1 to 1 correspondence of items exists*

Types of coupling [pressman]



Structure Chart (SC)

- also called a hierarchy chart or program structure chart
- a design technique that employs functional decomposition to specify the functions of an application program

A method for designing the detailed structure of individual programs or modules

Is a ‘top-down’ method
- it starts with the overall task of the program

Also called ‘stepwise refinement ‘

Structure Chart (SC)…cont

- A structure chart differs from a physical DFD
- Transform a physical DFD into a structure chart by performing 2 forms of analysis :
  - ✓ <u>Transform analysis</u> *: used to segment sequential DFD processes into input, processing & output functions*
  - ✓ <u>Transaction analysis</u> *: used to segment DFD processes into alternate flows of control depending on a condition or user selection*

- Each DFD process will appear as a structure chart module

A few guidelines to use structure charts effectively :

- Level 1 of a SC may have only 1 module (*the boss module, executive module or main module*)
- Each module should perform only 1 function
- Modules at each level should be arranged in sequential order (Left – Right). [*exception to : modules that represent alternative functions selected by higher-level module*]
- Selection symbol used to indicate that not all modules at the next lower level will be performed

A few guidelines to use structure charts effectively :

- Loop symbol used to indicate that some or all of the modules at the next lower level will be performed repeatedly until a specified condition is met
- Data elements passed between modules should be specified by labeling each data flow symbol with the name assigned to each element in the physical DB design

A few guidelines to use structure charts effectively :

- Control flows used to indicate a control flag or other value that is direct the flow of control or to report the status of a condition
- Each program structure chart should be reviewed & evaluated ➔ to ensure the design works before coding begins

~The End~

Source of Materials :

- Pressman, R. (2003). *Software Engineering: A Practitioner's Approach*. 6th & 5th edition. New York: McGraw-Hill.
- Somerville I. (2001). *Software Engineering*. 6th edition. Addison Wesley