DESIGN CONCEPTS

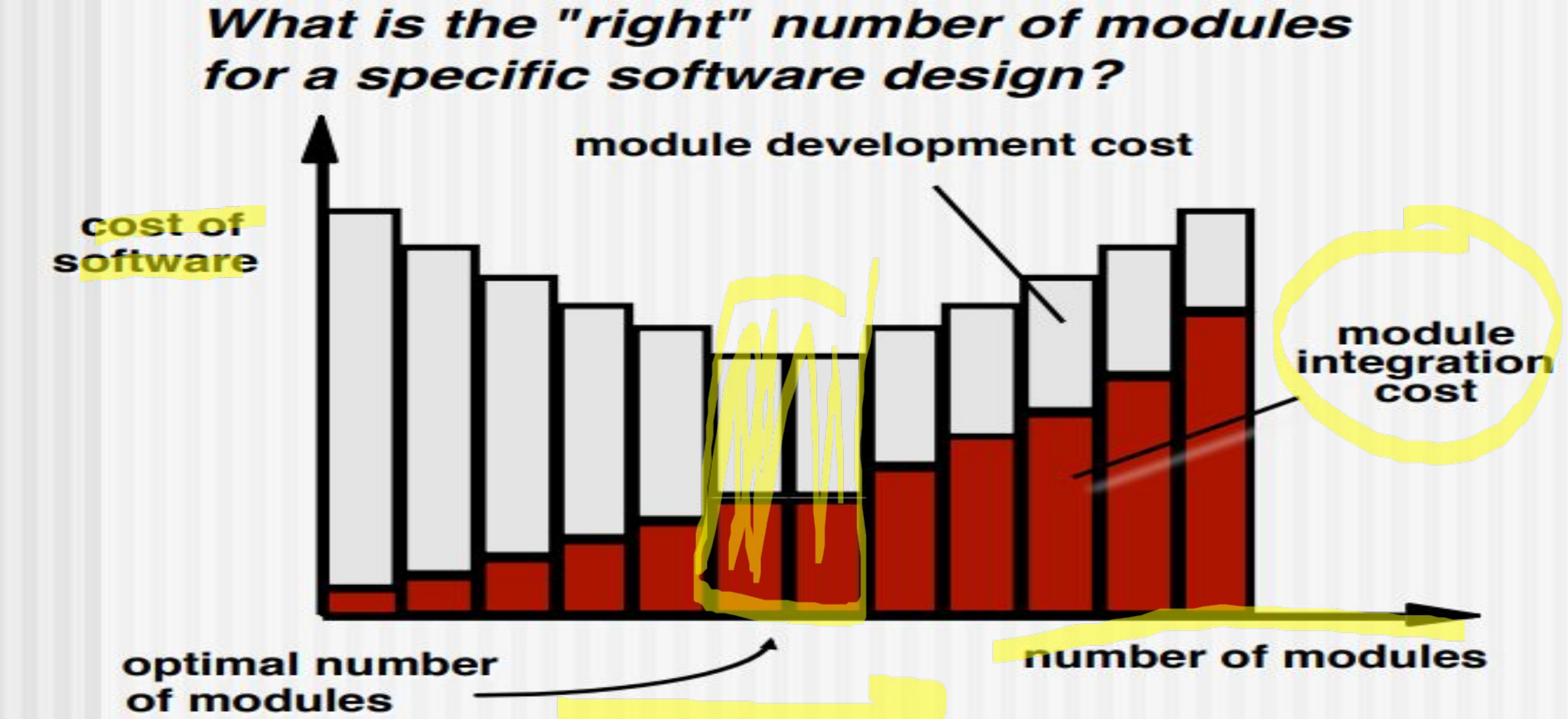
Lecture # 21

Lecture by Engr. Sidra

MODULARITY

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.
  - Modular decomposability: A design method provides a systematic mechanism for decomposing the problem into sub-problems -->reduce the complexity and achieve the modularity
  - Modular composability: A design method enables existing design components to be assembled into a new system.
  - Modular understandability: A module can be understood as a standalone unit it will be easier to build and easier to change.
  - Modular continuity: A small changes to the system requirements result in changes to individual modules, rather than system- wide changes.
  - Modular protection: An aberrant condition occurs within a module and its effects are constrained within the module.

MODULARITY TRADE-OFFS

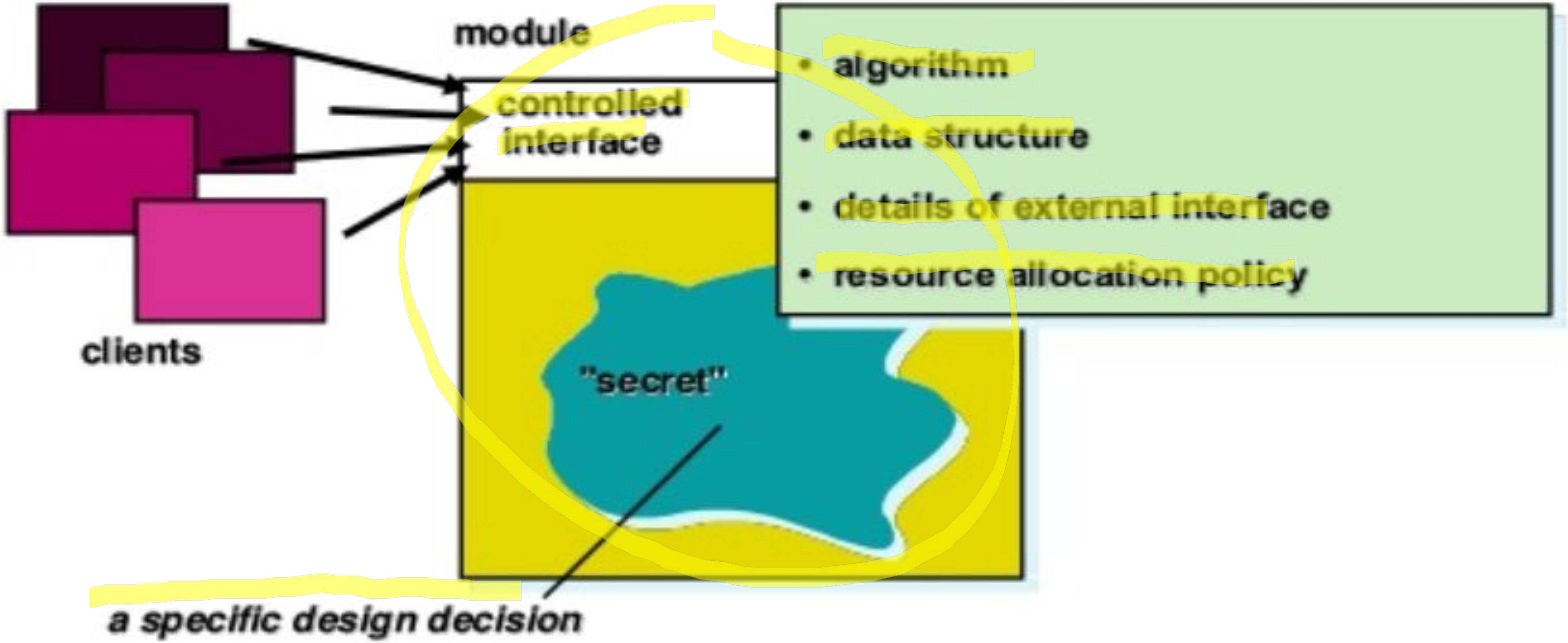


Lecture by Engr. Sidra

INFORMATION HIDING

- Information (data and procedure) contained within a module should be inaccessible to other modules that have no need for such information.
- Hiding defines and enforces access constraints to both procedural detail within a module and any local data structure used by the module
- Why Information Hiding?
  - Decreases the probability of adverse effects
  - Restricts the effects of changes in one component on others
  - Emphasizes communication through controlled interfaces
  - Discourages the use of global data
  - Leads to encapsulation—an attribute of high-quality design

INFORMATION HIDING



- algorithm
- data structure
- details of external interface
- resource allocation policy

FUNCTIONAL INDEPENDENCE

- Functional independence is achieved by developing modules with "single-minded" function and an "aversion" to excessive interaction with other modules.
- Independence is assessed using two qualitative criteria:
- Cohesion is an indication of the relative functional strength of a module.
  - A cohesive module performs a single task, requiring little interaction with other components in other parts of a program. Stated simply, a cohesive module should(ideally) do just one thing.
- Coupling is an indication of the relative interdependence among modules.
  - Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

COHESION

- A cohesive module performs a single task in a procedure with little interactions with others.
- Goal
  - Achieve high cohesion for modules in a system.
- Different types of cohesion:
  - Communication cohesion: data sharing among processing elements
  - Procedural cohesion: order among processing elements
  - Coincidentally cohesive: a set of tasks related to each other loosely
  - Logically cohesive: logical connection among processing elements

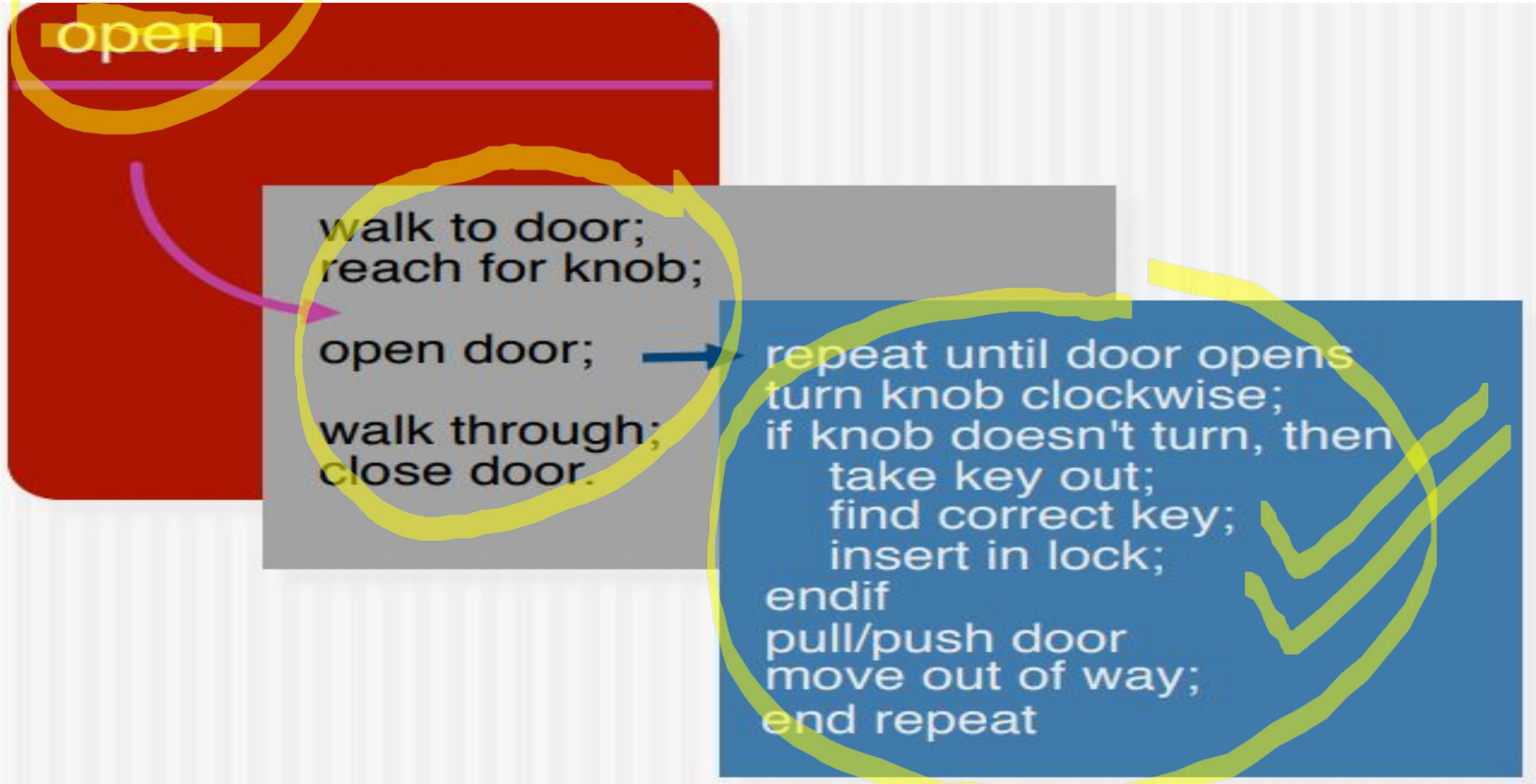
COUPLING

- Goal:
  - Strive for lowest possible coupling among modules.
- Good coupling:
  - Reduce or avoid change impact and ripple effects.
  - Reduce the cost in program changes, testing, maintenance
- Types of coupling:
  - Data coupling: parameter passing or data interaction
  - Control coupling: share related control logical (for a control data)
  - Common coupling: common data sharing
  - Content coupling: module A use of data or control information maintained in another module.

REFINEMENT

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.
- Abstraction and refinement are complementary concepts

STEPWISE REFINEMENT



open

```
walk to door;
reach for knob;

open door;

walk through;
close door.
```

```
repeat until door opens
turn knob clockwise;
if knob doesn't turn, then
    take key out;
    find correct key;
    insert in lock;
endif
pull/push door
move out of way;
end repeat
```

ASPECT

- A crosscutting concern is some characteristic of the system that applies across many different requirements.
- It is important to identify aspects so that the design can properly accommodate them as refinement and modularization occur.
- In an ideal context, an aspect is implemented as a separate module (component) rather than as software fragments that are “scattered” or “tangled” throughout many components.
- To accomplish this, the design architecture should support a mechanism for defining an aspect—a module that enables the concern to be implemented across all other concerns that it crosscuts.

REFACTORING

- It is a reorganization technique which simplifies the design of components without changing its function or behavior.
- Fowler [FOW99] defines refactoring in the following manner:
  - "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.”
- When software is refactored, the existing design is examined for:
  - Redundancy
  - Unused design elements
  - Inefficient or unnecessary algorithms
  - Poorly constructed or inappropriate data structures
  - or Any other design failure that can be corrected to yield better design.