SOFTWARE ENGINEERING

Spring 2024

SOFTWARE DESIGN

- The design process comprises a set of principles, concepts and practices, which allow a software engineer to model the system or product that is to be built.

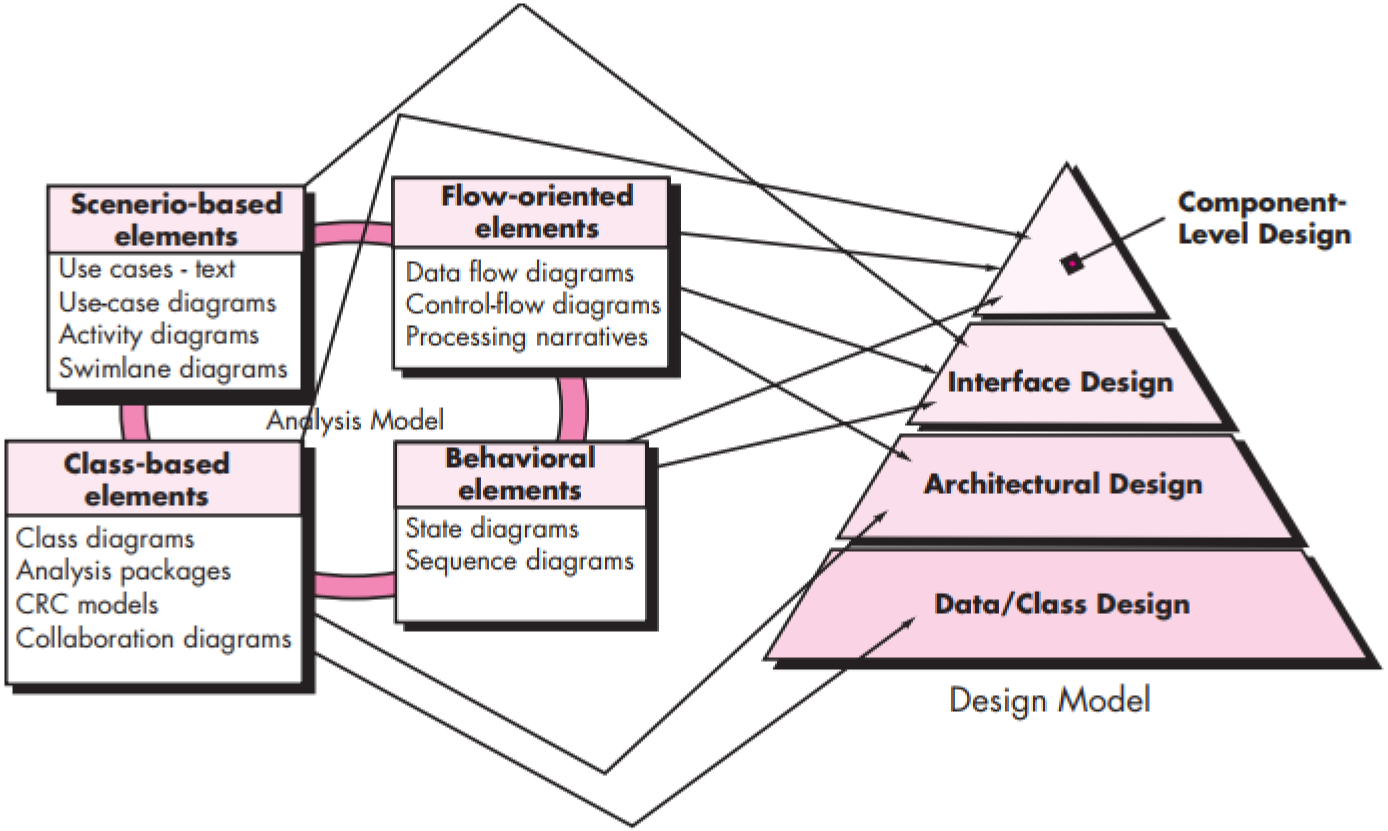
- Software design is a phase in software engineering, in which a blueprint is developed to serve as a base for constructing the software system.

- Many critical and strategic decisions are made to achieve the desired functionality and quality of the system.

DESIGN

- Mitch Kapor, the creator of Lotus 1-2-3, presented a "software design manifesto" in *Dr. Dobbs Journal.* He said:
  - Good software design should exhibit:
  - *Firmness:* A program should not have any bugs that inhibit its function.
  - *Commodity:* A program should be suitable for the purposes for which it was intended.
  - *Delight:* The experience of using the program should be pleasurable one.

ANALYSIS MODEL -> DESIGN MODEL



DESIGN AND QUALITY

- the design must implement all of the explicit requirements
- the design must be a readable, understandable guide
- the design should provide a complete picture of the software

QUALITY GUIDELINES

- A design should exhibit an architecture
- A design should be modular
- A design should contain distinct representations
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.
- A design should lead to interfaces that reduce the complexity
- A design should be derived using a repeatable method
- A design should be represented using a notation that effectively communicates its meaning.

ATTRIBUTE

- Hewlett-Packard [Gra87] developed a set of software quality attributes that has been given the acronym FURPS.
- *Functionality* is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system.
- *Usability* is assessed by considering human factors, overall aesthetics, consistency, and documentation.
- *Reliability* is evaluated by measuring the frequency and severity of failure, the accuracy of output results, the mean-time-to-failure (MTTF), the ability to recover from failure, and the predictability of the program.
- *Performance* is measured by considering processing speed, response time, resource consumption, throughput, and efficiency.
- *Supportability* combines the ability to extend the program (extensibility), adaptability, serviceability

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" [DAV95] between the software and the problem as it exists in the real world.
- The design should exhibit uniformity and integration.
- 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 CONCEPTS

- Abstraction—data, procedure, control
- Architecture—the overall structure of the software
- Patterns—"conveys the essence" of a proven design solution
- Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces
- Modularity—compartmentalization of data and function
- Hiding—controlled interfaces
- Functional independence—single-minded function and low coupling
- Refinement—elaboration of detail for all abstractions
- Aspects—a mechanism for understanding how global requirements affect design
- Refactoring—a reorganization technique that simplifies the design
- OO design concepts
- Design Classes—provide design detail that will enable analysis classes to be implemented

ABSTRACTION

- Powerful design tool to consider components at an abstract level
- The concept of abstraction can be used in two ways

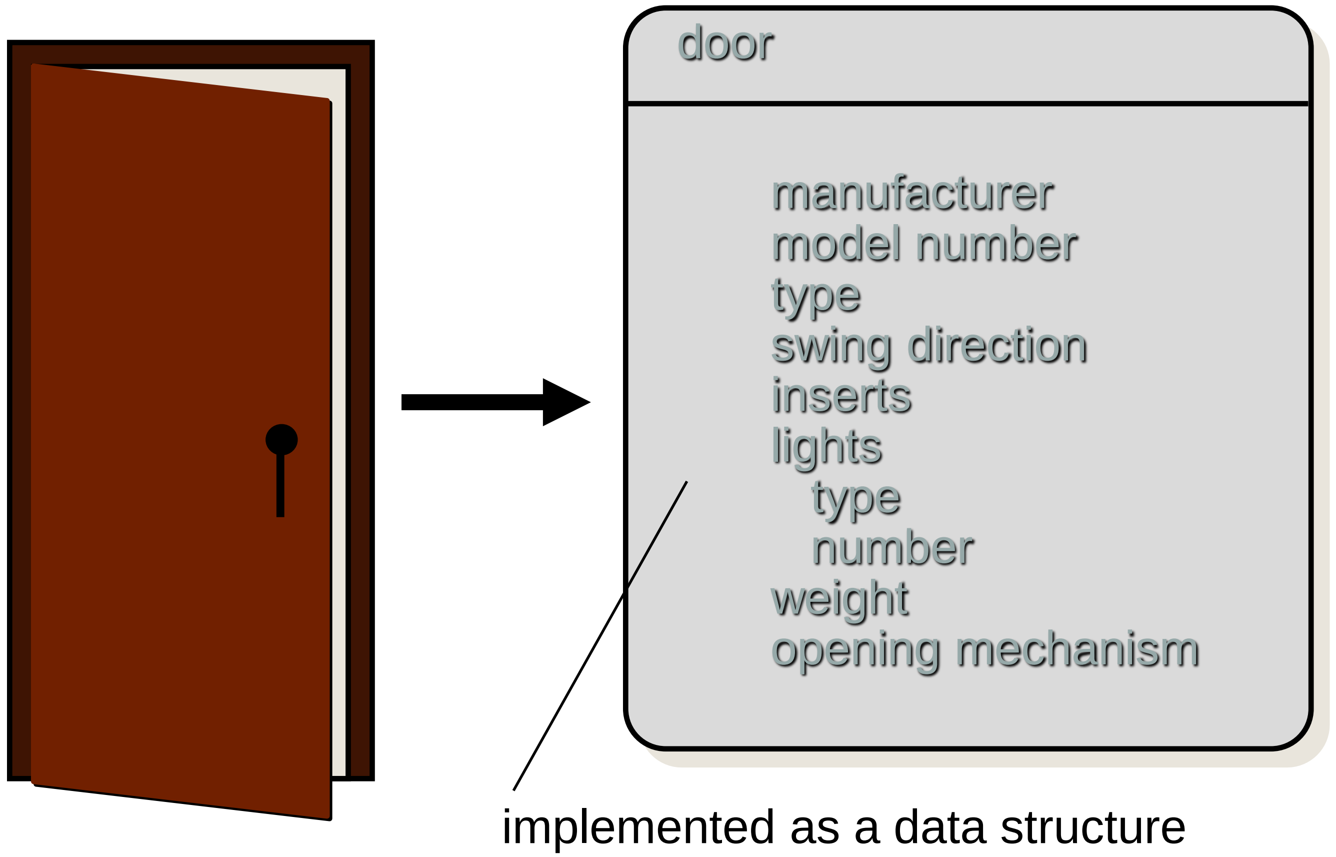
1)Process - Mechanism of hiding irrelevant details and representing only the essential features of an item so that one can focus on important things at a time

2)Entity - Model or view of an item.

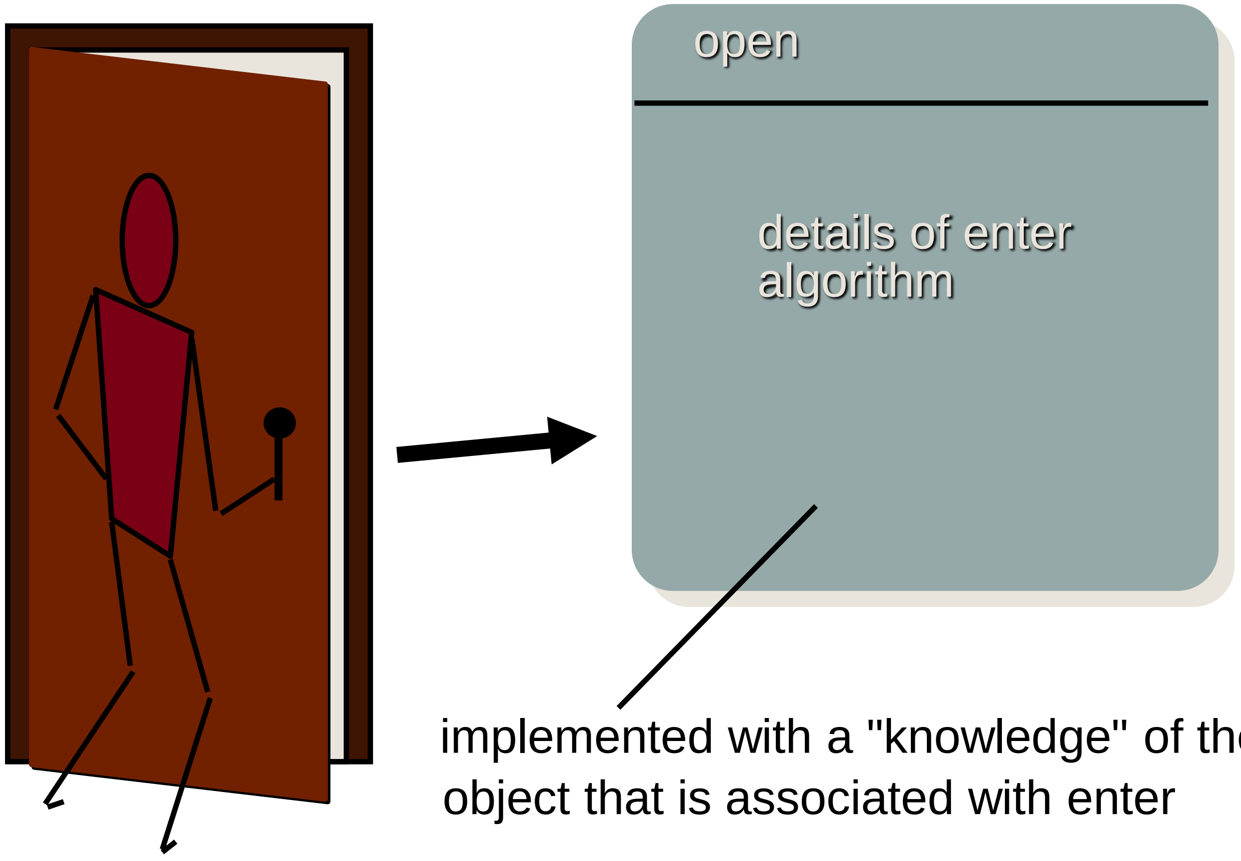
There are three commonly used abstraction mechanisms

- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

DATA ABSTRACTION



PROCEDURAL ABSTRACTION



implemented with a "knowledge" of the object that is associated with enter

ARCHITECTURE

- The whole structure of the software is called as software architecture.
- Structure gives conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The main aim of the software design is to obtain an architectural framework of a system.
- Shaw and Garlan [Sha95a] describe a set of properties that should be specified as part of an architectural design:
  - **Structural properties.**
  - **Extra-functional properties**
  - **Families of related systems.**

PATTERNS

- A design pattern represents a design structure and that structure solves a particular design problem in a specified content.

Design Pattern Template

Pattern name—**describes the essence of the pattern in a short but expressive name**

Intent—**describes the pattern and what it does**

Also-known-as—**lists any synonyms for the pattern**

Motivation—**provides an example of the problem**

Applicability—**notes specific design situations in which the pattern is applicable**

Structure—**describes the classes that are required to implement the pattern**

Participants—**describes the responsibilities of the classes that are required to implement the pattern**

Collaborations—**describes how the participants collaborate to carry out their responsibilities**

Consequences—**describes the "design forces" that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented**

Related patterns—**cross-references related design patterns**

SEPARATION OF CONCERNS

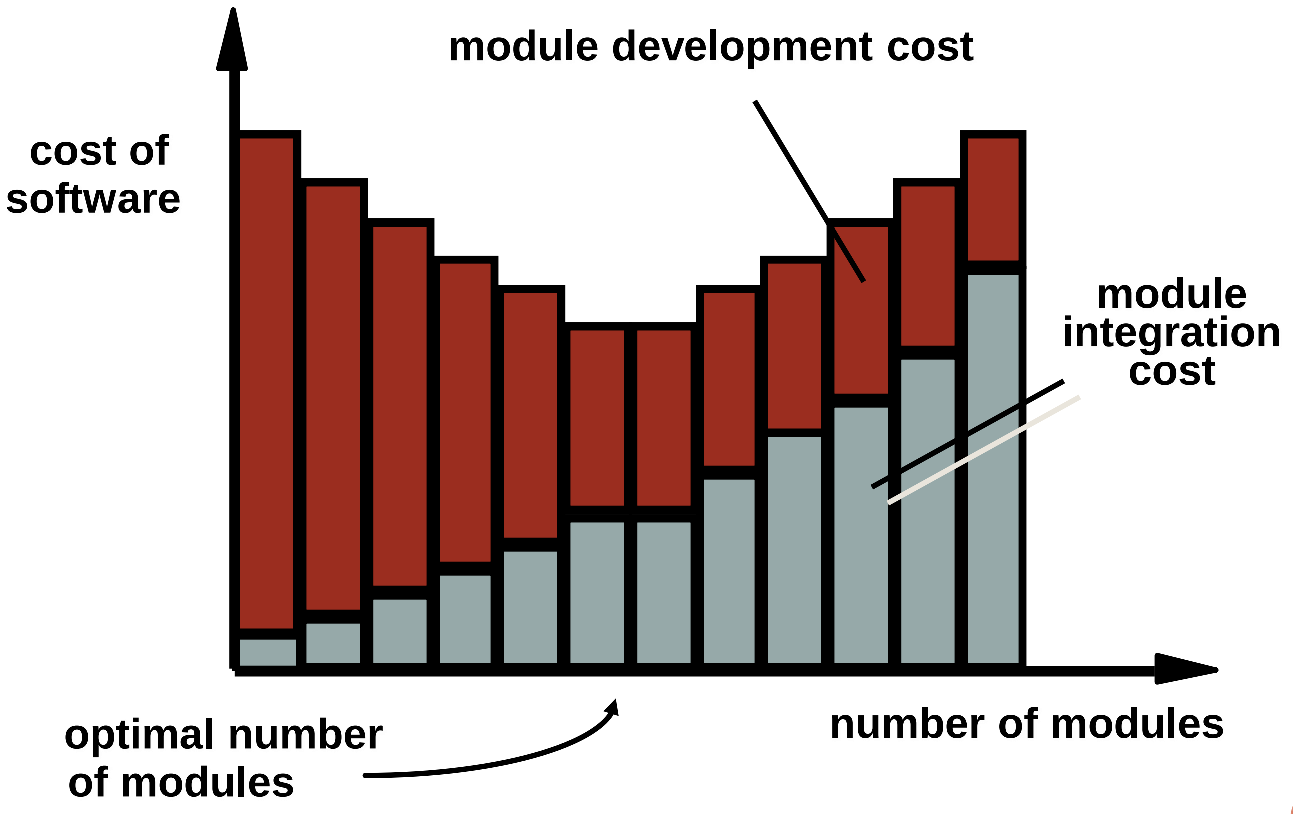
- Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently
- A *concern* is a feature or behavior that is specified as part of the requirements model for the software
- By separating concerns into smaller, and therefore more manageable pieces, a problem takes less effort and time to solve.

MODULARITY

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

MODULARITY: TRADE-OFFS

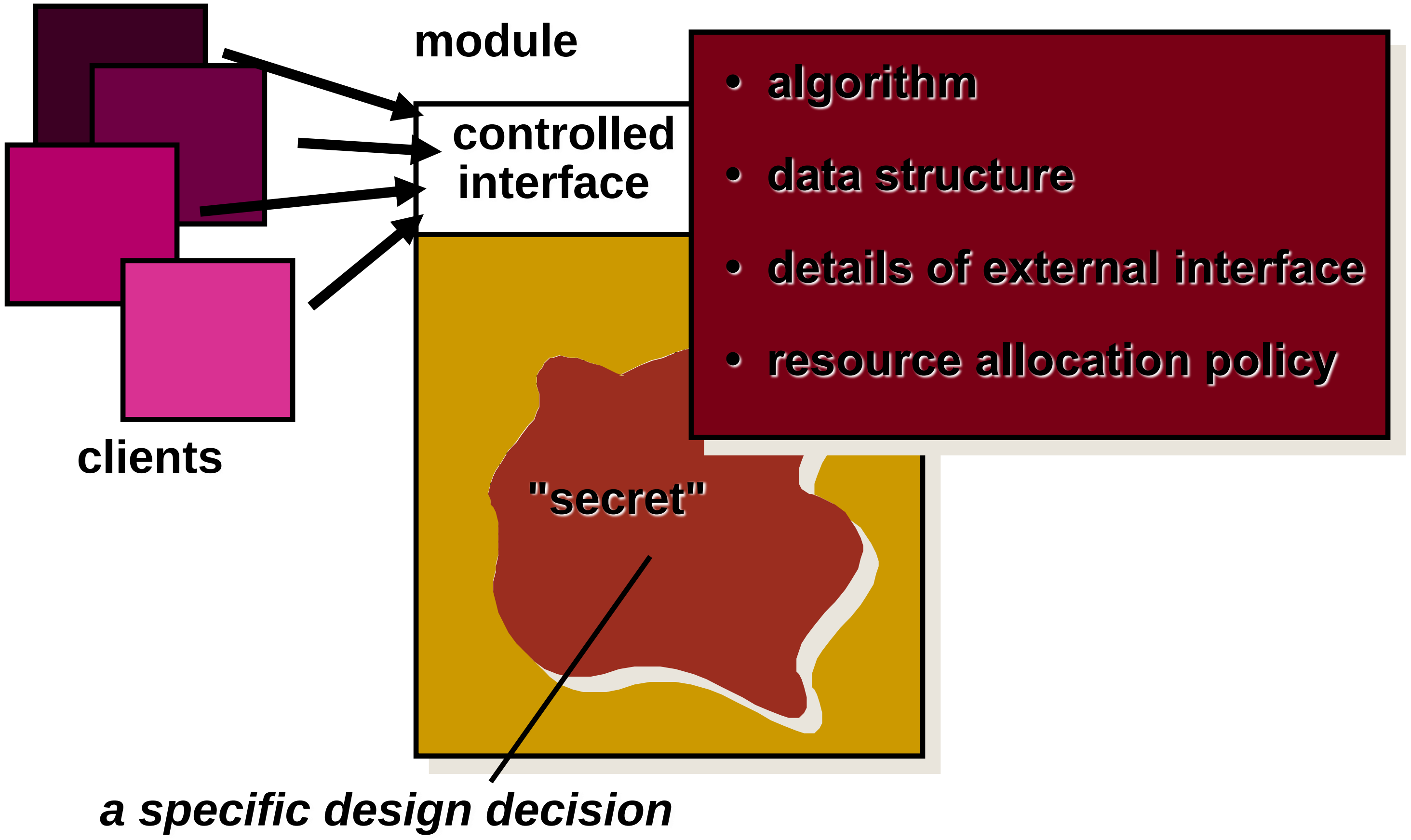
What is the "right" number of modules for a specific software design?



INFORMATION HIDING

- Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

INFORMATION HIDING



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

WHY INFORMATION HIDING?

- reduces the likelihood of "side effects"
- limits the global impact of local design decisions
- emphasizes communication through controlled interfaces
- discourages the use of global data
- leads to encapsulation—an attribute of high quality design
- results in higher quality software

FUNCTIONAL INDEPENDENCE

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.

Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

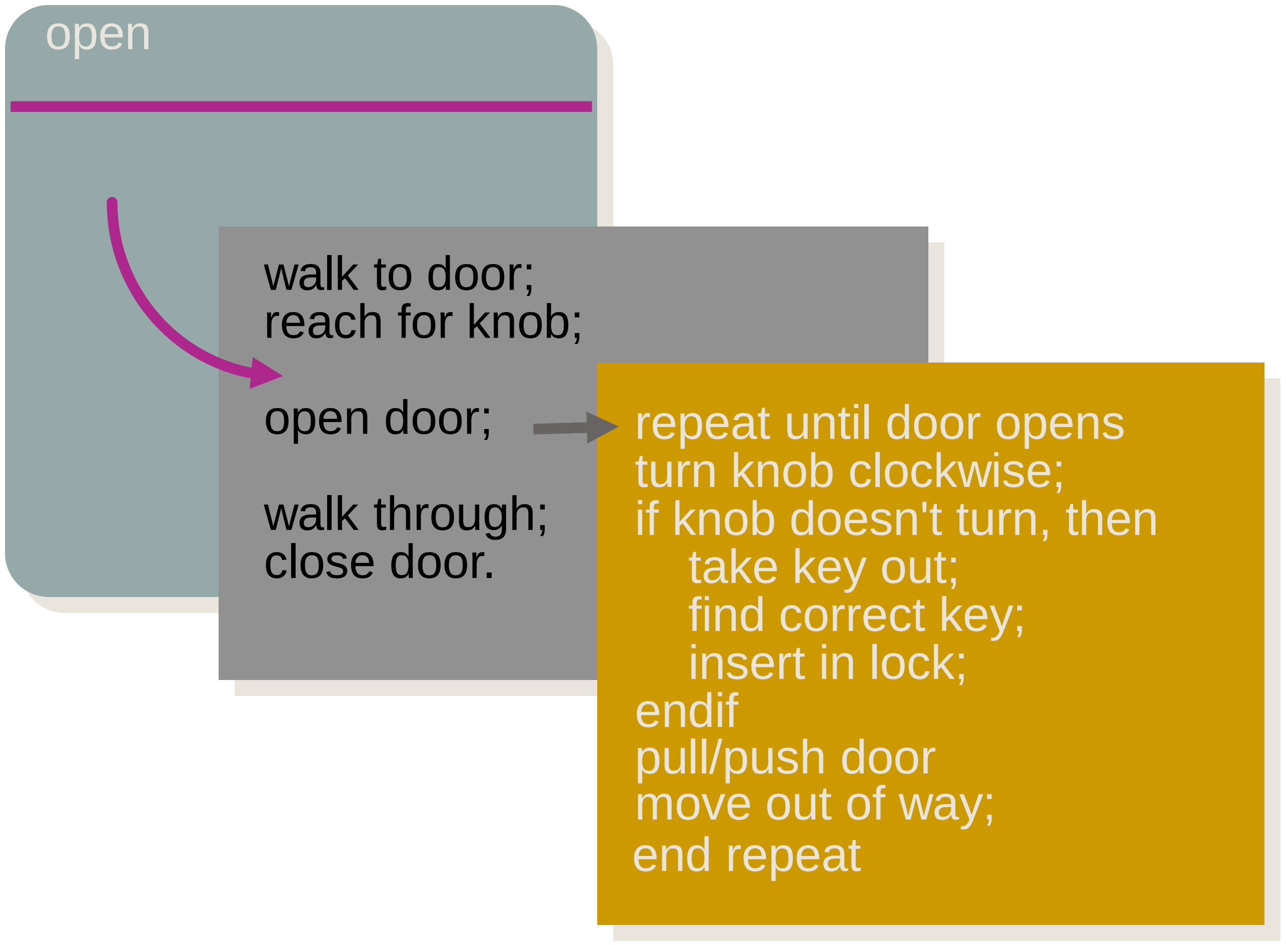
Coupling

Coupling is an indication of interconnection between modules in a structure of software.

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.

Stepwise Refinement



```
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
```

REFACTORING

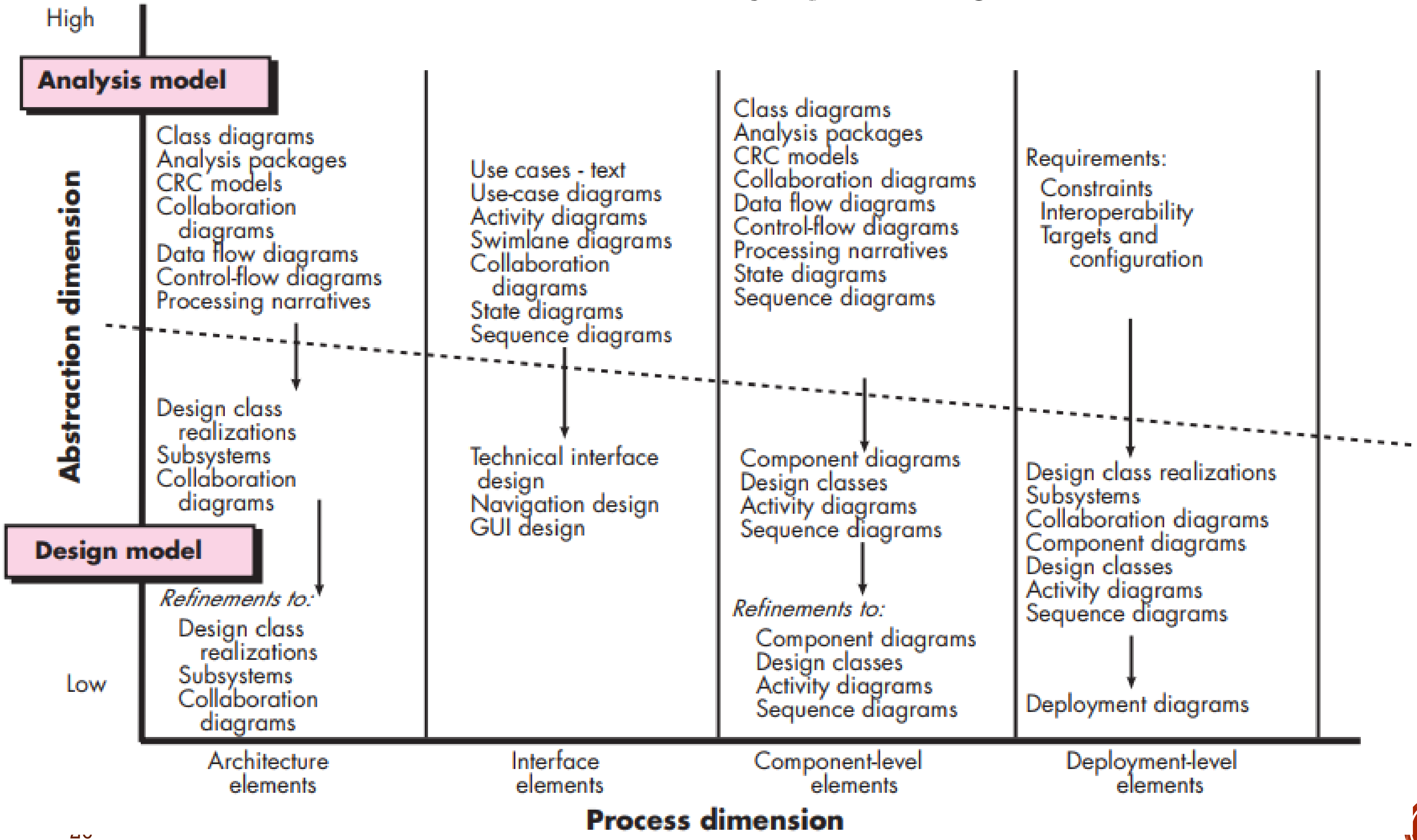
- It is a reorganization technique which simplifies the design of components without changing its function behavior.
- Refactoring is the process of changing the software system in a way that it does not change the external behavior of the code still improves its internal structure.

- 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 a better design.

DESIGN CLASSES

- Analysis classes are refined during design to become entity classes
- Boundary classes are developed during design to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.
  - Boundary classes are designed with the responsibility of managing the way entity objects are represented to users.
- Controller classes are designed to manage
  - the creation or update of entity objects;
  - the instantiation of boundary objects as they obtain information from entity objects;
  - complex communication between sets of objects;
  - validation of data communicated between objects or between the user and the application.

The Design Model



DESIGN MODEL ELEMENTS

- Data elements
  - Data model --> data structures
  - Data model --> database architecture
- Architectural elements
  - Application domain
  - Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
  - Patterns and "styles" (Chapters 9 and 12)
- Interface elements
  - the user interface (UI)
  - external interfaces to other systems, devices, networks or other producers or consumers of information
  - internal interfaces between various design components.
- Component elements
- Deployment elements