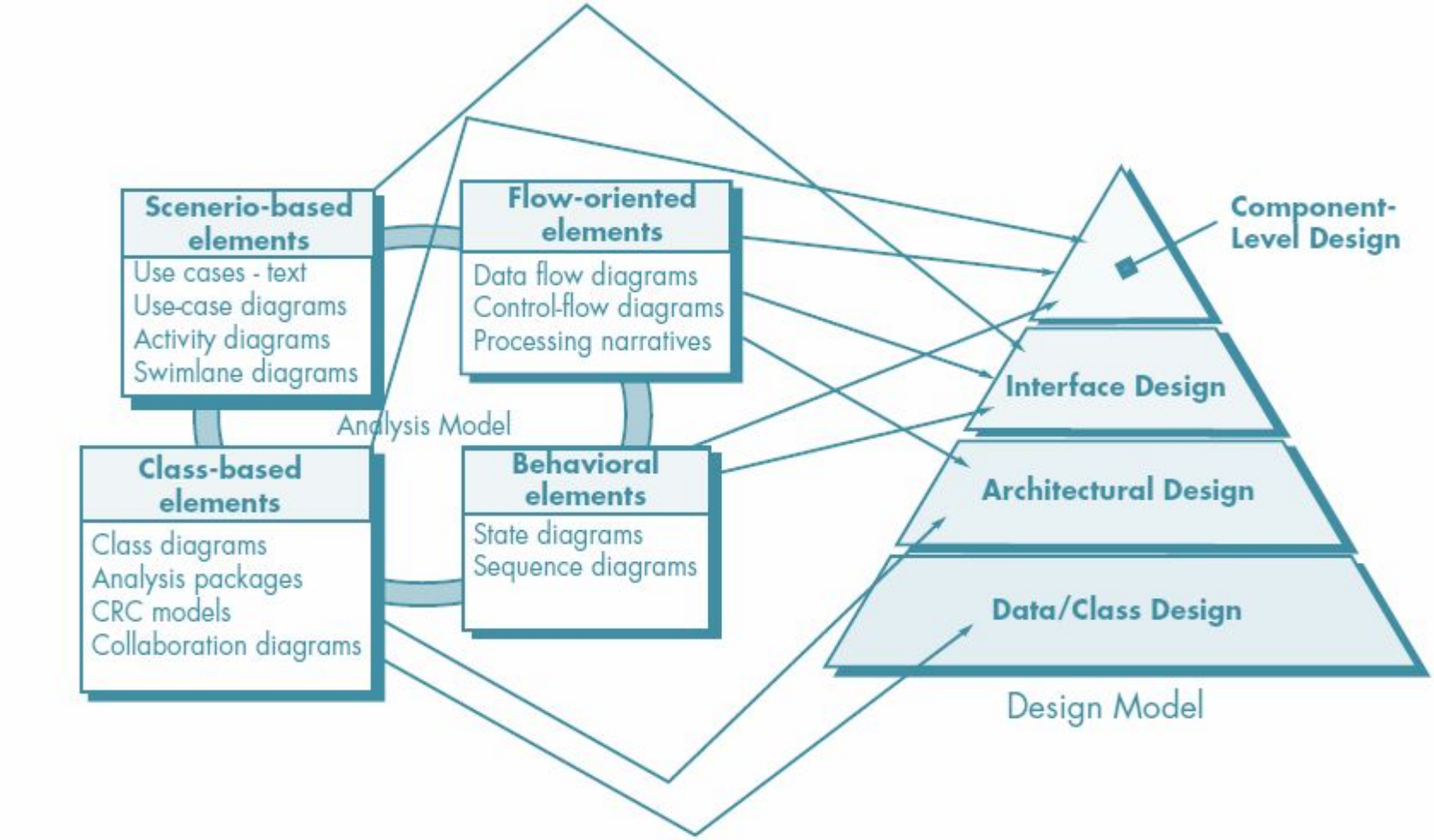
Design Engineering

The Design Process

- Software design
  - sits at the technical kernel of software engineering
  - is applied regardless of the software process model
  - is the last software engineering action within the modeling activity
  - sets the stage for construction (code generation and testing)


Scenerio-based elements
Use cases - text
Use-case diagrams
Activity diagrams
Swimlane diagrams
Flow-oriented elements
Data flow diagrams
Control-flow diagrams
Processing narratives
Analysis Model
Class-based elements
Class diagrams
Analysis packages
CRC models
Collaboration diagrams
Behavioral elements
State diagrams
Sequence diagrams
Component-Level Design
Interface Design
Architectural Design
Data/Class Design
Design Model

- Software design is an iterative process
- requirements are translated into a “blueprint” for constructing the software

Software Quality Guidelines and Attributes

- McGlaughlin [McG91] suggests three characteristics that serve as a guide for the evaluation of a good design
  - The design must implement
    - Explicit requirements of the requirements model
    - Implicit requirements of the stakeholders
  - The design must be
    - Readable
    - Understandable guide

    for coders & testers
  - The design should provide a complete picture of software with
    - Data
    - Functions
    - behavior

Quality Guidelines

- A design should exhibit an architecture that
  - has been created using recognizable architectural styles or patterns
  - is composed of components that exhibit good design characteristics
  - can be implemented in an evolutionary fashion that facilitates implementation & Testing
- A design should be modular
  - the software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of
  - Data
  - Architecture
  - Interfaces

- A design should lead to data structures that are appropriate for the classes
- A design should lead to components that exhibit independent functional characteristics
- A design should lead to interfaces to
    - reduce the complexity of connections between components and with the external environment
- A design should be derived using a repeatable method with the information obtained from requirement analysis
- A design should be represented using a notation that effectively communicates its meaning

Quality Attributes

- The FURPS quality attributes represent a target for all software design
- Functionality is assessed by evaluating the feature set and capabilities of the program
- Usability is assessed by considering
  - human factors
  - Overall Aesthetics
  - Consistency
  - Documentation
- Reliability is evaluated by measuring the

1. frequency and severity of failure
2. The accuracy of output results
3. the mean-time-to-failure (MTTF)
4. the ability to recover from failure
5. the predictability of the program

- Performance is measured by considering

  1.processing speed 2. response time

  3. resource consumption 4. throughput 5. efficiency

- Supportability combines the ability to
  - extend the program (extensibility)
  - adaptability,
  - Serviceability

  these three attributes represent a more common term (Maintainability)

- and in addition
  - Testability
  - Compatibility
  - configurability

Design Concepts

- ***set of fundamental software design concepts*** has evolved over the history of SE
- provides the software designer with a ***foundation for selecting the design methods***
- Helps you answer the following questions
  - What criteria can be used to ***partition software*** into ***individual components***?
  - How is ***function*** or ***data structure*** detail separated from a ***conceptual representation*** of the software?
  - What uniform criteria define the ***technical quality*** of a software design?

Design Concepts

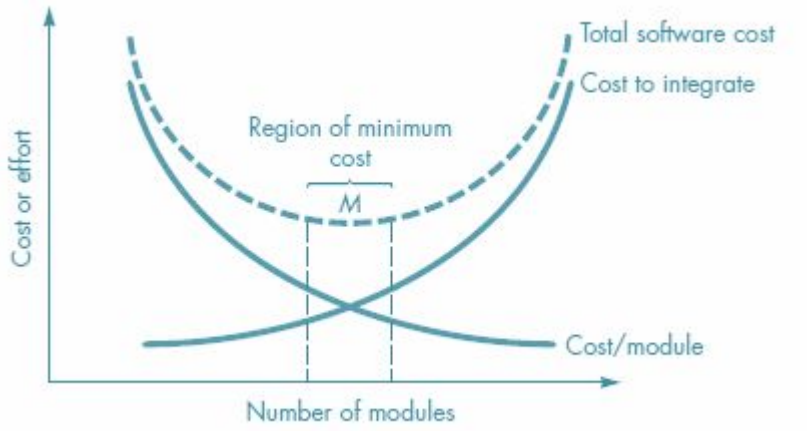
- Separation of concerns
  - modularity
    - Abstraction
      - Procedural
      - Data
    - Information hiding
    - Functional dependence
    - Aspects
    - Refinement
    - Refactoring
- Architecture

Separation of Concerns

- divide – and –conquer approach
- Subdivision of complex problem in to manageable pieces
- concern is a feature or behavior
- separating concerns takes less effort and time
- The complexity of two combined problems is greater than the sum of individual complexity
- important implications with regard to software modularity
- Separation of concerns is manifested in other related design concepts:
    - Modularity
    - Aspects
    - functional independence
    - refinement

Modularity

- Software is divided into separately named and addressable components
- modularity is the single attribute of software
- Intellectually manages software
- Monolithic software cannot be easily grasped by a software engineer
- Breaks the designs into many modules to
  - make understanding easier
  - reduce the cost required build software

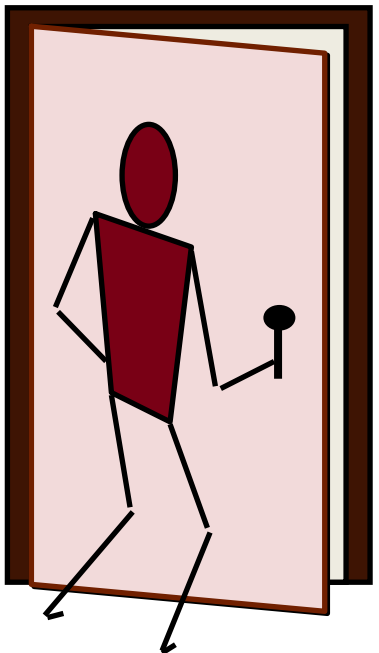


- care should be taken to stay in the vicinity of M
- Undermodularity or overmodularity should be avoided

- modularize a design so that
  - development can be more easily planned
  - software increments can be defined and delivered
  - changes can be more easily accommodated
  - testing and debugging can be conducted more efficiently
  - long-term maintenance can be conducted without serious side effects

Abstraction

- When you consider a modular solution to any problem
- ***procedural abstraction***
    - a sequence of instructions that have a specific and limited function

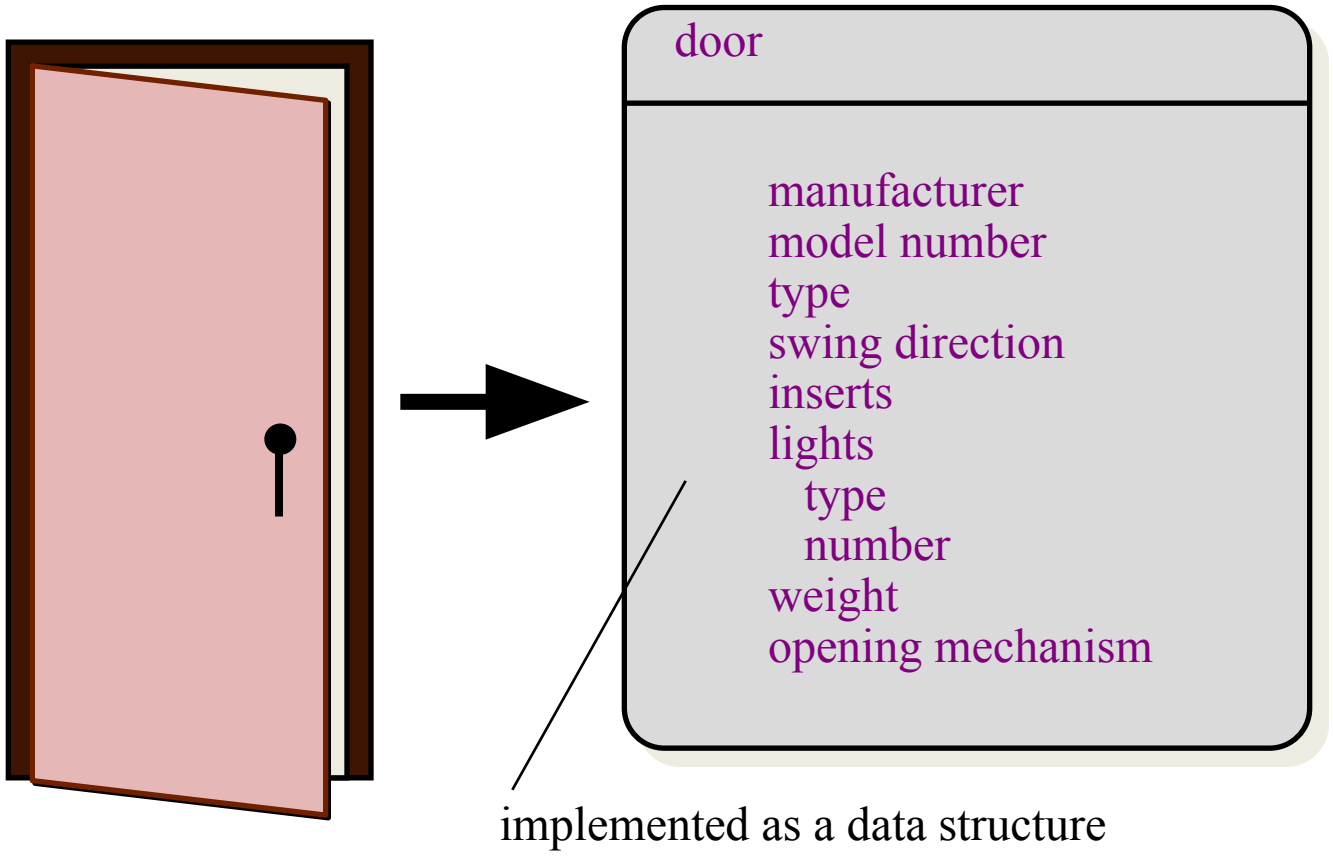


open

walk to the door
reach out and grasp knob
turn knob and pull door
step away from moving door

data abstraction

- is a named collection of data that describes a data object



Information Hiding

- How do I decompose a software solution to obtain the best set of modules?
  - characterize by design decisions that (each) hides from all others
  - The information contained within a module is inaccessible to other modules
- Facilitates effective modularity
- Enforces access constraints to both procedure and data

Functional Independence

- developing modules with
  - "singleminded" function and "aversion" to excessive interaction with other modules
- is a key to good design, and design is the key to software quality
- Independent modules are easier to maintain
- Error propagation is reduced
- Reusable modules are possible
- Independence is assessed using two qualitative criteria: cohesion and coupling

Aspects

- some concerns span the entire system and cannot be easily compartmentalized
- An aspect is a representation of a crosscutting concern
    - Requirement A crosscuts requirement B
        - B cannot be satisfied without taking A into account
- It is important to identify the aspects

For example, consider two requirements for the **SafeHomeAssured.com** WebApp. Requirement *A* is described via the **ACS-DCV** use case discussed in Chapter 6. A design refinement would focus on those modules that would enable a registered user to access video from cameras placed throughout a space. Requirement *B* is a generic security requirement that states that *a registered user must be validated prior to using* **SafeHomeAssured.com.** This requirement is applicable for all functions that are available to registered *SafeHome* users. As design refinement occurs, *A** is a design representation for requirement *A* and *B** is a design representation for requirement *B*. Therefore, *A** and *B** are representations of concerns, and *B* crosscuts A**.

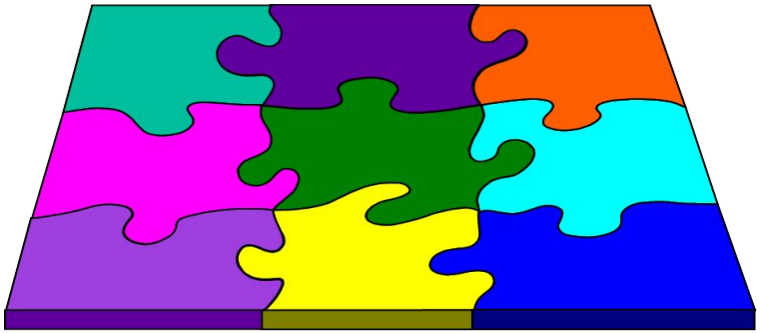
Refinement

- Stepwise refinement is a top-down design strategy
- Refinement is actually a process of elaboration
- program is developed by successively refining levels of procedural detail
- Develop macroscopic statement of function in a stepwise fashion until programming language statements are reached

Refactoring

- simplifies the design (or code) of a component without changing its function or behavior
- Without altering the external behavior of the code [design] yet improves its internal structure
- 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

Architecture



- alludes to
  - the overall structure of the software
  - conceptual integrity for a system

- architecture is
  - structure or organization of program components
  - The manner in which these components interact

Structural properties

- Defines the components of the system (modules, objects, filters)
- the manner in which those components are packaged and interact with one another

Extra-functional properties

- performance, capacity, reliability, security, adaptability, and other system characteristics

Families of related systems

- draw upon repeatable patterns that are commonly encountered in the design of families of similar systems

- architectural design can be represented using one or more of a number of different models
- ***Structural models***
  - an organized collection of program components
- ***Framework models***
  - repeatable architectural design frameworks encountered in similar types of applications
- ***Dynamic models***
  - behavioral aspects of the program architecture with regard to external events
- ***Process models***
  - business or technical process of the system
- ***Functional models***
  - functional hierarchy of a system

Patterns

- Pattern is a proven solution to a recurring problem
- Design pattern describes a design structure that solves a particular design problem
- Design pattern enables a designer to determine
  - Whether the pattern is applicable to the current work
  - Whether the pattern can be reused
  - Whether the pattern can serve as a guide similar pattern

THE DESIGN MODEL