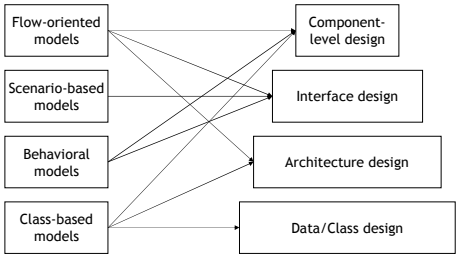
Design Engineering 1

Suradet Jitprapaikulsarn

Design

- Creative activity
- Diversification then Convergence

From Analysis to Design



Simplicity is difficult

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are obviously no deficiencies. The first method is far more difficult

C.A.R. Hoare

Design and Quality

- Explicit requirements are implemented while implicit requirements are accommodated
- Design must be comprehensible
- A complete picture—data, functional, and behavioral domains—is provided

Quality Guideline

1. Recognizable styles or patterns
2. Modularity
3. Separation of concerns
4. Coupling and cohesion
5. Complexity
6. Repeatability
7. Effective notation

Quality Attributes

- Functionality
- Usability
- Reliability
- Performance
- Supportability

Design Principle

- Avoiding “tunnel vision”
- Traceability
- Don’t reinvent the wheel
- Try to minimize the intellectual distance between the software and the problem
- Uniformity and integration
- Prepare for the change
- Degrade gently when encounter aberrant
- Design ≠ Coding
- Assessing quality while creating design
- Review, review, and review

Derived from Davis, A., *201 Principles of Software Development*, McGraw-Hill, 1995

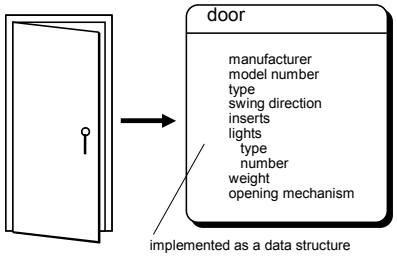
Design Concepts

- Abstraction—data, procedure, and control
- Architecture—overall structure of software
- Pattern—essence of proven solution
- Modularity—compartmentalization of data and function
- Information Hiding—know only what need to know
- Functional Independence—single minded function
- Refinement—elaboration of abstraction
- Refactoring—Rearrange for better

Abstraction

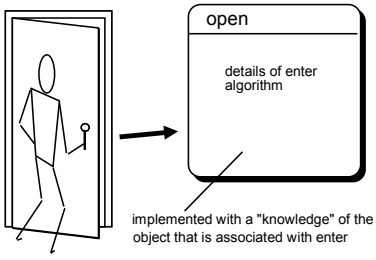
- Representing the essential features of something without including background or inessential details

Data Abstraction



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Procedural Abstraction



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Control Abstraction

- Use of control structure to hide the implementation detail
- E.g. if-else, while, for, etc.

Architecture

- **Structural models** represent architecture as an organized collection of program components
- **Framework models** identify repeatable architecture design frameworks
- **Dynamic models** indicate how the structure or system configuration may change as a function of external events
- **Process models** focus on the design of the business or technical process that the system must accommodate.
- **Functional models** represent the function hierarchy of a system

Pattern

Design Pattern Template

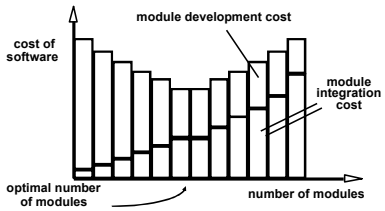
- ***Pattern name*—describes the essence of the pattern in a short but expressive name**
- ***Intent*—describes the pattern and what it does**
- ***Also-known-as*—lists any synonyms for the pattern**
- ***Motivation*—provides an example of the problem**
- ***Applicability*—notes specific design situations 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**

Modularity

- Divide and Conquer
- Allow a program to be intellectually manageable
- Dividing vs Integrating

Modularity

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



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Information Hiding

- Hiding details of data structure and/or operations behind an interface
- Users of module need not know the details of the modules

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

Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005