Design Concepts and Principles

Instructor: Dr. Jerry Gao
Design Concepts and Principles

- Software design and software engineering
- Design process
- Design principles
- Design concepts
  - Abstraction, refinement, modularity
- Software architecture
  - Control hierarchy
  - Structural partitioning
  - Data structure
  - Software procedure
  - Information hiding
- Effective modular design
  - Functional dependence, cohesion, coupling
- Design heuristics for effective modularity
- Design model
- Design documentation
Software Design and Software Engineering

Design -> The first step in the development phase for any engineered product. It serves as the foundation for all software engineering and software maintenance steps that follow.

The goal of a designer is to produce a model (or representation) of an entity that will later be built.

Input of software design: Req. analysis models and specification doc.

Output of software design: Design models and design specification doc.

Design - translates the requirements into a completed design model for a software product.
- provides the representations of software that can be assessed for quality.

Figure 13.1
Software Design

A number of design methods can be used to produce software design:
- Data design: transforms the information domain model into data structures.
- Architecture design: defines the relationship among major structural elements of the program.
- Interface design: describes how the software communicates with users.
- Procedure design: transforms structural elements of the program architecture into a procedural description of software components.

The evolution of software design:
- Modular program construction[DEN73] and top-down refining methods[WIR71].
- Structured programming [DAH71, MIL72].
- Translation of data flow/data structure into a design definition.[JAC75][WAR74].
- Object-oriented approach [JAC92][GAM95].

Common features of software design methods:
- A mechanism for translation of an analysis model into a design representation
- A notation for representing functional components and their interfaces
- Heuristics for refinement and partitioning
- Guidelines for quality assessment
Design Process

Software design --> an iterative process through which requirements are translated into a “blueprint” for constructing the software.

The design is represented at a high level of abstraction. As design iterations occur, subsequent refinement leads to design representation at much lower levels of abstraction.

Design quality is very important. Two methods are used to check the quality:
a) formal technical reviews, and b) design walkthroughs

McGlaughlin’s [McG91] three common features of a good design:
- The design must implement all of requirements (explicit/implicit)
- The design must be readable and understandable
- The design should provide a complete picture of the software in the aspects of data, functions, and behaviors.
Design Quality

To evaluate a software design, a design quality criteria can be used.

Here is the guideline for a good design:
- A design should exhibit a hierarchical organization about software
- A design should be modular based on logical partition.
- A design contains both data and procedural abstractions.
- A design should leads to modules with independent functional features.
- A design should leads to simplified interfaces between modules.
- A design should be derived using a repeatable method

Software design process encourages good design through the application of fundamental design principles, systematic methodology, and through reviews.
Design Principles

David [DAV95] suggests a set of principles for software design:

- 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 software and the problem 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.
- Design is not coding.
- The design should be assessed for quality.
- The design should reviewed to minimize conceptual errors.

External quality factors: observed by users.
Internal quality factors: important to engineers
Design Concepts

- **Abstraction:**

  Each step in the software engineering process is a refinement in the level of abstraction of the software solution.
  - Data abstractions: a named collection of data
  - Procedural abstractions:
    A named sequence of instructions in a specific function
  - Control abstractions:
    A program control mechanism without specifying internal details.

- **Refinement:** Refinement is actually a process of elaboration.

  Stepwise refinement is a top-down design strategy proposed by Niklaus [WIR71]. The architecture of a program is developed by successively refining levels of procedural detail.

  The process of program refinement is analogous to the process of refinement and partitioning that is used during requirements analysis. The major difference is in the level of implementation detail, instead of the approach.

  Abstraction and refinement are complementary concepts. Abstraction enables a designer to specify procedure and data w/o details. Refinement helps the designer to reveal low-level details.
Design Concept - Modularity

The concept of modularity has been espoused for almost four decades. Software is divided into separately named and addressable components, called modules. Meyer [MEY88] defines five criteria that enable us to evaluate a design method with respect to its ability to define an effective modular system:

- 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:
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.
Software Architecture

Software architecture is the hierarchical structure of program components and their interactions.

Shaw and Garlan [SHA95a] describe a set of properties of architecture design:

- Structural properties:
  The architecture design defines the system components and their interactions.

- Extra-functional properties:
  The architecture design should address how the design architecture achieves requirements for performance, capacity, reliability, adaptability, security.

- Families of related systems:
  The architecture design should draw upon repeatable patterns in the design of families of similar systems.

Figure 13.3.
Software Architecture

Different architectural design methods: (Figure 13.3)

- **Structural models**: represent architecture as an organized collection of components.

- **Framework models**: increase the level of design abstraction by identifying repeatable architecture design frameworks (patterns)

- **Dynamic models**: address the behavior aspects of the program architecture

- **Process models**: focus on the design of the business or technical process

- **Functional models**: can be used to represent the functional hierarchy of a system
Structural Partitioning

The program structure should be partitioned both horizontally and vertically. (Figure 13.4)

(1) Horizontal partitioning defines separate branches of the modular hierarchy for each major program function.

Simplest way is to partition a system into:
input, data transformation (processing), and output

Advantages of horizontal partition:
- easy to test, maintain, and extend
- fewer side effects in change propagation or error propagation

Disadvantage: more data to be passed across module interfaces
-> complicate the overall control of program flow

(2) Vertical partitioning suggests the control and work should be distributed top-down in program structure.

Advantages: good at dealing with changes:
- easy to maintain the changes
- reduce the change impact and propagation
Effective Modular Design

**Information hiding:**
Modules should be specified and designed so that the internal details of modules should be invisible or inaccessible to other modules.

Major benefits: reduce the change impacts in testing and maintenance

**Functional independence:**
Design modules based on independent functional features

Major benefits: effective modularity

**Cohesion:** a natural extension of the information hiding concept
A module may perform a number of tasks.

A cohesive module performs a single task in a procedure with little interactions with others.

Goal: to achieve high cohesion for modules in a system.

Different types of cohesion:
- coincidentally cohesive: a set of tasks related to each other loosely
- logical connection among processing elements --> logically cohesive:
- data sharing among processing elements --> communication cohesion
- order among processing elements --> procedural cohesion
Effective Modular Design

**Coupling:** (Figure 13.8)
A measure of interconnection among modules in a program structure.
Coupling depends on the interface complexity between modules.

**Goal:** to 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.