software engineering - design

- **design**
  - design process
    - Architectural design

      1. separate system into **sub-systems**
      2. identified and documented the relationship between subs
    - Abstract specification

      specification (services and constraints) of subs
    - Interface design

      design and documented **interfaces** between subs
    - Component design

      allocate services to subs
    - Data structure design
    - Algorithm design
  - design methods
    - modular design
      - modular programming

        break into subroutines (very similar to methods in OO design)
        - autonomous

          They do some functionality on their own
          without having to necessarily have other classes
        - coherent

          only do one particular task
          do not support diverse things
        - robust

          regardless of the context of the input data, they do not fail
    - design step 1 ----- procedure abstraction 13

      working out what all the functions and the methods are.
      Once you've got those methods and functions,you can decide which classes they belong to or which classes you need to describe them
      what the code has to do
      - functional decomposition

        break into procedures and each procedure represents distinct logical function
        - functions vs methods
          - functions do not have side effects

            每次调用函数，得到的结果都是可以预测的（根据函数本身）
            比如，调用square( ) 得到的结果都是输入的数据的平方

但是调用method不一样， 每次调用的结果不一定相同 (internal state may change)

- functional programming language

  work the same way with the same data, not causing side effects
  less efficient, not suitable for system with various states

- **Object-Oriented Design**
- step-wise refinement

  不能用于大系统， 主要是用来设计 methods
  从 high-level spec 开始， 不断地把spec 细分， 细分到每个 sub-problem 都可以被独立解决

- criteria for design methods
  - Modular **de-composability**
    - larger problem into smaller problems (sub)

      class can support a different level of function.
      sub problems can be solved independly

      - repetitive
      - top-down design

        遵循hierarchy
        working down the hierarchy level by level

  - Modular **composability (re-usability)**

    modules combined in another way to produce new system
    code reusable
    Component reuse

  - Modular **understandability**

    good abstractions and procedures
    should be understood on its own
    meaningful name -- camel case

    - no highly complexity

      如果设计的部分之间存在很复杂的relationships,那么设计就是极为复杂的

  - Modular **continuity**

    a small change in the problem specification leads to a change in just one (or a small number of) modules
    就比如，要求把整个页面从英文翻译为中文（change in specification）可以专门设计一个dictionary module，在要求改变之后，只需要改变dictionary module 就可以
    static array is not good: difficult to satisfy

  - Modular **protection**

    如果一个 module 出现了bug 那么这个module 不应该过度影响其他module

    - tight validation of data

- Repository Model

  where and how to store data
  persistent storage

  - 2 ways

- repository model

  all data store in the central database
  for example : database

  - advantage
    - efficient
    - do not need to organize the interface between the subsystems
  - disadvantage
    - cause stamp couplings

      一步错步步错，而且难以找到错误源头
    - different required security level

      如果有一些极为重要的数据，那么把这些数据单独存储会更安全
    - compromise on data type

      就比如， 有些 subsystem 专门存储 image，要和其他 subsystem（也许存储其他数据类型）共用数据库，就需要妥协
    - evolution is hard

      数据太庞大了，很难进化
- components maintain its own database

  sub-system exchange data via message passing

- design properties
  - interface

    abstraction of the module
    encapsulation: user can not see the internal detail, user can only see the interface
    in most of the cases, interface does not change
  - coupling

    a measure of the strength of the interconnections between system components.
    - loose coupling

      for example, keep all variables private or package access
      state decentralization
      use message passing and parameters to communicate
      - **components changes are unlikely to influence others**
    - tight coupling
  - cohesion

    A measure of how well a component fits together
    - levels
      - low cohesion (weak and highly undesirable)
        - **coincidental cohesion**
          - A module that only has *coincidental cohesion* is one supporting tasks that have no meaningful relationship to one another

            Miscellaneous Functions

- example

  for example:
  1. fix car
  2. calculate the cost
  3. walk dog(they do not have any relationships)

- **logical cohesion**
  - A *logically cohesive* module is one whose elements contribute to activities of the same general category in which the activity or activities to be executed are selected from outside the module.

    它的元素有助于相同的一般类别的活动，其中要执行的一个或多个活动是从模块外部选择的
    contains a number of activities of the same general kind
    - example
      - example 1 ------ transportation

        Go by Car
        Go by Train
        Go by Boat
        Go by Plane
        a person must choose a specific subset of these modes of transport. It's unlikely anyone would use them *all* on any particular journey. (都是交通工具，一个用户可以选择一个或者多个)
      - example 2 ------ math package in java
- **temporal cohesion**
  - A *temporally cohesive* module is one supporting tasks that are all **related in time.**

    modules are activated at the same time

- moderate cohesion (Acceptable)
  - **Sequential cohesion**
    - A module with (only) *procedural cohesion* is one supporting different and possibly unrelated activities, in which control passes from one activity to the next.
    - **The output for one part of a component is the input to another part**
  - **Communicational cohesion**
    - A module exhibits *communicational cohesion* if all the activities it supports **use the same input or output data - or access and modify the same part of a data structure.**

      for example
      Find Title of Book
      Find Price of Book
      Find Publisher of Book
      Find Author of Book

- high cohesion (desirable)

  work together for the OO programming
  object attributes are only modified inspected by the code that sits in the same class

  - **functional cohesion**
    - A module exhibits ''functional cohesion'' if it supports activities needed for the **execution for one and only one problem-related task.**
    - each part is necessary for the execution of a single function
  - **object cohesion**
    - each module provides functions which allow object attributes to be modified and inspected

      for example
      in a class, there are some private attributes, and inside the class, there are functions that have the ability to get or modify those private attributes

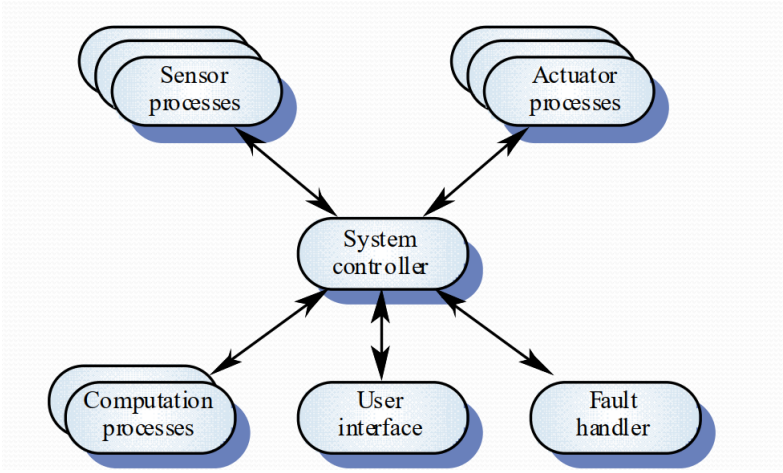
- Five Principles for Good Design
  - Linguistic modular units
  - Few interfaces
    - **Every module should communicate with as few others as possible**
    - facade structure

      如果modules 之间的 interaction 太多， codes are mess， 那么这些modules 不一定可以重复利用， 可以用package 包含他们， package 让这些modules 变得更加安全稳定
  - Small interfaces （Loose Coupling ）
    - **should exchange as little information as possible**
  - Explicit interfaces

    when we what to make modifications, we should know which classes should be checked (哪些modules 被影响了)
    - **If modules A and B communicate, this must be obvious from the documentation of A or B or both**
  - Information hiding (encapsulation)
    - **user can only see the interfaces, the internal detail of modules should be set as private**

- Architectural Design
  - Architectural Design
    - definition
      - identify the sub-systems
      - identify the framework for sub-system control and communication
    - characteristics
      - Represents the link between **specification and design processes**

- Architecture
  - definition
    - output of architectural design
- Architectural Design Process
  - System structuring
    - use **block diagram and repository module**
      - block diagram
      - repository module
    - break system into **sub-system**
      - benefits of sub-system
        - enhance reuseability
    - identify the communications between sub-systems
  - Control models
    - establish the control relationship between sub-systems
    - control flow between sub-system
      - Centralised control



一个 sub-system 掌控全局

- call-return model

  Applicable to **sequential** systems

  - top - down
- manager model

  Applicable to **concurrent** systems

  It is often used in real time systems which do not have very tight constraints

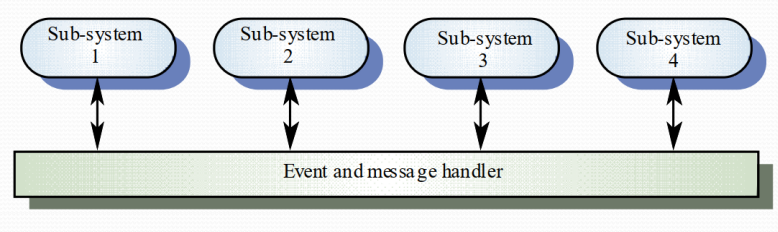
  each of these sub systems is happens all the time

  - in parallel

- Event-based (driven) control

基于事件的模型是每个子系统都可以响应来自其他子系统或系统环境的外部生成事件的模型。它是由外部生成的事件驱动的系统，其中事件的时间由处理事件的子系统控制

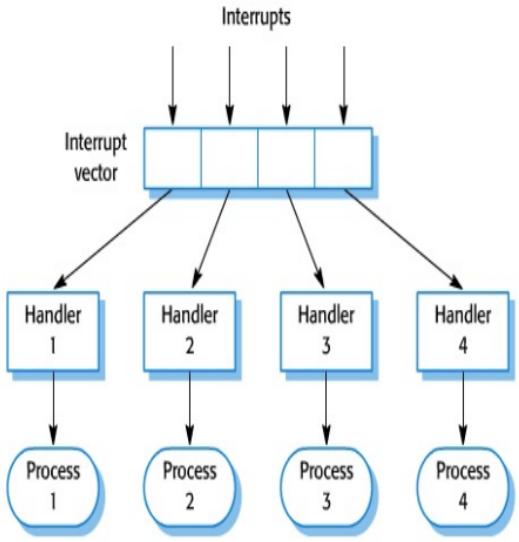
- Broadcast models



for example, library print server
you upload the files on the network (handler)
the one that is free will take the request and handle the print job
after it finishes, it will told you but will not broadcast to other sub-systems
components register an interest in specific events. When these events occur, **control is transferred to the component that can handle the event.**

  - advantage
    - simple and effective
  - disadvantage
    - sub-systems don't know if or when an event will be handled

- Interrupt-driven models



interrupts detected by handler and handler will pass to other components for processing
number of interrupt types with a handler for each type

  - advantage
    - respond very quickly

      prevent overload
      interrupt -driven routine is connected directly to CPU
  - disadvantage

- Modular decomposition
  - break sub-systems into modules

- Two modular decomposition models
  - An object model

    break into interacting loosely coupled objects
    - Object-oriented decomposition

      identifying
      - objects classes
      - attributes
      - operations
  - A data-flow model（pipeline model ）

    break into functional modules which transform inputs to outputs.
    Functional transformations process
    - for non OO languages
- `different between sub-system and modules`

  `sub-system本身就是一个系统，独立于其他sub-systems`

  `module是一个组成成分，通常不会被考虑为独立系统`

- Architectural Models

  models are abstract structure of the system
  - Static structural models

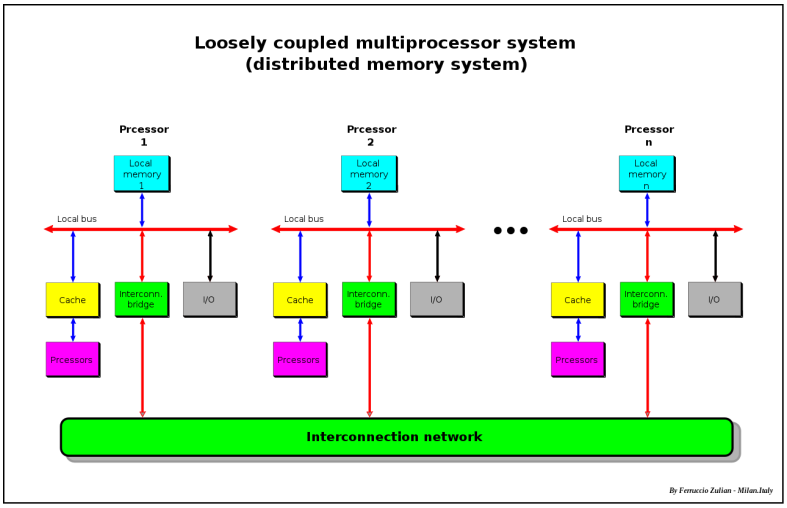
    major sub-system
  - Dynamic process models

    process structure
  - Interface models
  - Relationships models

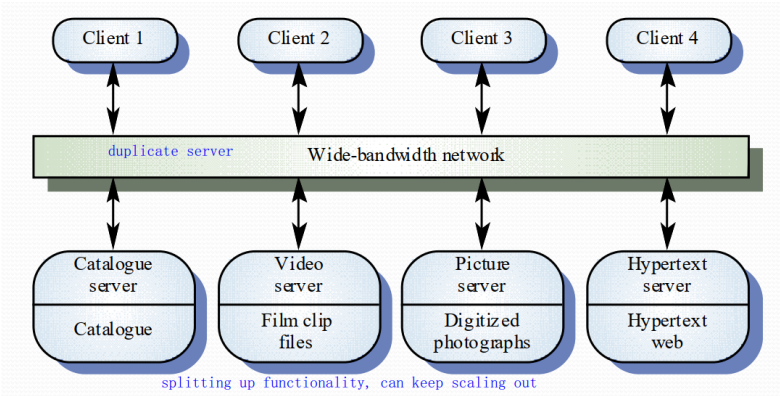
    data flow

- Distributed Systems Architectures
  - system type
    - personal system
    - embedded system
    - distributed system
      - definition of distributed system

        information processing is distributed over several computers
        more than one processor
      - characteristics
        - advantage
        - disadvantage
  - architecture



- Multiprocessor architectures
- Client-server architectures



clients and servers are treated differently

system is modelled as a set of services provided by servers to client processes

- component
  - stand-alone servers

    if we want to increase the volume, just add more servers
    - most servers are **reusable**
  - clients
  - network

    allow clients send query to servers

    allow servers send response back
- characteristics
  - advantage
    - economic

      cheaper hardware
    - effective
    - easy to modify

      add servers

      upgrade servers
    - scalability

        - disadvantage 15
- Distributed object architectures

  no distinction between clients and servers

  Any object on the system may provide and use services from other objects

以上内容整理于 幕布文档