Chapter 5 Designing the Architecture

- 关于体系结构/系统设计的说明：
- 需求定义和需求分析之后的步骤是对系统进行设计，说明软件系统是如何构造的：
  - 构建较小规模系统：可以直接从SRS（顺便提及软件部署问题—例如数据挖掘子系统可以直接部署到数据服务器上）简单进入到类设计、数据结构和算法设计，进而实现该系统。例：从DFD图的教材购销需求进入较详细设计。
  - 构建中大规模系统：在关注数据或代码的细节之前，则希望将系统分解为规模可管理的单元（诸如子系统或模块）,也考虑硬件支持，于是提出了体系结构设计问题。
- 软件体系结构：研究如何针对系统进行合理分解。

Chapter 5 Designing the Architecture

5.1 The Design Process （设计过程）

1. Introduction

①**design** : problem description $\xrightarrow[\text{process}]{\text{creative}}$ solution (P223)

(requirements(SRS))

（将需求中的问题描述转变成软件解决方案的创造性过程）

Chapter 5 Designing the Architecture

A: 软件设计是一种具有智力挑战性的任务。要考虑到软件系统可能遇到的所有情况，包括可预料的系统功能，还有各种非功能目标（除了质量需求之外，还有设计阶段要考虑的易维护、易扩展、易使用、可移植性等诸多单纯的设计目标）。

B: 不存在可以遵循的手册和可以完全套用的办法，要给出完全满足需求的设计，需要创造性、聪明才华、经验以及专业判断力。

C: 有经验的软件开发人员可以借鉴现有的各种方案或原理。

（图**5-2** 设计建议的来源）

体系结构（**Architecture**）:一种软件解决方案，用于解释如何将系统分解为单元，以及单元如何相互关联，还包括这些单元的所有外部特性。

设计模式（**Design pattern**）：一种针对单个软件模块或少量模块而给出的一般性解决方案，它提供较低层次的设计决策。(此设计决策低于体系结构) (注: 此处为说明,不是定义)

设计公约（**Design Convention**）：一系列设计决策和建议的集合，用于提高系统某方面的设计质量。

（例如：敏捷方法中的很多原则就属于设计公约，但不太成熟。）

当一种设计公约发展成熟时，将会被封装成设计模式或体系结构风格，最后可能被内嵌封入程序语言结构。例如：对象、模板等都是编程语言支持的原设计和编程公约）

Chapter 5 Designing the System

② example:

house building

A: one problem/requirement ⟷ several rough solutions

⟶ one real solution

B: one requirement ⟶ several prototypes

⟶ one design(solution)

③ notes:

A: in SE { problem ⟷ <SRS>
solution ⟷ practical software description

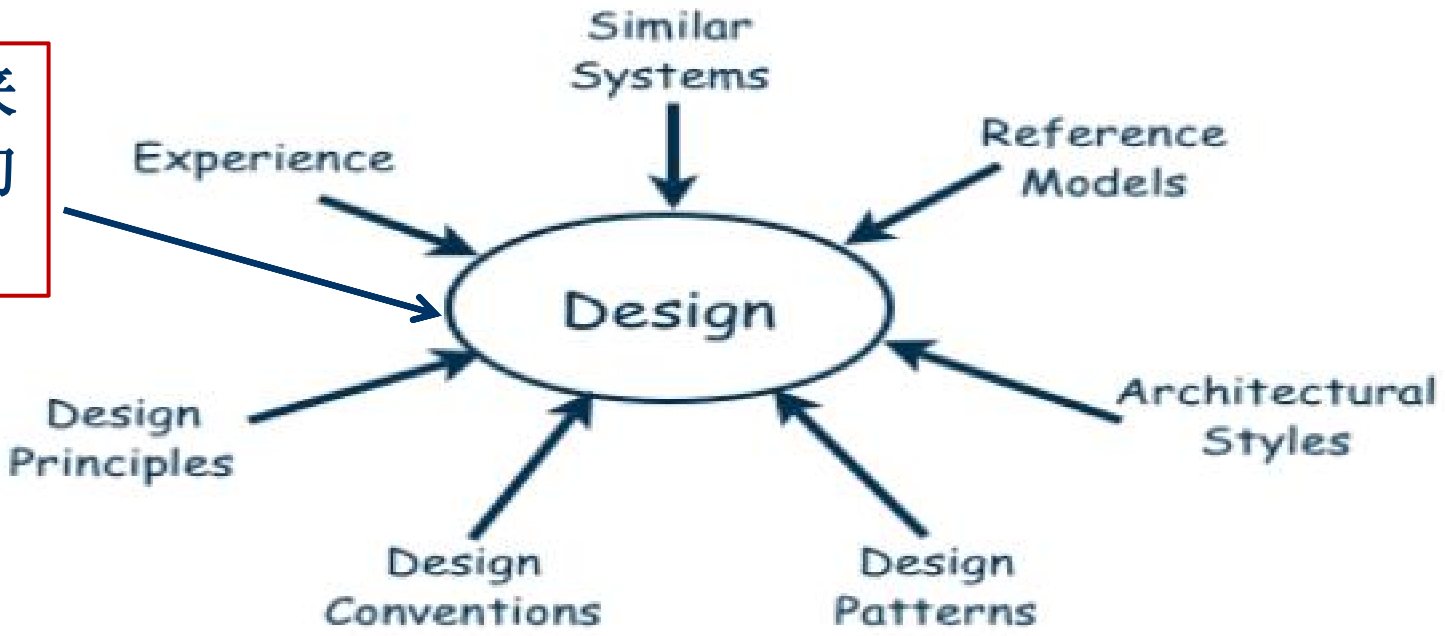
B: requirement changes（需求变更） will happen in software design process or latter processes

其频繁程度比建造房屋要大得多

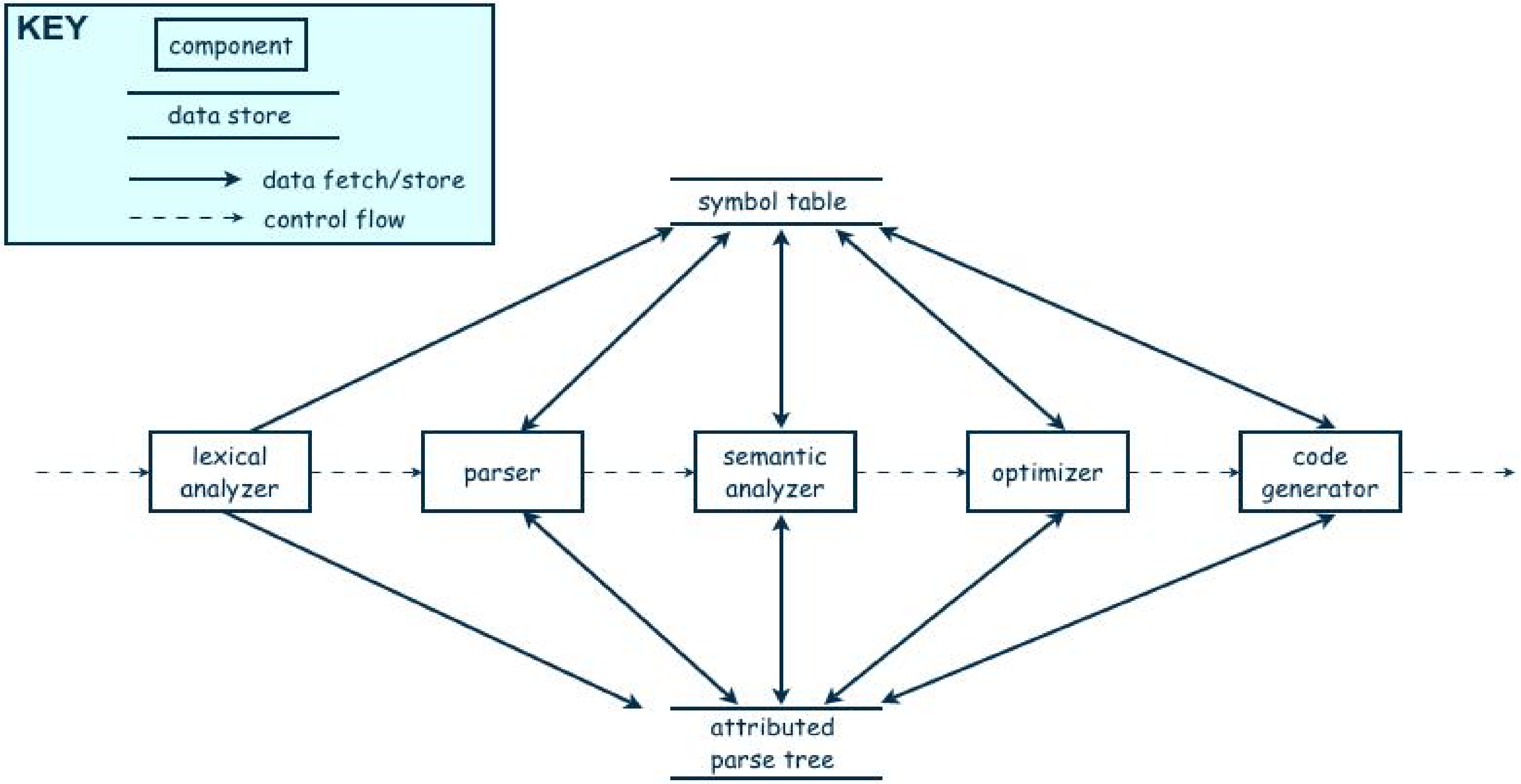
Design is a Creative Process (continued)

- Many ways to leverage existing solutions
  - Cloning: Borrow design/code in its entirety, with minor adjustments（如下编译器设计就可以是克隆级别的设计）
  - Reference models: Generic architecture that suggests how to decompose the system (Fig 5-3：编译器的设计参考模型)

大部分来自客户的建议



- Reference model for a compiler



Chapter 5 Designing the System

2. Conceptual and Technical Designs

（概念设计与技术设计）（关于设计的非主流解释）

① purpose: (two part iterative process)

A: conceptual design(≈system design)—satisfy customer

B: technical design –satisfy the system builders on our development team
(≈ program design)

② definition:

A: conceptual design: Tells the customer what the system will do (software architecture and function)

B: technical design: Tells the programmers how the system will do----software function and interface's implementation method (coder's reference document)

Chapter 5 Designing the System

③answering questions by conceptual design

1—6(see P225)

④characteristics of good conceptual design

1—5(see P226)

⑤contents by technical design (algorithm,etc.) (P226)

A: major hardware components and their function

B: hierarchy and function of software components

C: data structures and data flow

⑥note: (P225)

$$\left\{\begin{array}{l}\text{conceptual design}\\ \text{technical design}\end{array}\right. \xrightarrow[\text{are sophisticated}]{\text{when customers}} \text{combined}$$

Chapter 5 Designing the System

3. Design Process Model (体系结构设计过程模型)

A: Modeling (初始建模): 尝试可能的分解：根据需求描述的系统的关键特性等确定软件体系结构风格，进行系统级别的决策。(比如：MVC模式就不适合路由选择服务程序的设计，因为事件触发引起的服务程序指向很明确，没有太多的逻辑处理。其对于时间驱动的周期性任务也不适合。)

例如：信贷系统将把初级审核业务放在客户端。以及由此带来的数据迁移问题等

B: Analysis(分析): 关注软件系统的功能及质量属性（性能、安全性、可靠性等）、各种约束等等。(关注系统级别决策)

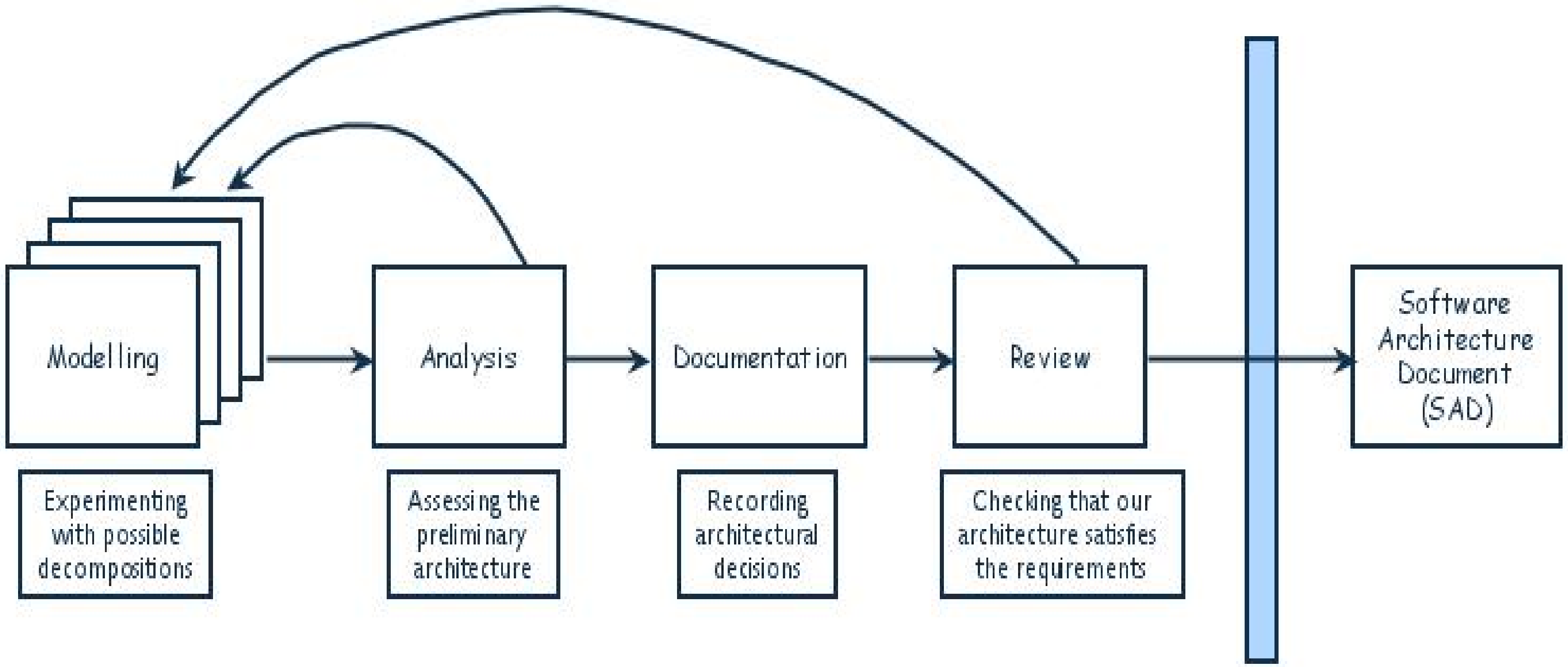
C: Documentation(文档化): 确定各个不同的模型视图。

D: Review(复审): 检查文档是否满足所有功能及质量需求。

E: final output: Software Architecture Document

(正式的 <SAD>：软件体系结构文档)

- Designing software system is an iterative process
- The final outcome is the software architecture document (SAD) (FIG5.4)


Modelling
Analysis
Documentation
Review
Software Architecture Document (SAD)
Experimenting with possible decompositions
Assessing the preliminary architecture
Recording architectural decisions
Checking that our architecture satisfies the requirements

Chapter 5 Designing the System

5.3 Decomposition and Modularity(conceptual design)

（系统设计中的分解与模块化）

1. Six ways to create designs（六大类设计方法）

① **Functional decomposition** : function module dividing
(example: layered DFD→SC diagram)

② Feature-oriented design : Function Feature implement.

③ **Data-oriented decomposition** : enact data structure
(example: Jackson approach)

④ **Process(进程)-oriented decomposition** ： confirm parallel processes .

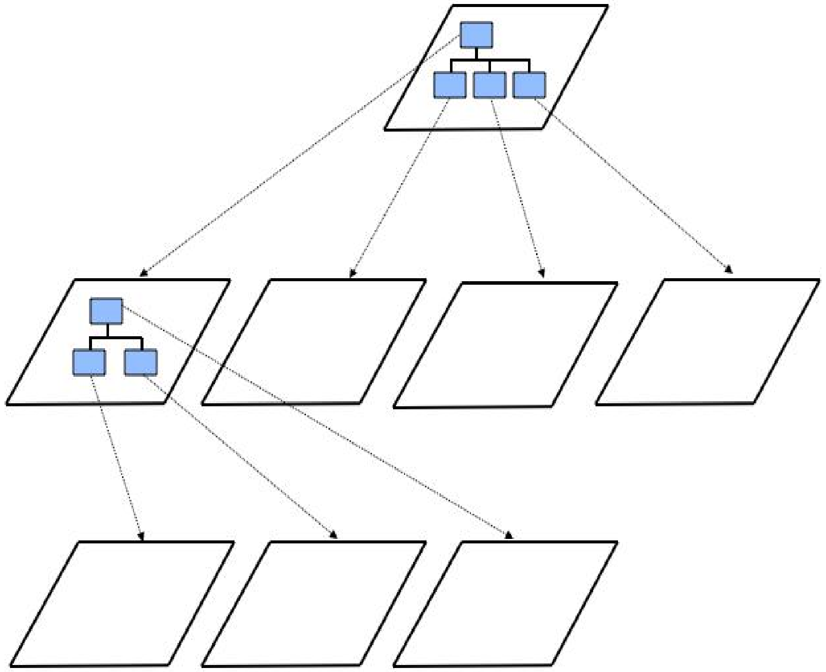
⑤ Event-oriented decomposition (Event=condition).

⑥ Object-oriented design: assign objects to modules (identify classes and relations)

2. Explain

① six ways ---- all have hierarchy (fig5.5)and units.

② example ---- data-oriented decomposition (including "deal with" different data in different layers)

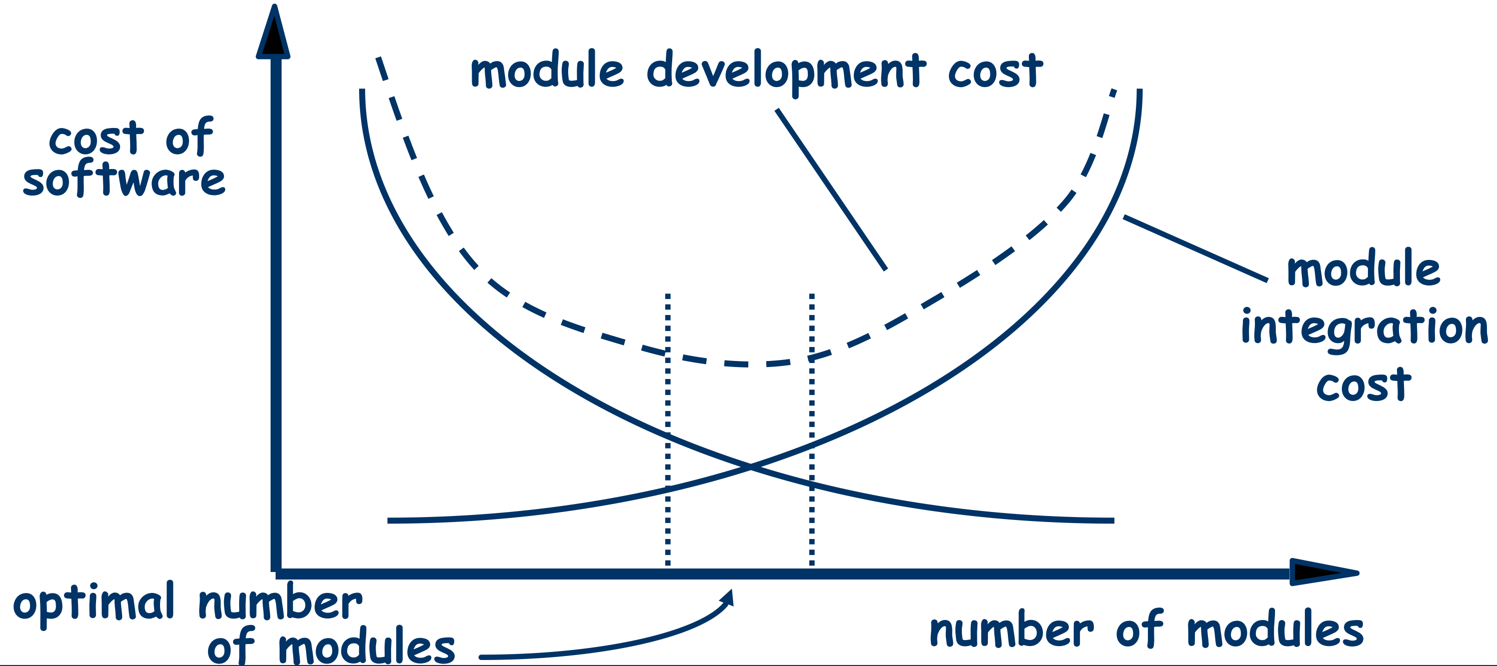


Modular : when each activity of the system is performed by exactly one component, and when inputs and outputs of each components are well-defined, a system is modular.

$C(x)$: complexity of problem x; $E(x)$: effort to solve x

If $C(p1)>C(p2)$ then $E(p1)>E(p2)$

$$C(p1+p2)>C(p1)+C(p2) \Longrightarrow E(p1+p2)>E(p1)+E(p2)$$



3. 体系结构视图 （大部分可以用UML来描述）

A: 分解视图 ----分层细化概念

B: 依赖视图 ----依赖关系

C: 泛化视图 ----泛化关系

D: 执行视图 ----传统的执行体的次序与输入输出关系

E: 实现视图 ----代码单元和源文件之间的传统映射对应关系

F: 部署视图 ----节点的计算能力与节点资源的展示

G: 工作分配视图 ----任务分配管理展示

Chapter 5 Designing the System

5.4 Architecture Styles and Strategies

（软件结构风格和设计策略）

1. Three design levels （三种设计层次）(P229)

①**architecture design**----associates the system
（体系结构设计） capabilities(in SRS) with the system components that will implement them
（由软件需求中的系统能力与系统部件关联起来而得到软件整体结构的过程）

note: architecture design≈system design (in chapter 1)

architecture(体系结构)：一种软件设计方案，说明如何将系统分解为单元以及这些单元的关联方式，以及所有单元的外部可见特性。

Chapter 5 Designing the System

②**code design**----involves algorithms and data
（代码设计）　structures for each component
（各个部件（模块）的算法、数据结构的设计）

components = programming primitives
+primitive operators
+composition mechanisms)

③**executable design**----lowest level of design, including
（运行设计）　memory allocation, data formats,
bit patterns
（最底层设计—内存分配、数据格式、位模式、相关显示匹配等等）。

Chapter 5 Designing the System

④note

A: top down work----designing an architecture, then the code design, and finally the executable design.

B: code design+executable design ≈ program design

C: repeatability (taking modification to architecture, code design，executable design several times, as the designers’s understanding and creativity permit)

D: <u>design style</u>----includes component, connector, constraints on combining components.

体系结构风格: 大规模系统结构模式, 包括规则、元素、技术。（是关于如何完成一般性设计所给予的建议，并不是完整的、细节化的解决方案）

Chapter 5 Designing the System

2. Several (system) design styles（几种体系结构风格）

① pipes and filters（管道与过滤器）（**5.4.1**）

(一种组件式设计,常用于语义分析、字典分析等设计)

A: pipe----streams of data for input and output

B: filter----a component finishing reading the input stream （数据转换构件）

② object-oriented design （面向对象设计）

用于一般软件设计

A: OOA--get abstract data type(classes)

B: OOD--get instance(object) and its detailed definition

characteristics----integrity of the data representation

----data representation must be hidden from other objects (encapsulation)

OO
观点:
软件
等于
一系
列对
象的
演化

Chapter 5 Designing the System

③ implicit invocation （隐含调用）

用于分组交换网络软件设计等等

focus on: event-driven, based on the notion of event broadcasting/announcement

application---- packet-switched networks

---- user interfaces (to separate the presentation of data from applications that manage the data)

类似于C/S结构的软件设计等

④ layering （分层设计）

A: meaning X: hierarchical structure

Y: each layer provides service to the one outside

Chapter 5 Designing the System

B: example1 ： file security system

----(1)加密解密算法,(2)文件加密解密接口,(3)密钥管理: 文件签名,检验签名,获得文件访问等. (4) 认证:加密文件的管理, 用户标识和密码的管理, 认证协议实现等.

example2： ----Fig5-9（OSI开放互联参考模型）

C: feature: take advantage of the notion of abstraction

⑤repositories(信息库/资源仓库)

(常用于信号处理、知识发现、模式识别系统等设计)

A:construction: a central data store and a collection of components (Fig5-8)

B: black board system (one of repositories)

--includes the black board itself, knowledge sources and the control. When the state of black board changes, the knowledge source will responds.

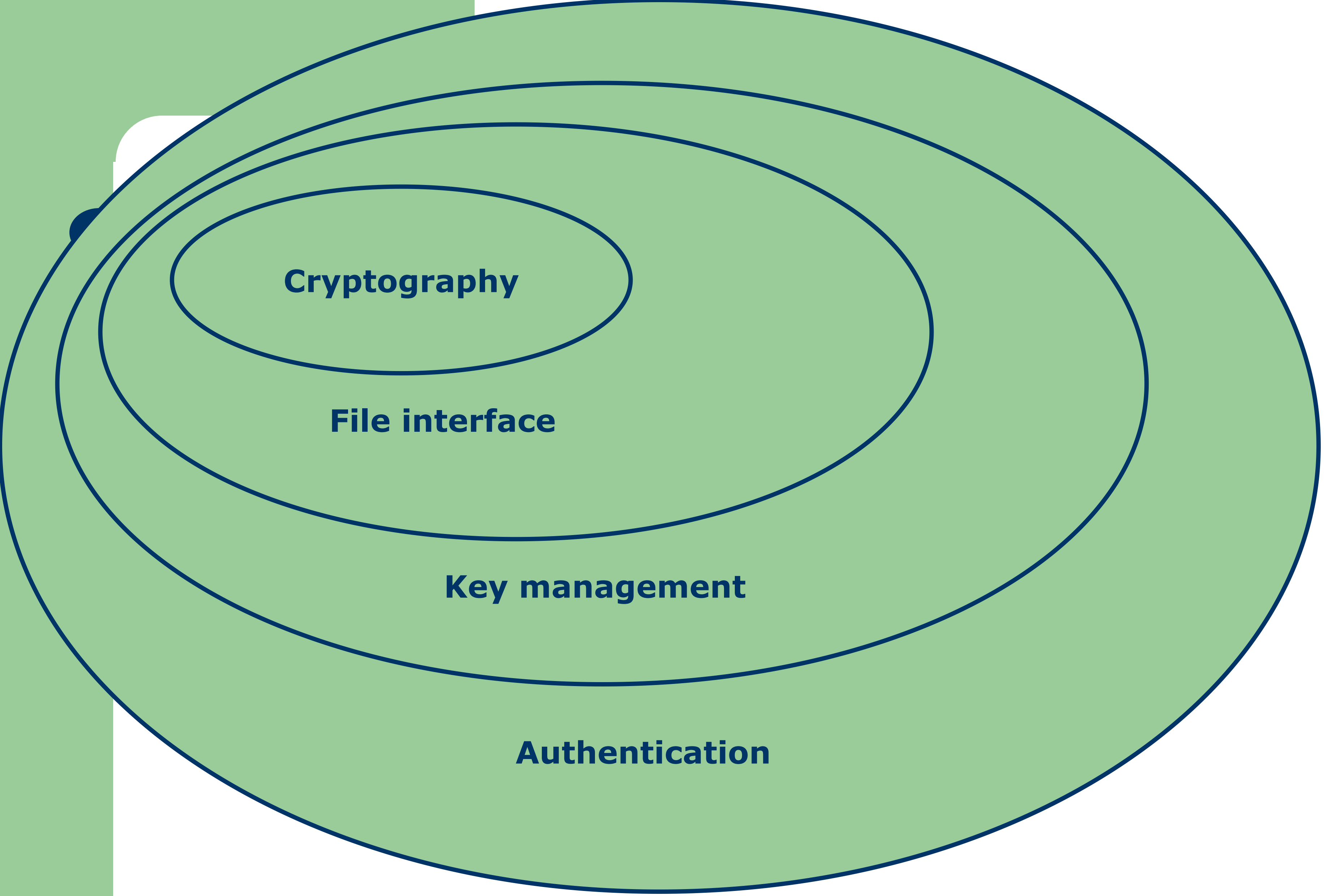


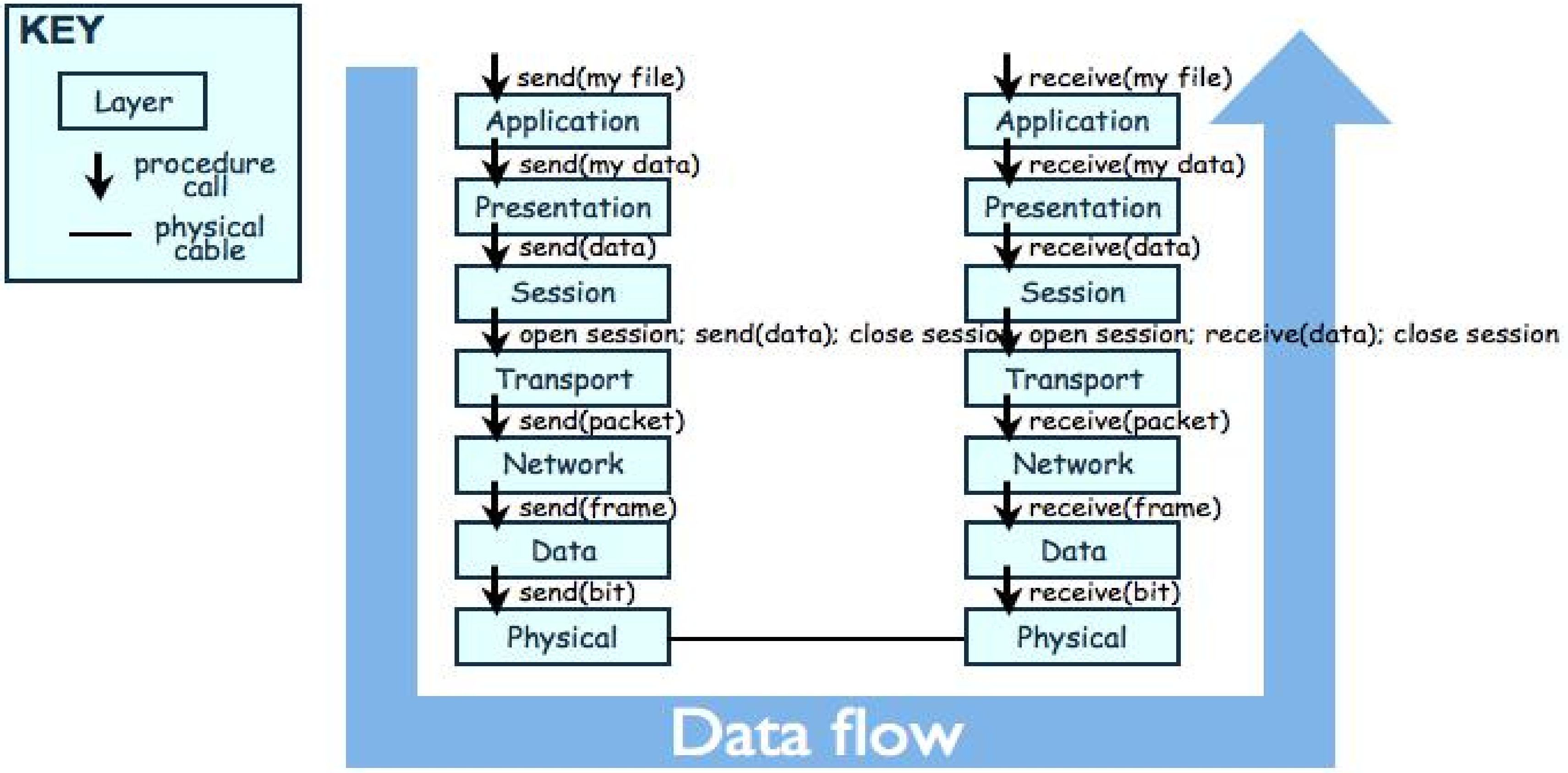
Fig 5.6 Layered security architecture

Layering

- Layers are hierarchical
  - **Each layer provides service to the one outside it and acts as a client to the layer inside it**
  - Layer bridging: allowing a layer to access the services of layers below its lower neighbor
- The design includes protocols
  - Explain how each pair of layers will interact
- Advantages
  - **High levels of abstraction**
  - **Relatively easy to add and modify a layer（例如数据链路层的LLC之下的MAC层802.3至802.12等的添加）**
- Disadvantages
  - Not always easy to structure system layers
  - System performance may suffer from the extra coordination among layers

Example of Layering System

- The OSI Model



Chapter 5 Designing the System

⑥ interpreters (用于解释器专用设计)

⑦ process control（过程控制）

用于工业过程控制的软件设计

A: explain—maintain specified properties of process output at or near specified reference values

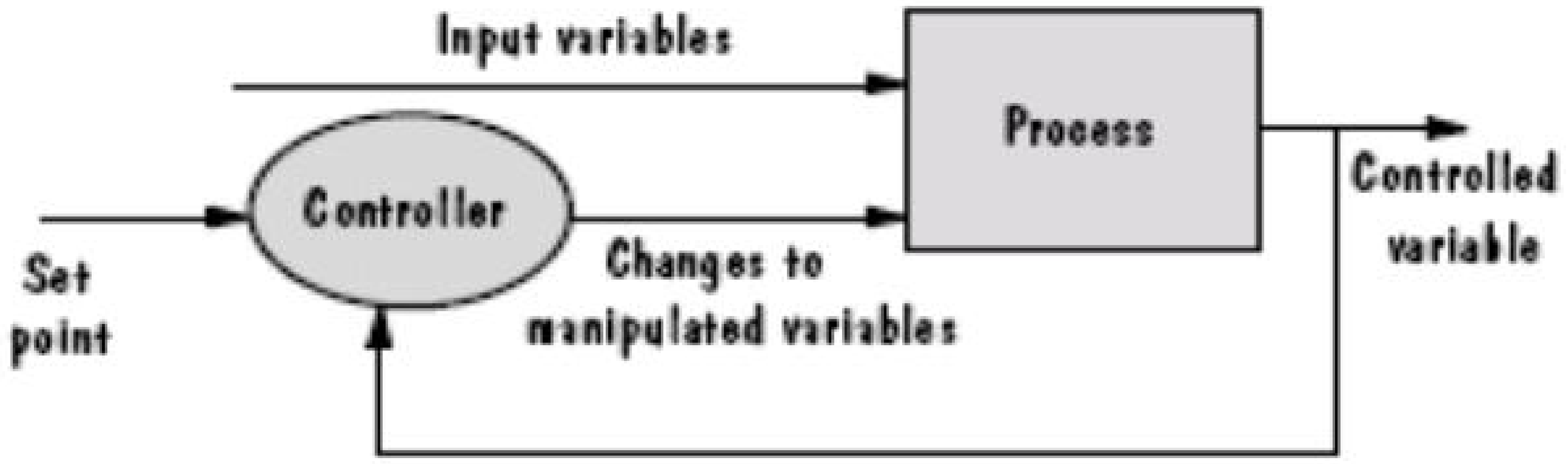
B: two types X: feedback
Y: feedforward

⑧ other styles

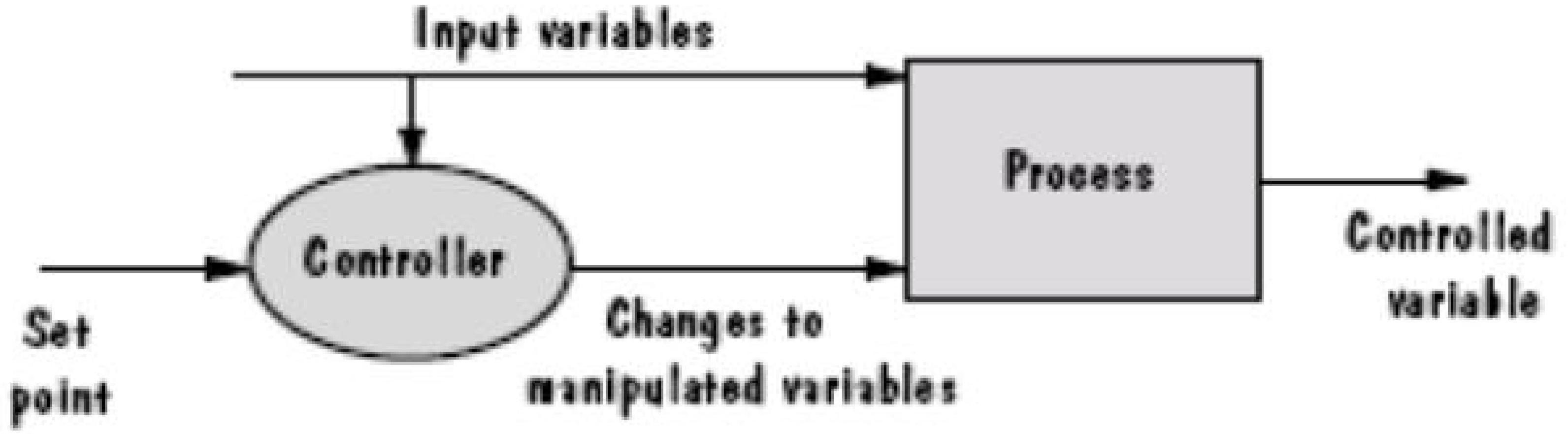
A: distributed system architecture

B: client-server architecture or B/S architecture

FEEDBACK LOOP:



FEEDFORWARD LOOP:



Chapter 5 Designing the System

#9: peer-to peer,P2P （对等网络---延伸到类似AI系统）

A: meaning X: every component-----client +server

Y: the interface-----provide service, and can receive service

B: useness: X: self-organizing (rapidly.)

Y: scale extending (is easy.)

C: example : ----file sharing network.

----战地无线组网的快速构建，无人机集群。（实时系统—门槛防渗透、单兵能耗控制等）

----Napster: 音乐共享系统（巧妙之处：没有中央文件服务器，各个节点之间可以直接传送）

Chapter 5 Designing the System

5.5 Issues in Design Creation(设计过程中的若干问题)

1. Several Conceptions(涉及质量属性的几个概念)

A: Modifiability (可修改性)---- 更改软件系统的难易程度。

-------设计时的诸多特性：预测预期改变、高内聚、低耦合、接口设计多重性（分层、新接口的可增加性）等等。

-------自我管理软件：检测环境改变进而做出适当反应。

B: Maintenability (可维护性)----是指理解、改正、改动、改进软件的难易程度

-------改正性维护

-------适应性维护

-------完善性维护

-------预防性维护（包括增加自我管理能力等）。

Chapter 5 Designing the System

C: Performance (性能)----软件的性能是指在完成软件功能时展示出来的及时性、吞吐量等特性。

-------响应时间。（请求的反映时间）。

-------吞吐量（处理请求的速度）。

-------负载量：并发用户数。

提高性能的策略：------提高资源利用率。（从规划的层面上增加软件并行程度；分布式数据共享、云共享等等）

-------有效管理资源分配。（先到先服务；时限长段优先等等）

-------设计适应性能要求的拓扑结构。或者允许有限变更等等。

D: Security (安全性)----实现软件各种安全保密措施的有效性度量。

------免疫力：改动体系结构以保证系统安全特征，最小化系统安全漏洞。

------弹性：功能分段使攻击影响最小化，快速恢复能力。

------其他重要措施：权限划分处理、软件加密分析、硬加密措施分析、预警或查杀木马等。

------P2P网络安全性问题。

E: Reliability (可靠性)----软件产品在假设的环境下，按照规定的条件和规定的时间区间完成规定功能的能力。

------主动故障检测：周期性检查或预测系统故障，如无法提供服务，提供错误服务，数据损坏，死锁等等。

------故障恢复：实时处理故障的能力，如撤销事务，回退，备份，服务降级，实时修正，实时报告等等。

F: Robust(健壮性)----系统在不正确的输入或意外的环境条件下依然保持正确工作的能力。

问题：我们设计的系统是安全的吗？存在哪些安全风险？（以后再探讨）（智能驾驶：电池蓄电量提醒问题）

G: usability(易使用性)----用户能够操作软件系统的容易程度

------用户界面放在次体系结构中，便于用户定制。

------有的用户命令需要体系结构支持。如撤销命令（维护一个过去状态链表），展示多重视图等等。

H: 商业目标----涉及购买与开发，最初的开发成本与维护的成本，新的技术与已知技术到来的副作用等。

Chapter 5 Designing the System

当探讨或分析两个模块共享某数据时,模块各自的私有细节应隐藏

and levels of Abstraction

象的层次）

模块化是优秀设计的特征之一

design(<u>modularity</u>)---- 模块有清晰的输入

和输出, 设计目的明确, 功能独立, 可以做独立检测。

B: <u>abstraction</u>(<u>levels of abstraction</u>)—hiding details

（对细节的隐藏称为抽象）

C: 抽象的层次----设计师的考虑：设计从抽象开始!

----hierarchy (of components)---- 模块/部件都以某种不同层次结构的抽象的形式出现, 越上层、越早期的模块层次或框架是越抽象的设计。

设计师的标准

Chapter 5 Designing the System

② advantage (of combining modular components with abstraction hierarchy)

A: top level give a view to solution(by hiding details), other levels show major functions and implementation details

B: OOD—(ADT: high level objects and relationships, not care the instance designing)

C: the ability to design different components in different ways

（例如：可以用OO和原型化的方法设计用户界面，而在考虑安全性设计时使用状态图等）

Chapter 5 Designing the System

③ example of using abstraction – sidebar 5.2

（具体使用抽象的概念时，其无处不在！）

A: sorting a L(list) (see next page)

(1): 三个抽象层次各有用处;

----若只关心L在排序前后的样子, 仅需第一层次.

----若关心算法的速度, 第二层次足亦.

----若要考虑编写源码, 则需要第三层次.

(2): 三个层次的连续表达不可或缺

----三种不同的抽象形式

----完整文档供不同开发阶段的不同使用者参考.

B: using abstraction in OOD –--stack: ADT(P239)

Chapter 5 Designing the System

Rearrange L in non-decreasing order

```
DO WHILE I is between 1 and (length of L)-1:
  Set LOW to index of smallest value in L(I), ..., L(length of L)
  Interchange L(I) and L(LOW)
END DO
```

```
DO WHILE I  is between 1 and (length of L)-1
  Set LOW to current value of I
    DO WHILE J is between I+1 and (length of L)-1:
      IF L(LOW) is greater than L(J)
        THEN set LOW to current value of J
      ENDIF
    ENDDO
  Set TEMP to L(LOW)
  Set L(LOW) to L(I)
  Set L(I) to TEMP
ENDDO
```

Chapter 5 Designing the System

3. Collaborative design（设计中的协作关系）

① several tasks (for developing team)(group behavior)

（关于开发团队的几个任务）

A: personnel choice（人员选择与分工）

（谁最适合设计系统的某个部分？！）

B: problem expression and document organization

（问题的表达语言和文档组织形式）（可以自成一体）

（目标：每个团队成员都能理解他人的设计！）

C: coordinating the design components

（设计的部件之间的协调与有效交互----以达到有效运行）

Chapter 5 Designing the System

② **the cause of design breakdown—siderbar5.7**

设计过程崩溃的主要类型（**------**小组练习项目之通病）

A: 缺少具体的设计方案。

B: 缺少设计过程的元方案。（最高抽象模型**----**顶层设计）

C: 问题和解决方案的优先级选择不合理。

D: 约束不明。

E: 没有可验证的模拟设计。（必要的原型系统）

F: 进度难以跟踪。

G: 解决方法不够完整。（不完整到一定程度，设计崩溃）

Chapter 5 Designing the System

③ major problems in performing Collaborative design

A: addressing differences in personal experience, understanding, and preference .

(解决个人经验,理解能力和爱好等方面的差异,妥善搭配)

B: people behave difference between doing in groups from do individually

(解决人们在团队的行为方式与单独的行为方式的不同)

(日本的团队, 美国团队, 混合团队)

④ design is a collaborative and iterative process

(not just building a product, also building a shared understanding of ……P241)

Outsourcing（外包式协作设计）

- Coordination becomes increasing difficult
- Collaborative team may be distributed around the world
- Four stages in distributed development(分布式开发)
  - **Project performed at single site with on-site developers from foreign countries**
  - **On-site analysts determine system requirements, which are in turn provided to off-site groups**
  - **Off-site developers build generic products and components that are used worldwide**
  - **Off-site developers build products that take advantage of their individual areas of expertise**

Outsourcing（外 包）

- 外包式协作设计要注意的问题：
  - 工程设计次序有新的要求：设计师反复与需求分析人员、测试人员、编程人员交流。
  - 时差与不稳定的网络。
  - 不了解当地的商业规则、客户、法律。
  - 多种语言交流的难度问题。

Chapter 5 Designing the System

4. Designing the User Interface

（设计用户界面要注意的问题）

note: User interface can be tricky things to design,
because difference people have different styles
of perceiving, understanding, and working .

①key elements(设计界面要注意解决的要素)

----metaphors(寓意/比喻) （基本术语、图像和概念等）

----mental model(思维模型)（数据、功能、任务的组织与表示）

----navigation rules for the model(模型的导航规则)

（如何在数据、功能、活动和角色中移动及切换）（京东购物经历）

----look (外观)

----feel(感觉)

Chapter 5 Designing the System

② cultural issues（文化差异问题）

issues ----we must consider the user’s beliefs, values, norms, traditions, mores, and myths.（信仰，价值观，道德规范，传统，风俗，传说）

solution ---- international design/Bias-free design (it is best for the interface to be tested in the culture in which it will be used)

---- tailoring the interface

③ user preferences（用户爱好问题）

– making alternative interfaces for selecting by those who have different preferences

- 补充1：用户界面设计之重要性：
  - 用户界面设计是否成功将直接影响着系统的质量，并最终影响着用户对系统的满意程度
  - 用户界面的好坏直接影响软件的价值
  - 图形用户界面的设计要求以用户为中心，应该使用用户术语实现与用户的交互 。

- 补充2：界面设计之原则
  - 一致性原则
  - 能够及时提供信息反馈
  - 合理布局，保持界面的简洁
  - 合理利用颜色
  - 对用户出错的宽容性
  - 减少重复的输入
  - 支持快捷方式的使用
  - 尽量减少对用户记忆的要求 （逐次回退等）
  - 快速的系统响应
  - 符合用户的工作环境与工作习惯
  - 用户联机支持

Chapter 5 Designing the System

5. Concurrency (并发性)

note: in many systems, actions must take place concurrently rather than sequentially

① synchronization(同步) ---- allowing two activities to take place concurrently without their interfering with one another使两个活动同时进行但互不干涉的方法.

② mutual exclusion(互斥) ---- making sure that when one process is accessing a data element, no other process can affect(改动) that element（大公司的缓存方案）

③ examples A: “can’t synchronization” (P245-246)

B: “mutual exclusion” (P246)

C: “in network---locking shared data”

Chapter 5 Designing the System

6. Design patterns and reuse (X)

① design patterns -- the key aspects of a common design structure ,including objects and instances, roles and collaborations, distributions of responsibility

② reuse – reuse the design patterns

Chapter 5 Designing the System

5.6 Characteristics of good design （含§6.2节）

（优秀软件设计所具备的特征）

high-quality design should have such characteristics:

- ease of understanding
- ease of implementation
- ease of testing
- ease of modification ---is especially important
- correct translation from the requirement

1. Component Independence（部件/模块独立性）

Focus : degree of component independency

---- decide by two concepts :
- cohesion（内聚）
- coupling（耦合）

Chapter 5 Designing the System

① coupling : inter-dependency among software components（两个软件部件之间的相互关联程度）

A: highly coupled,loosely coupled, uncoupled(Fig6.1)

B: examples (P249)—the ways of components depend on each other. The measured degrees see Fig6.2

C: purpose for studying coupling
----minimize the dependence among modules (P297)

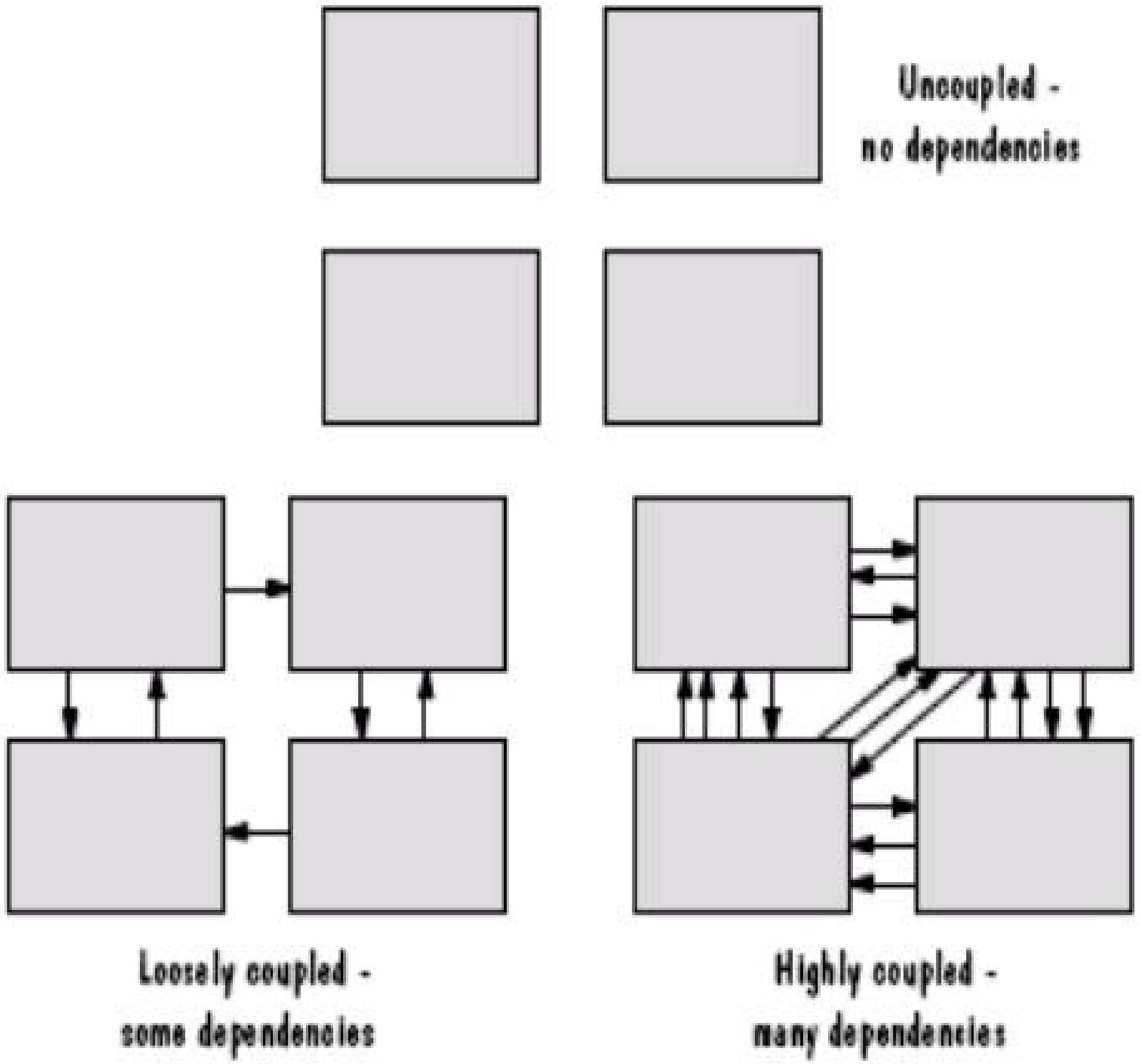
D: six levels(of coupling measures)

uncoupled（非直接耦合）：模块相互之间没有信息传递

data coupling（数据耦合）：模块间传递的是数据

stamp coupling（特征耦合）：模块间传递的是数据结构

control coupling（控制耦合）：模块间传递的是控制量


Uncoupled - no dependencies
Loosely coupled - some dependencies
Highly coupled - many dependencies

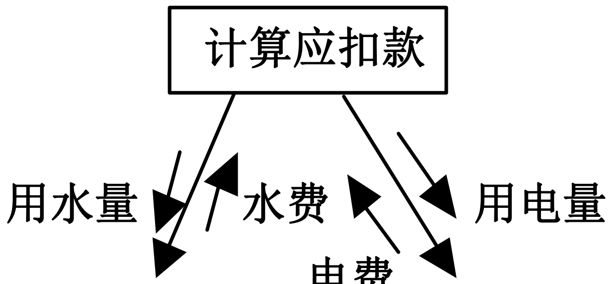


图 3.6 数据耦合

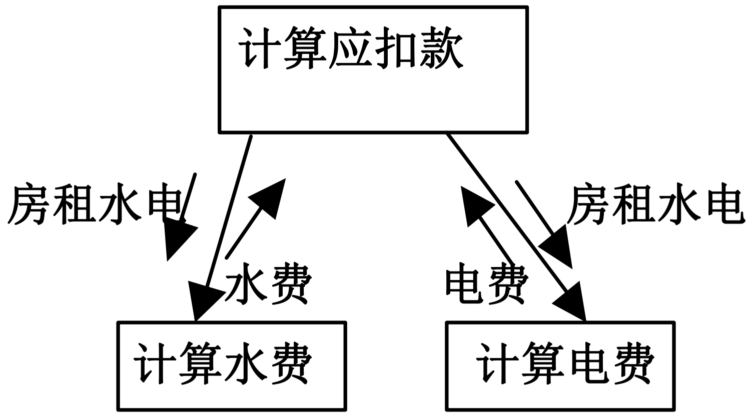
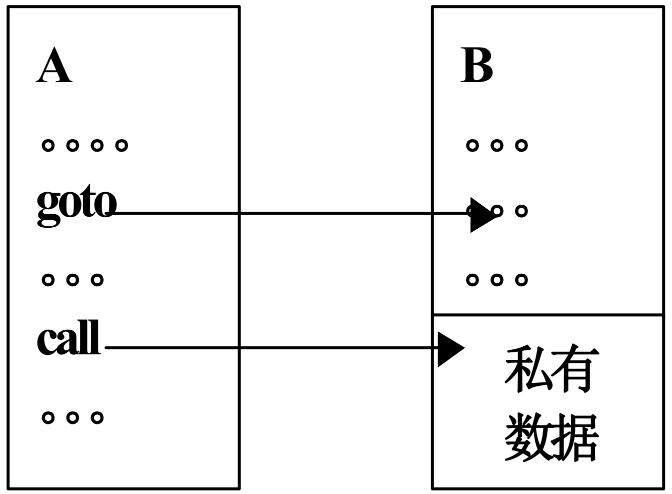


图 3.7 特征耦合

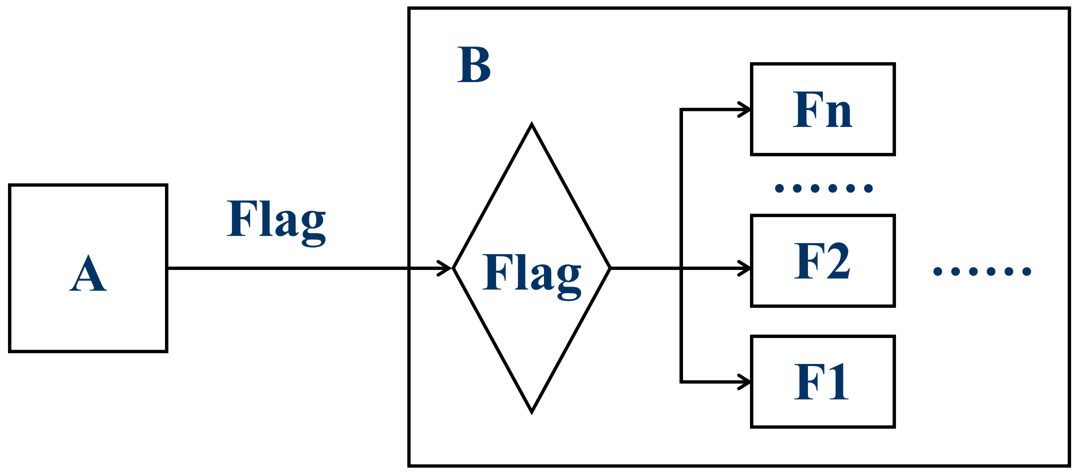
☺ The most desirable.

房租水电=房租十用水量＋用电量

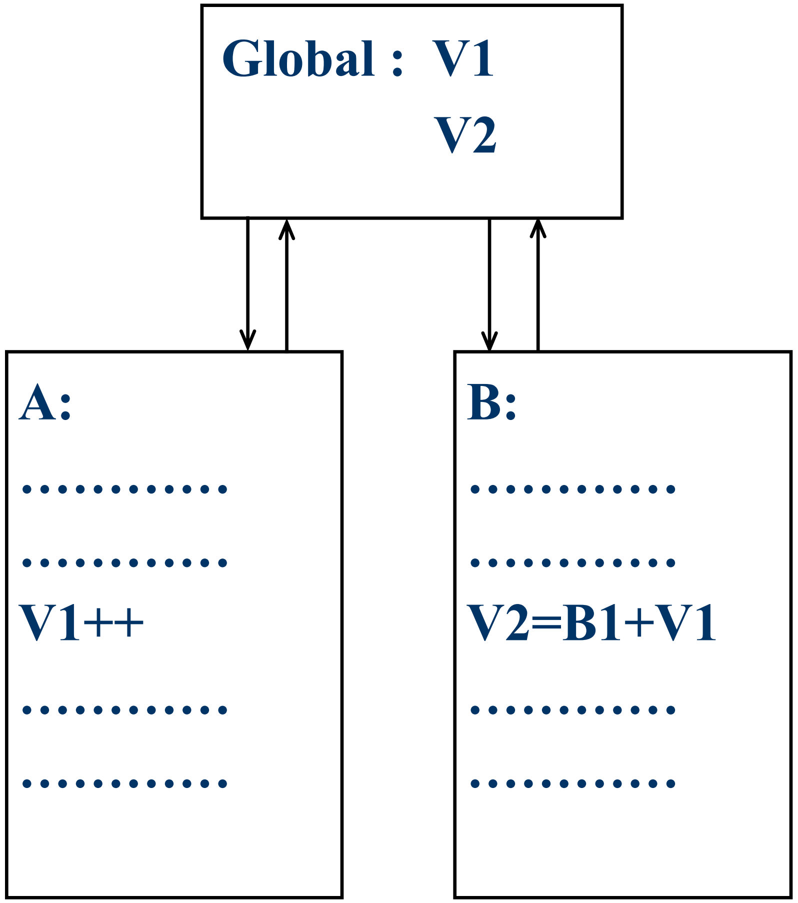
内容耦合



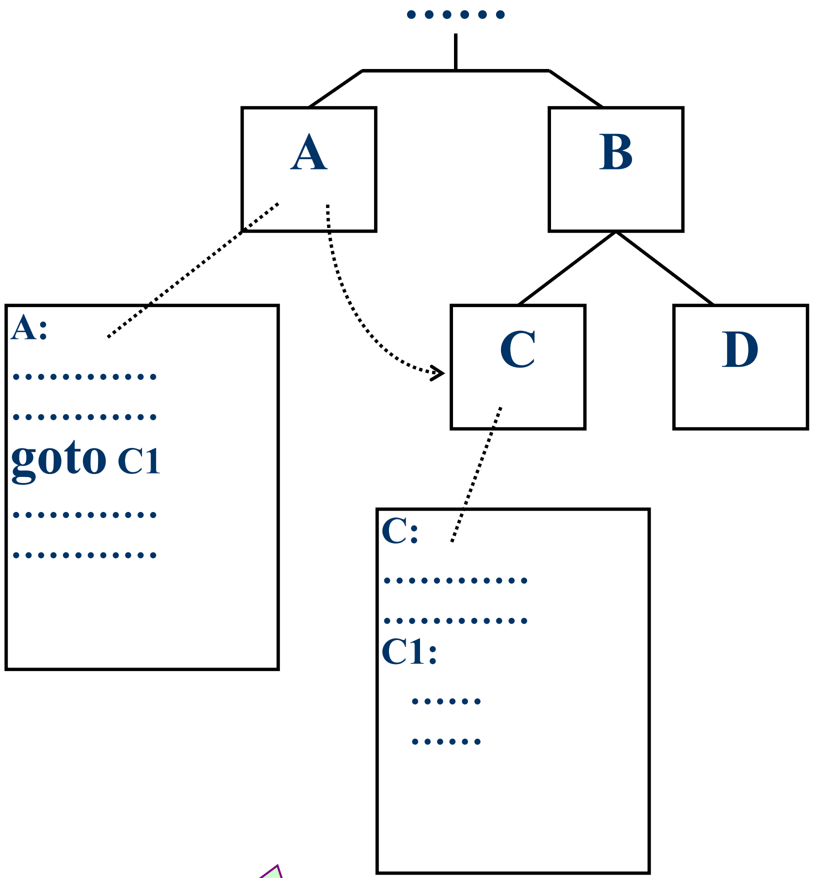
控制耦合



公共耦合



内容耦合



☹ The least desirable

Chapter 5 Designing the System

common coupling（公共耦合）：不同模块访问公共数据

content coupling（内容耦合）：一个模块直接修改另一个模块(A模块直接调用B模块的私有数据, 或直接转移到B模块中去)

E: OOD—with low coupling (P251)

F: purpose(in design)—pursue low coupling (but we can’t forbid using the highly coupled components)

② cohesion: internal glue among software components（软件部件内部各组成成分的关联程度）

A: purpose(for study cohesion)

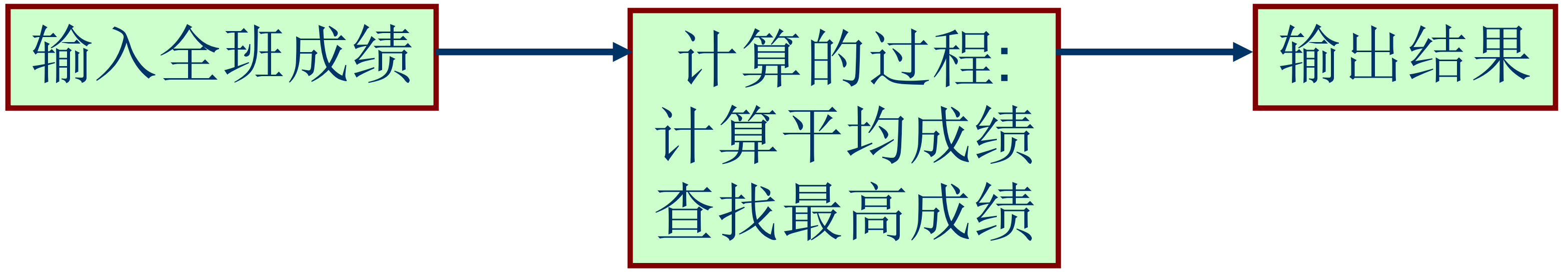
---- make each component as cohesive as possible

Chapter 5 Designing the System

B: seven levels (of cohesion measures)

- coincidental（偶然性内聚）：不相关的功能, 过程,数据等出现在同一个部件中.(例如，如果有几个模块都需要执行“读A”、“写Ｂ”等相同的一组操作，为了避免重 复书写，可以把这些操作汇成一个模块，供有关的模块调用)（例如：某些数据准备模块等等。）
- logical（逻辑性内聚）：逻辑上相关或相似的功能或数据放置在同一个部件内　(例如一个用于计算全班学生平均分和最高分的模块，无论计算哪种分数，都要经过读入全班学生分数，进行计算，输出计算结果等步骤。实际上除中间的一步须按不同的方法计算外，前、后这两步都是相同的。把这两种在逻辑上相似的功能放在一个模块中，就可省去程序中的重复部分)

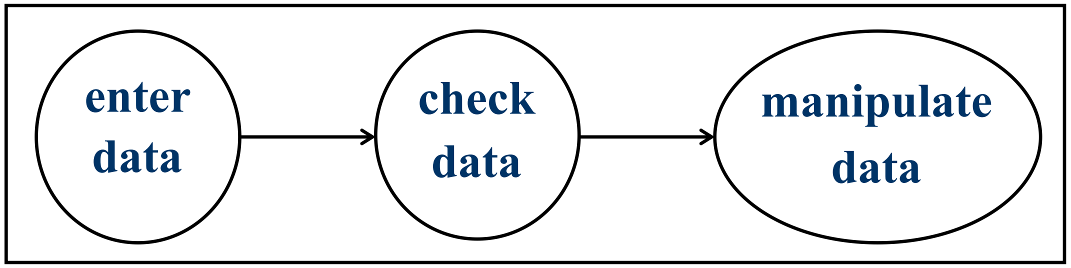
Chapter 5 Designing the System



temporal（时间性内聚）：部件各部分要求在同一时间完成 (例如:一个初始化模块可能包含“为变量赋初值”、“打开某个文件”等为正式处理作准备的功能。由于要求它们在同一时间内执行，故称为时间性内聚。)

Chapter 5 Designing the System

procedural（过程性内聚）：各部分有特定次序。



communicational（通讯性内聚）：各个部分访问共享数据（或私有共享，或远程共享等等）

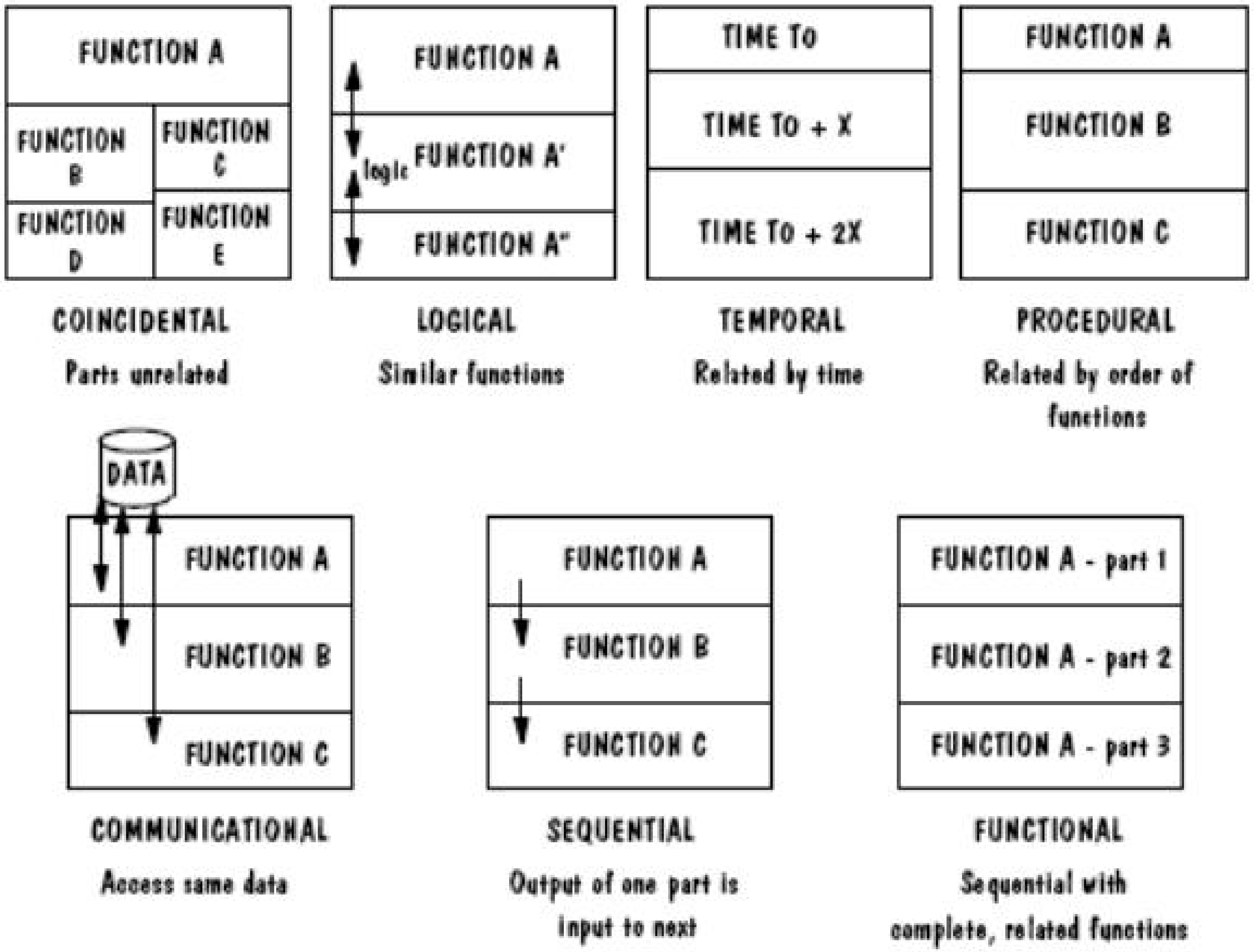
sequential（顺序性内聚）：各部分有输入输出关系。

functional（功能性内聚）：各部分组成单一功能

C: OOD----with high cohesion (P253)

D: purpose(in design)—pursue high cohesion (but we can't forbid using the low cohesive components)

E: 内聚度和耦合度的度量是设计质量测度的重要指标之一。


FUNCTION A
FUNCTION B
FUNCTION C
FUNCTION D
FUNCTION E
COINCIDENTAL
Parts unrelated
FUNCTION A
logic
FUNCTION A'
FUNCTION A''
LOGICAL
Similar functions
TIME To
TIME To + X
TIME To + 2X
TEMPORAL
Related by time
FUNCTION A
FUNCTION B
FUNCTION C
PROCEDURAL
Related by order of functions
DATA
FUNCTION A
FUNCTION B
FUNCTION C
COMMUNICATIONAL
Access same data
FUNCTION A
FUNCTION B
FUNCTION C
SEQUENTIAL
Output of one part is input to next
FUNCTION A - part 1
FUNCTION A - part 2
FUNCTION A - part 3
FUNCTIONAL
Sequential with complete, related functions

Chapter 5 Designing the System

2. Exception(意外) Identification and handling

① note: the <SRS> tell us what the system is supposed to do, but it does not usually make explicit what the system is not supposed to do

② exception: unanticipated action/event
exception handling: a dealing method(P254)

③ types: (P254)
A: failure to provide a service
B: providing the wrong service or data
C: corrupting data

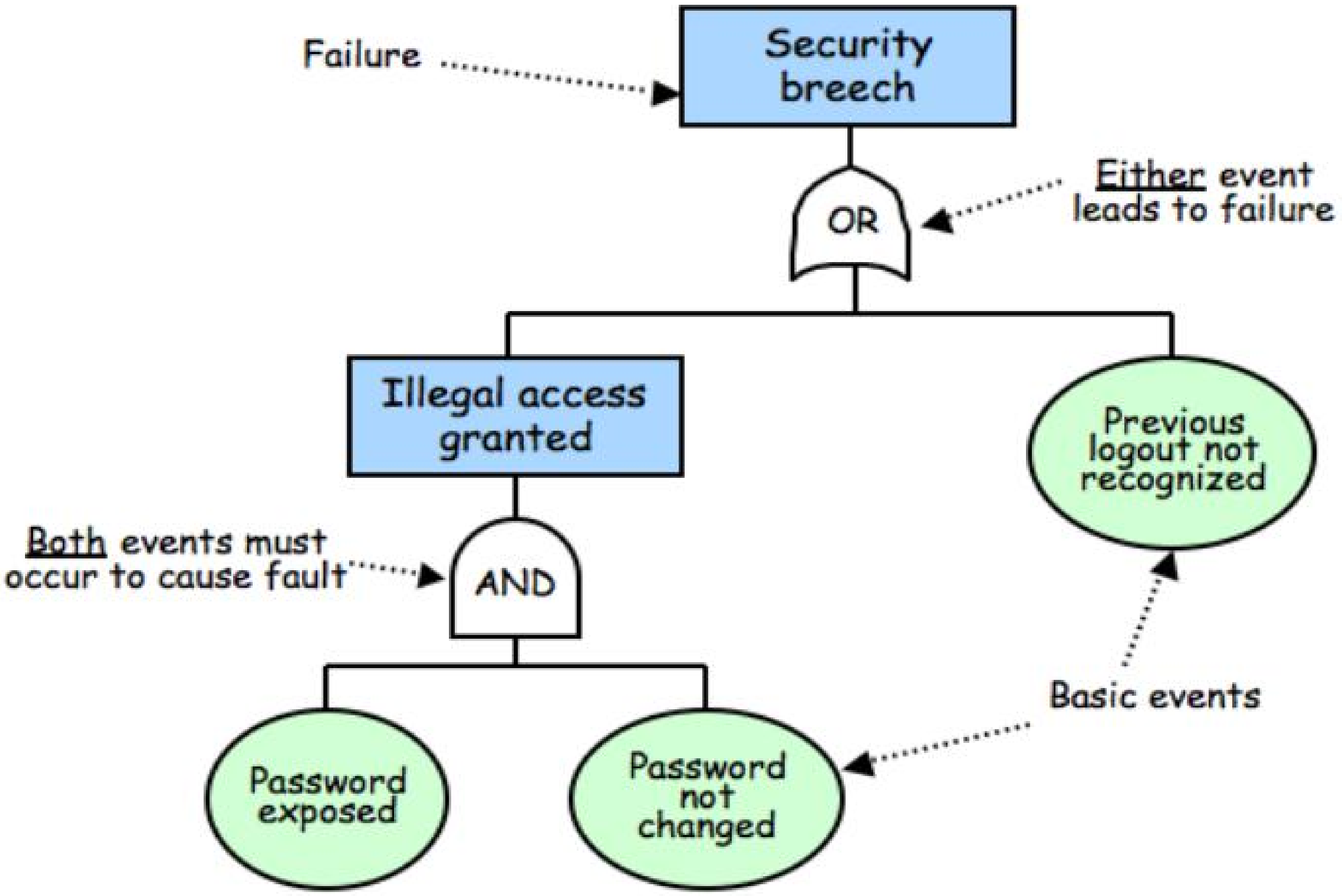
④ dealing ways (P255)
A: retrying
B: correct

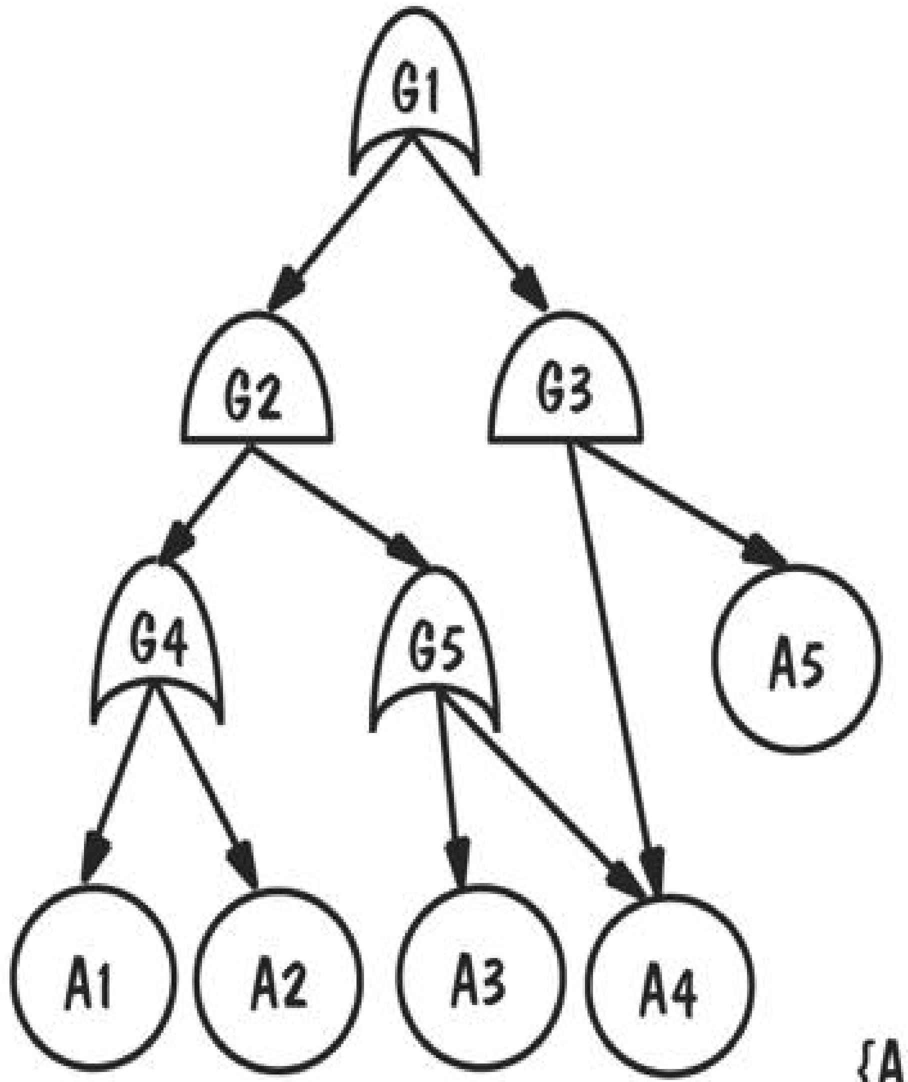
C: report ----优秀的设计应该允许用户向系统报告异常，系统设计时应该嵌入故障树分析（由此生成割集树）和失效模式分析技术，来表示和识别异常，以进行最后的异常处理，甚至对软件系统进行有限更正。

故障树：图**5-11**：倒置的树，根节点表示想分析的故障/失效，其他节点表示事件或者表示导致根节点失效所发生的故障。边表示节点间的关系，含有“与”、“或”等关系。

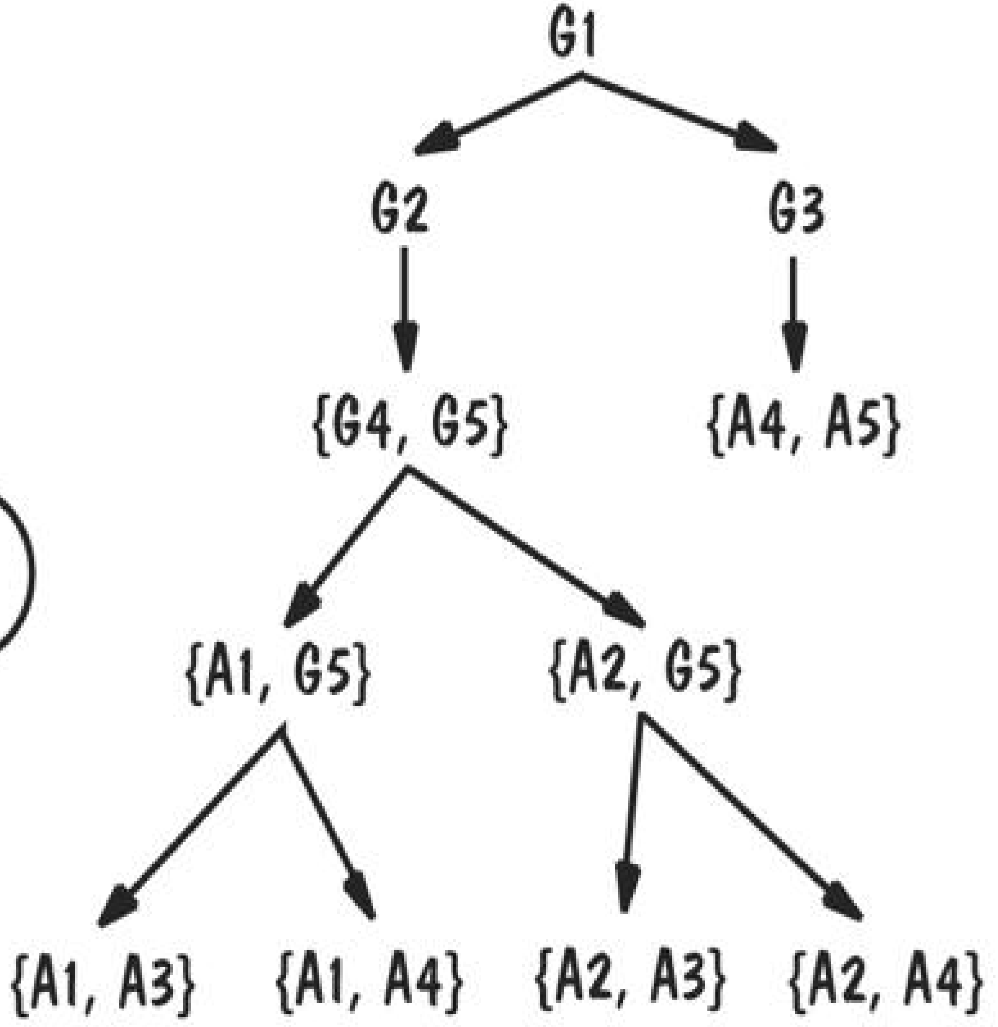
割集树：是对故障树的简化。见图**5-12**。

⑤ 注：最难处理的异常是所谓的“不规律”异常，包括提供的完全没有规律的错误服务和错误数据。


Failure
Security breech
Either event leads to failure
OR
Illegal access granted
Previous logout not recognized
Both events must occur to cause fault
AND
Basic events
Password exposed
Password not changed



Fault tree



Cut-set tree

Chapter 5 Designing the System

3. Fault prevention and tolerance

① error(mistake), fault, failure (P6)

② active fault detection—periodical examine symptoms of faults or anticipate failures

A: mutual suspicion : -----------做更多的预先检查和防范。

B: redundancy : ------------一个功能有多个预留的实现途径

③ passive fault detection (P256)----等待故障发生再处理

④ fault tolerance (容错设计) 当软件失败发生时,采取措施减少损失并将损害隔离开来, 在用户接受的条件下使系统继续运行 .

A: explain: failure occurs $\longrightarrow$ isolate the damage $\longrightarrow$ system continue running

Chapter 5 Designing the System

B: example

X: memory allocation

Y: “conveyor belts in an assembly line” (P258)

5.6 Techniques for Improving Design (X)

1. Reducing Complexity
2. Design by contract
3. Prototyping design
4. Fault-tree analysis

Chapter 5 Designing the System

4. 安全性分析（§5.7.3）

把不安全因素看成风险，在设计系统体系结构或程序结构时，有关安全问题必须进行以下分析：

①软件特征化：深入理解分析系统的目标和实现方式。

②威胁分析：范围很广，价值很大。（针对威胁来源和各种威胁活动）

③漏洞评估：寻找系统设计的各种问题。（新的高技术职业：白帽子黑客的日常工作。）（白帽子：杜塞漏洞，调查问题；黑客：利用漏洞）

④风险可能性分析（每个漏洞暴露的可能性大小）

⑤风险影响决策（攻击成功后，系统收到的影响评估）：

⑥风险缓解计划（应对措施等）

5. 成本效益分析（§5.7.5）

软件设计本身带来的效益分析，涉及软件冗余度和硬件分配等等。

Chapter 5 Designing the System

5.7 Design evaluation and validation (X)

note: two ways of design inspection:

A: validation

B: verification

1. Mathematical validation
2. Measuring design quality
3. Comparing designs
   - one specification, many designs
   - comparison table
4. Design reviews

Chapter 5 Designing the System

4. Design reviews （§5.9）

- 确认与验证 （结合练习题10）
  - **preliminary design review (概念设计复审)**
    - examines conceptual design with customer and users
  - **critical design review (技术设计复审)**
    - presents technical design to developers
  - **program design review (技术/程序设计复审)**
    - programmers get feedback on their technical designs before implementation

Chapter 5 Designing the System

5.8 Documenting the design（设计文档化）

1.Document

design rationale

menus and other display-screen formats

……

fault-handling approach

2. Note

①if it’s a distributed network, then the configuration document should include:

topology, synchronization, control and routing message of the nodes, etc. .

Chapter 5 Designing the System

② design description is cross-reference with <SRS> (two documents should have completeness and consistency), by“配置管理技术和工具” .

木 桶 定 律

盛水的木桶是由许多块木板箍成的，盛水量就是由这些木板共同决定的。若其中一块木板很短，则此木桶的盛水量就被短板所限制。这块短板就成了这个木桶盛水量的“限制因素”（或称“短板效应”）。人们把这一规律总结为“木桶原理”，或“木桶定律”，也称“短板理论”。

启示：

- 木桶定律无论对个体还是对一个组织，都有启迪。
- 木桶可以象征企业、部门、项目团队等，也可以象征一个独立的人，其盛水量就是企业、部门、项目团队或一个人的最大的能力、实力、竞争力。木桶之于企业，木板就是资金、技术、人力资源、产品、营销、内部管理等。
- **软件需求与软件设计不能有明显的弱点！**劣势决定优势，劣势决定生死，这是竞争的残酷法则！

课程设计及课堂拟演讲命题内容范围示例

1：在OOD的实施过程中，界面类、控制类、实体类等在软件工程中是如何实际协调使用的？举例说明。

2：在OOD的实施过程中，一般的系统设计和详细设计的区别与联系是什么？举出简单的事例加以说明。

3：在软件工程具体实施时，有时采用某些开发模式(如MVC)，请将您使用的模式予以归纳，举例说明其与软件工程课程中的系统设计或详细设计的联系与区别以及对应关系。

4:你的软件测试是如何进行的？请举例介绍一般软件测试的数据分类及彻底性原则。例如手机APP测试

5：自由讲述自己课程设计的内容，主要讲需求分析和软件设计。讲自己的收获和体会，可以整体讲，也可以讲其中的某个阶段或片段。

6. 结合实际案例，阐述本课件5.4节某些质量属性的实现与保证措施。

课程设计及课堂拟演讲命题内容范围示例

课堂演讲者的各种经验体会：

1：需求考虑的问题：格式转换，数据处理等；UML用例图，时序图，类图，界面设计，界面原型根据UML时序图设计，这样不需要反复修改。收获：改变了看待项目的角度。

2：个人经验：不写代码，单纯画UML图很难，不得不写完代码后又返回头修改UML图。

3：设计框架选择探讨，有的有各种坑（如何坑的？）。Rxjava框架。Er图等。

4：远程启动个人电脑，演示UML制图。

第五章作业：

1：本章练习题第4、第5题.

2：上一章第12题的延续：继续进行设计阶段的工作。
或者：上一章第17题的延续：继续进行设计阶段的工作。

注意：提交个人作业和课程设计的WORD文档名称：

学号+姓名+课程作业
学号+姓名+课程设计

5.10 Software Product Lines软件产品线

- Organizations can find success by reusing their expertise and software assets across families of related products
- The corporate strategy for designing and developing the related products is based on the reuse of elements of a common **product line**（产品线）
- A distinguishing feature of building a product line is the treatment of the derived products as a **product family**（产品系列）；their simultaneous development is planned from the beginning
- The family's commonalities are described as a collection of reusable assets (including requirements, designs, code, and test cases), all stored in a **core asset base**（核心资产库）

软件产品线：

是指具有一组可管理的公共特性的软件密集性系统的合集，这些系统满足特定的市场需求或任务需求，并且按预定义的方式从一个公共的核心资产集开发得到。

5.10 Software Product Lines
Core Asset Base

- Candidate elements （候选元素）in a core asset base:
  - Requirements
  - Software architecture
  - Models and analysis results
  - Software units
  - Testing
  - Project planning
  - Team organization

5.10 Software Product Lines
Strategic Scoping（战略范围）

- Product lines are based not just on commonalities among products but also on the best way to exploit them
  - First, employ strategic business planning to identify the family of products we want to build, using knowledge and good judgment to forecast market trends and predict the demand for various products（确定产品系列，判断市场趋势）
  - Second, **scope** the plans, so that the focus is on products that have enough in common to warrant a product-line approach to development. That is, the cost of developing the (common) product line must be more than offset by the savings we expect to accrue from deriving family members from the product line（划定计划范围，确保产品线方式开发）

5.10 Software Product Lines
Sidebar 5.8 Product-line Productivity

- CelsiusTech AB, a Swedish naval defense contractor, motivated by desperation, transitioned from custom to product-line development. In 1985, the company, then Philips Elektronikindustier AB, was awarded two major contracts simultaneously, one for the Swedish Navy and one for the Danish Navy.
    - senior managers questioned whether they would be able to meet the demands of both contracts, particularly the promised (and fixed) schedules and budgets, using the company’s current practices and technologies.
- Development of the product line and the first system were initiated at the same time; development of the second system started six months later. The two systems plus the product line were completed using roughly the same amount of time and staff that was needed previously for a single product. Subsequent products had shorter development timelines. On average, 70–80 percent of the seven systems’ software units were product-line units (re)used as is.

5.10 Software Product Lines
Advantages of Product-Line Architecture

- A product lines promotes planned modifiability
- Examples of product-line variability:
  - Component replacements
  - Component specializations
  - Product-line parameters
  - Architecture extensions and retractions

5.10 Software Product Lines

Sidebar 5.9 Generative Software Development

- **Generative software development**（生成式软件开发） is a form of product-line development that enables products to be generated automatically from specifications
- The domain engineer defines a **domain-specific language (DSL)** that application engineers then use to specify products to be generated
- Lucent developed several product lines and generative tools for customizing different aspects of its 5ESS telephone switch

5.10 Software Product Lines
Product-Line Evolution

- Key contributor to product-line success is having a product-line mindset
  - Company’s primary focus is development and evolution of product-line assets as opposed to individual products
  - Changes made to improve capability to derive products
  - Backwards capability

Chapter 9 Testing the System

Note A:unit and integration testing----by yourself or a small part of the development team

B:system testing----by the entire develop team

- 9.1 Principles of system testing

Focus A: ① <u>objective of unit and integration</u>

------ensure the code implemented the <u>design</u> properly ①②③④⑤⑥⑧⑨ ⑦ ⑩