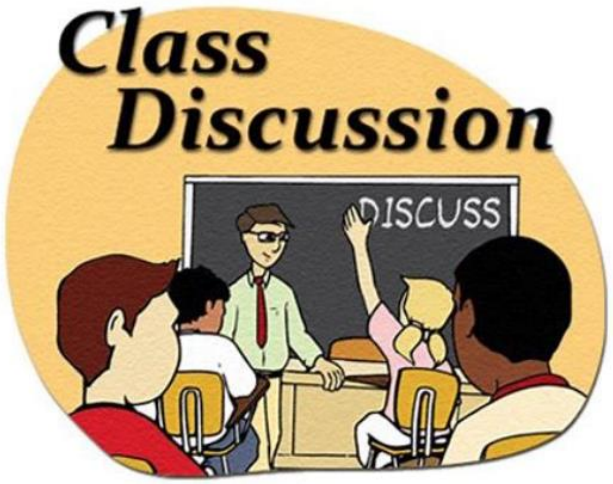
CS2001
软件工程

8. 软件设计
—软件设计概述

课件来源　　复旦大学软件学院

课堂讨论：设计的意义



课堂讨论：盖一栋高楼（例如上海中心）为什么要先进行设计然后再施工？

需要定义全局视图和分工协作关系
客户确认，尽早发现问题，避免修改
涉及很多方面，规划分工合作和专业能力
对各部分方案的可行性提前进行考虑
对材料等成本进行估算
规划开发流程和进度
对最终质量进行估计并从候选方案中选择

复杂和变化性：软件开发的本质挑战

Year	Operating System	SLOC (Million)
1993	Windows NT 3.1	4–5[1]
1994	Windows NT 3.5	7–8[1]
1996	Windows NT 4.0	11–12[1]
2000	Windows 2000	more than 29[1]
2001	Windows XP	45[2][3]
2003	Windows Server 2003	50[1]

Operating System	SLOC (Million)
Debian 2.2	55–59[4][5]
Debian 3.0	104[5]
Debian 3.1	215[5]
Debian 4.0	283[5]
Debian 5.0	324[5]
OpenSolaris	9.7
FreeBSD	8.8
Mac OS X 10.4	86[6][n 1]
Linux kernel 2.6.0	5.2
Linux kernel 2.6.29	11.0
Linux kernel 2.6.32	12.6[7]
Linux kernel 2.6.35	13.5[8]
Linux kernel 3.6	15.9[9]

软件规模

团队规模

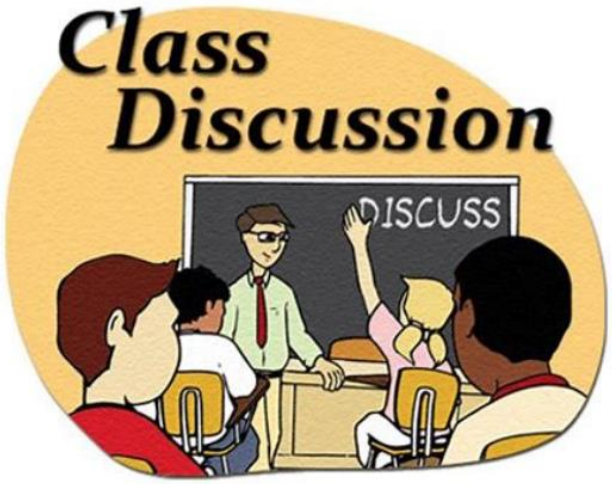
易变的需求

复杂的技术环境

为什么要进行设计

- 通过分解和抽象应对复杂性
- 作为需求和实现之间的桥梁
- 考虑候选方案并进行质量评估
- 作为任务分解和分工的依据
- 针对全局的技术决策作出约定
- 为开发人员交流提供基础
- 为相似系统的开发提供复用机会
- 为未来可能的变化预留空间

课堂讨论：软件设计的内容



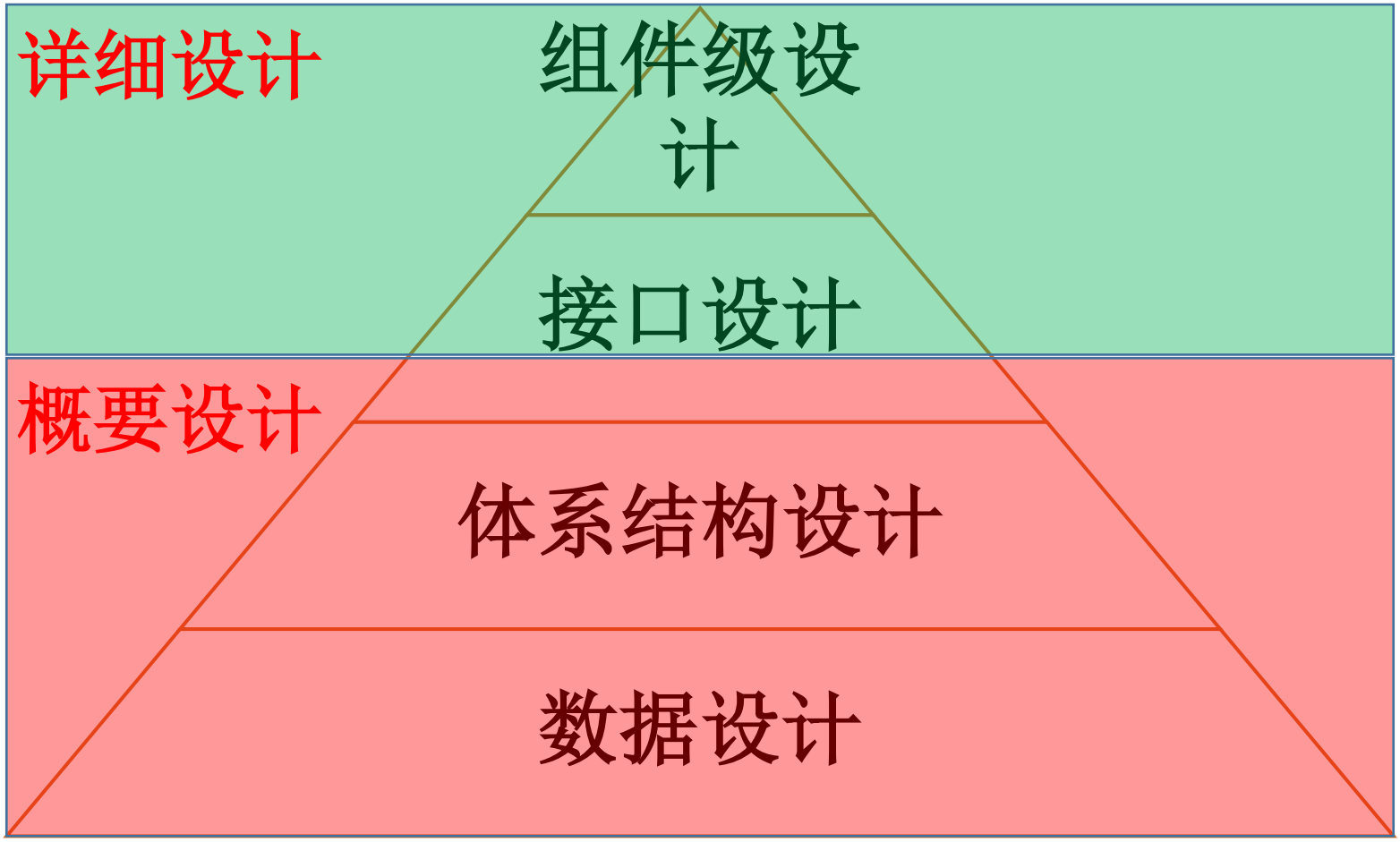
课堂讨论：如果请你来设计一个复杂的软件系统，那么你需要考虑哪些问题？

确定技术选型（C/S、B/S、微服务）
选择合适的技术平台和开发框架
定义好模块化划分（实现良好的信息隐藏和松耦合等）及其接口
定义好编程语言、平台和命名惯例
明确开发任务及其优先级和顺序

软件设计：承上启下

- 承上：满足需求
  - ✓ 实现所有显式定义的需求，如需求文档中所定义的需求
  - ✓ 适应所有隐式定义的需求，如未明确定义但符合客户、用户和开发方价值的需求
- 启下：实现和测试的基础
  - ✓ 从实现角度给出了软件系统的全貌：覆盖数据、功能、行为、交互、部署等多个方面
  - ✓ 实现、测试和维护的基础
  - ✓ 面向相关读者（开发者）的可读性和可理解性

软件设计的层次



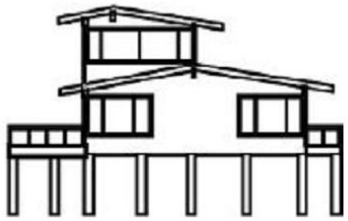
软件设计的内容

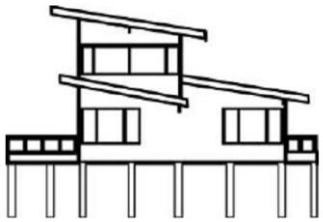
- 数据设计：全局的数据结构设计，例如实体及其属性和关系
- 体系结构设计：模块/组件划分及其交互关系、部署结构定义，语言、异常处理方式等其他全局性约定
- 接口设计：外部（如用户、硬件）及内部（组件间）交互接口设计
- 组件级设计：组件内的类与类之间关系定义，局部的数据结构、算法设计

设计是一个约束求解过程

一个海滨小屋的设计，仅仅考虑屋顶的样式就有很多种不同的方案，此外还要考虑：

- 建筑材料
- 房屋尺寸
- 门窗位置
- …





设计==搜索

根据问题的**目标**和**约束**，在**解空间**中寻找**最优**解决方案的过程

Frederick P. Brooks

《The Design of Design: Essays from A Computer Scientist》

海滨小屋设计的约束求解

- 目标
  - ✓ 面向大海的海滨小屋，为宾客提供难忘的风景
  - ✓ 应该坚固，以抵御飓风
  - ✓ 应具备至少14个人躺卧和就坐的空间
- 约束
  - ✓ 法律约束：小屋必须位于海滨场地的边界线后至少10英尺
  - ✓ 有一定弹性的约束：小屋应在春季前完工
  - ✓ 隐含的约束：小屋必须符合各种建筑法规

容易漏雨吗？房屋高度合适吗？

美观吗？　　阻碍阳光吗？

...各种各样的效用评价

软件设计的目标

- 功能性方面：完整实现功能需求
- 非功能性方面：充分实现质量需求
  - ✓ 外部质量：客户和用户所感受的质量
    - ■ 性能、可用性、可靠性、信息安全、安全性等
  - ✓ 内部质量：内部开发人员所感受的质量
    - ■ 可维护性、可扩展性、可移植性、可复用性等

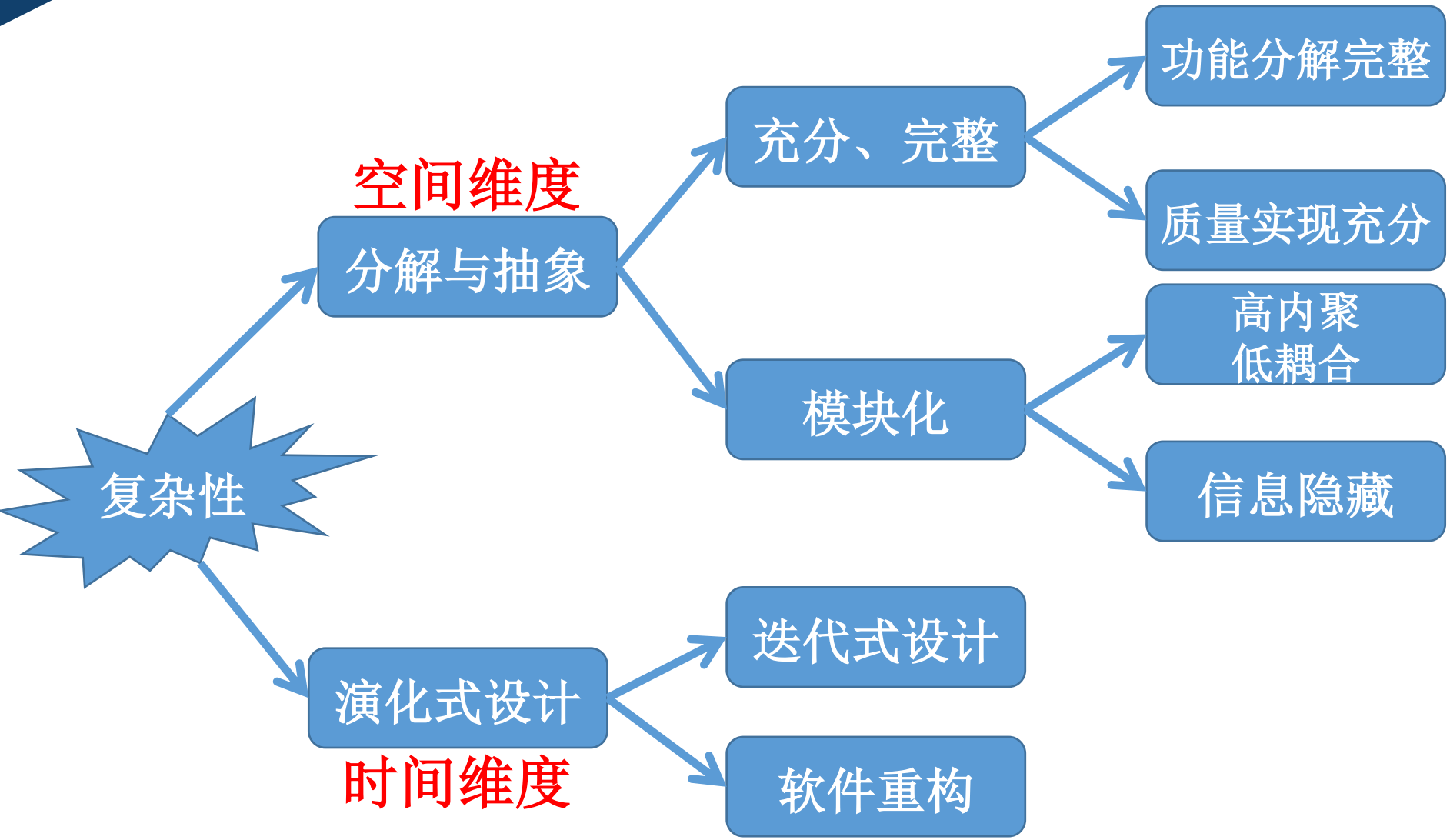
非功能性需求的特点

— 往往是全局性的（与全局体系结构相关）

— 无绝对的满足（满足程度）

— 经常存在冲突（需要权衡）

软件设计的基本原则


复杂性
空间维度
分解与抽象
充分、完整
功能分解完整
质量实现充分
模块化
高内聚
低耦合
信息隐藏
时间维度
演化式设计
迭代式设计
软件重构

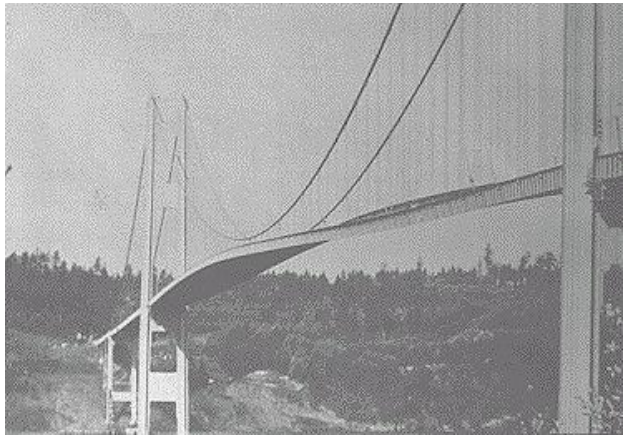
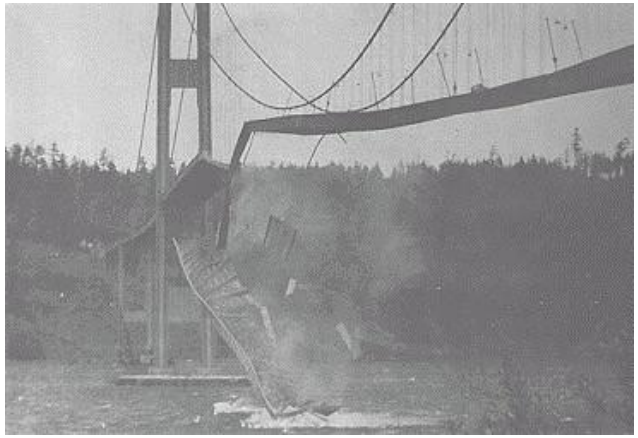
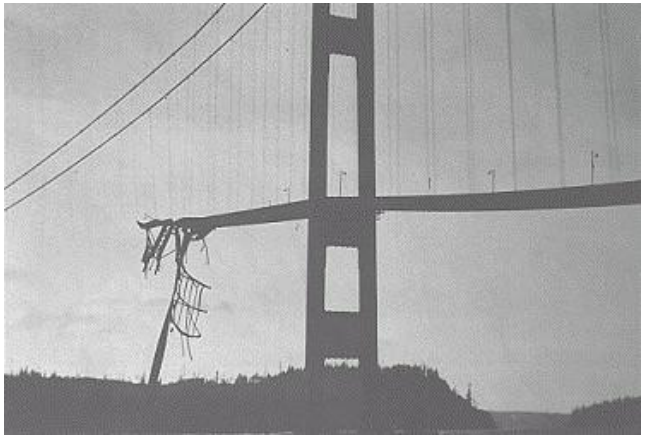
体系结构设计决策

- 是否存在一个通用的应用体系结构可以用作所设计的系统的模板?
- 系统将如何分布到各个硬件核或者处理器上?
- 可以使用什么体系结构模式或风格?
- 用来组织系统的基本方法是什么?
- 用什么样的策略来控制系统中的构件的运行?
- 系统中的结构构件将如何分解为子构件?
- 什么样的体系结构组织是实现系统的非功能性需求的最佳选择?
- 系统的体系结构应当如何描述?

非功能性质量与体系结构设计决策

性能（Performance）	多副本部署、负载均衡、关键性操作局部化
信息安全（Security）	层次化体系结构，关键性资产在最内层保护并进行高级安全确认
安全性（Safety）	安全性相关操作集中化，引入保护系统在失效时安全关闭系统
可用性（Availability）	使用冗余组件，支持不停机情况下的组件更换或更新
可维护性（Maintainability）	使用易修改的细粒度、自包含的组件，松耦合（如数据生产者与消费者分离，避免共享数据结构）

设计是一个险恶的（wicked）问题

塔科马海峡大桥（Tacoma Narrows Bridge）设计时主要考虑了承重负荷，但1940年的某一天狂风大作，大桥在横向谐波的作用下坍塌。这以后工程师们才知道应该考虑空气动力学因素。

要在严苛的标准下全面考虑各方面的质量问题

前路艰险，各种意想不到的情况都有可能发生

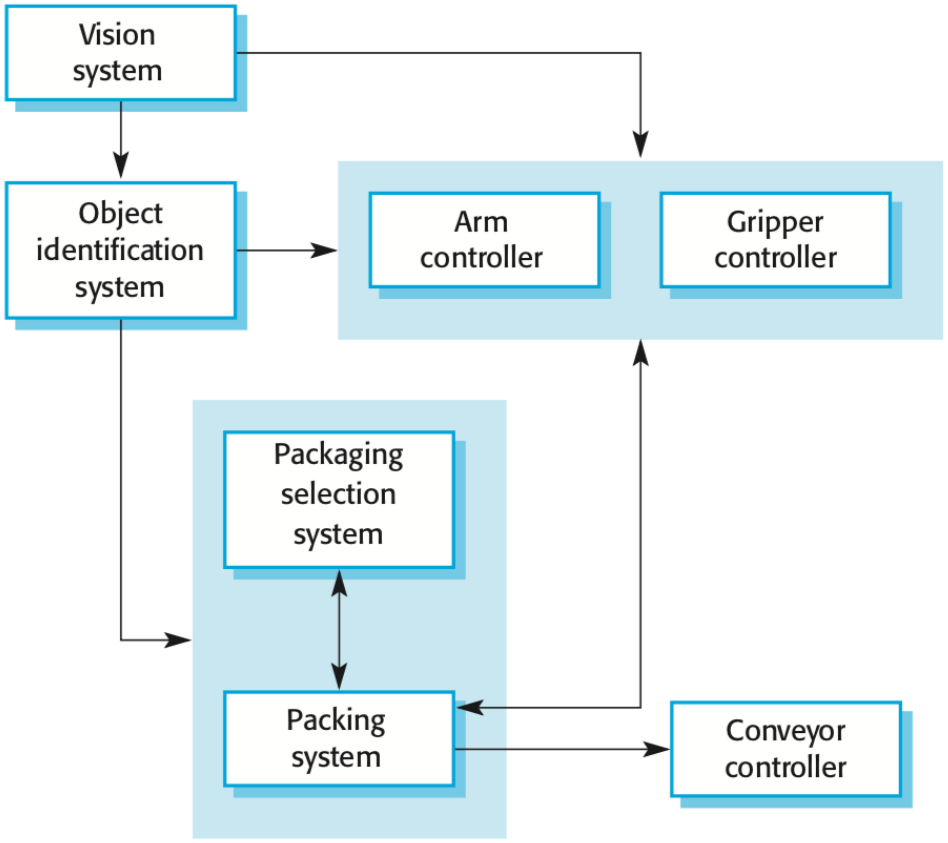
没有完美的设计，在演化中不断适应和完善

软件设计思想

- 体系结构（Architecture）
- 分解（Decomposition ）
- 抽象（Abstraction）
- 逐步精化（Stepwise Refinement）
- 风格和模式（Style & Pattern）
- 模块化（Modularity）
- 信息隐藏（Information Hiding）
- 功能独立（Functional Independence）
- 重构（Refactoring）

软件体系结构

- 软件系统的总体设计结构
  - ✓ 组件与组件之间的关系：反映软件系统的总体分解结构
  - ✓ 很大程度上决定了系统的整体质量
  - ✓ 是一个多视图的复杂设计结构：静态结构、动态交互、分布式部署结构等
  - ✓ 同时包括涉及系统全局的各种约定和决策



打包机器人体系结构

体系结构：分解、抽象、约定...

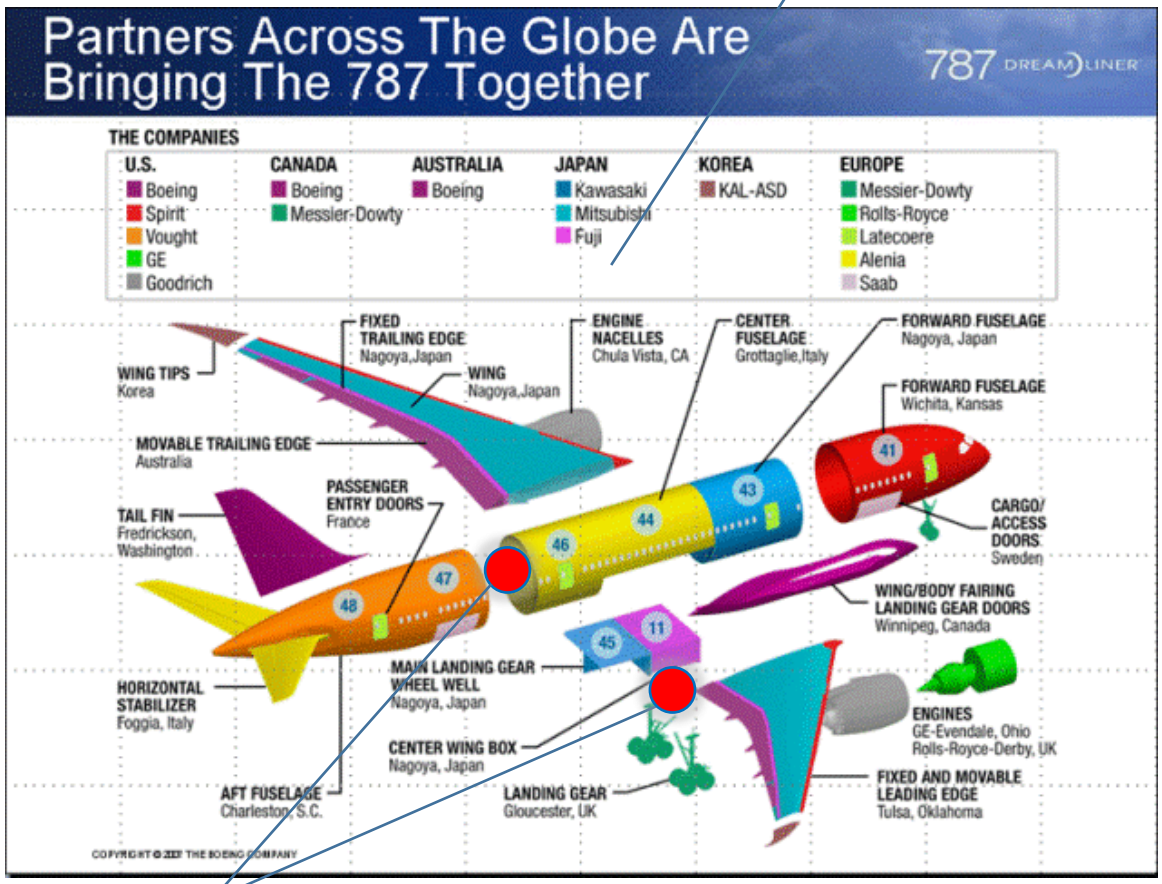
分解

约定

约束

标准

.....



接口（抽象）

总体质量在很大程度上取决于体系结构设计

好的体系结构设计

- 容易理解
  - ✓ 设计结构与领域概念对应关系清晰
- 容易开发和集成
  - ✓ 每个组件关注于有限的事情
  - ✓ 不同组件之间交互尽可能少
- 容易扩展和修改
  - ✓ 扩展不需要修改已有的组件
  - ✓ 修改的影响局部化，尽量减少全局影响
- 容易复用
  - ✓ 整个设计的复用
  - ✓ 单个组件的复用

抽象

- 控制软件设计复杂性的基本策略
- 从特殊到一般的过程
- 数据抽象
  - ✓ 突出与所关心的问题密切相关的核心信息，忽略无关信息
  - ✓ 抽象数据类型，如类、结构体
- 过程抽象
  - ✓ 突出过程操控的输入输出及相关条件，忽略具体执行过程
  - ✓ 抽象过程描述，如接口

  屏蔽底层细节，突出事物的本质特性

课堂讨论：抽象数据类型

课堂讨论：下面这段字体大小设置（12号小四字体，16像素）代码有什么问题吗？

```
currentFont.size = 16
```

这样似乎好一点

```
currentFont.size = PointsToPixels(12)
```

这样更好一点

```
currentFont.setSize(12)
```

屏蔽底层细节（像素），突出本质特性（逻辑字体大小）

好处：符合人的思维方式，降低复杂性，提高可迁移性

抽象数据类型：更多的例子

List
Initialize list
Insert item in list
Remove item from list
Read next item from list

Stack
Initialize stack
Push item onto stack
Pop item from stack
Read top of stack

Light
Turn on
Turn off

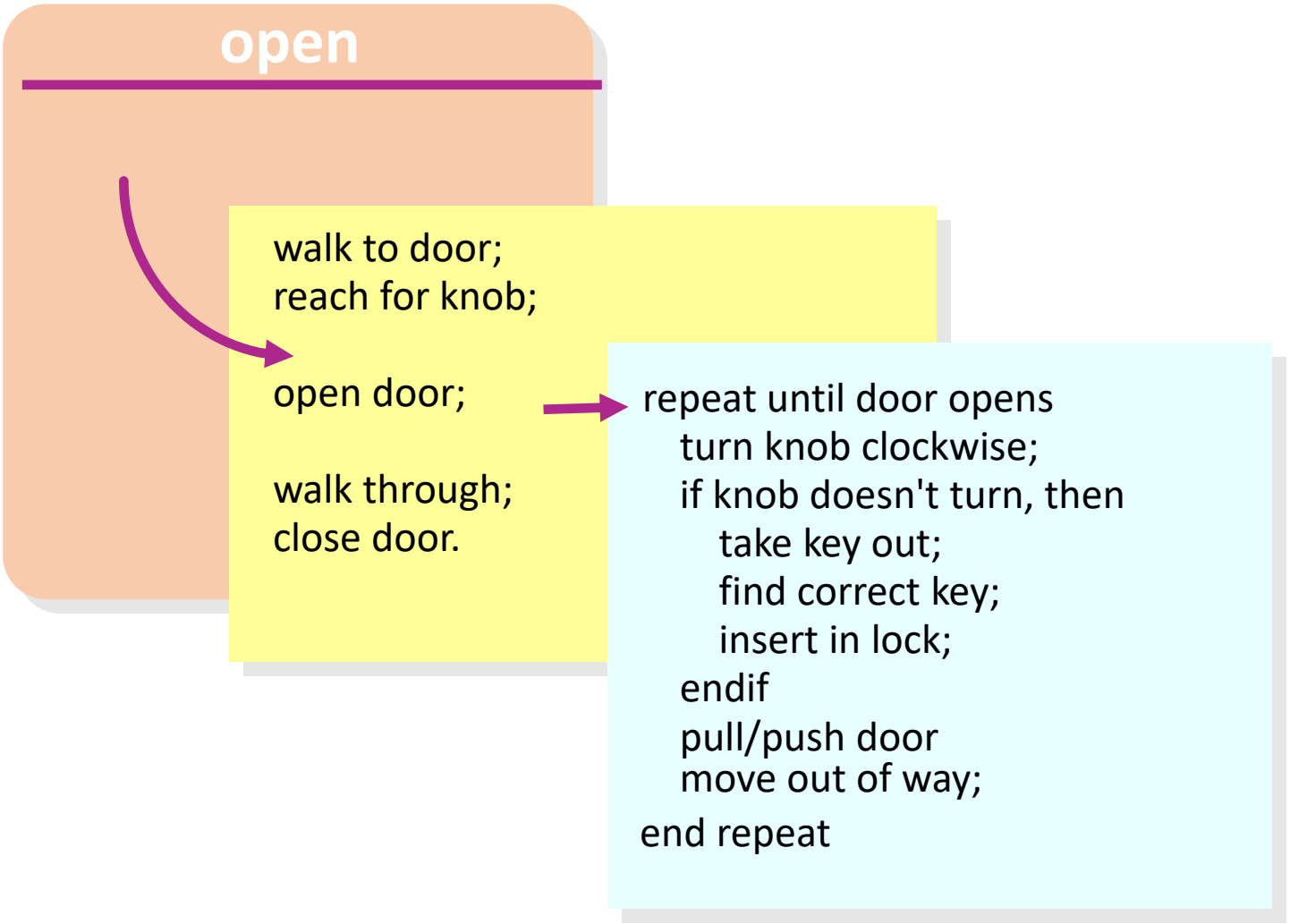
Elevator
Move up one floor
Move down one floor
Move to specific floor
Report current floor
Return to home floor

逐步精化

- 将问题的求解过程分解为若干步骤或阶段，每步都比上步更精化，更接近问题的最终解决方案
- 抽象与精化
  - ✓ 抽象使得设计者在考虑高层设计时能够忽略过程和数据的底层细节
  - ✓ 精化使得设计者能够逐步揭示底层的细节，得到更加具体的解决方案

逐步精化

Door: open



软件设计中的逐步精化

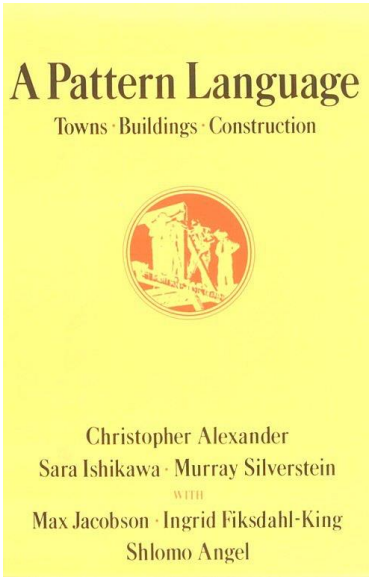
- 体系结构设计
  - ✓ 考虑系统的总体设计，涉及系统中所有的部件，复杂性较高
  - ✓ 抽象：仅考虑每个部件的外部属性（接口、质量要求等）
- 组件级详细设计
  - ✓ 仅考虑当个部件，允许逐步考虑更多细节
  - ✓ 精化：考虑部件的内部设计（如类结构、内部数据结构等）—如何实现外部属性要求

风格和模式：以建筑为例

中国古典园林风格

哥特式风格

巴洛克风格



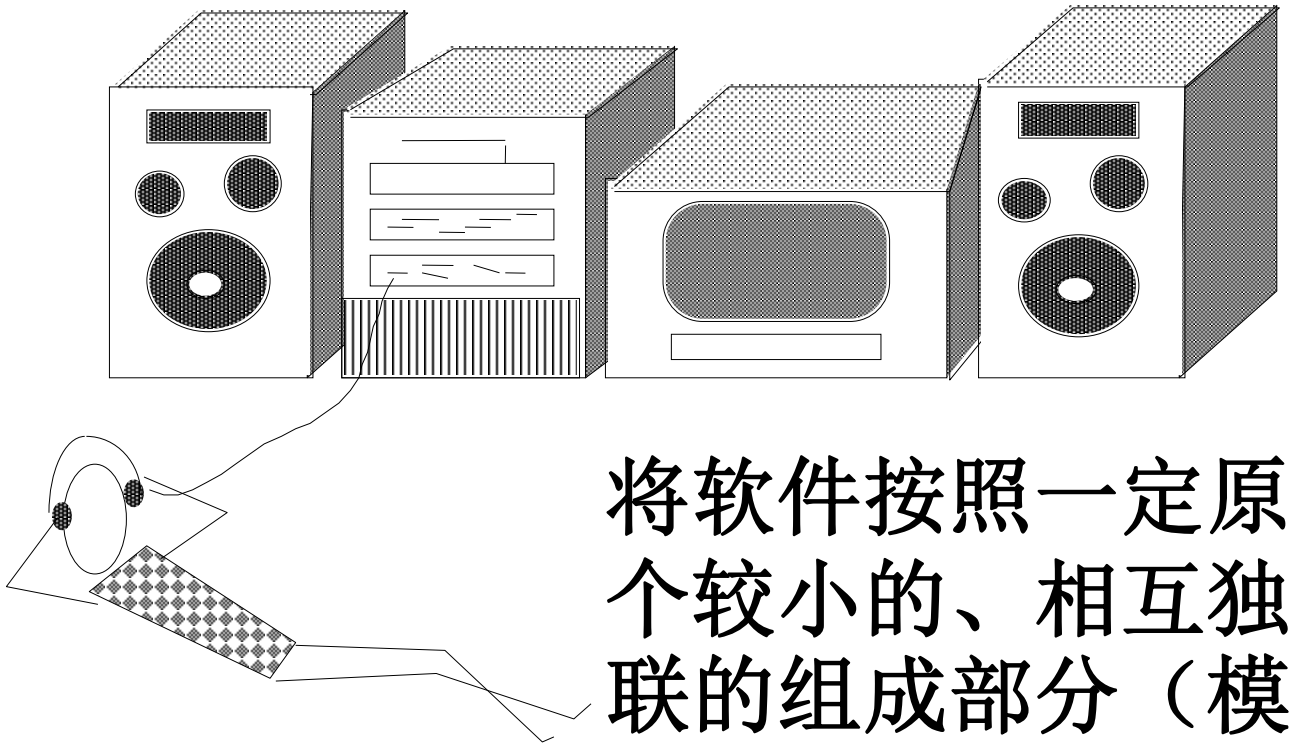
针对住宅：儿童的领域、农家的厨房、私家的沿街露台、个人居室、床卧室、浴室、大储藏室

针对办公室和公共建筑物：灵活办公空间、共同进餐、共同小组、宾至如归、等候场所、小会议室

……

模块化

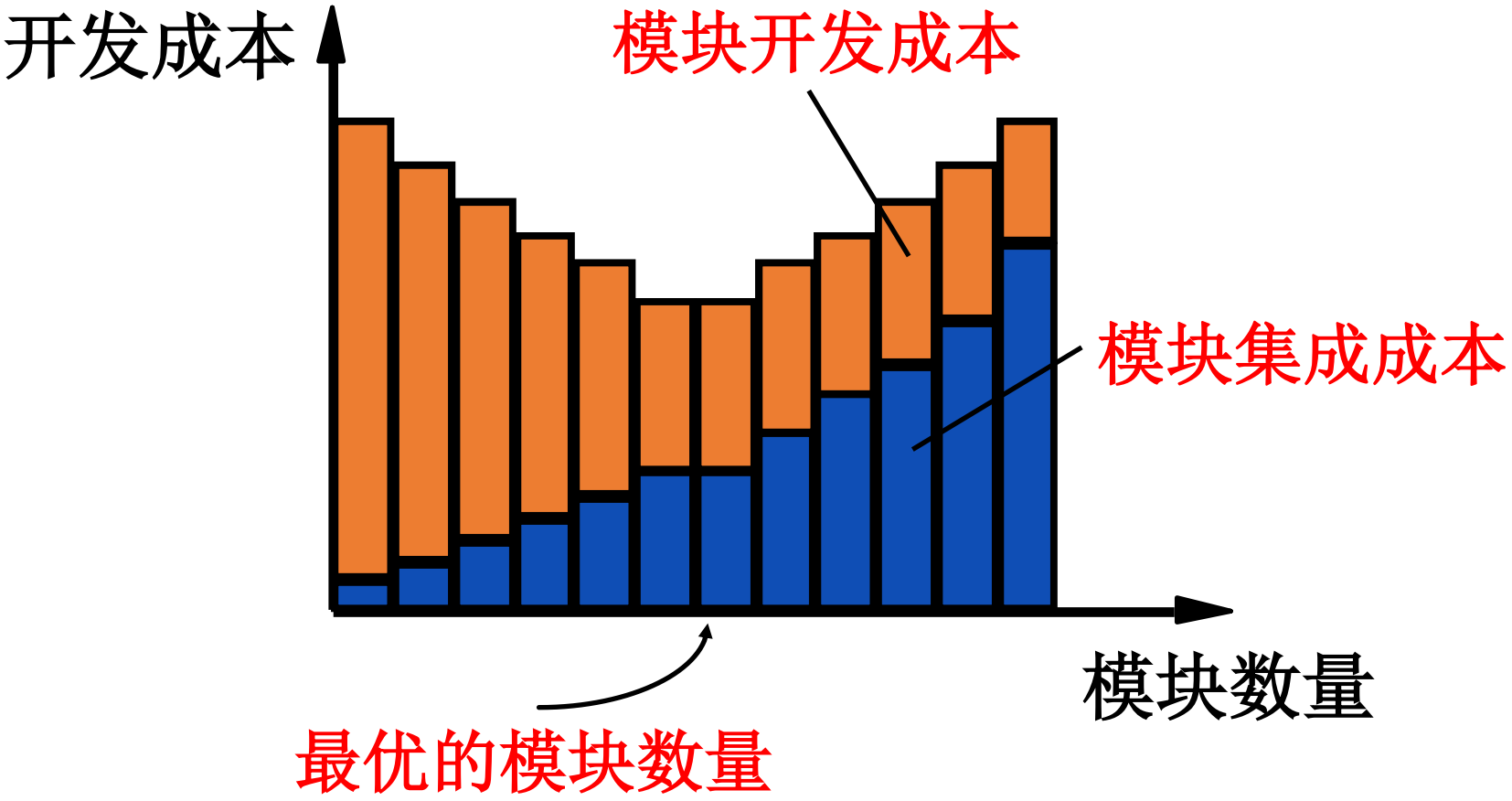
容易构造、容易变化、容易修理（错误）



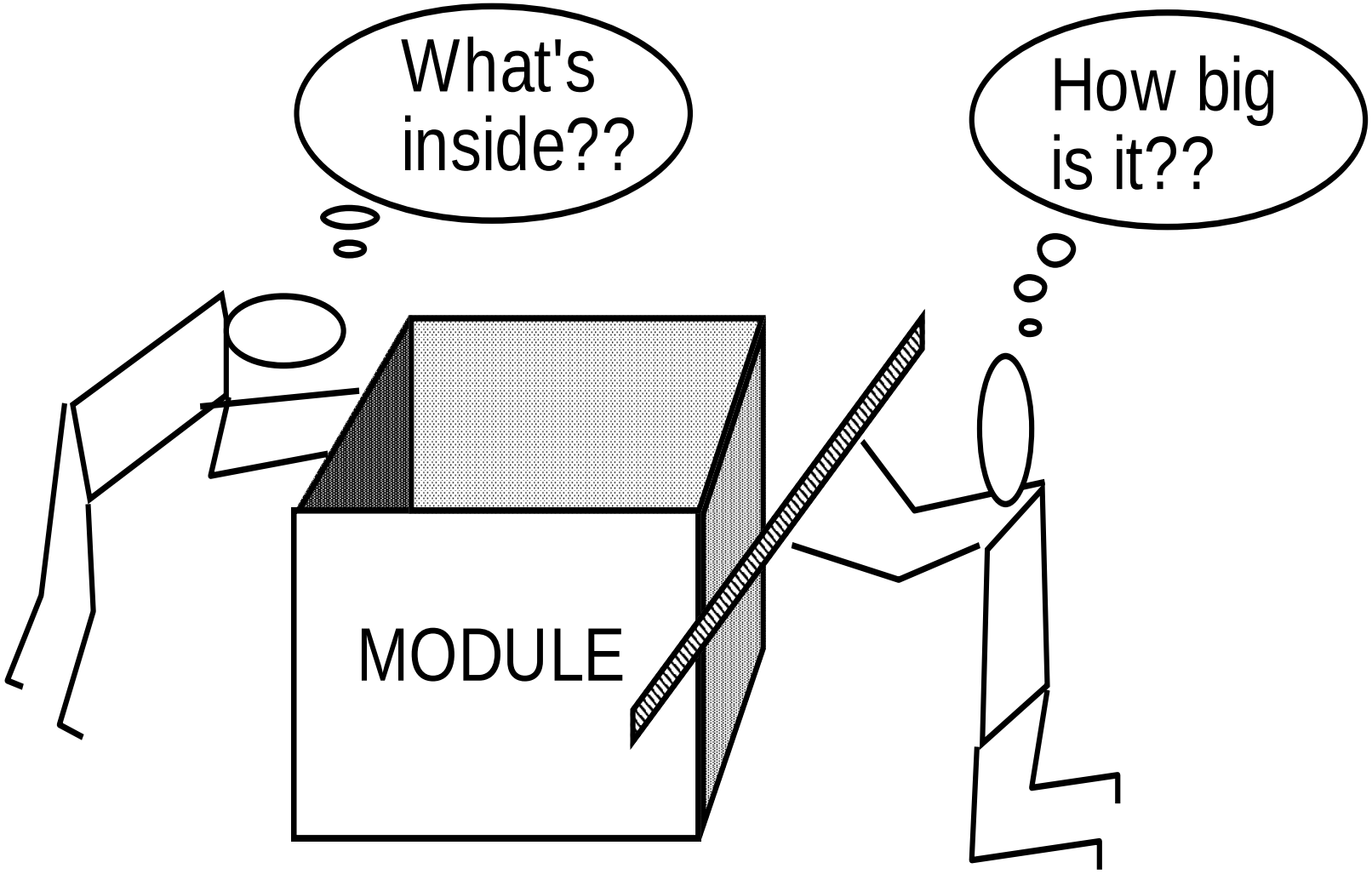
将软件按照一定原则划分为一个个较小的、相互独立而又相互关联的组成部分（模块）

是系统分解和抽象过程的体现

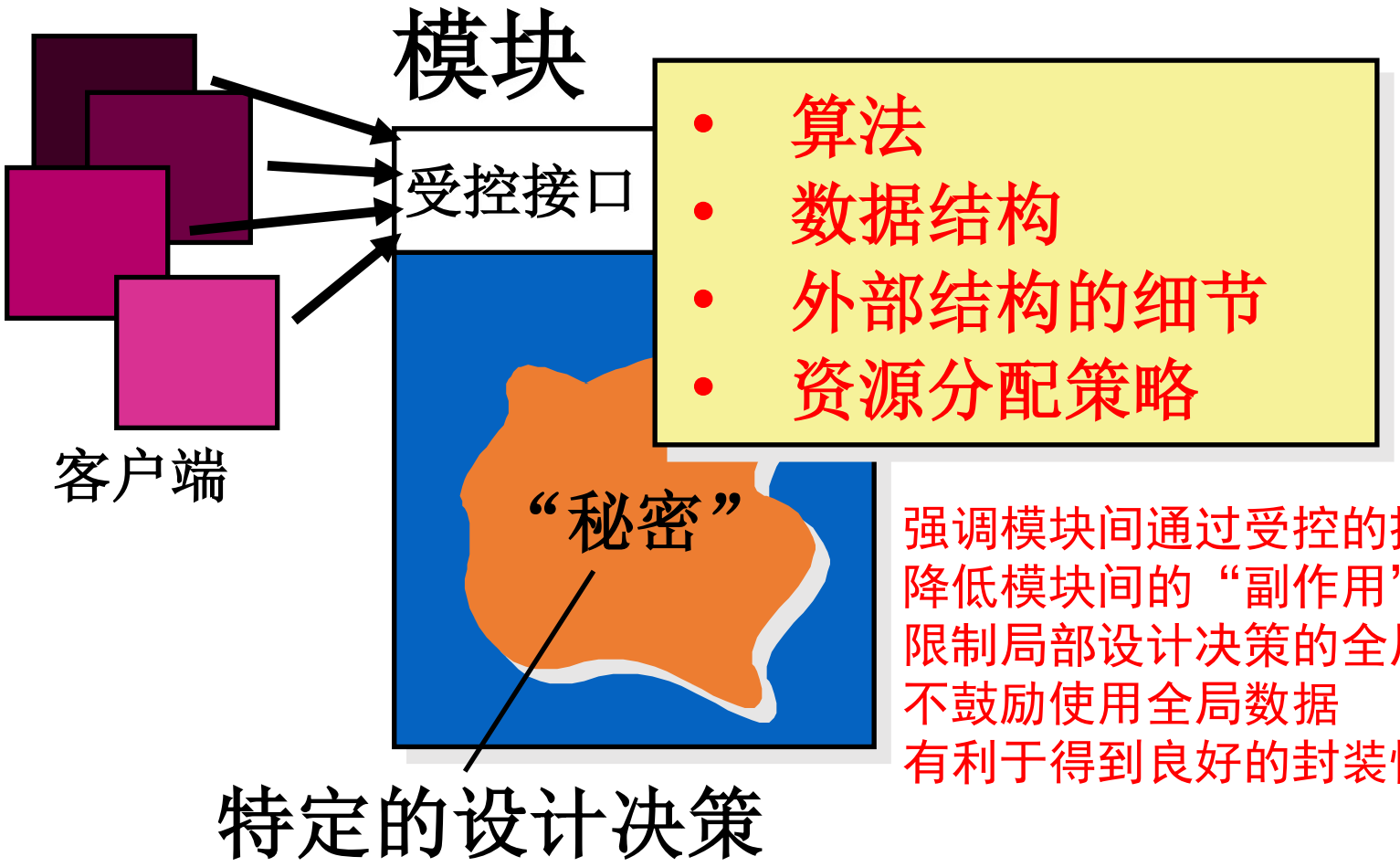
软件模块的最佳数量



软件的内部和外部视图



信息隐藏



- 算法
- 数据结构
- 外部结构的细节
- 资源分配策略

强调模块间通过受控的接口进行通信
降低模块间的“副作用”的可能性
限制局部设计决策的全局影响
不鼓励使用全局数据
有利于得到良好的封装性和模块化

为什么需要信息隐藏

- 限制局部设计决策的全局影响
- 降低“副作用”发生的可能性
- 强调模块/组件/类间通过受控接口进行通信
- 不鼓励使用全局数据

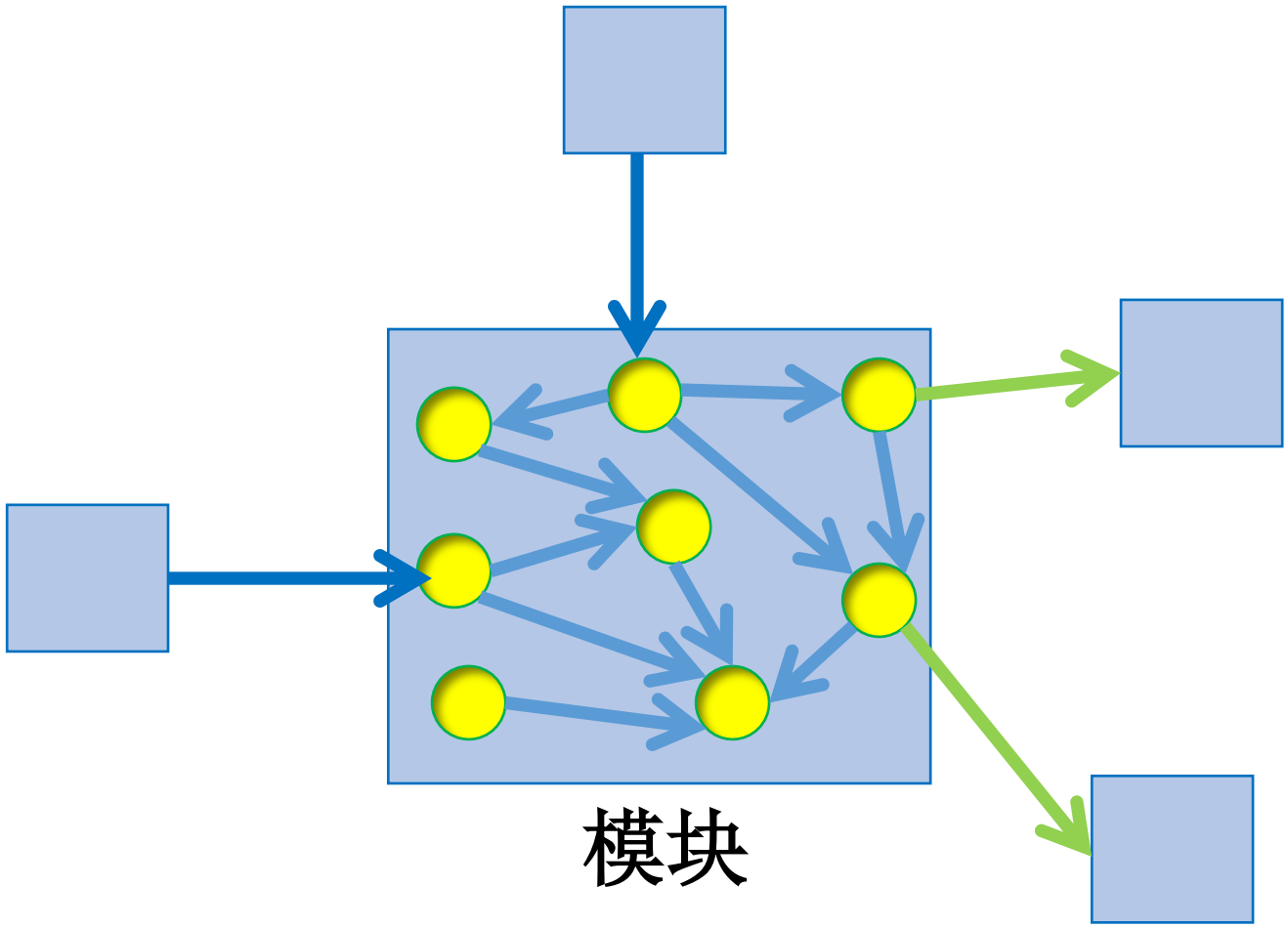
实现良好的封装和模块化，从而获得更高质量的软件

功能独立

- 每个模块只完成单一的功能，同时尽量限制与其他模块的交互
- 衡量标准
  - ✓ 内聚度（Cohesion）：模块内部的元素密切结合，完成且只完成单一功能或职责的程度
  - ✓ 耦合度（Coupling）：模块与其他模块相互联系的程度

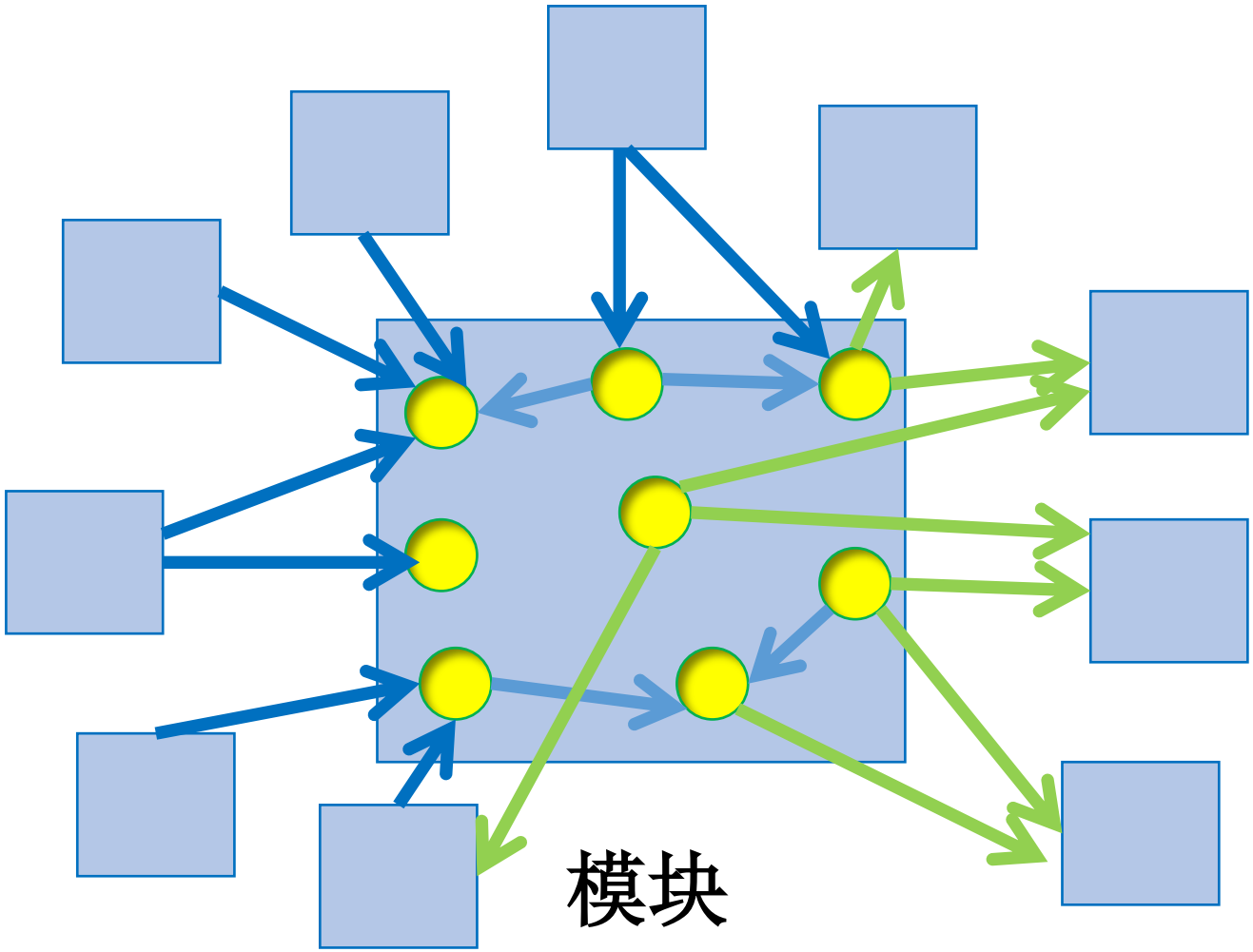
目标：高内聚、低耦合

功能独立性好的设计



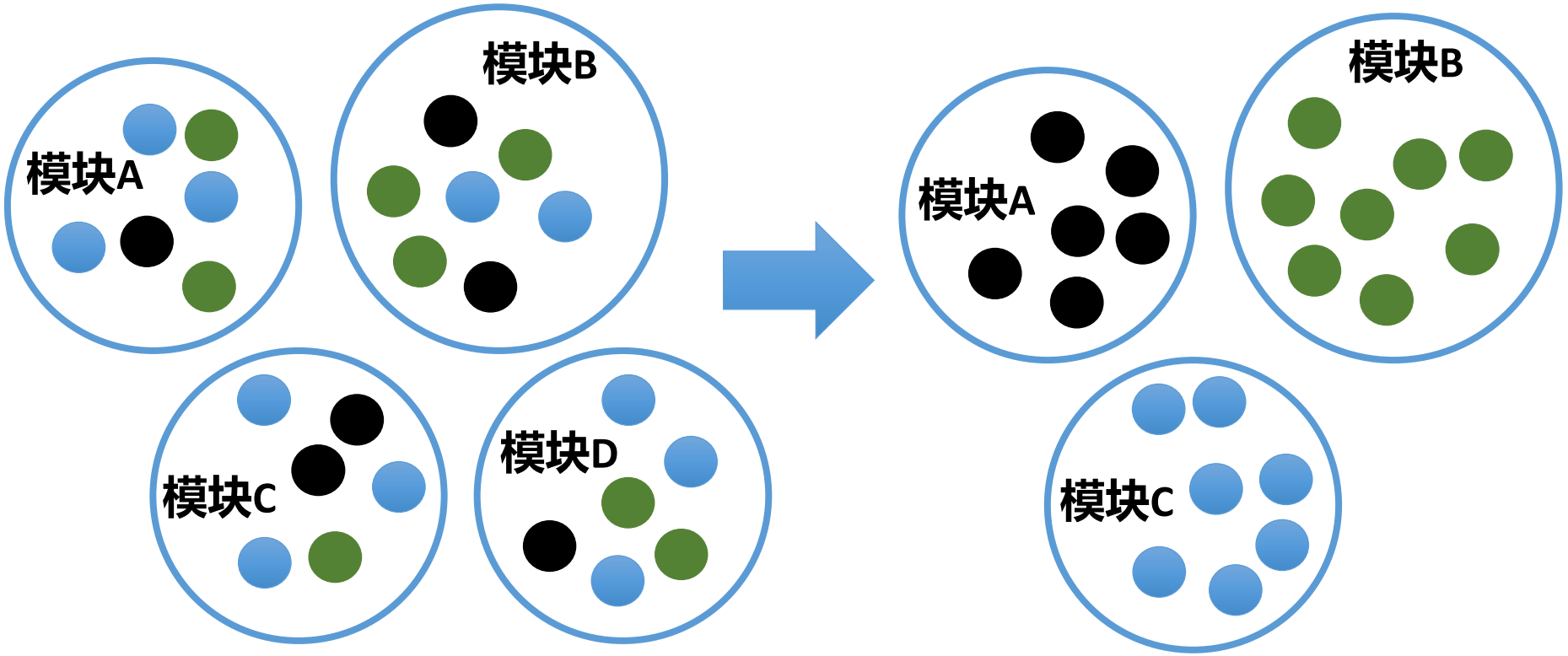
高内聚、低耦合

功能独立性不好的设计



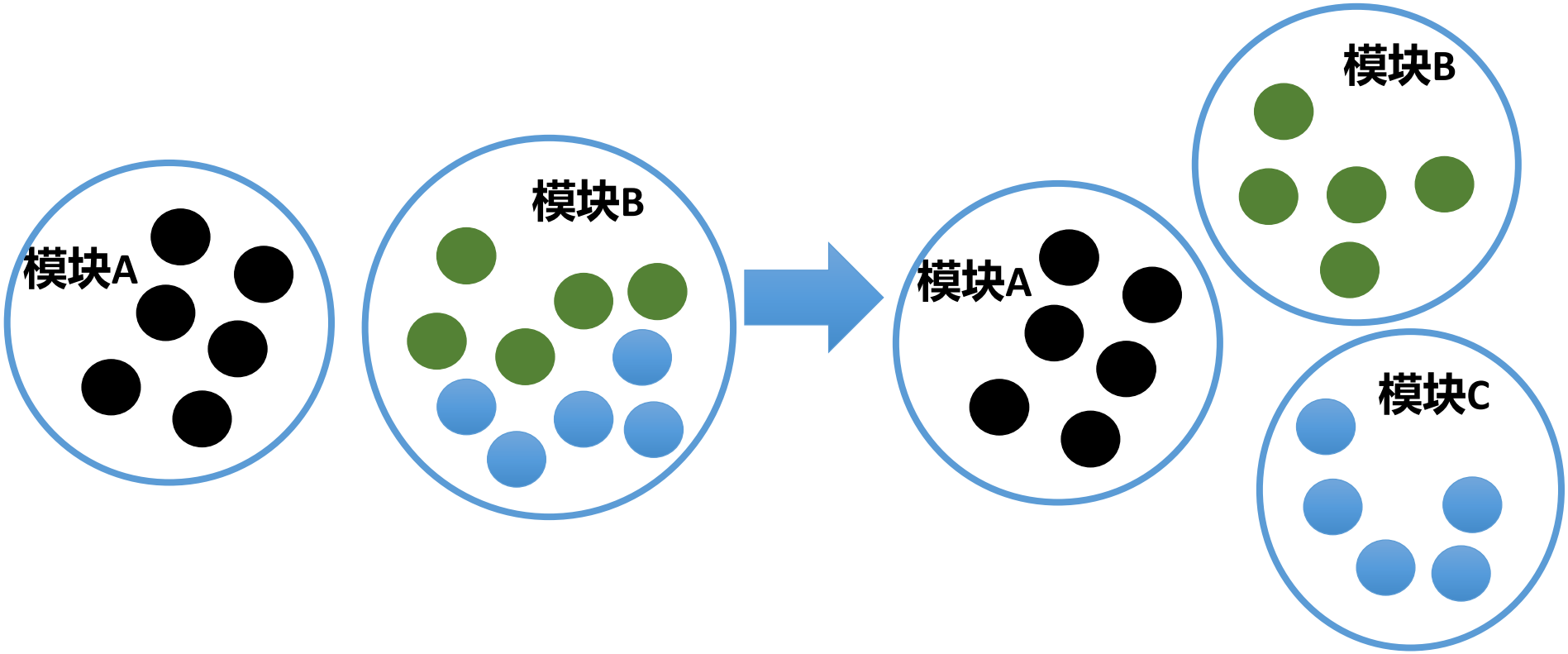
低内聚、高耦合

如何实现高内聚



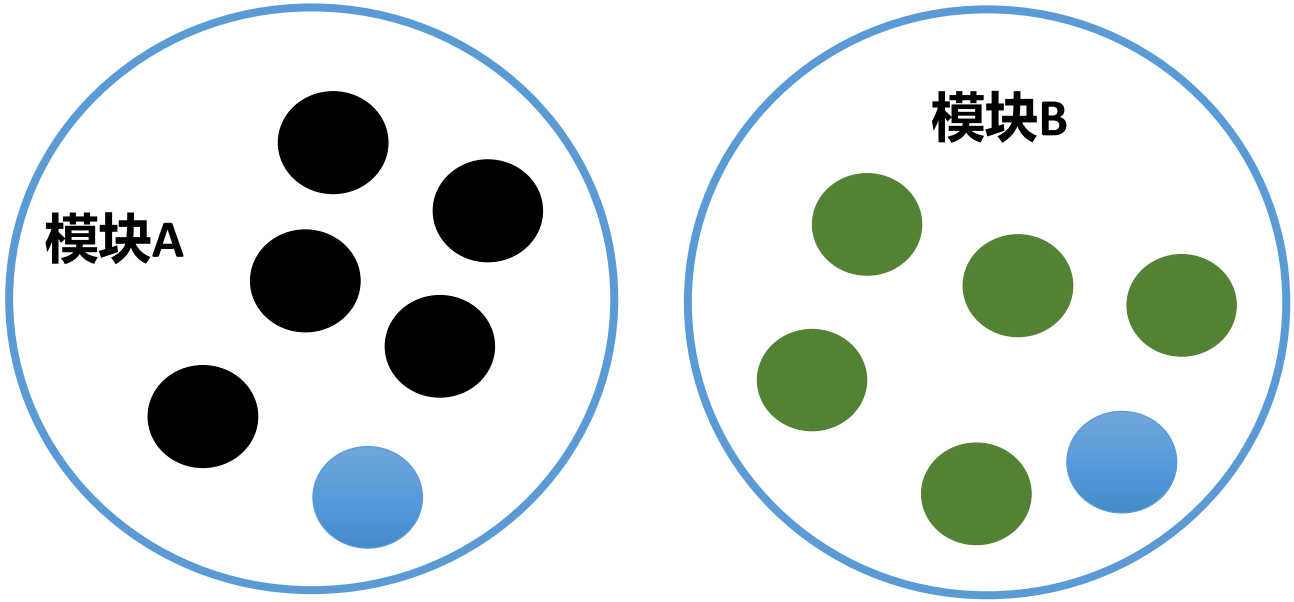
将密切相关的内容放在一起

如何实现高内聚



只将密切相关的内容放在一起

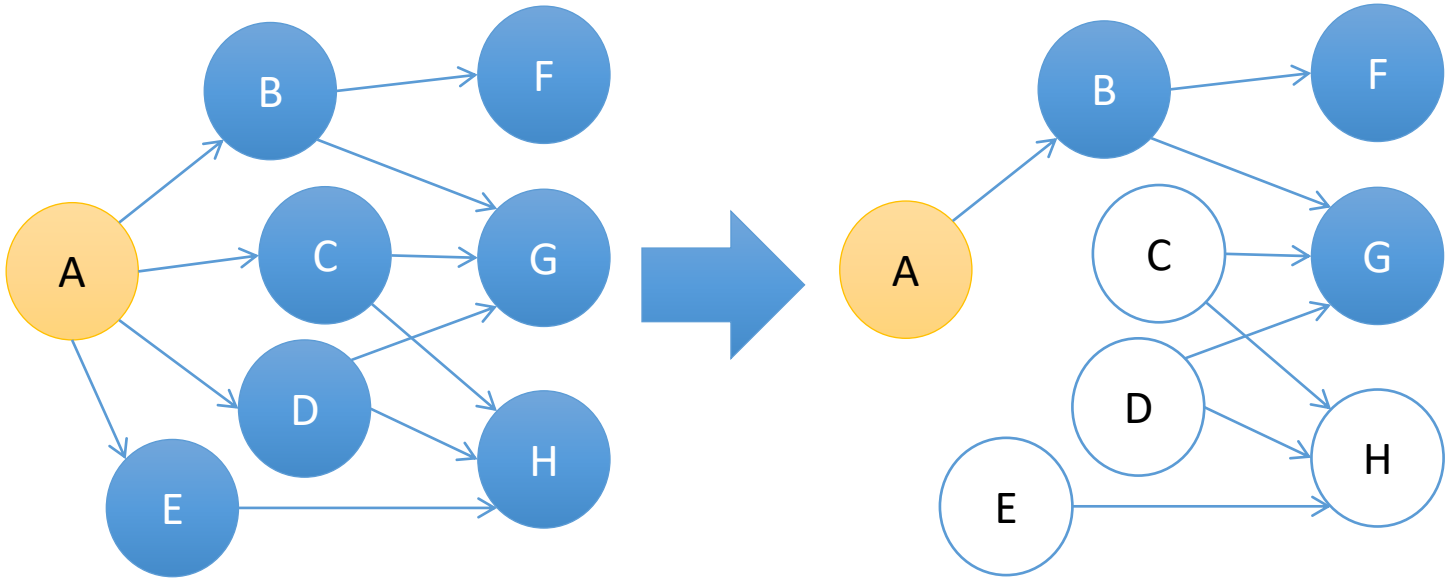
如何实现高内聚



重复意味着相关内容并不属于所在模块，而是影响很多模块（横切）！

消除重复

高耦合导致难以复用

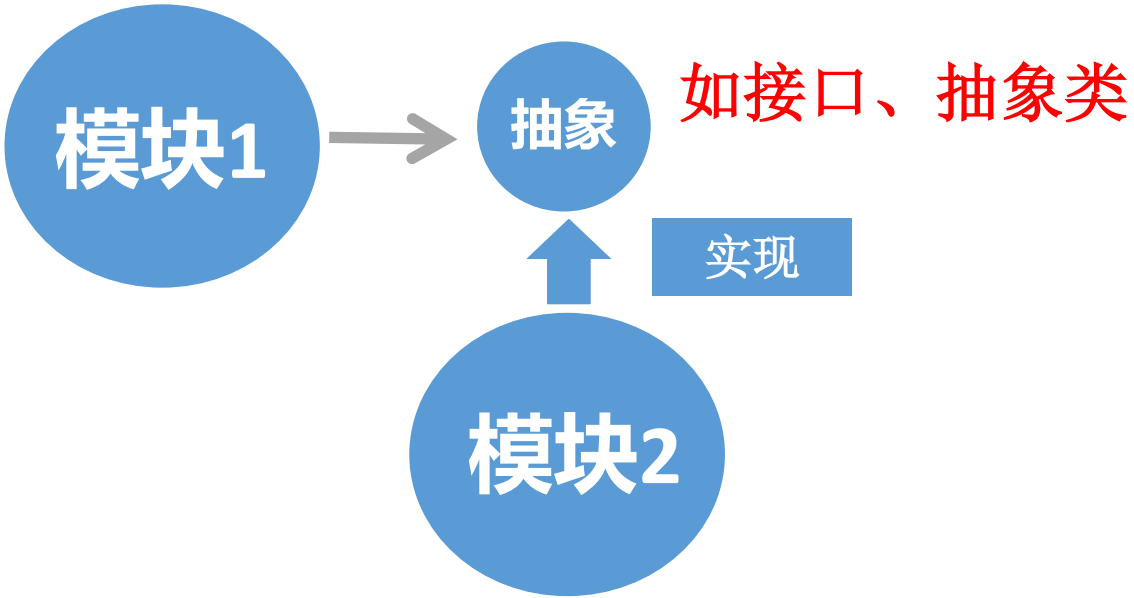


逻辑结构及模块间边界不清、粘连严重，外部依赖过多

形象比喻：大泥球、一团毛线

如何实现低耦合

- 最少知识原则（不要和陌生人说话）
- 减少不必要的依赖
- 依赖于稳定（抽象）的东西



重构

重构是以一种不改变代码外部行为、同时改进其内部结构的方式对软件系统进行修改的过程。

Martin Fowler

为什么重构?

- 软件的外部质量与内部质量可能会严重分离
- 持续的修改和演化会使得代码质量发生退化
- 内部质量对于未来的维护至关重要（成本、质量等方面）

重构：消除代码坏味道（bad smell）

1. Duplicate Code
2. Long Method
3. Large class
4. Long Parameter List
5. Divergent Change
6. Shotgun Surgery
7. Feature Envy
8. Data Clumps
9. Primitive Obsession
10. Switch Statements
11. Parallel Inheritance Hierachies
12. Lazy Class
13. Speculative Generality
14. Temporary Field
15. Message Chains
16. Middle Man
17. Inappropriate Intimacy
18. Alternative Classes with Different Interfaces
19. Incomplete Library Class
20. Data Class
21. Refused Bequests
22. Comments

阅读建议

- 《代码大全》3.5、第5章

快速阅读后整理问题
在QQ群中提出并讨论

CS2001
软件工程

End

8. 软件设计
—软件设计概述