软件设计基础

刘钦

Outline

- 什么是软件设计
- 软件设计的分层
- 软件设计过程、方法和模型、描述

Outline

- 什么是软件设计?
  - 软件设计思想的发展
  - 软件设计的核心思想
  - 理解软件设计
- 软件设计的分层
- 软件设计过程、方法和模型、描述

讨论

- 什么是软件设计?
- 为什么要做设计?
  - 软件设计与编程是什么关系?
  - 需求分析与软件设计是什么关系?
- 怎么做设计?

体系结构设计

产品线

框架

大规模软件设计

重构

构件

为复用而设计

信息隐藏

设计模式

结构化设计

面向对象设计

抽象数据类型

独立于软件实现

软件设计

自顶向下

与过程的关系

逐步求精

函数

程序设计

与人的关系

软件

过程

程序

软件设计思想的发展

Outline

- 什么是软件设计?
  - 软件设计思想的发展
  - 软件设计的核心思想
  - 理解软件设计
- 软件设计的分层
- 软件设计过程、方法和模型、描述

为什么要设计?

- 事物的复杂性 VS 思维的有限性
  - 7±2规则
  - 关注点分离与层次性

Why software is inherently complex?

“The complexity of software is an essential property, not an accidental one”

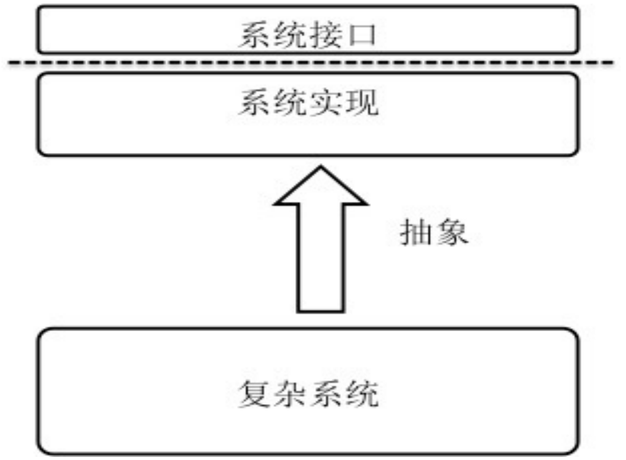
-- F. Brooks

Why software is inherently complex?

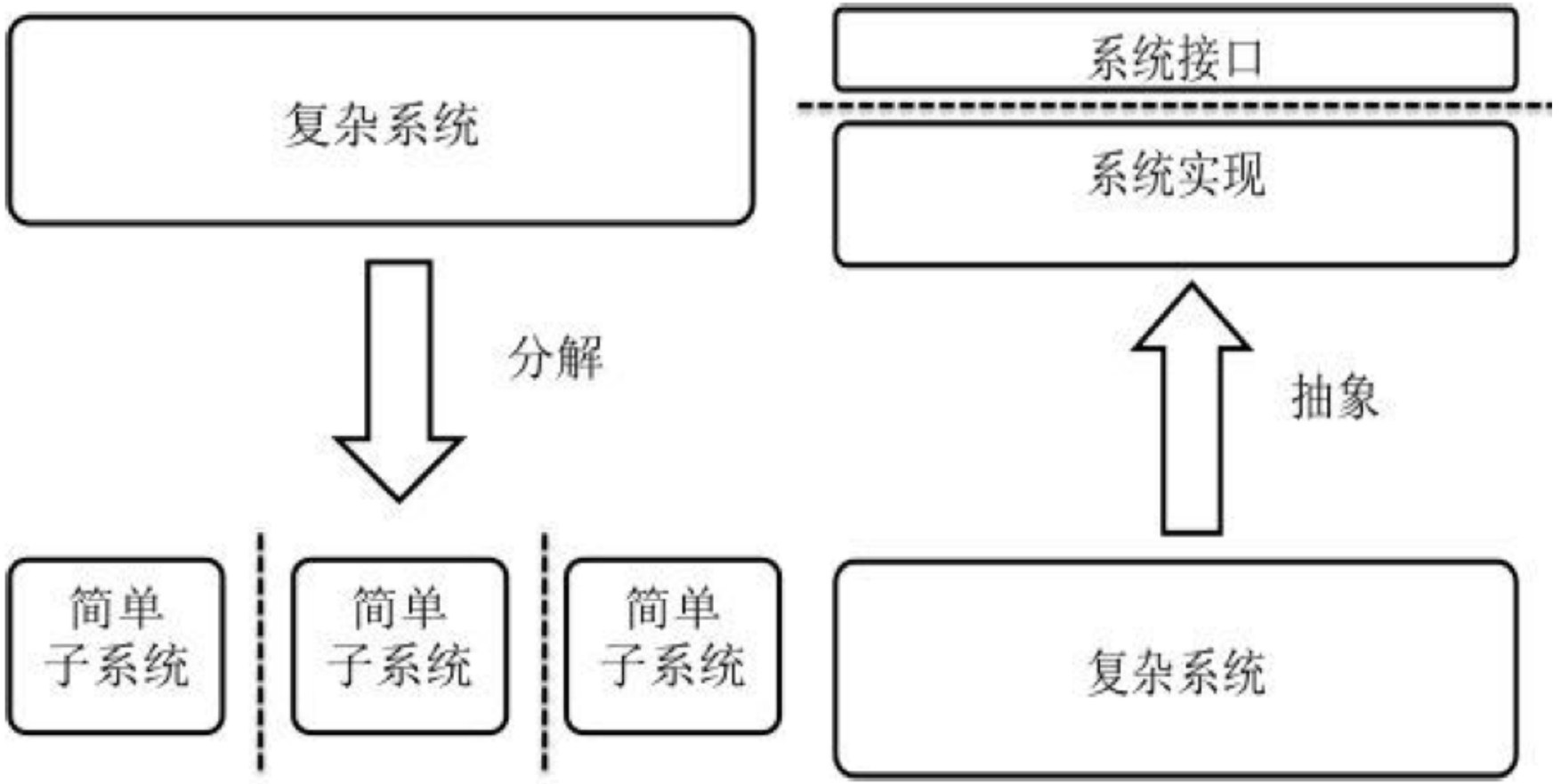
- The complexity of the problem domain
- The difficulty of managing the development process
- The flexibility possible through software
- The problems of charactering the behavior of discrete systems

设计的复杂度

- 事物复杂度 VS 载体复杂度
  - 对外表现 与 内部结构
- 设计复杂度 = 事物复杂度+ 载体与事物的适配复杂度

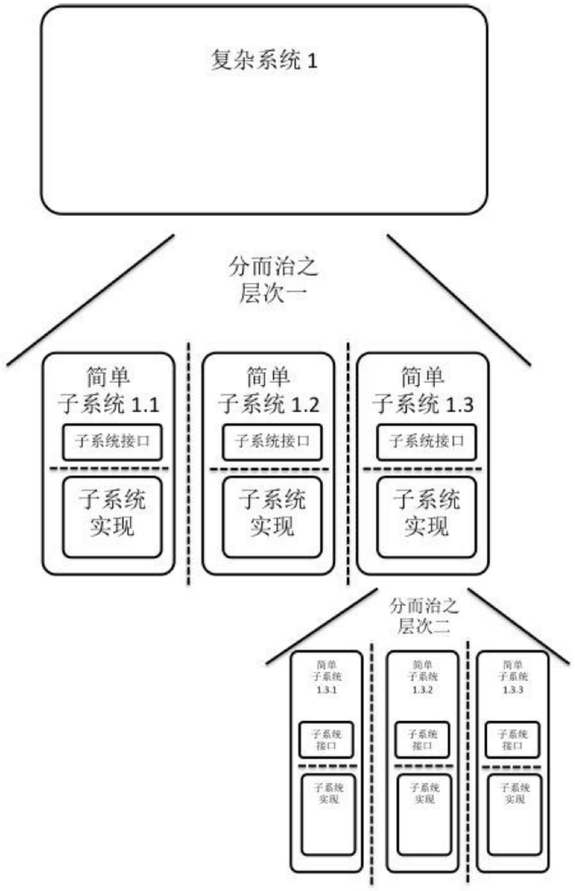


How to control the complexity?



软件设计的核心思想

分解与抽象



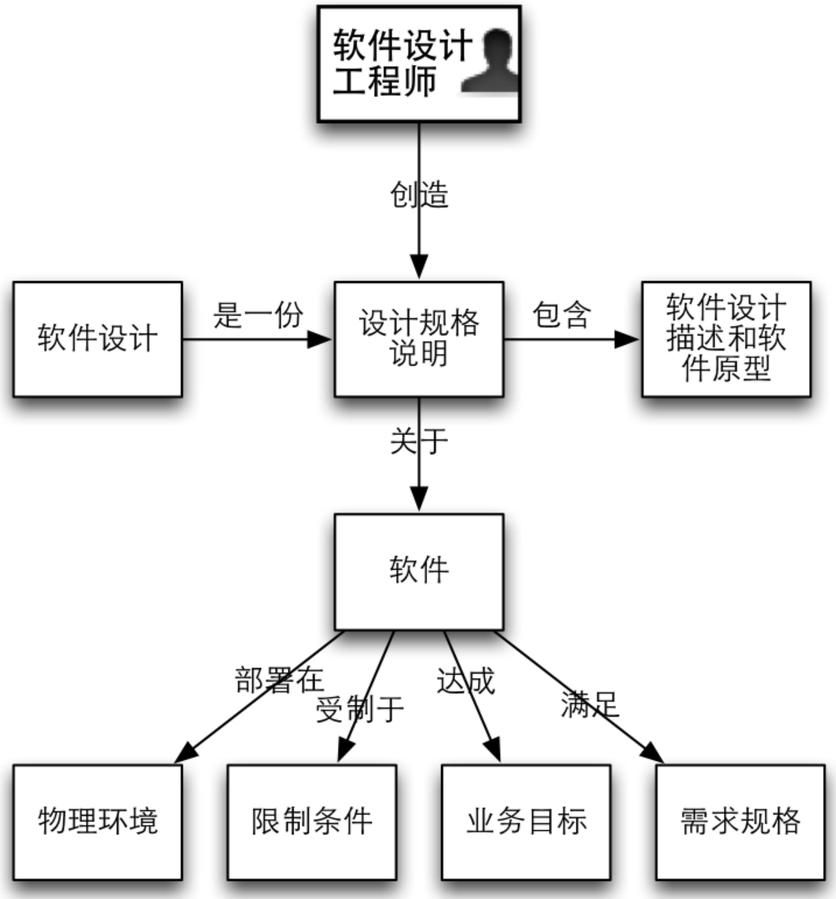
分解与抽象的并用和层次性

Outline

- 什么是软件设计?
  - 软件设计思想的发展
  - 软件设计的核心思想
  - 理解软件设计
- 软件设计的分层
- 软件设计过程、方法和模型、描述

什么是设计?

- [wiki]Design is the planning that lays the basis for the making of every object or system.
- [Brooks 2010]认为上述定义的精髓在于计划、思维和后续执行。
- [Ralph 2009]将设计定义为:
  - 设计(名词):一个对象的规格说明。它由人创造,有明确的目标,适用于特殊的环境,由一些基础类型构件组成,满足一个需求集合,受一定的限制条件约束。
  - 设计(动词):在一个环境中创建对象的规格说明。


软件设计
工程师
创造
软件设计
是一份
设计规格
说明
包含
软件设计
描述和软
件原型
关于
软件
部署在
受制于
达成
满足
物理环境
限制条件
业务目标
需求规格

软件设计

什么是设计?

- Designing often requires a designer to consider the aesthetic, functional, and many other aspects of an object or a process, which usually requires considerable research, thought, modeling, interactive adjustment, and re-design.

- 设计经常需要一个设计师考虑一个对象或过程的审美、功能以及其他方面，这通常需要进行相当的研究、 思考、 建模、 交互调整和重新设计。

Engineering design or Art design?

工程设计与艺术设计

- [Faste2001]认为软件设计要:时刻保持以用户为中心,为其建造有 用的软件产品;将设计知识科学化、系统化,并能够通过职业教育产生合格的软件设计师; 能够进行设计决策与折中,解决设计过程中出现的不确定性、信息不充分、要求冲突等复杂 情况。

- Vitruvius(《De Architectura》,公元前 22 年)认为好的建筑架构要满足“效用(Useful)、 坚固(Solid)与美感(Beautiful)”三个方面的要求。

- Brooks2010]认为重要的美感因素包括:简洁、结构 清晰和一致。

- Smith1996]认为在软件设计(尤其是人机交互设计)中艺术始终都处于中心地位,比工程性更加重要,为此设计师需要学会:发散性思维和创新;与用户共情,体会他们的内心感受;进行相关因素的评价和平衡,例如可靠性与时尚(Fashion)、简洁性与可修改性等;构思与想象,设计软件产品的可视化外观(Visualization)。

Design = Engineering +Art

Rational or Pragmatic?
理性主义or经验主义?

理性主义代表

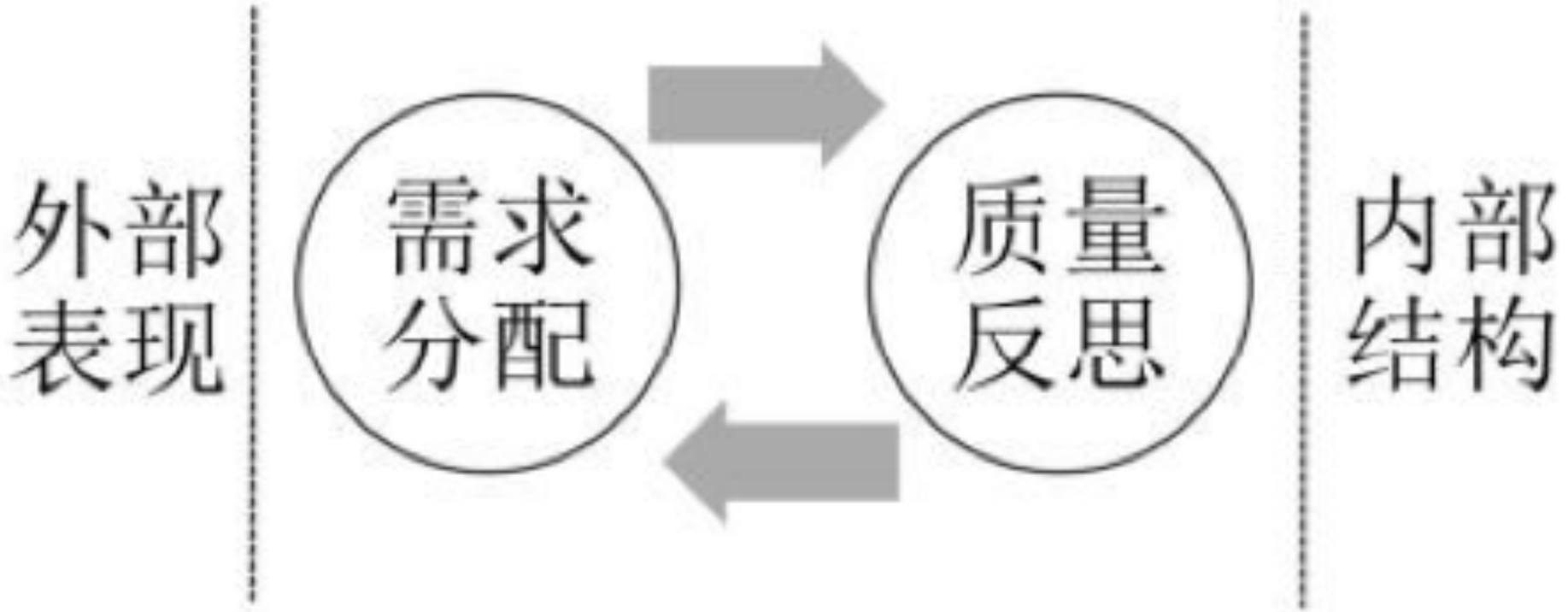
- 理性主义更看重设计的工程性,希望以科学化知识为基础,利用模型语言、建模方法、 工具支持,将软件设计过程组织成系统、规律的模型建立过程[McPhee1996]。
- 在考虑到人的 因素时,理想主义认为人是优秀的,虽然会犯错,但是可以通过教育不断完善自己 [Brooks2010]。
- 设计方法学的目标就是不断克服人的弱点,持续完善软件设计过程中的不足,最终达到完美[McPhee1996]。
- 形式化软件工程的支持者是典型的理想主义。

经验主义代表

- 经验主义者则在重视工程性的同时,也强调艺术性,要求给软件设计过程框架添加一些灵活性以应对设计中人的因素。

- [Parnas1986]曾指出没有过程指导和完全依赖个人的软件设 计活动是不能接受的,因为不能保证质量和工程性。但是[Parnas1986]也指出一些人的因素 决定了完全理性的设计过程是不存在的: 1 用户并不知道他们到底想要怎样的需求; 2 即使 用户知道需要什么,仍然有些事情需要反复和迭代才能发现或理解; 3 人类的认知能力有限; 4 需求的变更无法避免; 5 人类总是会犯错的; 6 人们会固守一些旧有的设计理念; 7 不合 适复用。

- 所以,[Parnas1986]认为软件设计需要使用一些方法弥补人的缺陷,以建立一个尽 可能好的软件设计过程。文档化、原型、尽早验证、迭代式开发等都被实践证明能够有效弥 补人类的缺陷[Parnas1986, Brooks 2010]



软件设计的演化性

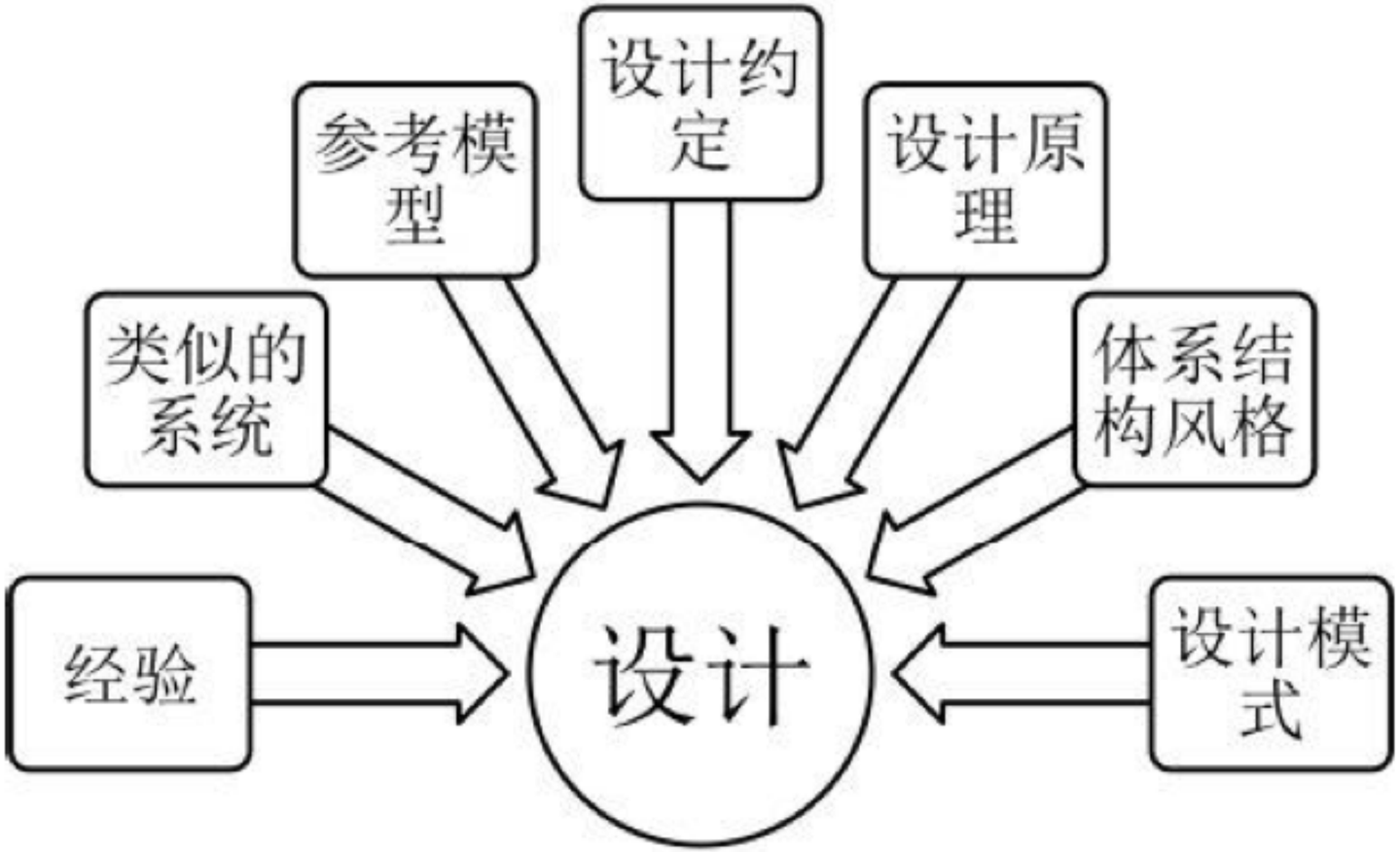
设计的决策

- [Freeman1980]认为:软件设计是一种问题求解和决策的过程;问题空间是用户的需求 和项目约束,解空间是软件设计方案;从问题空间到解空间的转换是一个跳跃性的过程,需 要发挥设计师的创造性,设计师跳跃性地建立解决方案的过程被称为决策。

- 软件设计的问题求解与决策比普通数学问题的求解与决策要困难的多,因为软件设计面对的问题通常都是不规则的(Ill-Structured)[Simon1978],包含有很多的不确定性和信息不充分情景,所以进行设计决策时并不能保证决策的正确性,往往需要在很长时间之后才能通 过验证发现之前决策的正确与否。

决策的约束性

- 约束满足与决策
  - 约束：
    - 需求；环境；资源；技术…
    - An initial need determines the most basic constraints and requirements on a design situation.
    - In general, more constraints are eventually discovered during the design work itself.
    - The constraints that apply both to the designed artifact and to the processes and participants involved during the design activity
  - 约束满足



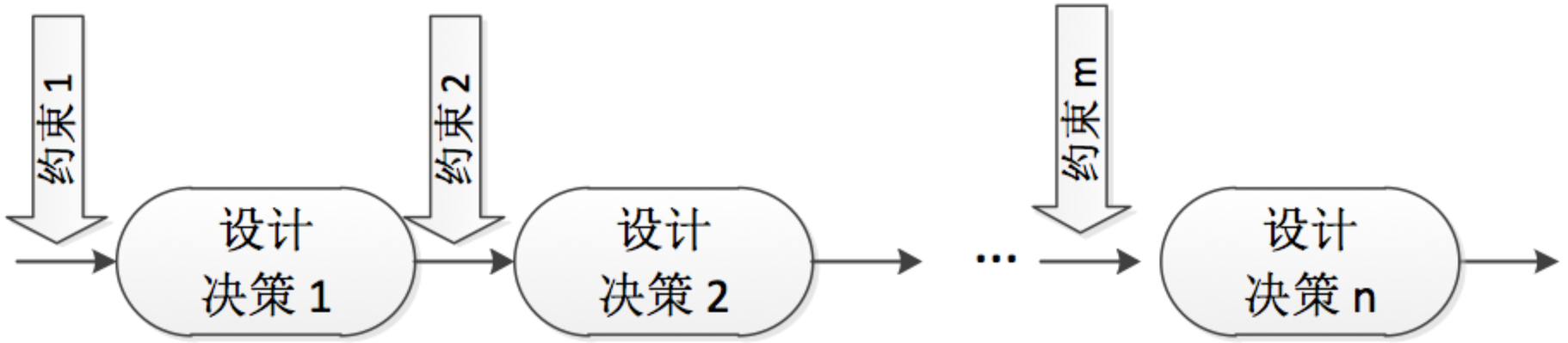
设计的决策依据

决策的多样性

- 决策的选择性
- 多个同样好的方案，选择一个
- The solution space for design problems is very large and its sheer size eliminates exhaustive search as a possible problem solving technique
- “design” is characterized by a series of decisions between various design alternatives

决策的演化性

- 遵循路线：问题决策验证下一个问题
- 决策的顺序影响
- 前一个决策会影响后一个，而且不可预见
- Each divergent perspective may influence the progress of the design in different and unpredictable ways.
- 决策的不可逆性
- 无法完全消除一个前期错误的决策
- 决策要一致！

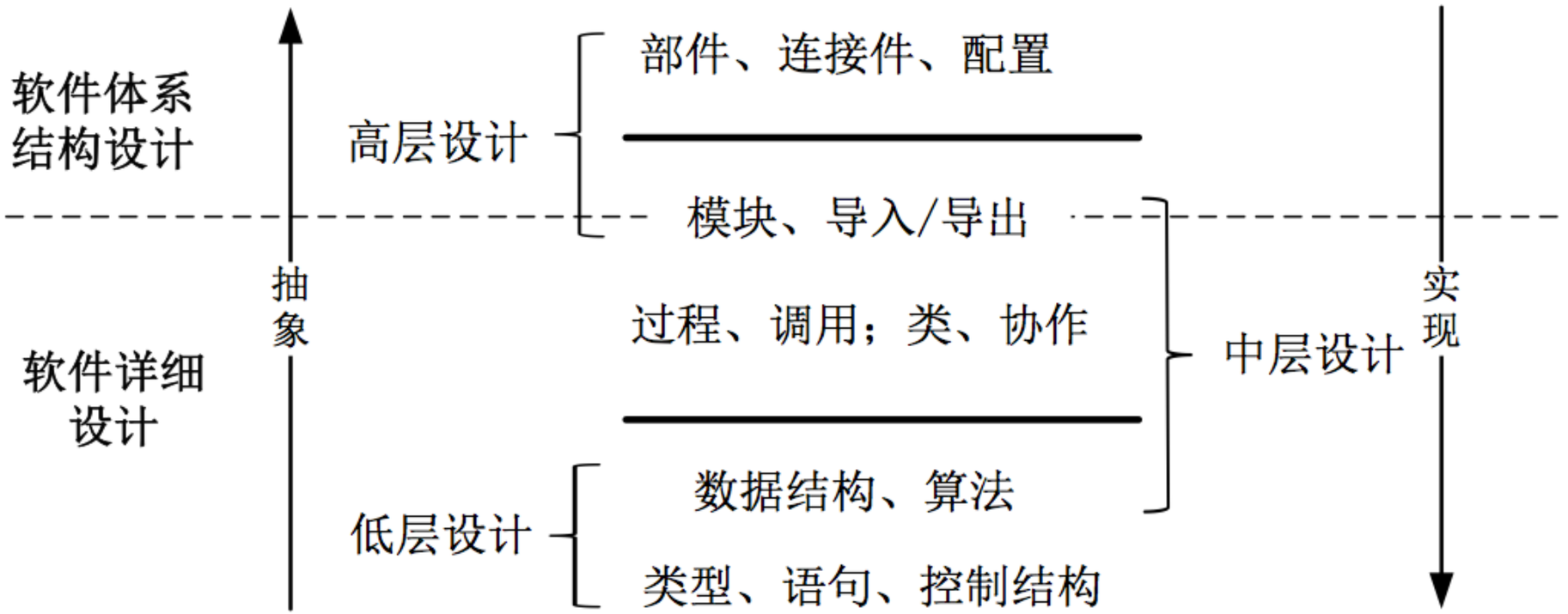


设计的演化性

决策的概念完整性也十分重要!

Outline

- 什么是软件设计?
- 软件设计的分层
  - 低层设计
  - 中层设计
  - 高层设计
- 软件设计过程、方法和模型、描述


软件体系
结构设计
软件详细
设计
抽象
实现
高层设计
部件、连接件、配置
模块、导入/导出
中层设计
过程、调用；类、协作
数据结构、算法
低层设计
类型、语句、控制结构

软件设计的分层

程序设计的建立

- 1950s
  - 第一代语言，第二代语言
  - 语句为最小单位
- 1960s
  - 第三代语言
  - 类型与函数：第一次复杂系统分割
- 1970s
  - 类型与函数的成熟：形式化方法
  - 数据结构+算法=程序

低层设计

- 将基本的语言单位（类型与语句），组织起来，建立高质量的 数据结构+算法
- 常见设计场景：
  - 数组的使用，链表的使用，内存的使用，遍历算法，递归算法…
- 经典场景：
  - 堆栈，队列，树，排序算法，查找算法…
- 数据结构与算法审美：
  - 简洁、结构清晰，坚固（可靠、高效、易读）
  - 数据结构与算法　课程
  - <计算机程序设计艺术>

低层设计的本质

- 屏蔽程序中复杂数据结构与算法的实现细节！

抽象层 **接口层**	数据结构的含义与使用	算法的语义与 复杂度
精化层 **实现层**	结构的类型定义 存储空间的使用与更改	书写控制语句 分支的处理技巧 对数据的操纵

低层设计：代码设计

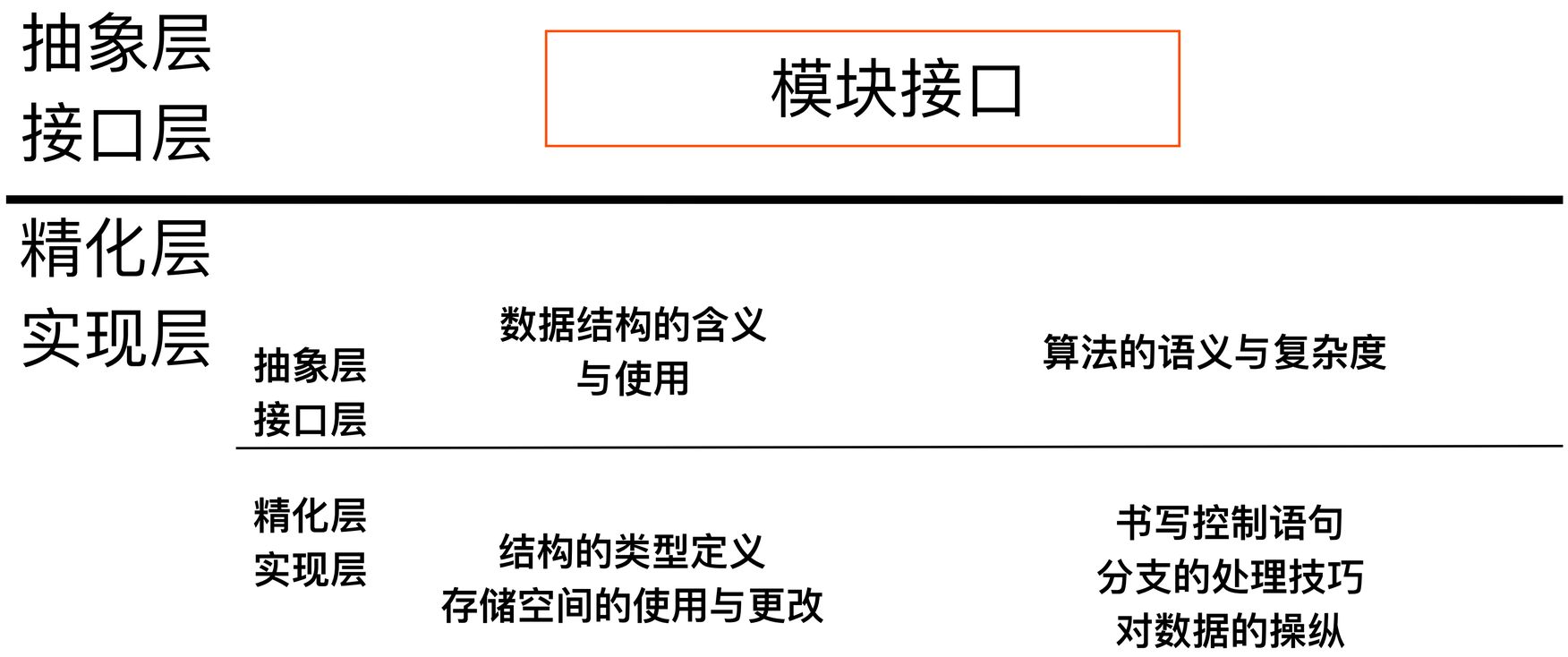
- 对一个方法/函数的内部代码进行设计
- 又被称为软件构造“software construction”，通常由程序员独立完成
- 依赖于语言提供的机制（程序设计课程）
  - OO or Structural
  - Reference or Address Point
  - …

模块划分

- 1970s
  - 函数的成熟与模块的出现
- 模块划分
  - 将系统分成简单片段
    - 片段有名字，可以被反复使用
- 名字和使用方法称为模块的抽象与接口
- 模块内部的程序片段为精化与实现

中层设计的开始

- 模块划分隐藏一些程序片段（数据结构+算法）的细节，暴露接口于外界

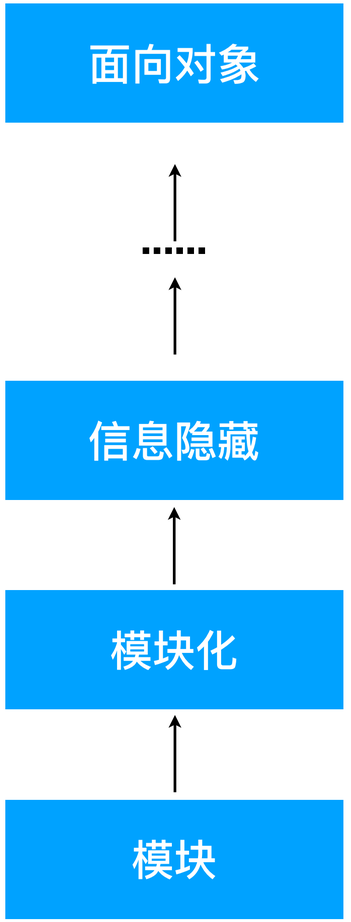


模块化的目标：完全独立性

- 完全独立有助于
  - 理解
  - 使用与复用
  - 开发
  - 修改

模块化的问题与困难

- 程序片段之间不可能是完全独立的
- 方法：实现尽可能的独立
  - 模块化
  - 信息隐藏
  - 抽象数据类型
  - 封装
  - …



中层设计总结——设计目标

- 最终审美目标：简洁性、结构清晰、一致性、质量（可修改、易开发（易理解、易测试、易调试）、易复用）

- 直接评价标准：模块化；信息隐藏；OO原则

抽象层
接口层

对象、模块

精化层
实现层

抽象层 接口层	数据结构的含义 与使用	算法的语义与复杂度
精化层 实现层	结构的类型定义 存储空间的使用与更改	书写控制语句 分支的处理技巧 对数据的操纵

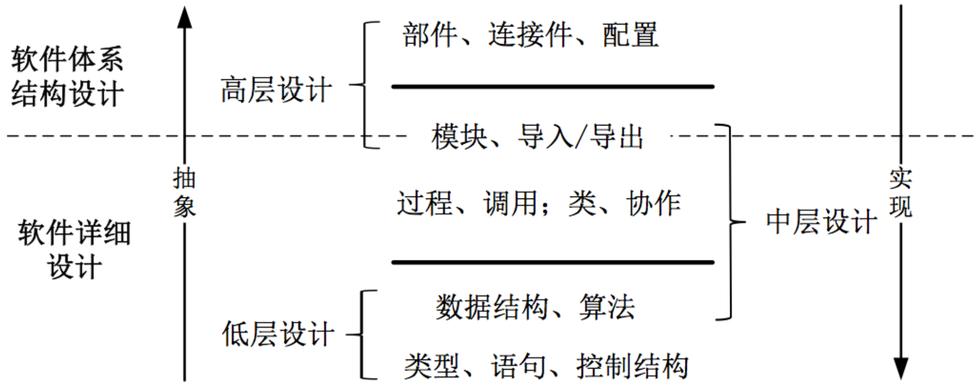
中低层设计的问题

- 《Programming-in-the-Small VS Programming-in-the-Large》
- 过于依赖细节
  - 连接与依赖，接口与实现
- 忽略的关键因素: 无法有效抽象部件的整体特性
  - 总体结构
  - 质量属性

大型软件开发的一个根本不同是它更关注如何将大批独立模块组织形成一个“系统”，也就是说更重视系统的总体组织

高层设计：体系结构

- 部件承载了系统主要的计算与状态
- 连接件承载部件之间的交互
- 部件与连接件都是抽象的类型定义（就像类定义），它们的实例（就像类的对象实例）组织构成软件系统的整体结构，配置将它们的实例连接起来
- 连接件是一个与部件平等的单位

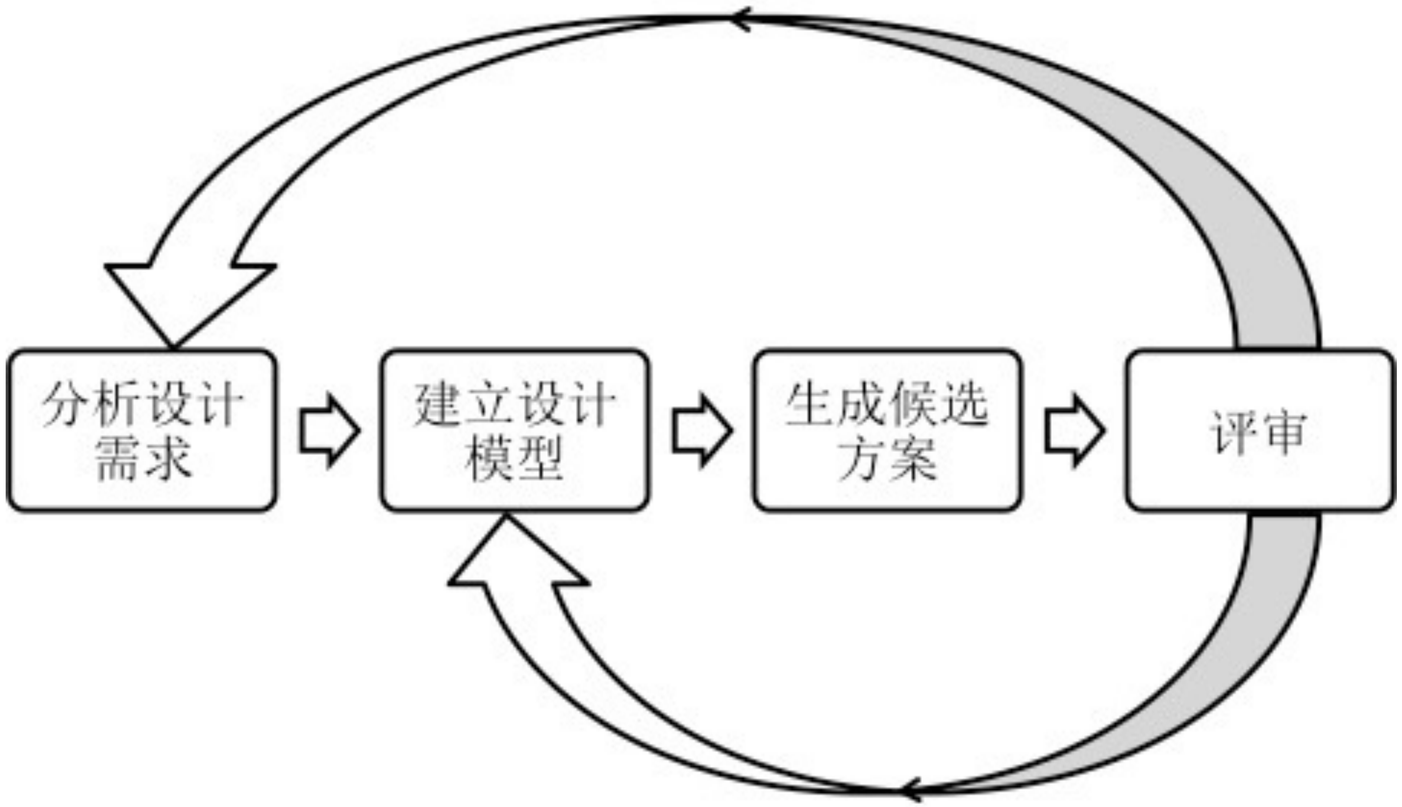


敏捷视点

- 只有高层设计良好，底层设计才可能良好
- 只有写完并测试代码之后，才能算是完成了设计！
- 源代码可能是主要的设计文档，但它通常不是唯一一个必须的。

Outline

- 什么是软件设计?
- 软件设计的分层
- 软件设计过程、方法和模型、描述
  - 软件设计过程的主要活动
  - 软件设计的方法和模型
  - 软件设计描述



设计的活动

Outline

- 什么是软件设计?
- 软件设计的分层
- 软件设计过程、方法和模型、描述
  - 软件设计过程的主要活动
  - 软件设计的方法和模型
  - 软件设计描述

设计的方法

- 软件设计的方法可以分为以下几种[SWEBOK 2004]:
  - 结构化设计方法、
  - 面向对象设计、
  - 数据为中心设计、
  - 基于构件的设计、
  - 形式化方法设计。

静态模型vs动态模型

- 描述软件设计的模型,通常可以分为两类:
  - 静态模型
    - 静态模型是通过快照 的方式对系统中时间不变的属性进行描述。通常描述的是状态,而不是行为。
      - 比如:一个数 字的列表是按大小排序好的。
  - 动态模型。
    - 动态模型通常描述的是系统行为和状态转移。
      - 比如:排序的 过程中如何进行排序。

结构化设计模型和面向对象设计模型

- 在结构化设计中
  - 静态模型
    - 实体关系图
  - 动态模型
    - 数据流图和结构图(Structure Chart)
- 在面向对象设计中
  - 静态模型
    - 类 图、对象图、构件图、部署图;
  - 动态模型
    - 交互图(顺序图和通信图)、状态图、 活动图等

Outline

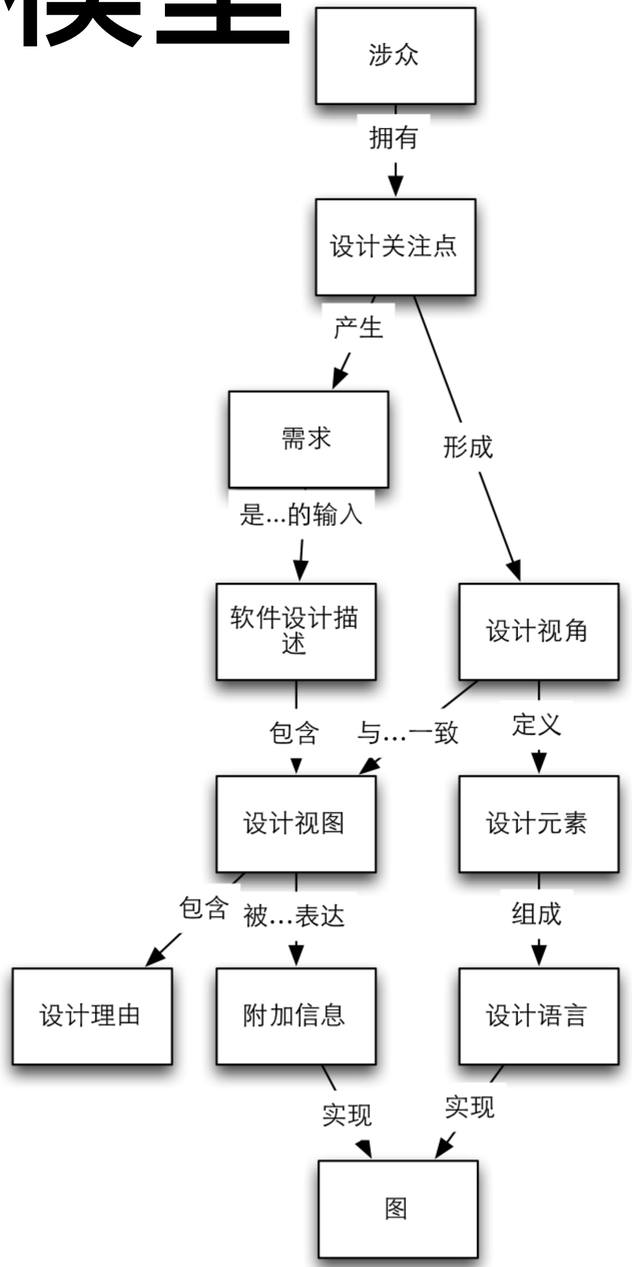
- 什么是软件设计?
- 软件设计的分层
- 软件设计过程、方法和模型、描述
  - 软件设计过程的主要活动
  - 软件设计的方法和模型
  - 软件设计描述

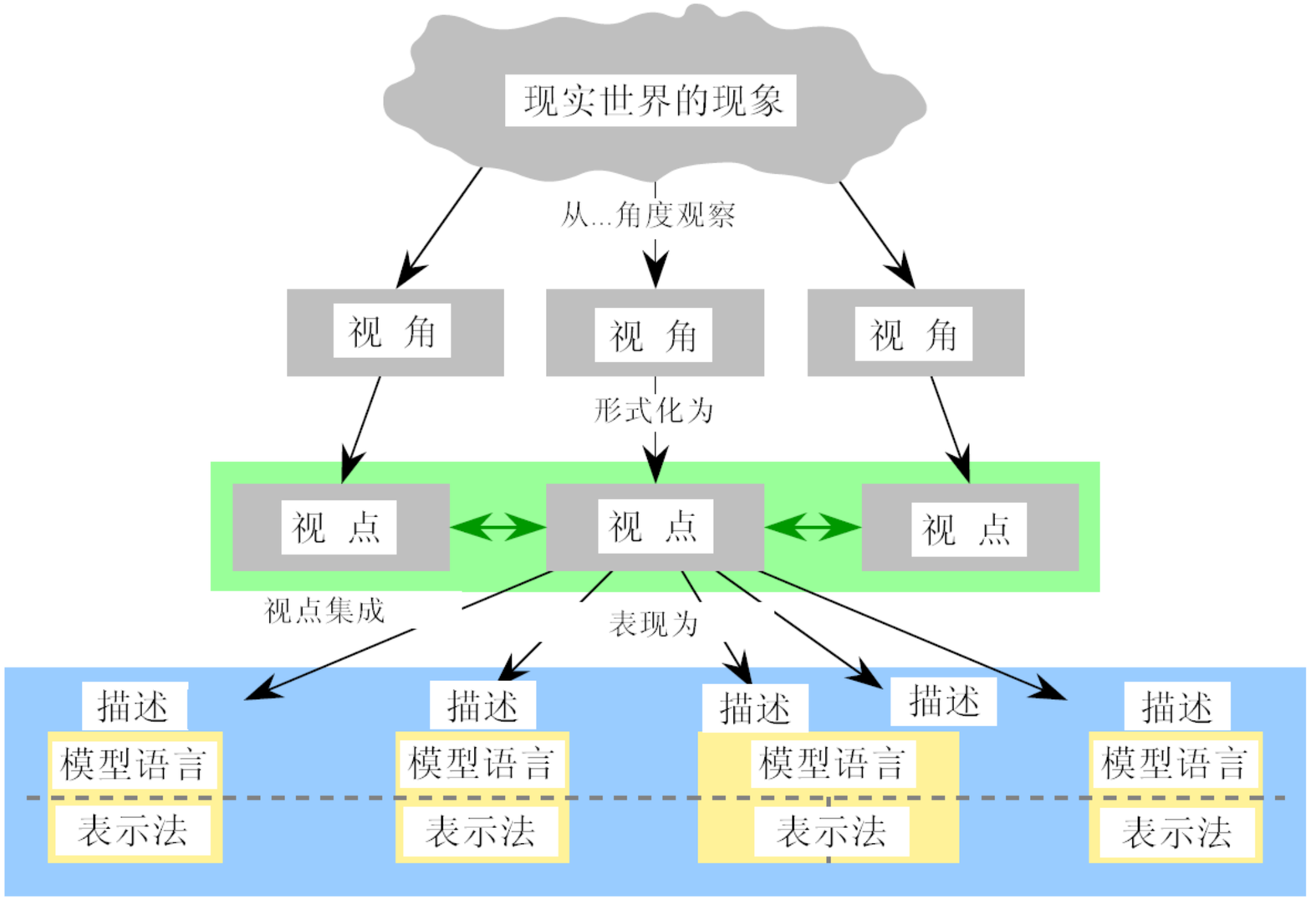
软件设计描述规范

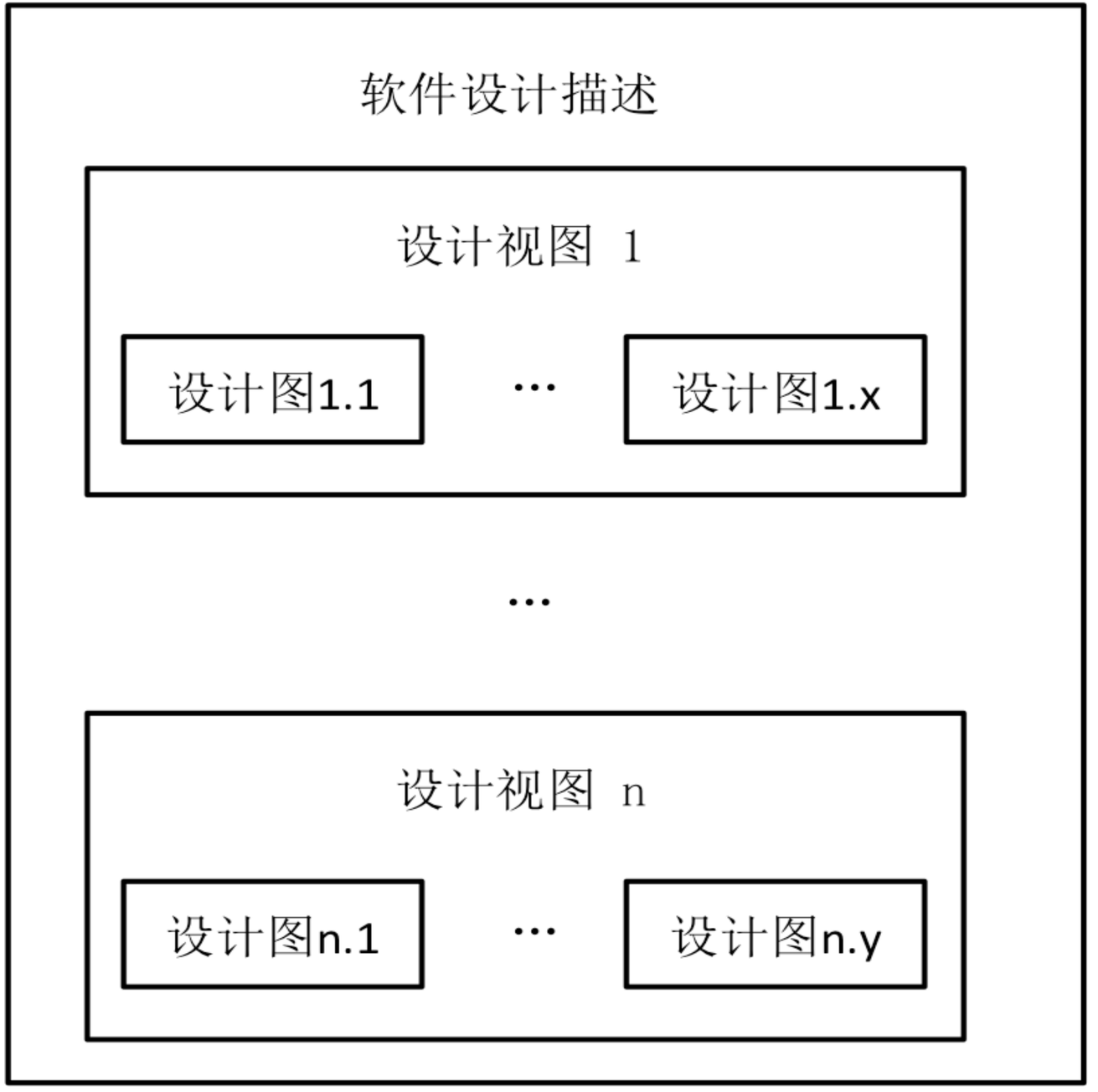
- 早在 1998 年,IEEE 组织就提出了 IEEE 1016-1998 号标准[IEEE1016-1998]以规范软件设 计的文档化描述。

- 在 2009 年 3 月,IEEE-SA 标准委员会修订了 1016 号标准[IEEE1016-2009]。 IEEE 的标准并没有限定设计描述文档的具体结构,而是通过解释一系列概念说明了软件设计 描述文档应该遵循的思路和基本内容。

软件设计描述模型


涉众
拥有
设计关注点
产生
形成
需求
是...的输入
软件设计描述
设计视角
包含
与...一致
定义
设计视图
设计元素
包含
被...表达
组成
设计理由
附加信息
设计语言
实现
实现
图


现实世界的现象
从...角度观察
视 角
视 角
视 角
形式化为
视 点
视 点
视 点
视点集成
表现为
描述
模型语言
表示法
描述
模型语言
表示法
描述
模型语言
表示法
描述
描述
模型语言
表示法


软件设计描述
设计视图 1
设计图1.1
…
设计图1.x
…
设计视图 n
设计图n.1
…
设计图n.y

设计视角	设计关注	样例设计语言
上下文（Context）	系统服务和用户	UML 用例图，结构化分析上下文图
组合（Composition）	功能分解和运行时分解、子系统的构造、购买 vs 建造，构件的复用	UML 包图，构件图，体系结构定义语言 ADL
逻辑（Logical）	静态结构（类，接口，及其之间关系），类型和实现的复用（类、数据类型）	UML 类图，对象图
依赖（Dependency）	互联，分享，参数化	包图、构件图
信息（Inoformation）	持久化信息。	关系实体图，类图
模式（Patterns）	模式和框架的重用	UML 组织结构图
接口（Interface）	服务的定义，服务的访问	接口定义语言 IDL
结构（Structure）	设计主体的内部构造和组织	UML 结构图，类图
交互（Interaction）	对象之间的消息通讯	顺序图，通讯图
动态状态（StateDynamics）	动态状态的转移	状态图
算法（Algorithm）	程序化逻辑	决策表
资源（Resources）	资源利用	UML OCL

设计视角

利益相关者

- 设计视角必须符合在需求(Requiement)中利益相关者(Stakeholder)的设计关注点 (Design Concern)。
- 设计对不同的利益相关者来讲,有着不同的意义[Pfleeger 2009]:
    - 顾客想要确认现有的软件设计方案能够保证其所期望的系统的功能和行为可以实现。
    - 架构师想要知道软件单元、计算平台和系统环境足够实现需求中要求的非功能性需求。
    - 设计人员想要了解系统的整体设计,以确保系统的每个设计决策和系统功能都能实现。
    - 开发人员想要知道所开发的单元的精确描述,以及与其它单元的关系。
    - 测试人员想要确认怎样测试设计的所有方案。
    - 维护人员系统想要在修复问题和添加新特性时,能够保持体系结构的完整性和设计的一致性。
- 因为每个利益相关者目的不一样,对设计就会产生不同的设计关注点。而我们的设计得全面考量各个利益相关者的设计关注点,给出相应的设计视角下的设计视图。

难点	YourTour 系统将应用于少量（可能少于 50）必须定期更新的业务规则。它的目的是允许业务用户（而不是开发人员）更新这些规则。这些规则将用于检查某些条件（例如根据以往的预定是否可以得到折扣）或执行计算（例如计算一个旅行线路的全部费用，包括所有的税和附加费）
架构决策	开发一个定制规则引擎组件，而不是使用一个封装的解决方案或将规则写入代码
假设	需要管理的业务规则数量相对较少（少于 50 条），并且变化频率较低（每年少于两次）
可替代方法	选项 1：在代码中嵌入规则 选项 2：开发一个定制规则引擎组件 选项 3：购买规则引擎库
选取的选项	选项 2
理由	在代码中嵌入规则是一个糟糕的方法，因为这不符合软件工程分解问题的原则，而且使规则难于维护。购买一个规则引擎的性价比不高，因为规则的数量很少且发生变化的频率也不高

设计理由

设计描述文档模版

设计文档书写要点

- 充分利用标准的文档模版,根据项目特点进行适当的裁剪。
- 可以利用体系结构风格的图,让读者更容易把握高层抽象。
- 利用完整的接口规格说明定义模块与模块之间的交互。
- 要从多视角出发,让读者感受一个立体的软件系统。
- 在设计文档中应体现对于变更的灵活性。

Q&A