RUP大讲堂（第六讲） -软件架构的原理和实践原则

北京恒讯时代信息技术有限公司
肖勇
xiaoy@henxu.com

内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

问题-瓦萨战舰的故事

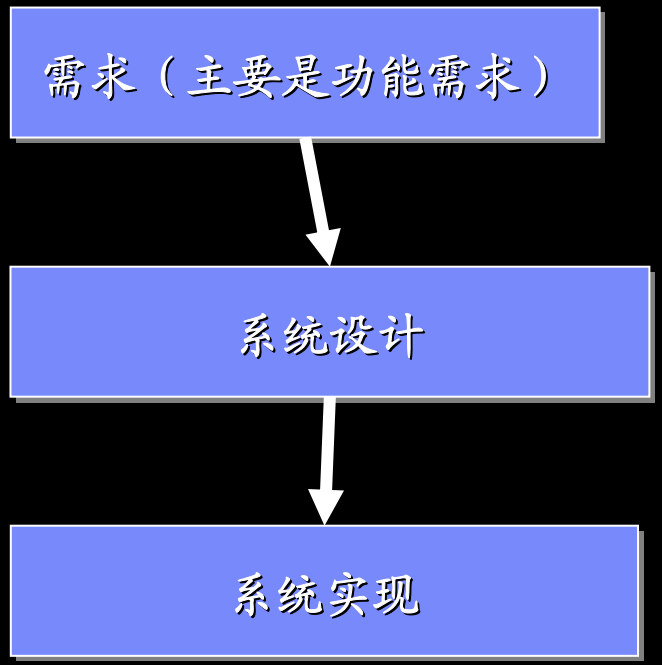
- 17世纪上半叶，北欧新教势力与中欧天主教势力发生了一场“三十年战争”，作为北欧新教势力的代表，瑞典的军事力量达到鼎盛时期。
- 1625年，号称“北方飓风”的瑞典国王古斯塔夫斯．阿道弗斯（Gustavs Adolphus）决心建造一艘史无前例的巨型新战舰——瓦萨（Vasa）战舰。瓦萨战舰确实是一艘令人望而生畏的战舰：舰长70米，载员300人，在三层的甲板上共装有64门重炮，火力超强。
- 1628年8月10日，这艘巨大的战舰终于完工。在斯德哥尔摩的王宫前，瓦萨战舰举行了盛大的下水典礼。礼炮声中，战舰扬帆起航，乘风前进。
- 在1万多名围观者的目光注视下，忽然，瓦萨号奇怪地摇晃了一下，便向左舷倾斜。海水从炮孔处涌入船舱，战舰迅速翻入水中，几分钟后，这艘雄伟战舰的处女航——也是唯一的一次航行结束了。瓦萨战舰在它壮丽的起航时刻，带着全身飘扬的彩旗，沉没于它诞生的港口。

问题-信息系统的“瓦萨”问题

- 瓦萨的故事已经过去300多年了，在船舶工业领域，作为学科和工业的基石——“架构”早已形成完整的理论和方法体系。瓦萨的故事，基本上不会重演了。
- 但是，在今天的软件系统领域，“瓦萨”问题依然是需要解决的关键问题。

问题-基本假设

- 体系结构提出之前的系统设计思路



特点:
技术性需求，特别是功能需求是产生设计的唯一（最主要的）的驱动力。
由此
- 非功能需求因素
- 非技术因素

的考虑很少。

基本假设:
设计是系统的技术需求分析的产物。

什么是软件架构-架构一词的来源

- 建筑行业：
  - 建筑学认为，所有的高楼大厦（复杂建筑），应该是由建筑结构、暖通系统、强电系统、弱电系统（监控系统、综合布线等）、给排水系统等构成。
  - 具体体现在建筑图、总平面图、综合管线、结构图、给排水、暖通、强电、弱电等图纸上。
  - 这种建筑学的思想方案，就是建筑设计的“架构体系”。
  - J建筑架构师需要把所有的层次结合起来：
    - 使客户理解
    - 在建造的过程中为施工者提供指导
- 架构相关于所有事情，架构为所有人提供一个共同的远景目标。
- 架构不包括每个部分的细节

内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

什么是软件架构-如何理解

- 架构是针对某种特定目标系统的具有体系性的、普遍性的问题而提供的通用的解决方案。
- 架构往往是对复杂系统的一种共性的体系抽象。
- 架构让我们能够正确、合理地理解、设计和构建复杂的系统。

- 理解1：高楼大厦是由钢筋、水泥和砖块构成。
- 理解2：信息系统是由数据和代码构成
- 理解1：高楼大厦是由一个个楼层、一个个房间构成。
- 理解2：信息系统是由一个个模块、一个个对象和组件构成。
- 理解1：高楼大厦是由支撑框架、管道系统、强弱电系统、给排水系统……等构成。
- 理解2：信息系统是由组织机构、业务流程、业务功能、业务信息……等构成。

什么是软件架构-定义

- 期望其与建筑架构起到相同的作用：
  - 将软件的所有层次组合在一起
  - 便于客户理解
  - 为建造过程提供指导
- 软件架构包含了过于下列方面的重要决定：
  - 软件系统的组成
  - 对所包括的系统及其接口的结构元素的选择，以及元素间的协作行为
  - 结构和行为元素如何组成不断增长的更大的子系统
  - 架构风格：组成元素与接口、相互协作、相互组合
- 架构元素不仅与结构和行为有关，也和用法、功能、性能、适应性、重用、可理解性、经济和技术的限制、折中、美学等有关

什么是软件架构-RATIONAL的定义

- 软件架构也关系到
  - 功能性Functionality；可用性Usability；系统弹性Resilience
  - 性能Performance；重用Reuse；可理解性Comprehensibility
  - 经济和技术的约束及相关折中Economic and technology constraints and tradeoffs；美学的考虑Aesthetic concerns
- 软件架构的描述包含
  - 构成系统的各个组件的描述
  - 组件间的交互（interactions）
  - 组件构成与组件合成的模式（pattern）以及在这些模式上的约束
- 注：前两部分的描述，对于任意由不同部分构成的系统而言都是需要的，软件体系结构作为一门学科，是将此提伸到设计层原则的高度，同时，用通过实践过程总结的模式作为设计的指导。

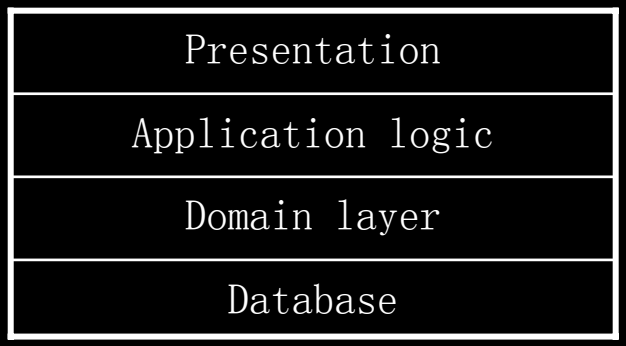
什么是软件体系结构-特征

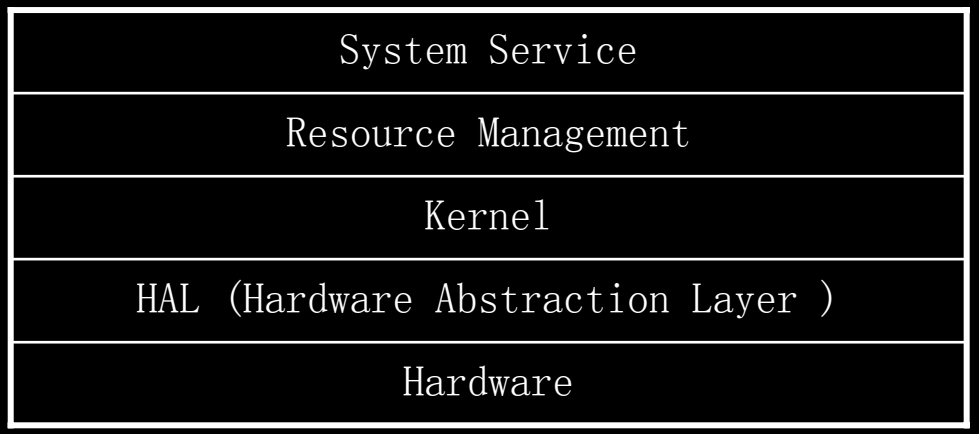
- 体系结构
  - 学科：经常面对的设计问题的抽象、形式化、准确的描述与严格的分析
  - 经验：从实践中浮现的、非正式的解决方法
  - 个性：目标系统的特性、隐含与显性的要求以及软件设计者（Architect）的习惯与个性
- 类比：素描的蓝图
  - 人体结构与比例的科学数据（事实上为统计数据）
  - 既定的风格（表现手法）
  - 个人的经验、积累与个性等
- 类似之处：将抽象或具体的系统模型化用另一种表达方式描述出来，体系结构即为这种模型的结构，忽略了细节而决定了细节。

什么是软件体系架构-复杂系统的架构本质

- 复杂系统的理解、设计和开发，普遍遵从层级理论的思路。诺贝尔奖获得者赫伯特.西蒙曾论述到：“*要构造一门关于复杂系统的比较正规的理论，有一条路就是求助于层级理论……复杂系统是层级结构的*”。
- **架构体系**就是一个由不同层级构成的、描述复杂系统的体系。

管理信息系统　层次结构：基于功能的划分　**操作系统：NT**





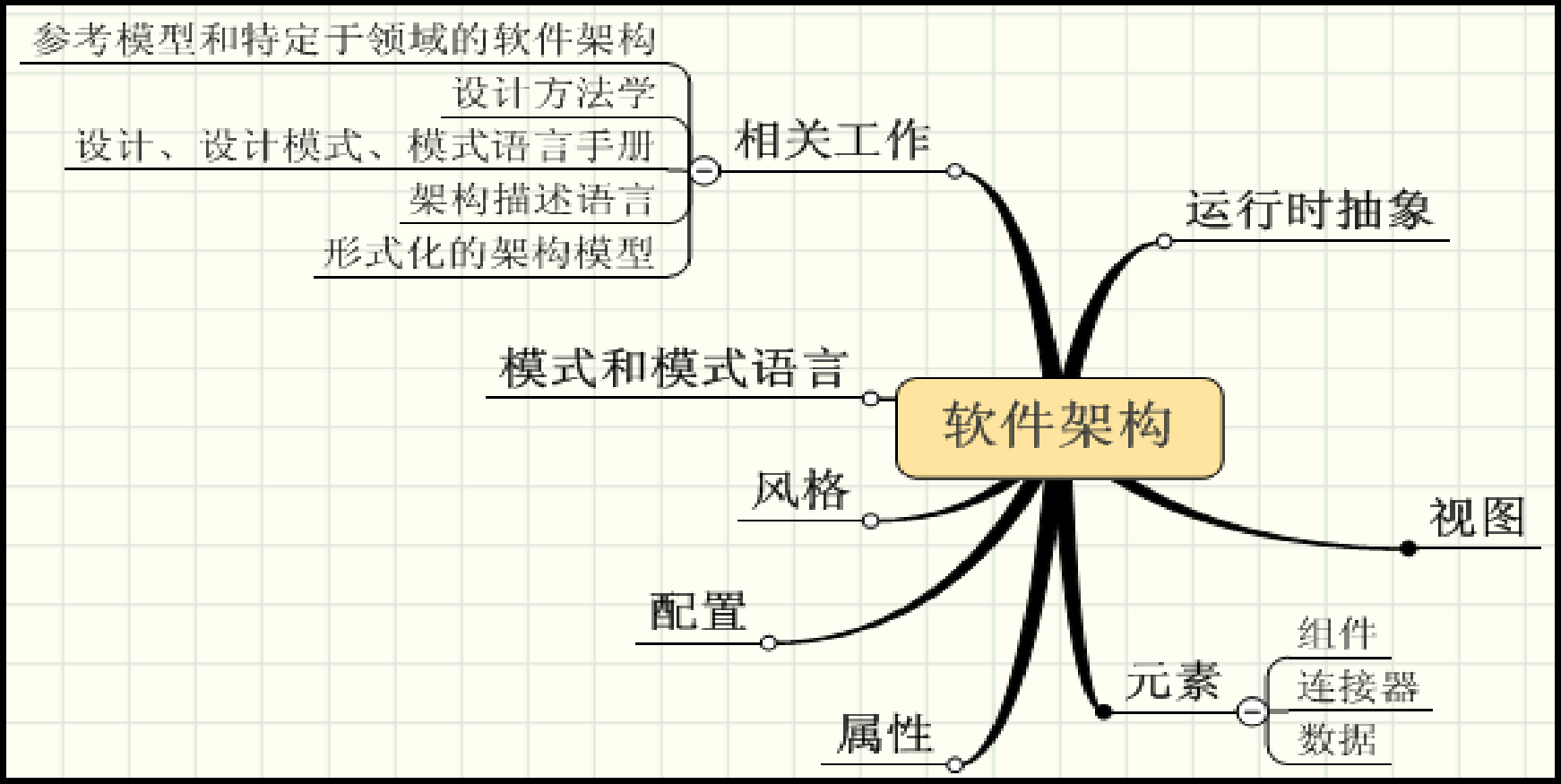
什么是软件体系架构-复杂系统的架构本质（续）

- 系统（不限于软件系统）设计划分为若干个层次，区分各个层次之间的区别，划分各层所需解决的设计问题是至关重要的。每个层次上定义
  - 组件：primitive and composite
  - 组成规则：composite组件或系统的构造规则
  - 行为规则：系统的语义，组件间交互的规则
- 这些规则在每一个层次上是不同的，每个层次上有各自的符号系统、设计问题、分析技术，由此，各个层次上的设计可以独立进行，而每一层是上一层的实现。

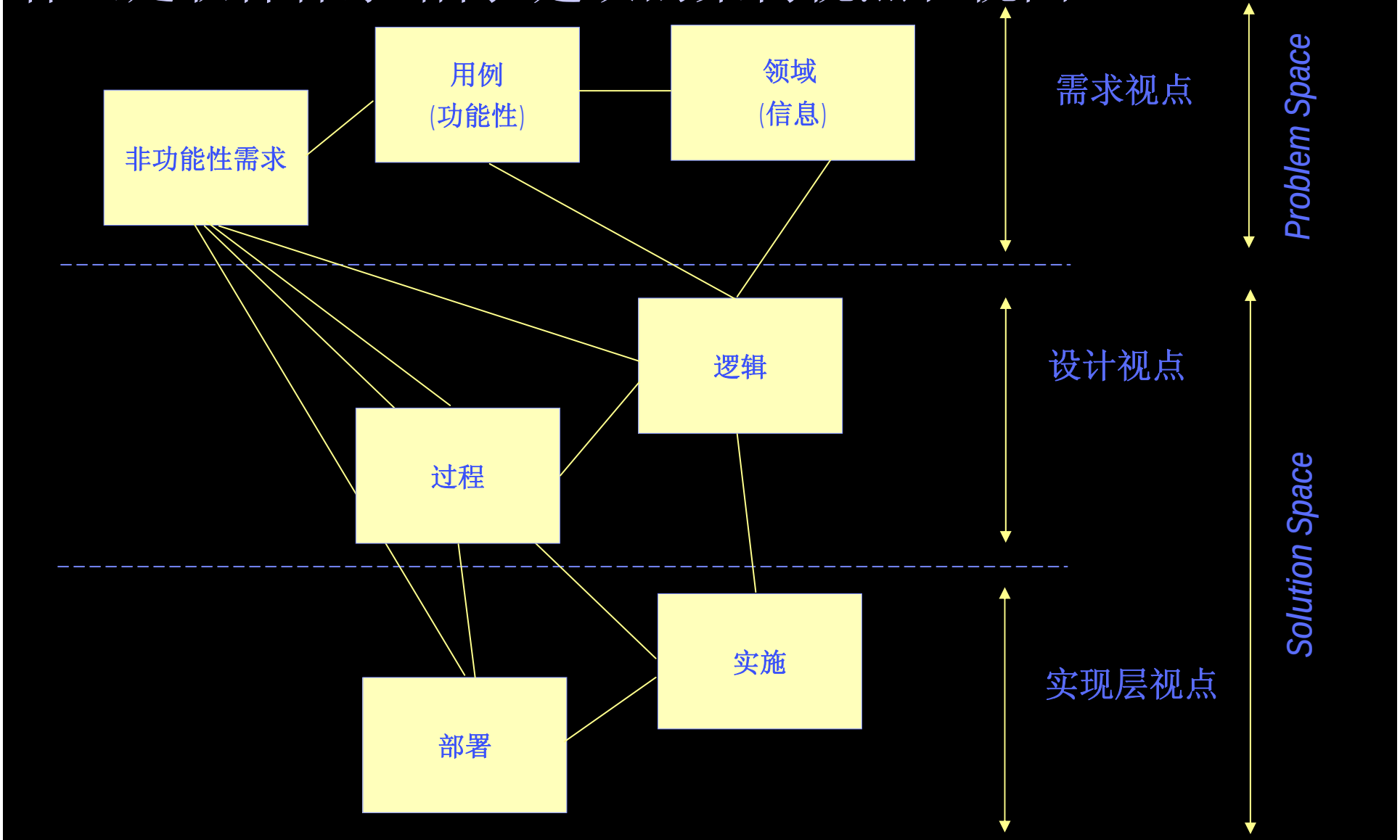
什么是软件体系结构-范围

- 体系结构是系统的抽象，用抽象的组件及其外部可见的性质与他们之间的关系来描述系统。
- 由于体系结构是抽象的，因此压缩了存局部信息，即组件的私有细节不属于体系结构的描述范围。
- 系统由许多结构构成，这些结构通常称为View(视图),一种视图只能描述从某个角度看到的系统，而非全部，同时，视图的集合不是固定的。

什么是软件体系结构-涉及的知识体系

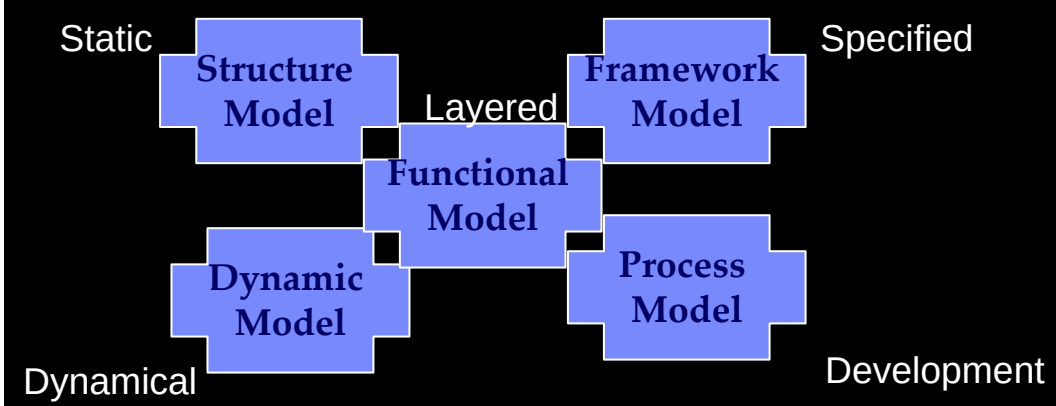


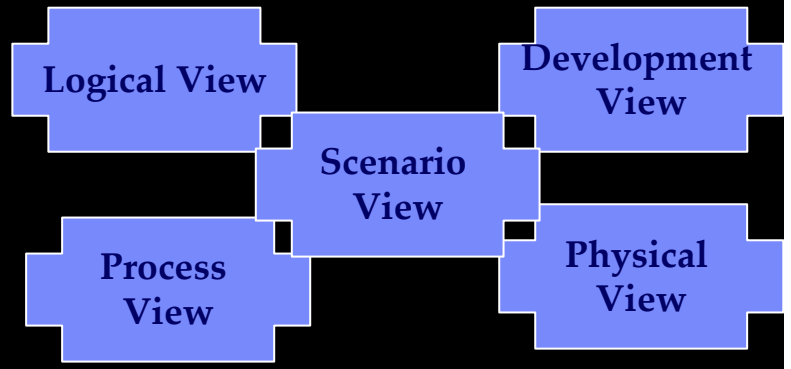
什么是软件体系结构-建议的架构视点和视图



什么是软件体系结构-视图

- 体系结构作为高层抽象提供能被来自各种背景的系统参与者（经理人、用户、客户、测试者、实现者、设计分析者等）所接受的描述，同时，是系统的“蓝图”。
- 体系结构视图包括
  - Functional view：描述系统功能即它们之间的关系
  - Concurrency view：描述线程的通讯与资源共享
  - Code view：程序员所用的描述类、对象、过程、子系统等的视图
  - Development view / Physical view ：系统的软硬件分布



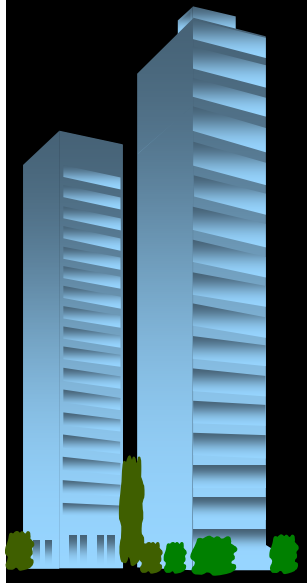


内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

为什么需要架构体系？

- 这和为什么需要建筑架构体系是同一个道理。
- 架构体系是为了帮助我们正确理解、设计一个复杂的系统，以确保我们最终可以成功构建出这种复杂系统的基础。
- Architecture as a *process* or *discipline*
  - 建造供所有类型的人使用的大厦的艺术或科学
  - 建造的活动或过程The action or process of building
  - 依据建筑物的详细的结构和装饰的组织而采用的特殊的方法或风格
  - 架构设计工作：结构，建筑物
  - 总体的构造或结构
- 好的架构能带来和谐、弹性、可靠的整体(whole)。同样地，好的系统architecture 能带给企业和谐、弹性、可靠的整体信息系统



为什么需要体系架构-基本假设

- 基本假设：系统不需要在超越“现在”的环境下生存

- 如果一个系统从来就不会变化
  - 该系统本身不需随外部环境而变化
  - 不会有该系统的变种有待开发

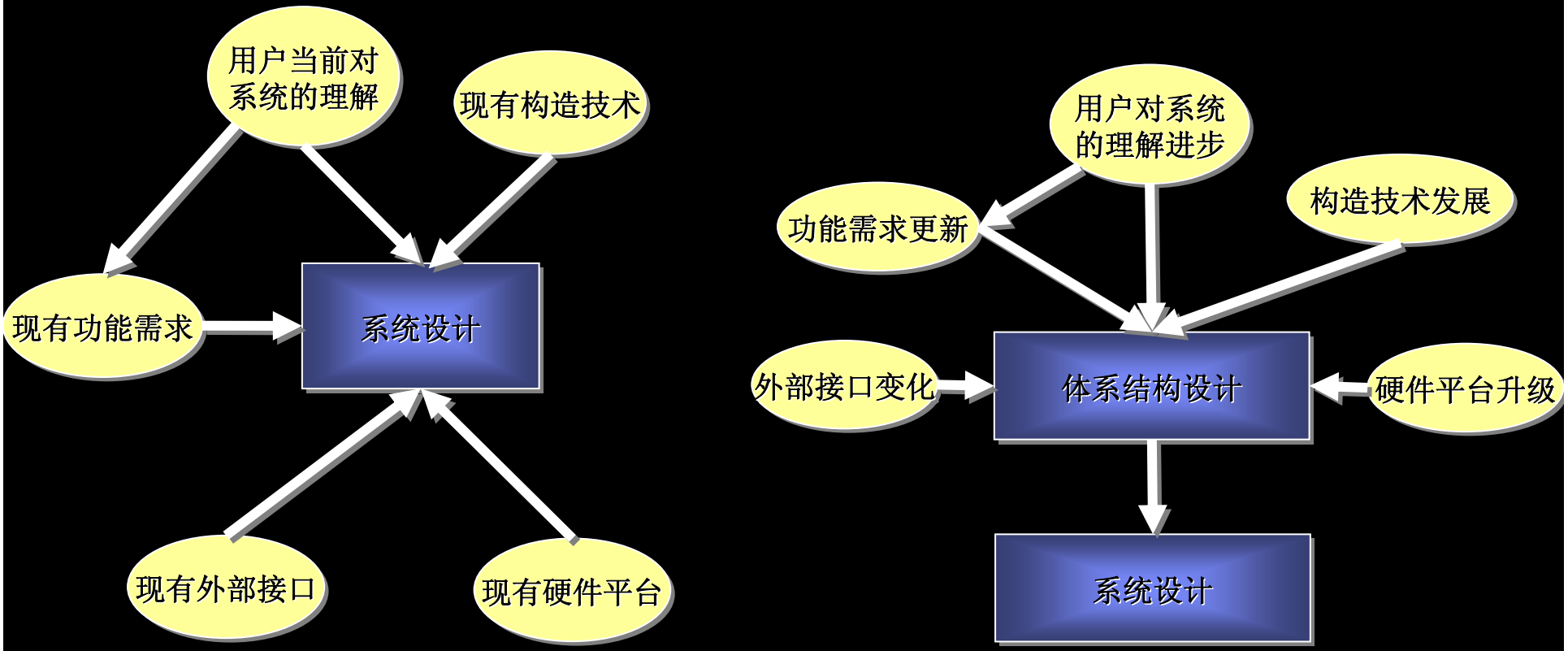
  就不需要有体系结构，只要”一锤子买卖“，如我们现在常做的这样。

- 但是
  - 辩证法说：世界是物质的，物质是变化的
  - 没有两个东西是完全一样的，就如不能两次踏入同样的流水一样
  - 人总是要偷懒的，才会有进步，所以尽量多动脑，少付出劳动代价

为什么需要软件体系架构-场景

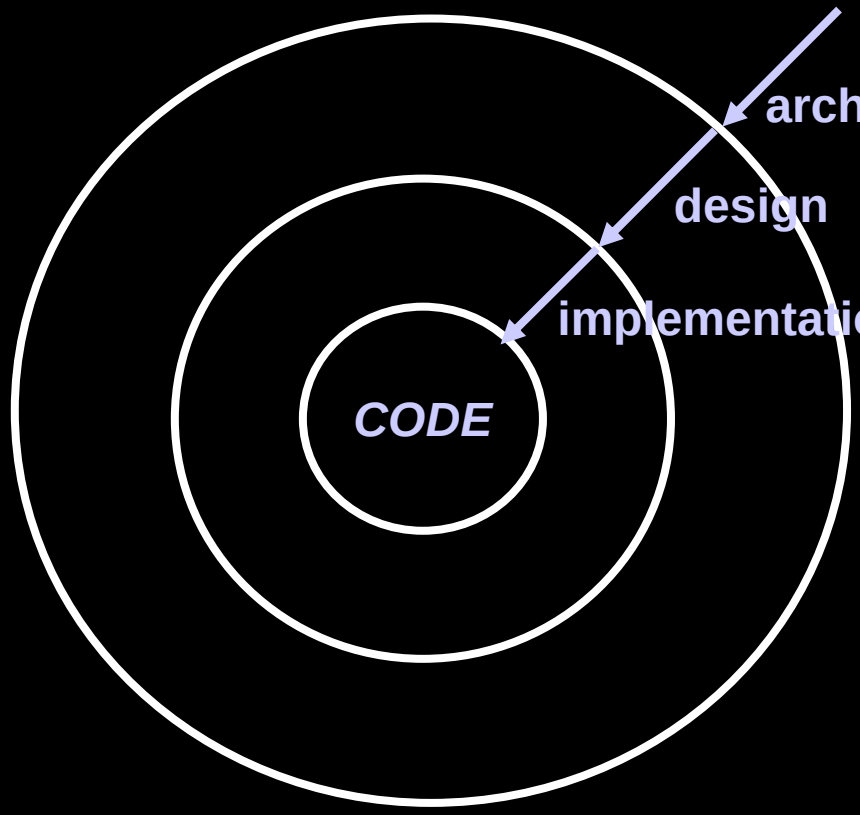
- 对于大型、复杂、软件密度高(software-intensive)的系统而言，软件体系结构是至关重要的
  - 是在系统的参与者对系统的抽象描述达成共识并以此为基础进行交流
  - 软件架构是软件开发过程中最早产生的系统描述，需要保证系统中相互冲突的需求的优先级、决定系统构造的原则，如如何在安全性与效率、可维护性与可实现性、特殊性与可扩展性等取得折衷
  - 软件架构由一些相对较小、易于理解与掌握的模型构成，从关键上描述系统的构成与行为。这些模型具有相当地可扩展性，易于在类似的系统中运用。从而达到软件复用的目标。

为什么需要软件体系架构—静止观点->变化观点



为什么需要软件体系架构-架构约束设计和实施

- 架构涉及到一系列对设计和建造起约束作用的的策略性的设计决定、规则和模式



架构的决定是最为基本的决定，改变他们将掀起宣然巨波

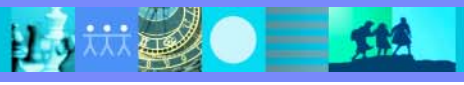
为什么需要软件体系架构-架构很昂贵（?）

- 管理通常总是忽略先期支付的项目成本
- 健壮的架构对于复杂的、关键性的、尺寸增大的项目尤为关键
- 好的架构帮助估算和控制开发成本

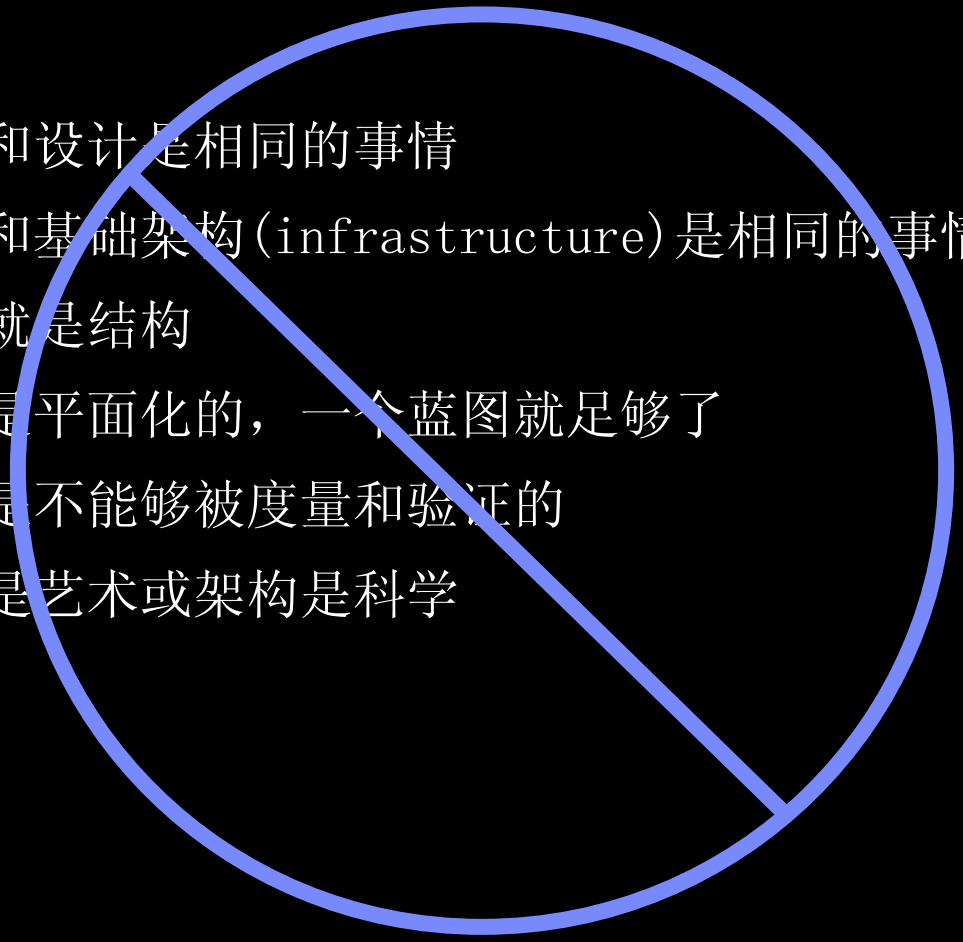
从长期的角度看，，没有架构的系统是十分昂贵的！

内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程



架构的常见错误理解



- 架构和设计是相同的事情
- 架构和基础架构(infrastructure)是相同的事情
- 架构就是结构
- 架构是平面化的，一个蓝图就足够了
- 架构是不能够被度量和验证的
- 架构是艺术或架构是科学

错误概念：架构 = 设计

- 架构是设计的一个表象，它关注于：
  - 对于结构来讲关键的的元素
  - 对于性能、可靠性、成本、适用性等有重大影响的元素
- 架构捕获了一组重要的设计决定
- 架构不关心每个独立元素的详细设计

错误概念：架构 = 基础架构Infrastructure

- 架构比基础架构包含的内容更多
- 基础架构是一个完整的、重要的架构的组成部分
- 架构也定义了也定义了应用在基础架构上的运行
- 架构定义了基础架构和应用组件之间的互操作性

错误概念：架构 = 结构

- 架构相关于结构、分解、接口等
- 架构比结构包含的更多：
  - 动态表象
  - 基本原理
  - 适合于上下文关系：业务上下文;开发上下文

错误概念：架构是平面化的

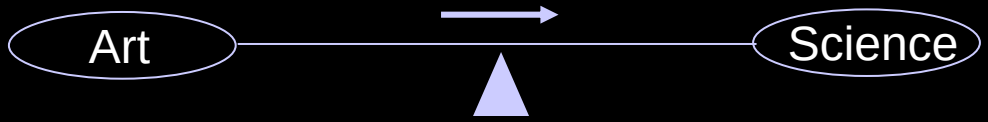
- 只有在那些非常细小的案例中，架构才是平面化的
- 架构有很多的维度，分别表述不同的涉众的不同的关注点
- 利用一个单一的蓝图来表达全部（大部分的）的维度将导致语义的过度使用和不完整
- 架构需要多个视图

错误概念：架构是不能够被度量和验证的

- 架构不是一个粗略的、用草纸和铅笔进行的顶层的设计
- 对于架构能够就功能和质量的需求、风险、和其他的系统关键属性进行系统化地评估:评审架构的工件 ;测试架构原型.

错误概念：架构是艺术或架构是科学

- 两者都不是/两者都是
- 难于应用分析方法



- 非常大的问题空间，却只有非常少的可选择的量化的度量标准
- 好的架构师拷贝过去成功地解决方案并与当前增加的改进结合
- 架构定义的过程有明确定义的步骤和明确描述的工件
- 架构相关的知识体系开始被编成法典
  - 模式，框架，服务，启发模式，…
- 直觉和创造性仍然重要

内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

架构带来的好处

系统完整性和质量

- 架构能够在变更中保持概念的完整性和系统质量
  - 所要求的功能
  - 质量属性
  - 新需求
  - 变更的需求
  - 融合技术
- 架构延长系统的寿命
  - 容易进化
  - 系统弹性
  - 弹性化的应变能力

控制复杂度

- “排除与克服”
  - 分解到组件
  - 隐藏实施的细节
- 关注点的分离
  - 组件（层次）封装细节
  - 不同组件可以由掌握不同专业技能的人员来实施

架构带来的好处（续）

可预见

- 过程可预见性
  - 架构原型允许你收集度量指标
    - 开发成本的度量指标
    - 进度的度量指标
- 行为可预见性
  - 架构迭代排除关键风险

可测试性

- 良好构件化了的系统支持更好的、更容易的
  - 诊断
  - 跟踪能力
  - 发现错误

架构带来的好处（续）

重用

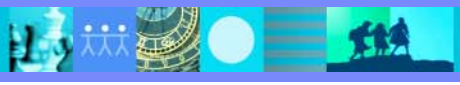
- 架构定义了替换规则
- 组件接口定义替换物的边界
- 架构使得各种粒度的重用成为可能
  - 组建级别的小范围的重用
  - 大范围的重用
    - 子系统
    - 产品
    - 框架

沟通

- 架构支持涉众间的沟通
- 不同的视图定位于不同涉众的关注点
- 架构沟通的是关键的设计决定
- 架构设计的基本原理沟通折中

架构带来的好处（续）-架构的成本是先期支付的

- 架构的投资发生在系统开发的早期阶段
  - 启动阶段
  - 精化阶段
- 在前期阶段的成本估算模型是不精确的
- 架构所带来的收益发生在实施和维护阶段
- 最小化返工带来的开销
- 重用带来的节省
  - 组件的重用
  - 开发可重用的组件
- 通过高效的资源利用带来的节省
- 通过准确的成本/进度估计带来的节省
- 从改进的维护和支持能力带来的节省



内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

软件架构的设计原则

- ◆系统的内聚和耦合度：这是保证一个系统的架构是否符合软件工程原则的首要标准。
- ◆ 层次的清晰和简洁性：系统每个部分完成功能和目标必须是明确的，同样的功能，应该只在一个地方实现。如果某个功能可以在系统不同的地方实现，那么，将会给后来的开发和维护带来问题。系统应该简单明了，过于复杂的系统架构，会带来不必要的成本和维护难度。在尽可能的情况下，一个部分应该完成一个单独并且完整的功能。
- ◆ 易于实现性：如果系统架构的实现非常困难，甚至超出团队现有的技术能力，那么，团队不得不花很多的精力用于架构的开发，这对于整个项目来说，可能会得不偿失。简单就是美。

软件架构的设计原则

- ◆ 可升级和可扩充性：一个系统框架，受设计时技术条件的限制，或者设计者本人对系统认识的局限，可能不会考虑到今后所有的变化。但是，系统必须为将来可能的变化做好准备，能够在今后，在目前已有的基础上进行演进，但不会影响原有的应用。接口技术，是在这个方面普遍应用的技巧。
- ◆ 是否有利于团队合作开发：一个好的系统架构，不仅仅只是从技术的角度来看，而且，它还应该适用于团队开发模型，可以方便一个开发团队中各个不同角色的互相协作。例如，将Web 页面和业务逻辑组件分开，可是使页面设计人员和程序员的工作分开来同步进行而不会互相影响。
- ◆ 性能：性能对于软件系统来说是很重要的，但是，有的时候，为了能让系统得到更大的灵活性，可能不得不在性能和其他方面取得平衡。另外一个方面，由于硬件技术的飞速发展和价格的下降，性能的问题往往可以通过使用更好的硬件来获得提升。

软件架构的设计原则-架构的评价

- 手段：正式的评审过程
  - 相关行业领域专家：列举可能发生的修改，修改发生的概率
  - 系统设计人员：给出对这些修改对模块结构将产生的影响
  - 根据对不同的模块结构的平均修改程度权重（或对比行业标准）评估该结构设计
  - 这种方法后来发展成较系统的软件体系结构评估方法

内容

- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

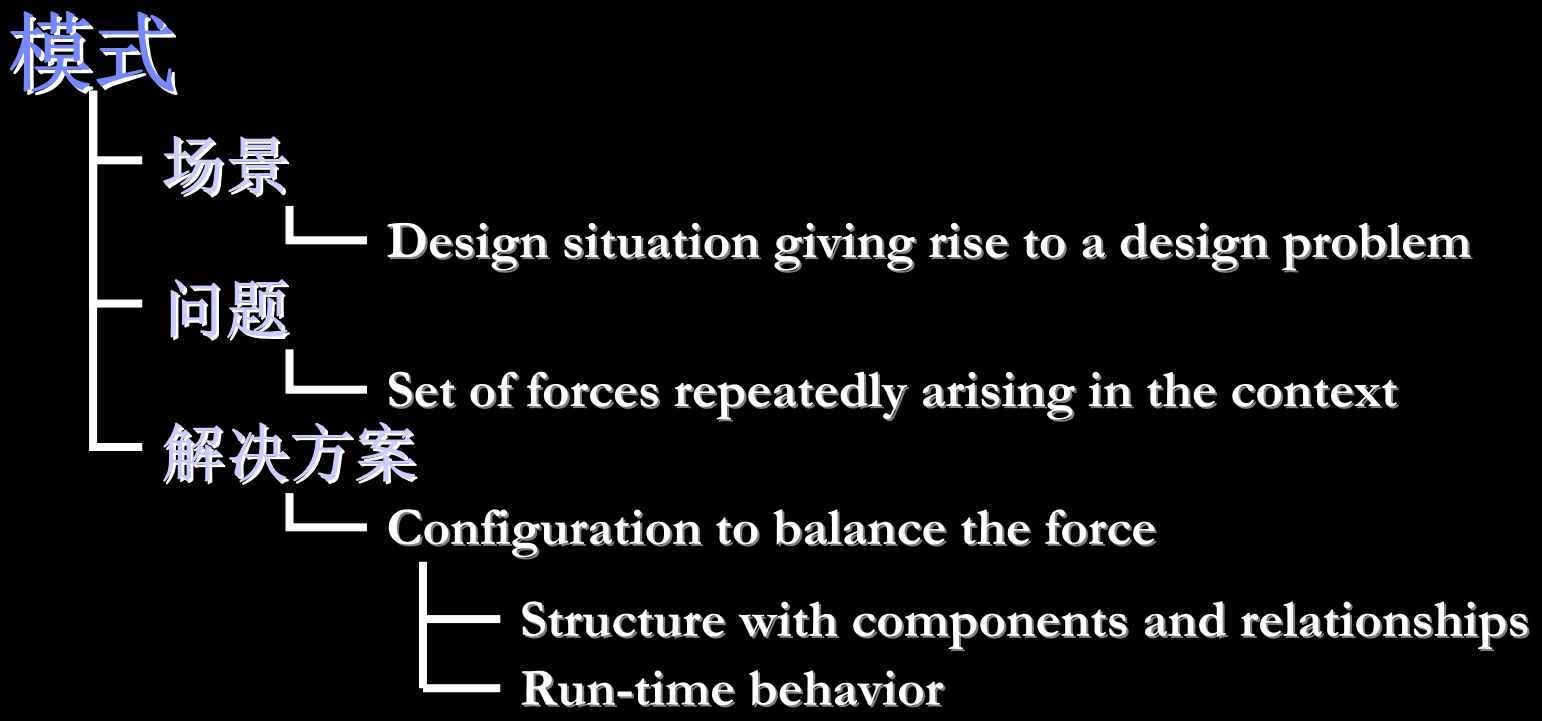
架构的风格及模式-什么是模式

- problem-solution pairs
  - 当一个行业专家解决一个特定问题时，很少使用完全不同于已有方案的全新的解决方法。
  - 一般回顾所解决的类似问题，然后用类似的方法解新的问题。
  - 这种“专家行为(expert behavior)”，是非常普遍的，例如建筑学(Architecture)、经济学、软件工程。
- 特定的问题解决方案中提取出共同点即为模式

架构的风格及模式-什么是模式（续）

- 我们可以由这里例子引出软件体系结构模式(此后除非特殊说明，简称模式)的一些特性
  - 针对特定情况下不断重复出现的设计问题给出解决方案。
  - 是已有的、实践证明有效的设计方案的总结。[existing & well-proven]
  - 标识并描述高于单个类/实例/组件的抽象。[interaction on comps]
  - 提供了设计原则的通用词汇与理解。[vocabulary and understanding]
  - 提供了描述软件体系结构的一种方法。[means of doc SA]
  - 对构造具有特定性质的软件系统提供支持。[skeleton to be extend]
  - 有助于建立复杂的、异质的软件体系结构。[++predefined set of comps + roles + relations - as building block . However , basic structure \= fully solution]
  - 有助于管理软件的复杂性。[reduce your work]

架构的风格及模式-模式的表达



架构的风格及模式-模式的分类

- 模式包含面广，我们根据其适用范围与抽象性划分为以下三类
  - 体系结构模式
  - 设计模式
  - 惯用法
- 基本的划分，如第一节基于模式的抽象程度的划分，同时也基本给出了按所相关的软件开发过程的阶段与活动的一种划分：
  - Architecture Pattern：用于粗略设计，体系结构设计，描述应用的基本结构。
  - Design Pattern：改进基本的体系结构，细化设计，同时提供一些局部实现的基础。
  - Idiom：实现阶段。
- 当然，该划分不是绝对的。

架构的风格及模式-模式的分类（续）

- 按问题种类的划分
  - From Mud to Structure ： 对系统的适当的组织划分（为子系统/任务），描述系统的整体结构
  - Distributed Systems ： 分布在不同的进程/子系统/物理位置的组件的组织结构
  - Interactive Systems ： 人机交互的系统结构
  - Adaptable System ： 针对可能演化与改变的功能需求可扩展可改变的系统结构
  - Structural Decomposition ： 将复杂的组件或(子)系统划分为协作组件的方法
  - Organization of Work ： 组件如何协作提供复杂的服务
  - Access Control ： 控制与分派对服务与组件的访问权限
  - Management ： 同种对象/服务/组件的集合的一体化处理
  - Communication ： 组织组件间通信的方法
  - Resource Handling ： 管理共享组件与对象的方法
  - Others ： 有些问题构成独立的类，有时放入到相关类中。注意到，问题是不断的出现的，这种划分本身需不断的更新。

架构的风格及模式-模式的分类（续）

	体系结构模式	设计模式	习惯用语
M-to-S	Layers / Pipes & Filters / Blackboard		
分布系统	Broker / Pipes & Filters / Microkernel		
交互系统	MVC / Presentation-Abstraction-Control		
自适应系统	Microkernel / Reflection		
结构分解		Whole-Part	
过程组织		Master-Slave	
访问控制		Proxy	
同类管理		Command Processor /View Handler	
通信		Publisher-Subscriber /Forward-Receiver / Client-Dispatcher-Server	
资源管理			Counted Pointer

架构的风格及模式-架构模式

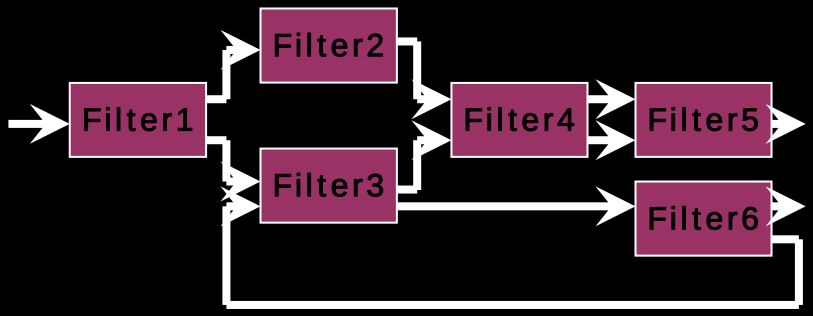
- 一般来说，一种体系结构风格描述了软件系统的基本结构类型以及其关联的构造方法、使用该风格的环境、变种与特例等。
- 体系结构模式是具体的软件体系结构的模板，描述系统级的结构特性，并影响到子系统的结构。因此，对体系结构模式的选择是开发软件系统时最根本的设计决策。
  - Model-View-Controller即为体系结构模式，提供了交互式软件系统的结构框架。

架构的风格及模式-架构模式（续）

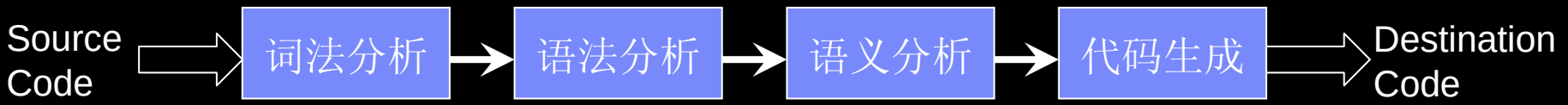
- General Architecture Styles for Software System
  - Pipe and Filters
  - Hierarchical Layers
  - Event System
  - Blackboards …
  - Interactive System
  - Adaptable System
- Domain Specific Software Architecture

架构的风格及模式-Pipe/Filter Pattern

- 为处理数据流的系统提供了一种结构.
- 每个处理步骤封装在一个过滤器组件中.
- 数据通过相邻过滤器之间的管道传输.
- 重组过滤器可以建立相关的系统族.



- 处理数据流的软件系统体系结构模式，如：编译器系统

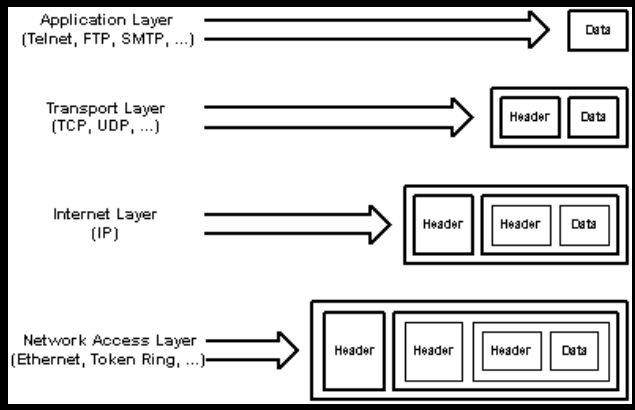


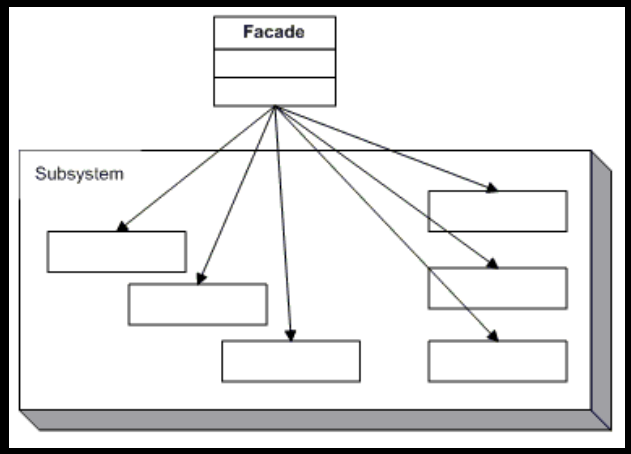
架构的风格及模式-Layered System

- 一个需要分解的大系统
- 如果系统的显著特征是混合了低层与层次问题。
- 系统的需求本身定义了多个层次上的需求。
- 系统规格说明描述了高层任务，并希望可移植性。
- 系统的外部边界要求系统附带功能接口。
- 高层任务到平台的映射不是直接的， 系统需要满足以下非功能性特性：
  - 后期源代码的改动应该不影响到整个系统。
  - 系统的各个部分应该可以替换。构件应该可以有不同的实现而不影响其他的构件。
  - 提高构件的内聚性。
  - 复杂构件的进一步分解。
  - 设计和开发系统时，工作界限必须清楚

架构的风格及模式-Layered System(续)

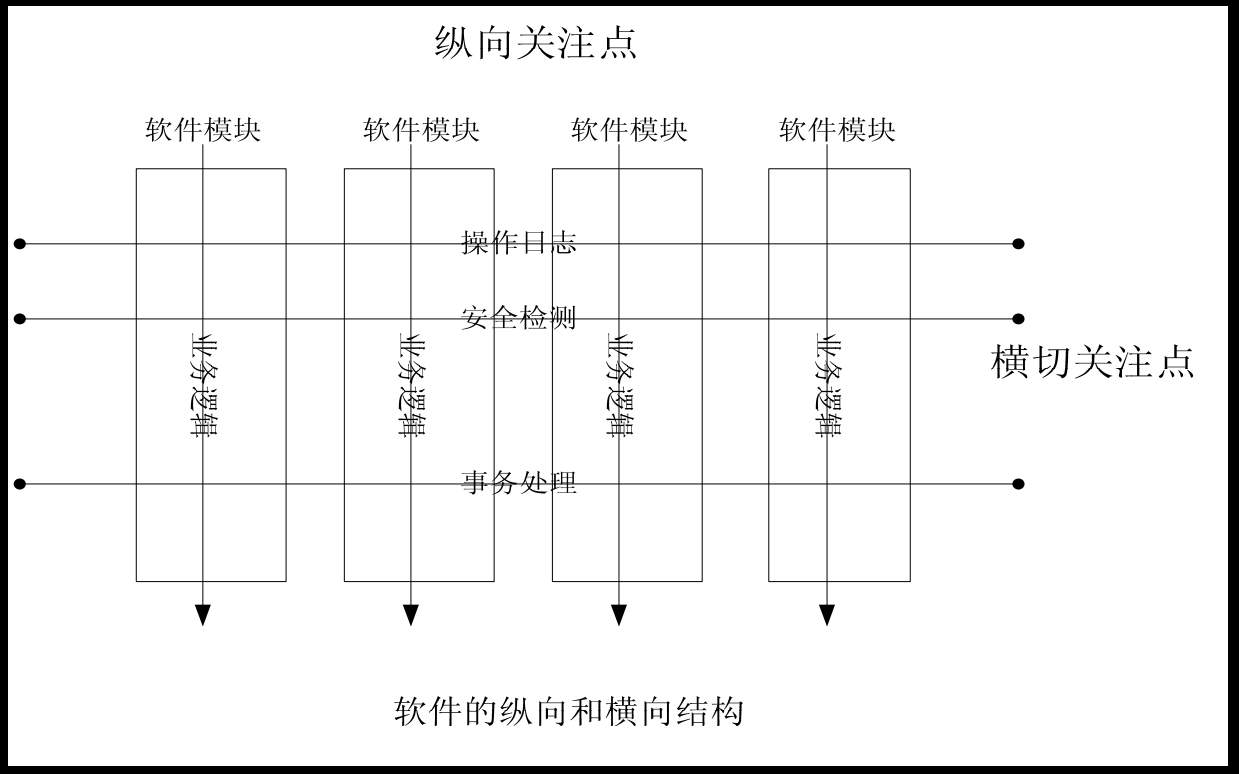
- 将系统分成适当层次，按适当次序放置。
- 从最低抽象层次开始，以梯状把抽象层次n放在n-1层顶部。直到功能顶部。
- 第n层提供的绝大多数服务由第n-1层提供的服务组成。
- 每个层次是一个独立的组件。它的责任是：提供了由上层使用的服务，并且委派任务给下一层次。
- 不允许较高层次直接越级访问较低层次。





架构的风格及模式-AOP

AOP给了我们一个新的视角来看待软件的架构，有的时候，即使不使用AOP技术，只使用AOP的某些观念和现有的技术来搭建系统架构，分离某些本来是紧耦合的关注点，对我们也是非常有益的。



软件的纵向和横向结构

架构的风格及模式-AOP（续）

通常，我们可以在如下情景中使用AOP技术：

- Authentication 权限
- Caching 缓存
- Context passing 内容传递
- Error handling 错误处理
- Lazy loading 懒加载
- Debugging 调试
- logging, tracing, profiling and monitoring 记录跟踪 优化 校准
- Performance optimization 性能优化
- Persistence 持久化
- Resource pooling 资源池
- Synchronization 同步
- Transactions 事务

架构的风格及模式-总结

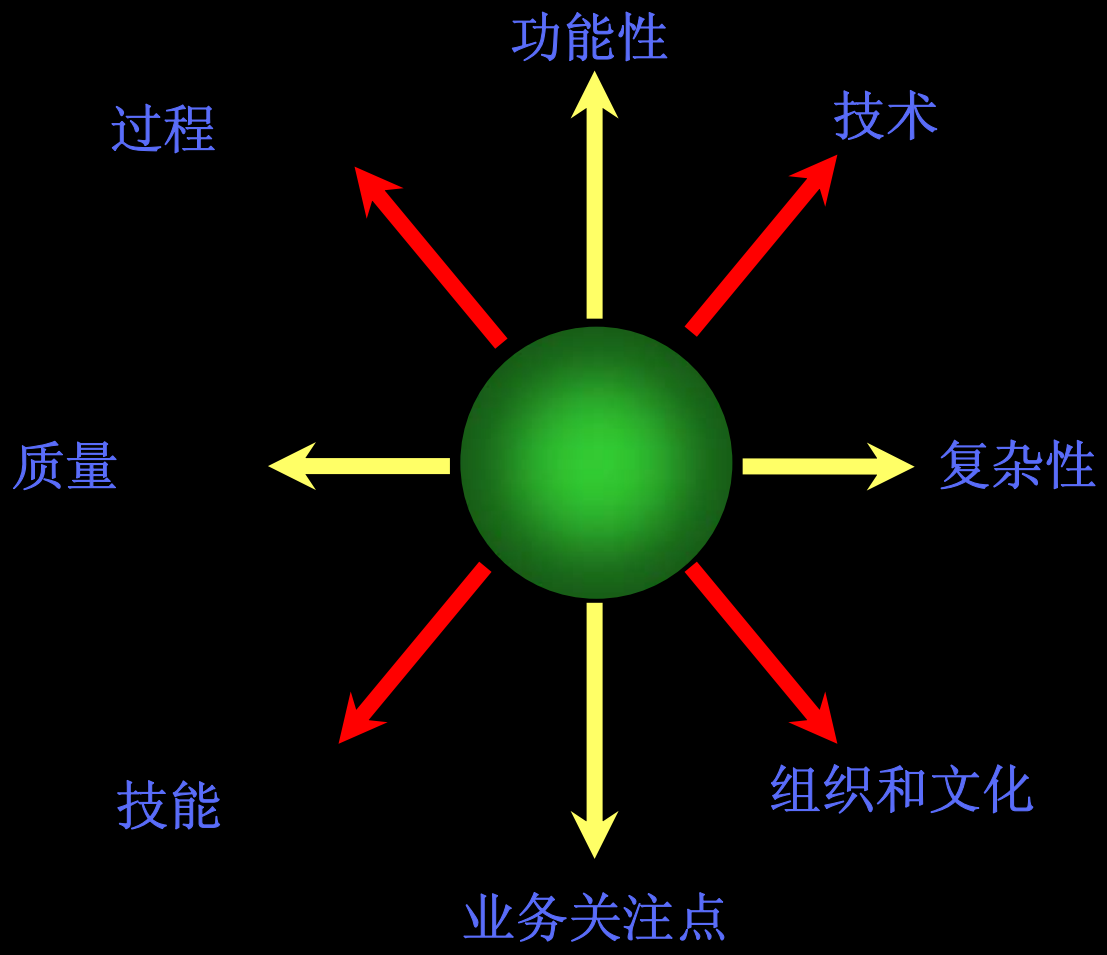
- 模式是软件体系结构的重要组成部分：
  - 提取软件系统的共同图式，明确建立在经过实践检验的软件系统构造技术基础上，如信息隐藏、界面与实现的分离等；
  - 提供了条例化的严格描述，包括对非功能属性的描述，有助于复杂系统高层属性的分析与描述，并提供有效的实现指南；
  - 与已有的独立于问题problem-independent的软件开发过程与方法相互补充，提供解决特定设计问题的手段，为在多种设计可能中做出有效选择体统基础。
- 由此在系统化的构造高质量软件之路上迈进一大步。

内容

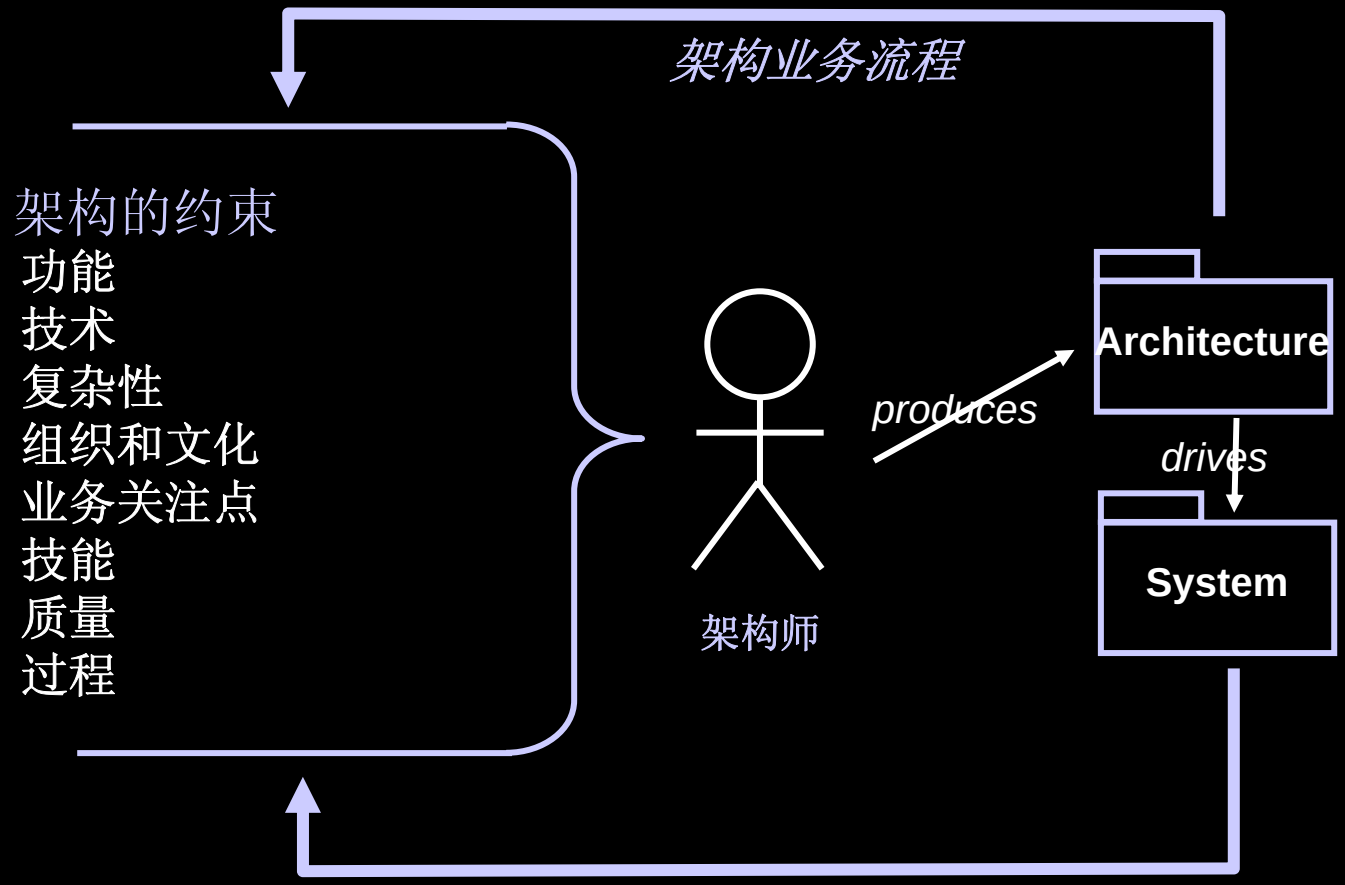
- 问题
- 什么是软件架构
- 为什么需要体系架构
- 架构的常见错误理解
- 架构带来什么好处
- 架构设计的原则
- 架构的风格及模式
- 架构设计的过程

架构设计的过程-多重影响

架构设计的过程-多因素的权衡



架构设计的过程–架构的基本工作流程



Software Architecture in Practice: L.Bass, P. Clements, R. Kazman, *Addison-Wesley, 1998*

架构设计的过程-需要定义中的公共元素

- 架构定义了关键 组件
- 架构定义了组件之间的 关系（结构）和 交互
- 架构忽略了组件中与组件之间交互无关的内容的信息
- 从另外一个组件的视点能够观察到的组件的行为是架构的一个部分
- 每个系统都有一个架构（即便这个系统只有一个组件组成）
- 架构定义了组件和结构背后的 基本原理
- 在架构定义中不包含组件的定义
- 架构不是简单的结构定义 - 在架构定义中不只是单一的结构定义

架构设计的过程-体系结构的构造

- 大步骤：
  1. 为软件系统构建一个基本商业目标
  2. 定义系统需求
  3. 构建或选用体系结构
  4. 正确描述该结构，并与各个方面交流
  5. 对此结构进行分析与评价
- 关键步骤：提及结构设计师所最关注的是第三步，需要进一步描述

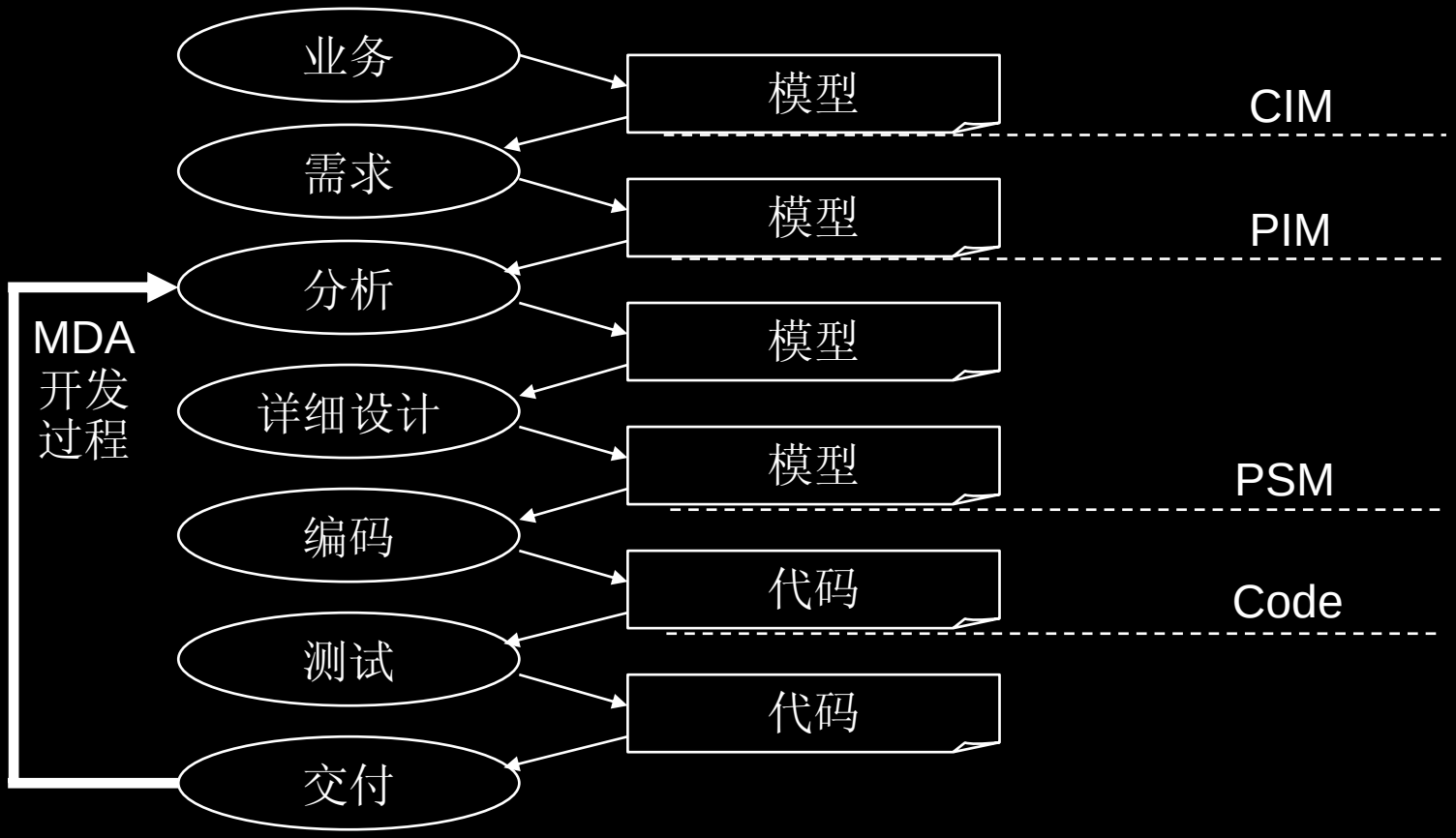
架构设计的过程-体系结构的构造（续）

- 体系结构的构造的前提：回答开始的问题
  - 理解处于整个软件开发生命周期中的体系结构的位置
  - 从简单的基于需求到受整体环境影响
  - 从静止的环境到变化的环境
  - 从开环到闭环：体系结构对环境的反馈
- 进一步的问题：
  - 基于体系结构设计有可遵循的过程标准吗？
- 简单回答
  - 如果，需要的标准是统一的、严格的、有文档标准的、保证正确的、且能回答你在体系结构设计中面临的具体问题的，那么，没有！
  - 只有：大概念上的几个步骤，如同以前大家学习的软件工程指导性过程，因为太一般化而不具备实际的指导意义，所以要做两件事情：
    - 了解这个过程：
    - 通过实例加强对现实的把握能力

架构设计的过程-什么阶段产出架构

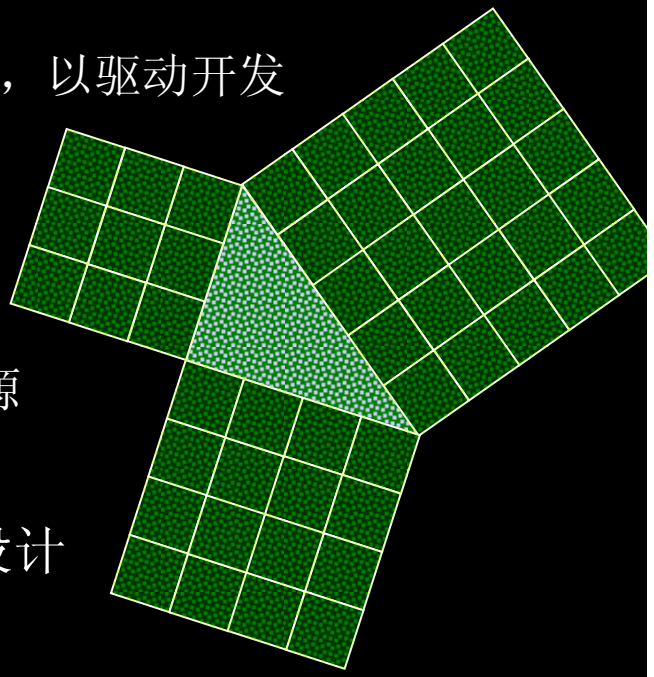
- 在项目的早期阶段开发架构
  - 在项目的启动阶段开始着手定义架构
  - 在项目的精化阶段稳定架构
- 架构由用例驱动和软件相关的环境产出
  - 对架构重要的用例来自两个地方
    - 帮助降低最严重风险的用例
    - 对系统用户来讲最重要的用例
- 形成架构基线——稳固的架构

架构设计的过程-MDA简介



架构问题总结

- 只有正确地开始，才能正确地结束
  - 过早地仓促地进入系统的构建是错误的开始
  - 正确地开始：确保需求正确，并使其处于良好状态，以驱动开发
- Architecture代表高质量
  - Pattern——复用设计的最佳实践
  - J2EE 13 Core Patterns
  - AntiPattern——应用性能、可靠性的主要问题根源
    - Butterfly, Hub, Breakable, Tangle
- 模型清楚地表达与记载了我们的发现、发明、设计
  - 我们才可以把它们传给别人
  - 我们才可以把它们传给未来




Questions


Thank
You