软件体系结构构造技术

北京大学软件工程国家工程研究中心

2004年7月12日

内容

- 软件体系结构概念
  - 定义
  - 重要性
- 特定领域的软件体系结构构造（DSSA)
  - DSSA定义
  - 体系结构风格
    - 体系结构风格定义
    - 典型体系结构风格
  - 质量属性与实现
    - 理解质量属性
    - 质量属性实现通用策略
  - 领域变化性控制机制
    - 变化性维度
    - 变化性控制机制
- 体系结构文档

从建筑业谈起

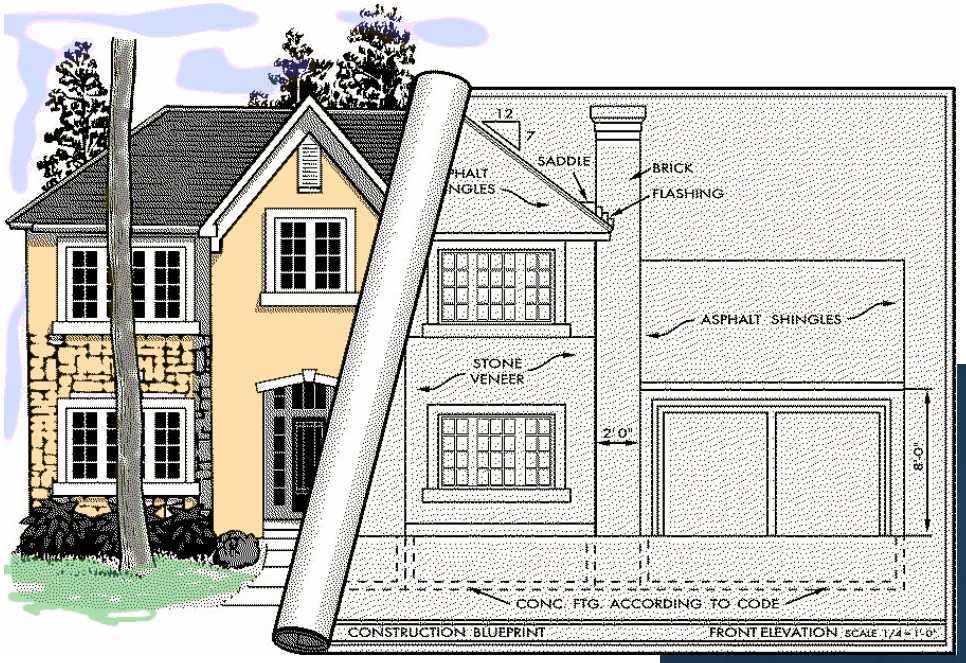
一个人搭建
需要

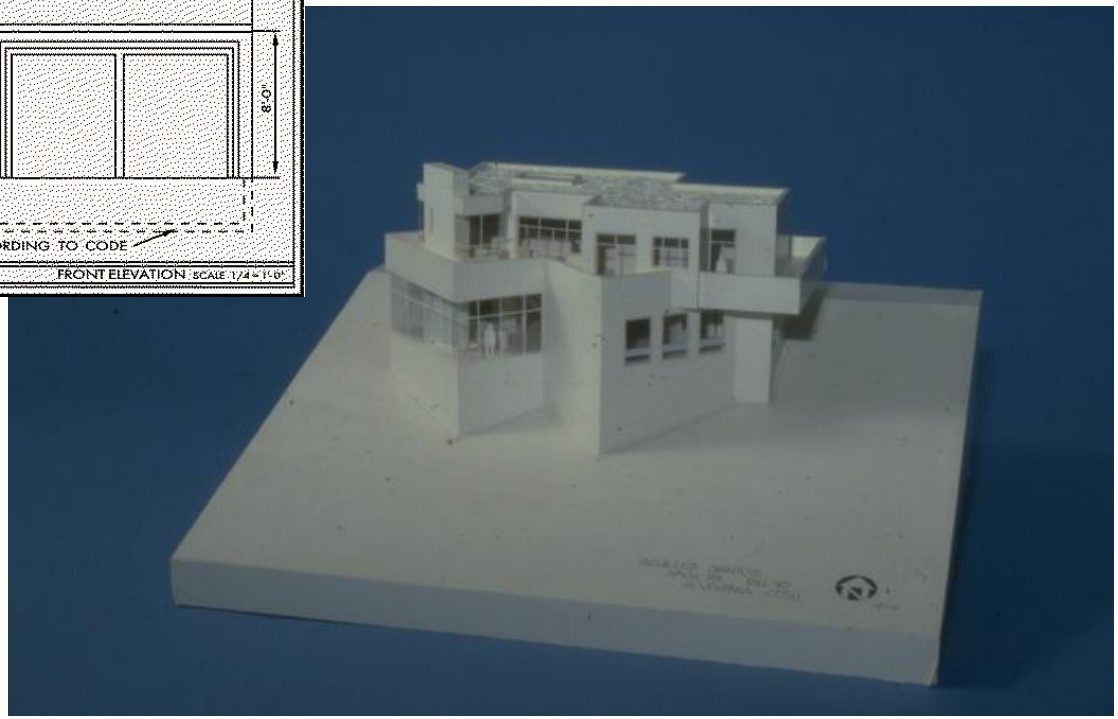
- 简单建模
- 简单的过程
- 简单工具

一个团队高效适时地建造，
需要

- 建模
- 良好的过程定义
- 良好的工具

对房子进行建模


12
7
SADDLE
BRICK
FLASHING
ASPHALT SHINGLES
STONE VENEER
2'-0"
8'-0"
CONC. FTG. ACCORDING TO CODE
CONSTRUCTION BLUEPRINT
FRONT ELEVATION SCALE 1/4" = 1'-0"



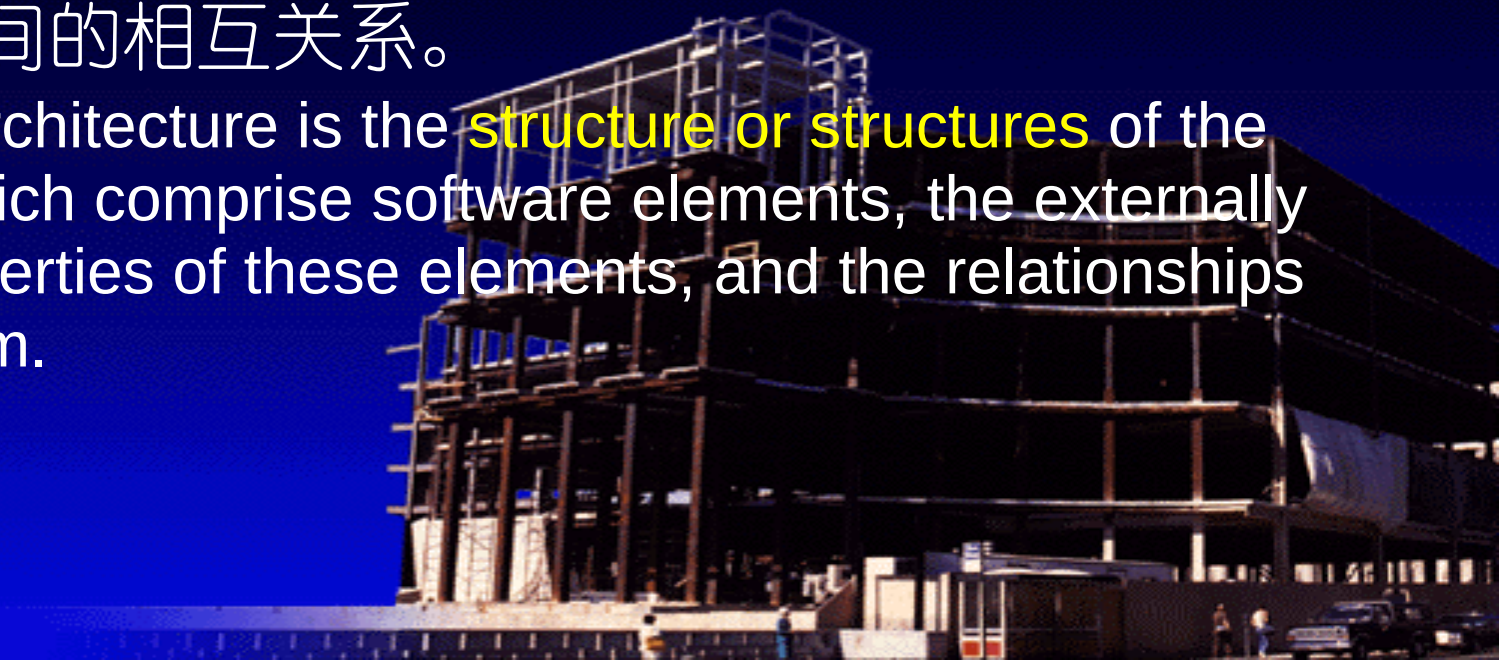
建模的重要性


...AND THEN THEY WANTED TO SEND ME ON A COURSE IN FOUNDATION DESIGN!!! WOULD HAVE TAKEN 3 DAYS OUT OF THE PROJECT.....

什么是软件体系结构

- 目前，关于软件体系结构的定义有60多种，有的定义是从构造的角度来审视软件体系结构，有的侧重于从体系结构风格、模式和规则等角度来考虑
- 软件体系结构是系统的一个或多个结构，它包括软件的组成元素，这些元素的外部可见的特性以及这些元素之间的相互关系。

Software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of these elements, and the relationships among them.



定义的含义 – 1

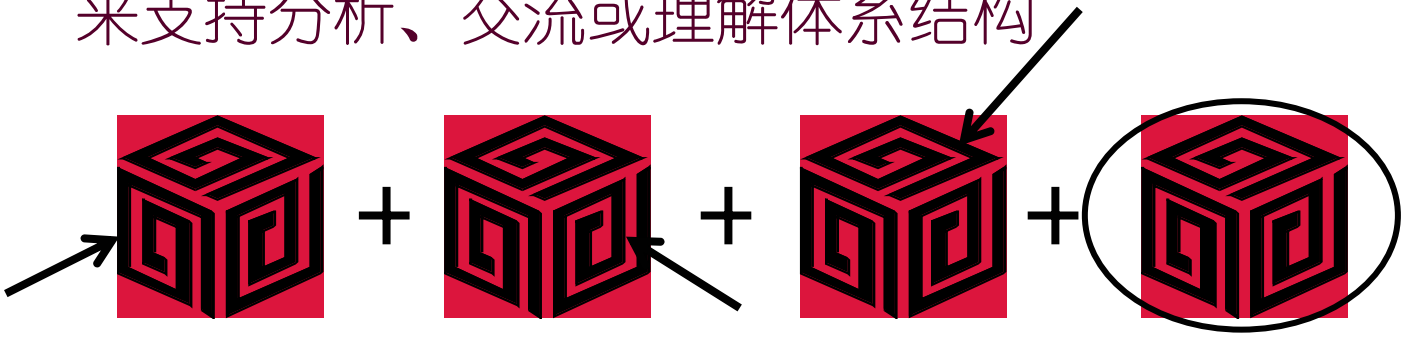
- 软件体系结构是系统的抽象
  - 体系结构定义了元素以及它们如何交互
  - 体系结构隐瞒了纯粹的属于局部的信息，元素的细节不属于体系结构。
- 元素外部可见的特性是指该元素对其他元素来说
  - 提供的服务，需要的服务，具备的性能特性，容错能力，共享资源的使用等

定义的含义 – 2

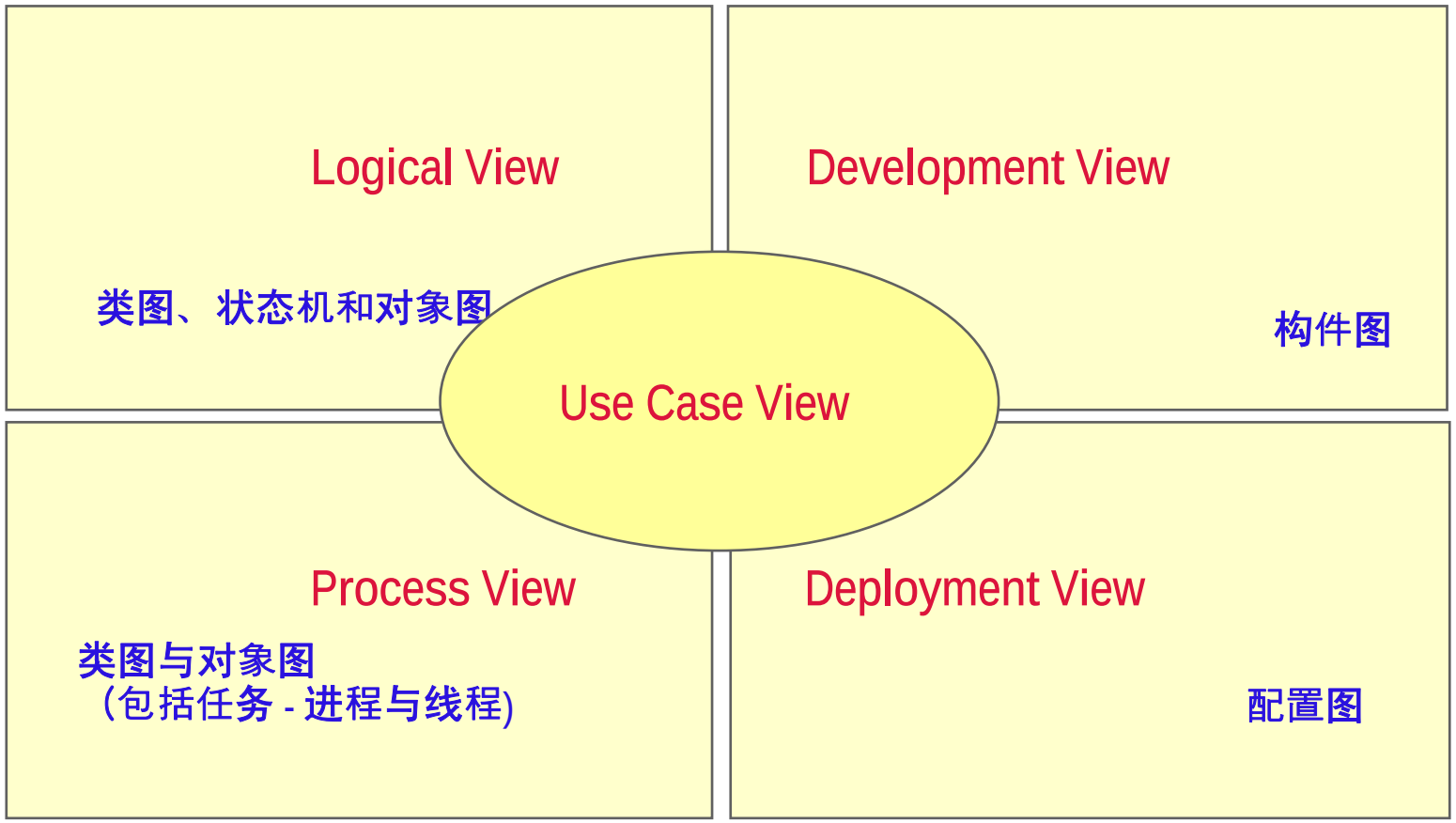
- 每个系统都有一个体系结构
  - 每个系统都是由元素以及元素之间的关系组成.
  - 最简单的例子，一个系统就是由一个元素和它自身的关系组成
- 每个系统都有体系结构，但并不意味着任何人都知晓该体系结构的存在
  - 软件体系结构与体系结构的规约的区别
- 如果你不明确地开发一个体系结构，那么你仍然拥有一个——只是不是你所喜欢或期望的

定义的含义 – 3

- 这意味着只有线框图不是体系结构的规约，但这是个起点
  - 用框能够表示“数据库”或者“执行代码”的行为——这还不够
  - 你需要增加规约和特性描述。
- 系统有多个结构structures，也叫做视图(views).
  - 类比于建筑，对同一栋大楼，承包商、设计师、室内设计人员、庭院设计家、电工都有不同的理解
  - 任何一个视图只能表示体系结构的部分内容
  - 候选视图的集合不是固定的也不是规定的：选择一组来支持分析、交流或理解体系结构

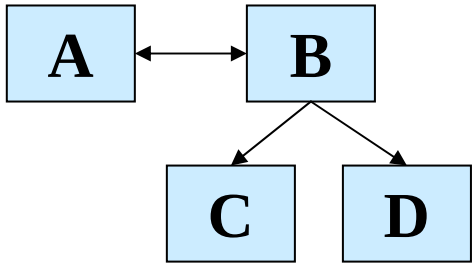


RUP中的体系结构视图

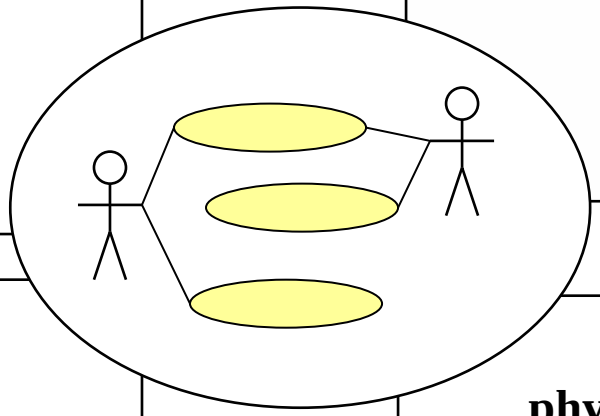


另一种看法：体系结构视图示例

Structure view: **components, packages, interfaces**



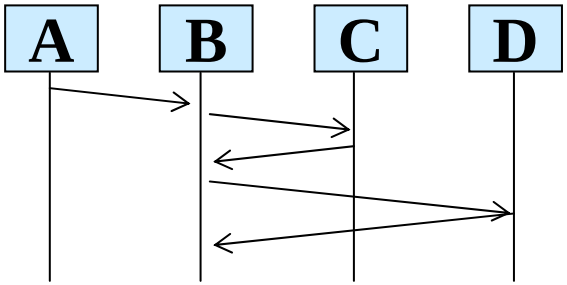
Stakeholders & Use cases view



Config. Mngt view

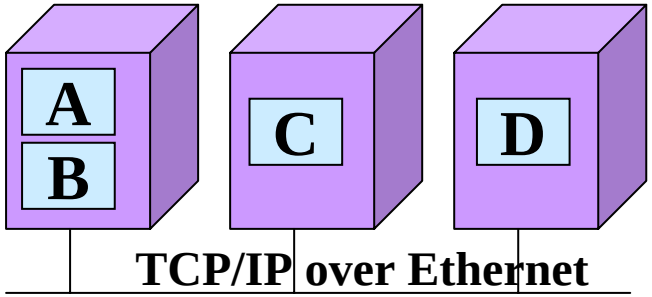
versioning policies
file ownership
…

Behaviour view:
MSC, state-diagrams



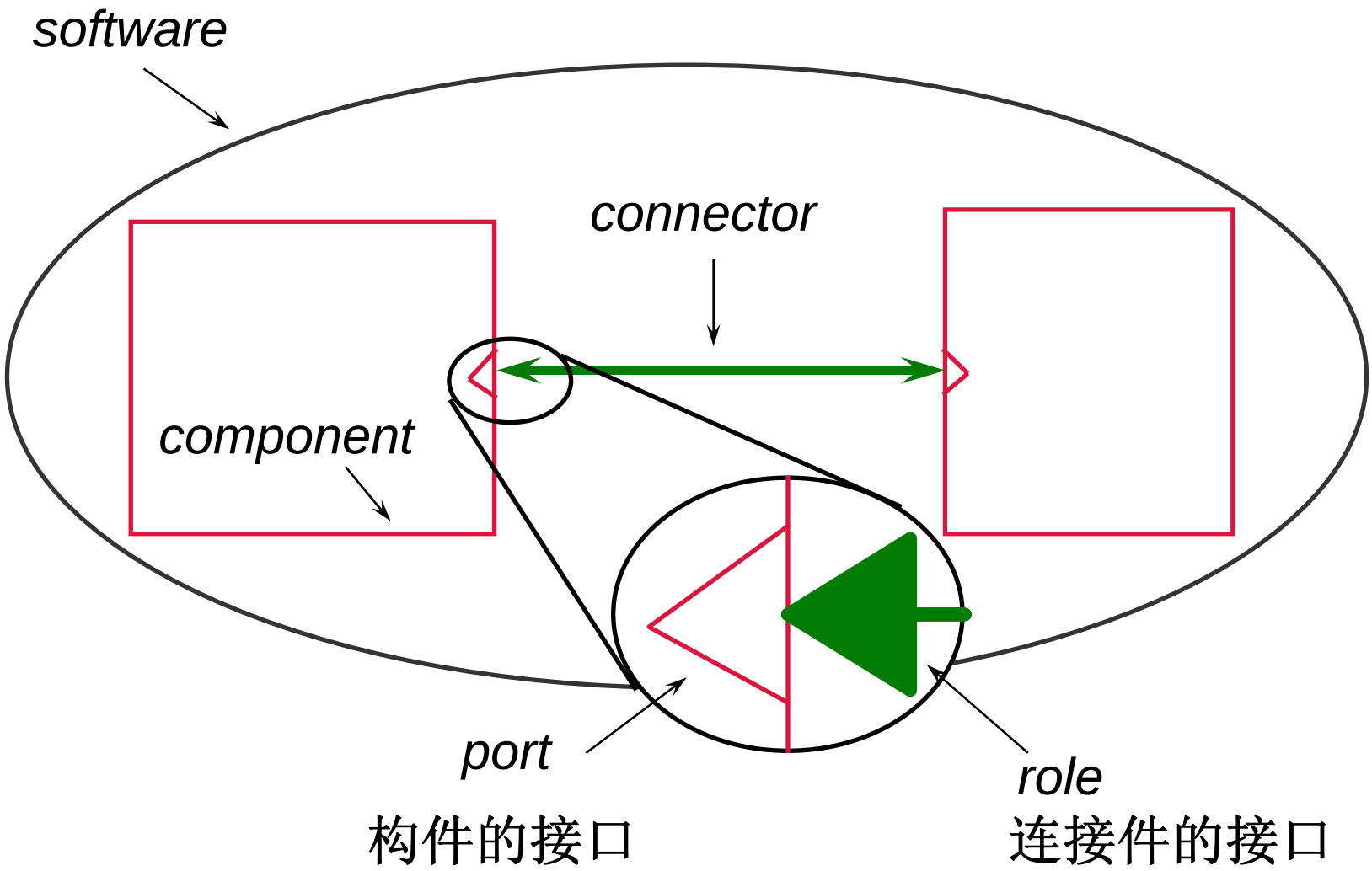
BC/WC e2e-response times, freq.

Deployment view:
physical model + mapping



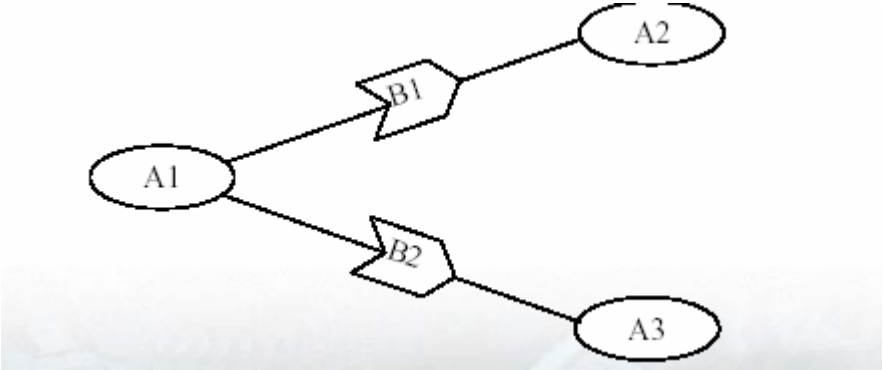
bandwidth, availability

软件体系结构的基本概念-1



软件体系结构的基本概念-2

- 体系结构最基本的组成元素：
  - 构件（components)
  - 连接件（connectors）
  - 约束（constraints)
- 构件是组成体系结构的基本计算单元或数据存储单元，用于实施计算和保存状态；
- 连接件用于表达构件之间的关系；
- 约束定义了相关因素的特性；

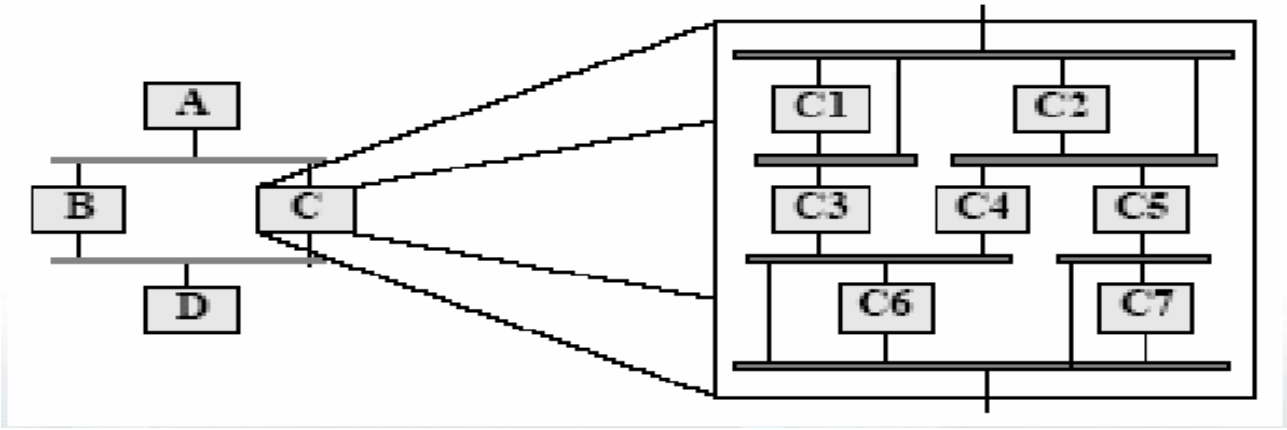


连接件

- 连接件是对以下内容进行建模的体系结构元素
  - 构件之间的交互
  - 指导这些交互的规则
- 简单交互
  - 过程调用
  - 共享变量访问
- 复杂和语义丰富的交互
  - 客户/服务器协议
  - 数据库访问协议
  - 异步事件广播
  - 管道数据流

配置

- 体系结构（配置）确定了体系结构的构件和连接件的关系
- 体系结构的拓扑是构件和连接件的连接图（connected graph），描述了系统结构的
  - 适当连接
  - 并发和分布特性
  - 符合设计规则和风格规则
- 复合构件本身就是配置



软件体系结构的重要性

- 体系结构是风险承担者进行交互的手段
  - 与用户进行需求协商
  - 找到恰当的解决方案
  - 作出管理决策和分配
- 体系结构是早期设计决策的体现
  - 指导系统实现
  - 体系结构决定了开发组织的组织结构
  - 实现系统的质量属性的途径之一
  - 有助于变化管理
  - 有助于循序渐进地原形设计
- 体系结构是可复用的模型
  - 产品线共享一个公共的体系结构
  - 使得系统开发可以使用已由其他组织开发完成的构件
  - 从构件的互联机制中分离功能性
  - 提供设计词汇表
  - 使得基于模板的构件开发成为可能

内容

- 软件体系结构概念
  - 定义
  - 重要性
- 特定领域的软件体系结构构造（DSSA)
  - DSSA定义
  - 体系结构风格
    - 体系结构风格定义
    - 典型体系结构风格
  - 质量属性与实现
    - 理解质量属性
    - 质量属性实现通用策略
  - 领域变化性控制机制
    - 变化性维度
    - 变化性控制机制
- 体系结构文档

DSSA定义-1

- 特定领域软件体系结构（Domain Specific Software Architecture,)对领域分析模型中表示的需求给出解决方案，它不是单个系统的表示，而是能够适应领域中多个系统需求的一个高层次的设计
- 它决定了
  - 结果产品的质量，例如性能、可修改性、可移植性
  - 组织结构和管理模式

质量属性实现策略

DSSA定义-2

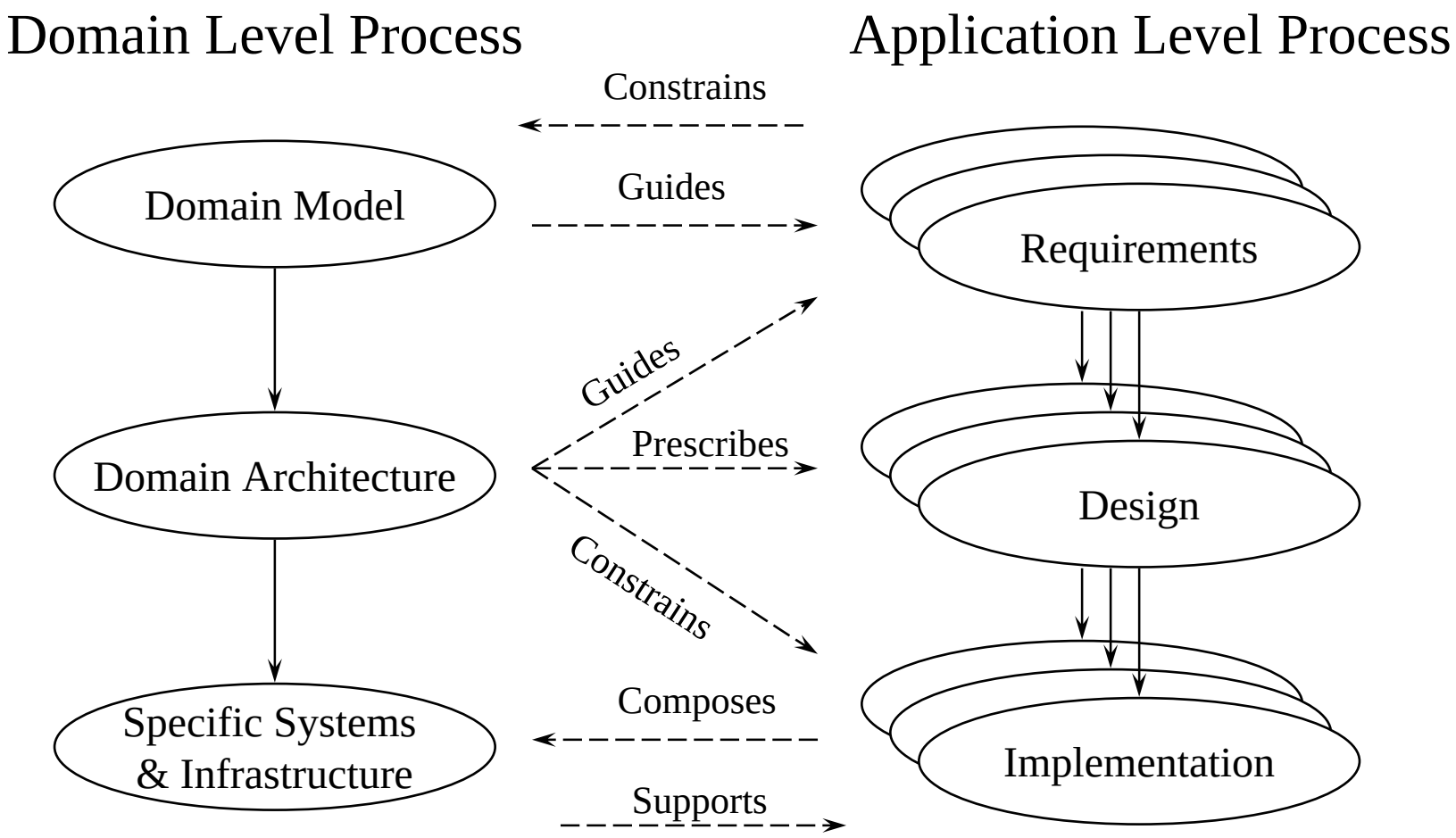
- 在领域工程中，DSSA作为开发可复用构件和组织可复用构件库的基础
  - DSSA说明了功能如何分配到其实现构件，并说明了对接口的需求，因此，该领域中的可复用构件应依据DSSA来开发
  - DSSA中的构件规约形成了对领域中可复用构件进行分类的基础，这样**组织构件库**，有利于构件的检索和复用

DSSA定义-3

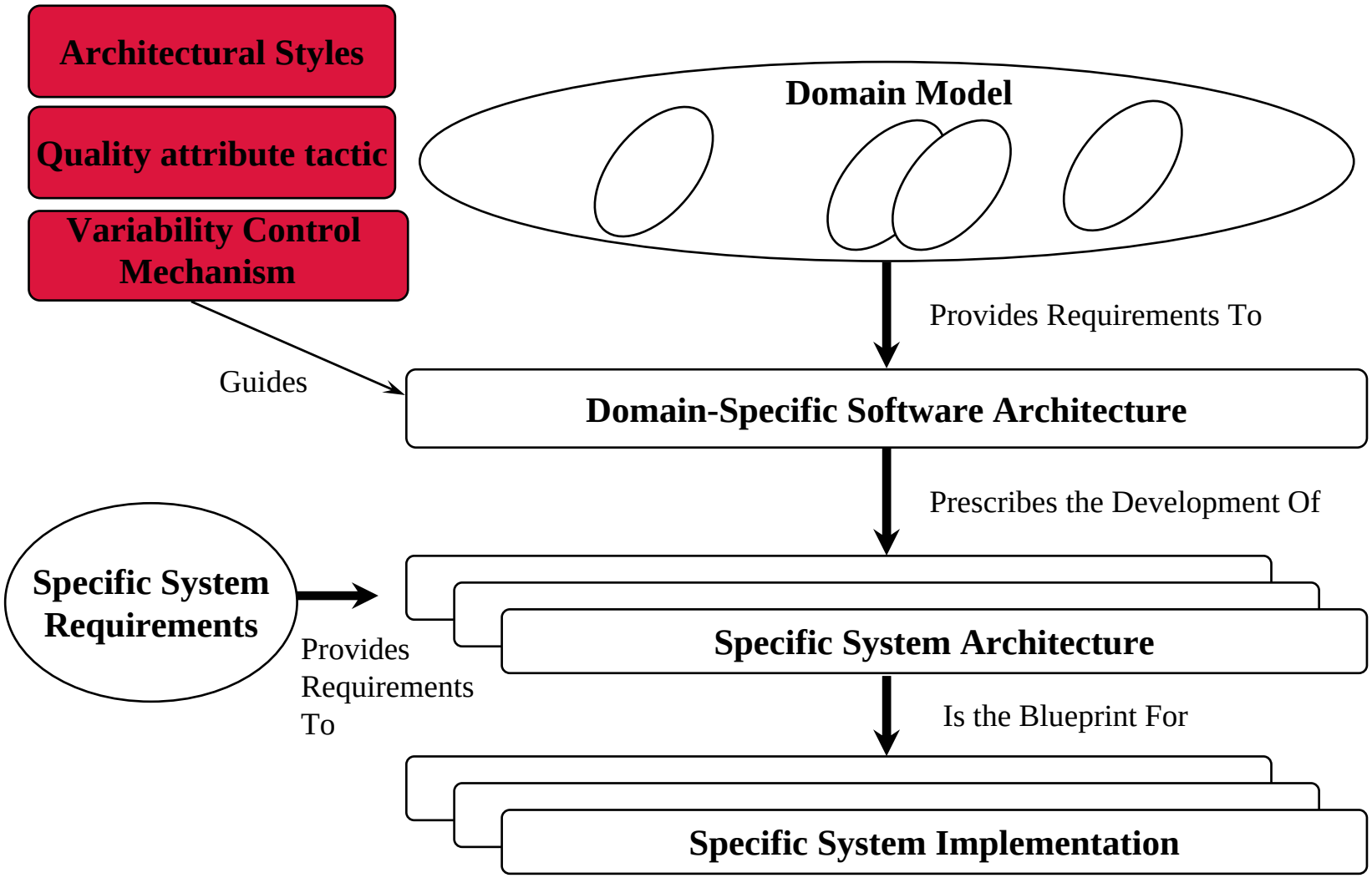
- 在应用工程中，经裁剪和实例化形成特定应用的体系结构
- 由于领域分析模型中的领域需求具有一定的变化性，DSSA也要相应地具有变化性，并提供内在的机制在具体应用中实例化这些变化性
- DSSA在变化性方面提出了更高的要求。具体应用之间的差异可能表现在行为、质量-属性、运行平台、网络、物理配置、中间件、规模等许多方面，例如，
  - 一个应用可能要求高度安全，处理速度较慢；而另一个应用要求速度快，安全性较低。体系结构必须具有足够的灵活性同时支持这两个应用

领域变化性控制机制

领域与应用的关系


Domain Level Process
Application Level Process
Constrains
Guides
Domain Model
Requirements
Guides
Prescribes
Domain Architecture
Design
Constrains
Composes
Specific Systems
& Infrastructure
Implementation
Supports

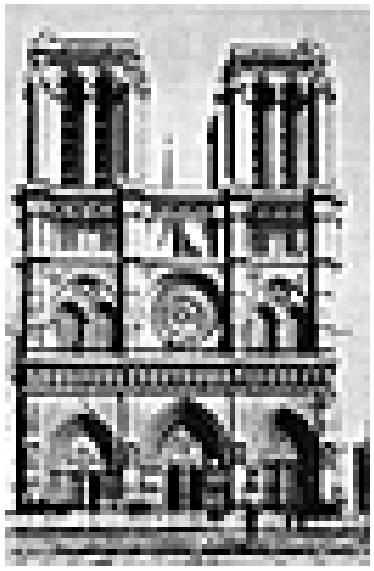
DSSA构造的关键技术



内容

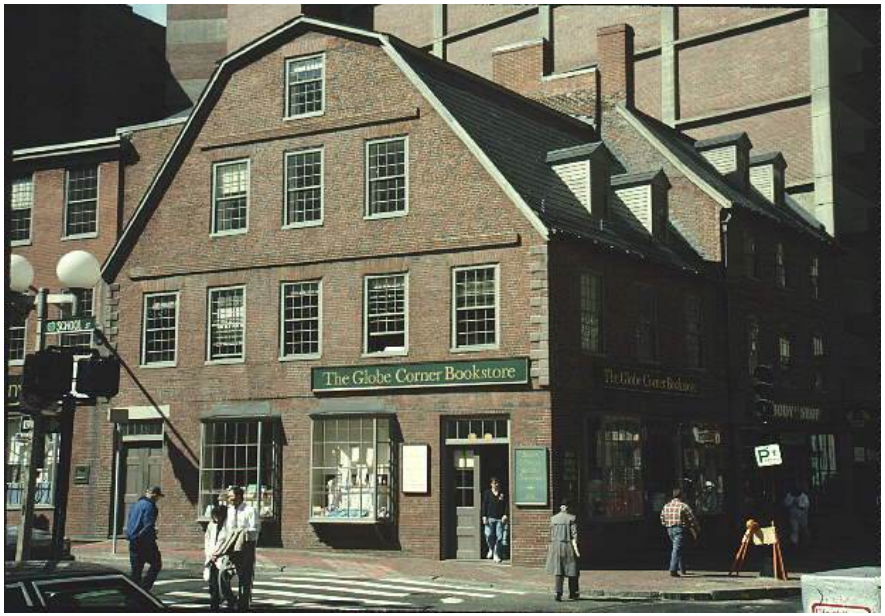
- 软件体系结构概念
  - 定义
  - 重要性
- 特定领域的软件体系结构构造（DSSA)
  - DSSA定义
  - 体系结构风格
    - 体系结构风格定义
    - 典型体系结构风格
  - 质量属性与实现
    - 理解质量属性
    - 质量属性实现通用策略
  - 领域变化性控制机制
    - 变化性维度
    - 变化性控制机制
- 体系结构文档

建筑风格



Early Gothic Architecture
Notre Dame, Paris
哥特式流行于西欧的12世纪到15世纪的一种建筑风格的，
特征是有尖角的拱门

18th Century Georgian Architecture
"Old Corner Bookstore"
乔治王朝的建筑风格
(1714 - 1830年乔治一世至乔治四世时代)



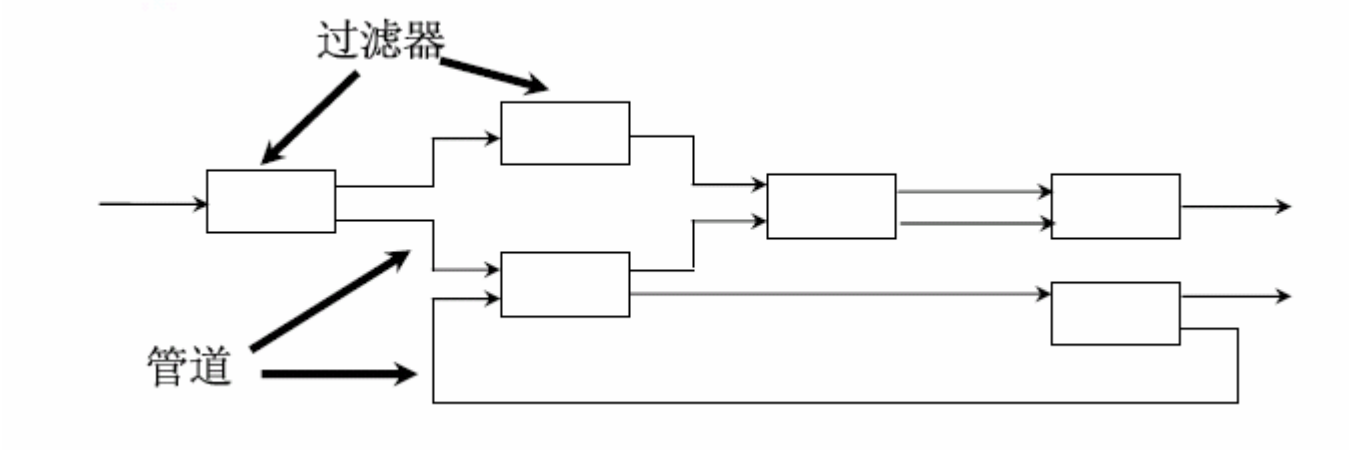
软件体系结构风格概念

- 软件体系结构风格（Architectural style）
- 一种体系结构风格定义了一个系统家族
  - 关于构件和连接件类型的术语
  - 一组约束它们组成方式的规定
  - 一个或多个语义模型，规定了如何从各成分的特性决定系统整体特性
- 概括地说，一种体系结构风格刻画了一个具有共享结构和语义的系统家族。

常见的体系结构风格

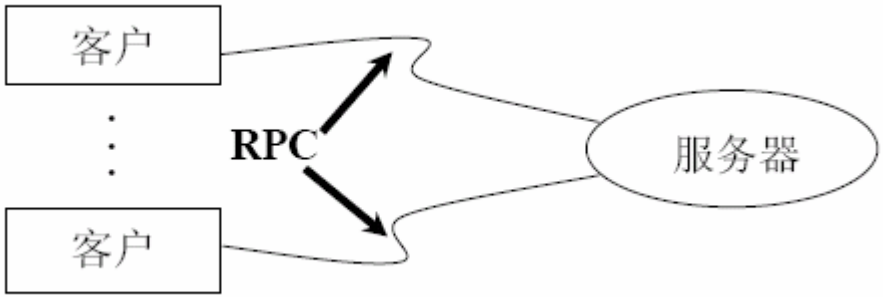
- 数据流系统
  - 批处理系统
  - 管道/过滤器系统
- 调用/返回系统
  - 主程序/子程序系统
  - 面向对象系统
- 独立构件
  - 通信进程系统
  - 事件系统
- 虚拟机
  - 解释器
  - 基于规则的系统
- 数据为中心的系统
  - 数据库
  - 超文本系统

管道/过滤器 Pipes and Filters



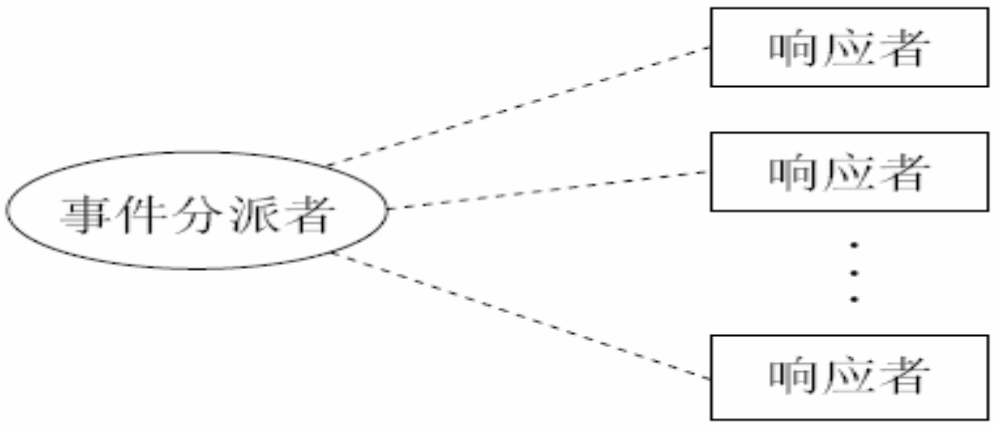
- 优点：(1) 结构简单：系统的行为是所有过滤器行为的简单复合；(2) 系统易于维护和增强：增加新过滤器，替换旧过滤器；(3) 支持复用：过滤器只同其输入、输出端口的数据相关；(4) 各过滤器可以并发运行。
- 缺点：(1) 容易导致批处理方式：每个过滤器从输入数据到输出数据的转换是一个整体。转换通常不适合交互式的应用；(2) 有时必须维护两个分离而又相关的流之间的对应关系；(3) 同实现有关，过滤器之间的数据传输率较低，而且每个过滤器都要作类似的数据打包和解包的工作。

Client-Server风格



- 优点：(1) 结构简单，系统中不同类型的任务分别由客户和服务器承担，有利于发挥不同机器平台的优势；(2) 支持分布式、并发环境，特别是当客户和服务器之间的关系是多对多时，可以有效地提高资源的利用率和共享程度；(3) 服务器集中管理资源，有利于权限控制和系统安全。
- 缺点：在大多数client-server风格的系统中，构件之间的连接通过（远程）过程调用，接近于代码一级，表达能力较弱。

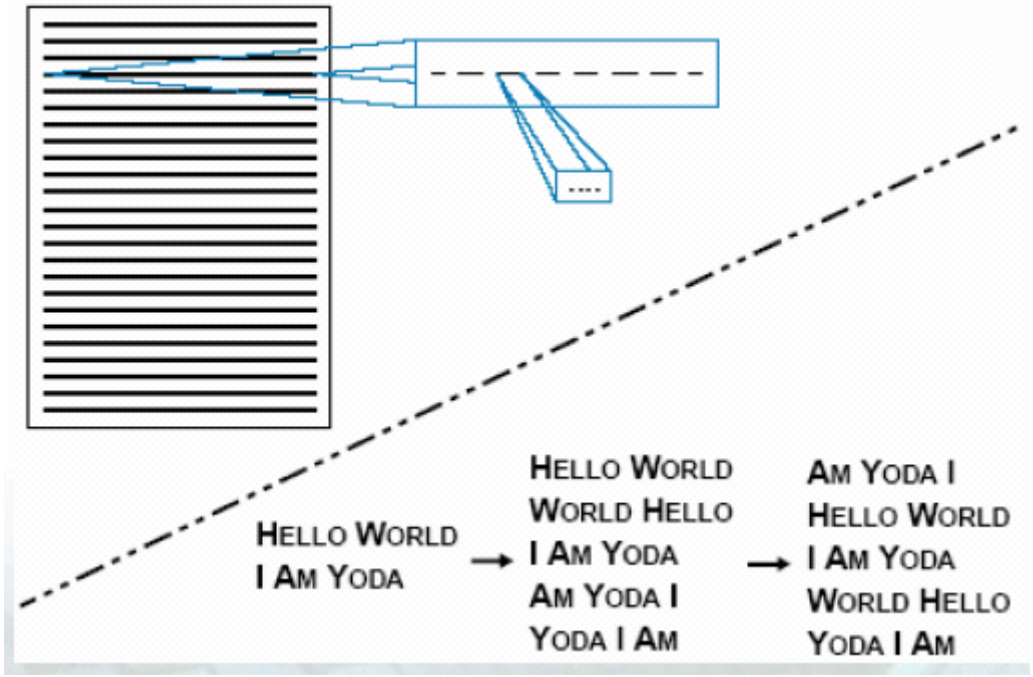
Event-based风格



- 优点：(1) 支持复用，构件通过登记它所感兴趣的事件被引入系统；(2) 便于系统演化，构件可以容易地升级或更换。
- 缺点：(1) 系统行为难以控制，发出事件的构件放弃了对系统的控制，因此不能确定系统中有无或有多少其它构件对该事件感兴趣，系统的行为不能依赖于特定的处理顺序。(2) 同事件关联的会有少量的数据，但有些情况下需要通过共享区传递数据，这时就涉及到全局效率和资源管理问题。

例子KWIC-1

- 问题陈述：KWIC(KeyWord In Context)，Parnas(1972)*KWIC*索引系统接收行*(Lines)*的有序集合，行是字*(Words)*的有序集合，字是字符*(Characters)*的有序集合。任何行可以字为单位循环移位。*KWIC*索引系统输出一个所有行、经过所有循环移位后的排序列表。

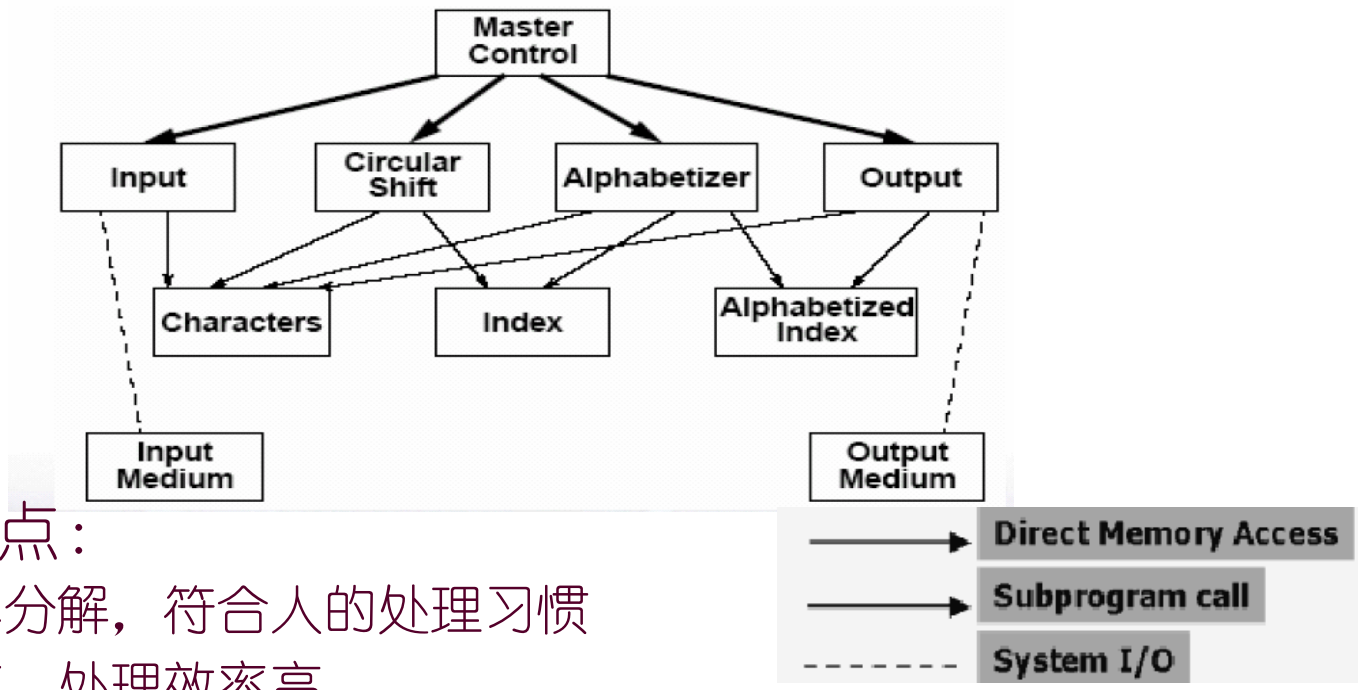


KWIC-2

- 目的：考察不同的体系结构风格对变化的适应能力
- 评价准则
  - 处理算法的改变：例如，行的移位可在每行读入后、在所有行读入后、或当排序要求一组移位的行时；
  - 数据表示的改变：例如，行、字、字符可以不同的方式存储；类似地，循环移位后的行可以显式或隐式存储（索引和偏移量的偶对）；
  - 系统功能的增强：例如，排除以某些“修饰词”(a, an, and等)打头的移位结果；支持交互，允许用户从原始输入表中删除一些行等；
  - 效率：时间和空间；
  - 复用：构件被复用的潜力。

KWIC-3

- 方案1：Main Program/Subroutine with Shared Data



方案优缺点：

+系统自然分解，符合人的处理习惯

+数据共享，处理效率高

+宜于功能完善，例如排除噪音词汇

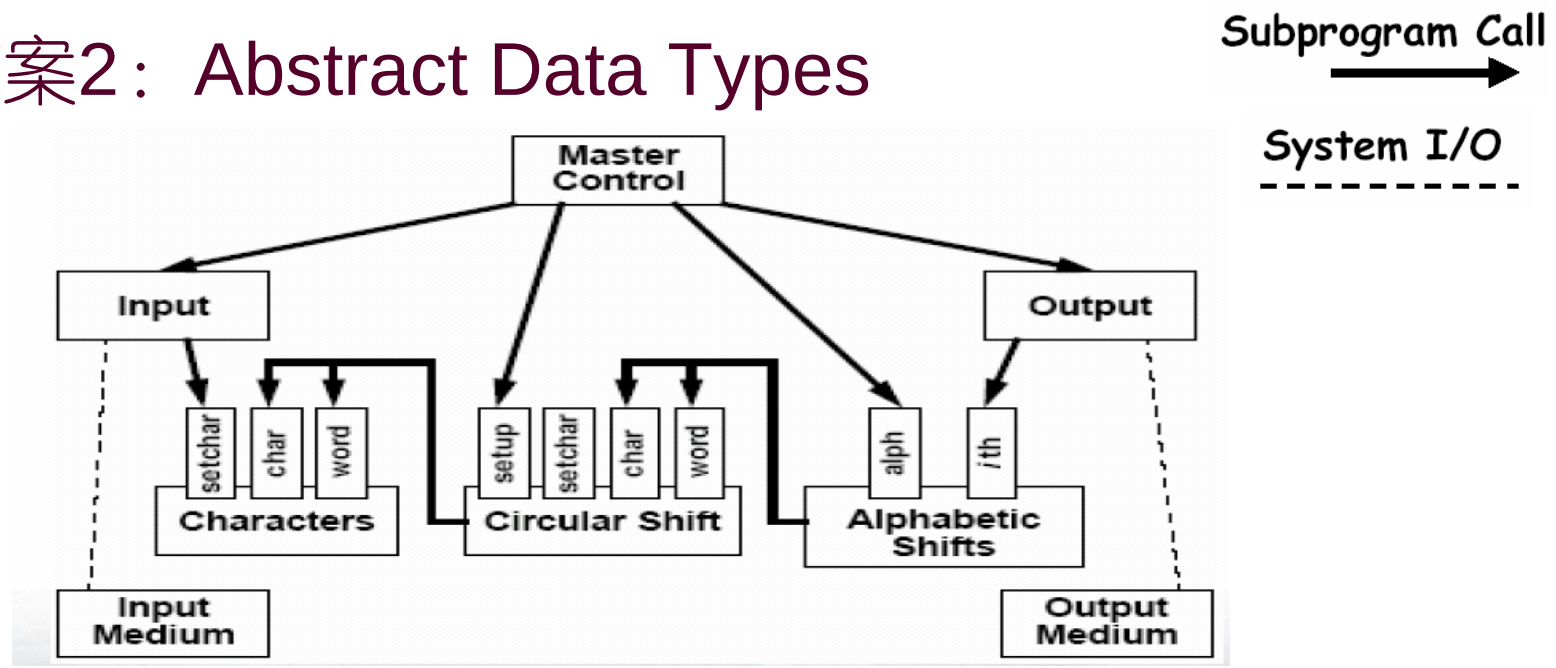
–改变数据的表示将影响所有的模块，数据存储格式对所有模块而言都是显式的，没有信息隐藏

–整体处理算法的变化

–系统构件难以支持复用，紧耦合。

KWIC-4

- 方案2：Abstract Data Types



the same data is stored in several components

方案优缺点：

+数据表示的变化对处理模块影响不大

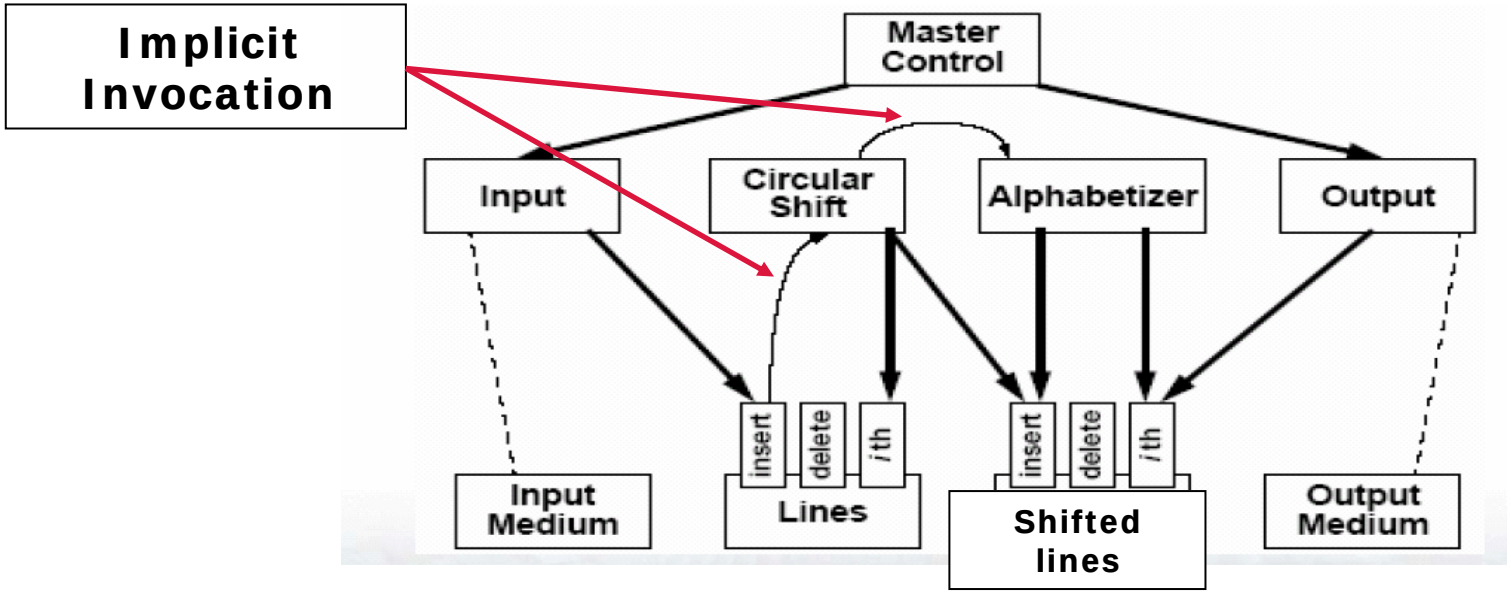
+系统构件可以较好地支持复用

–对系统功能的增强难以很好适应，必须牺牲概念简捷性或效率

–性能，可能需要更多的空间，通过接口访问可能稍微减慢速度

KWIC-5

- 方案3：Implicit Invocation with Shared Data



方案优缺点：

+适应变化：系统功能的增强只需登记其它对数据变化事件感兴趣的构件即可；数据表示的变化局部化，并且同处理相分离

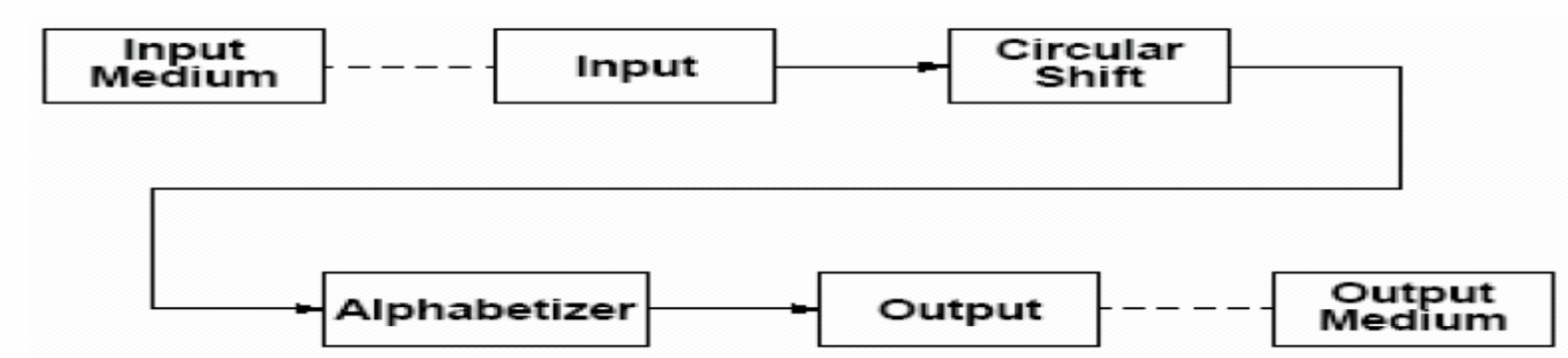
+系统构件可以较好地支持复用，模块只依赖于事件，而非其他模块

–难以控制隐式调用构件的处理顺序，特别是在并发系统中

–基于数据驱动的调用，倾向于使用更多的存储空间

KWIC-6

- 方案4：Pipe and Filter



方案优缺点：

+系统自然分解，符合人们直观的处理顺序

+系统构件支持复用，过滤器之间独立运行

+适应算法的变化，容易增强系统的功能

–难以支持系统的交互

–构件之间的数据传递通过拷贝进行，而非数据共享，易于造成空间和时间浪费

KWIC-7

方案比较：

1. Main Program/Subroutine with Shared Data
2. Abstract Data Types
3. Implicit Invocation with Shared Data
4. Pipe and Filter

	1	2	3	4
算法改变	−	−	+	+
数据表示改变	−	+	+	−
系统功能改变	+	−	+	+
效率	+	+	−	−
复用	−	+	+	+

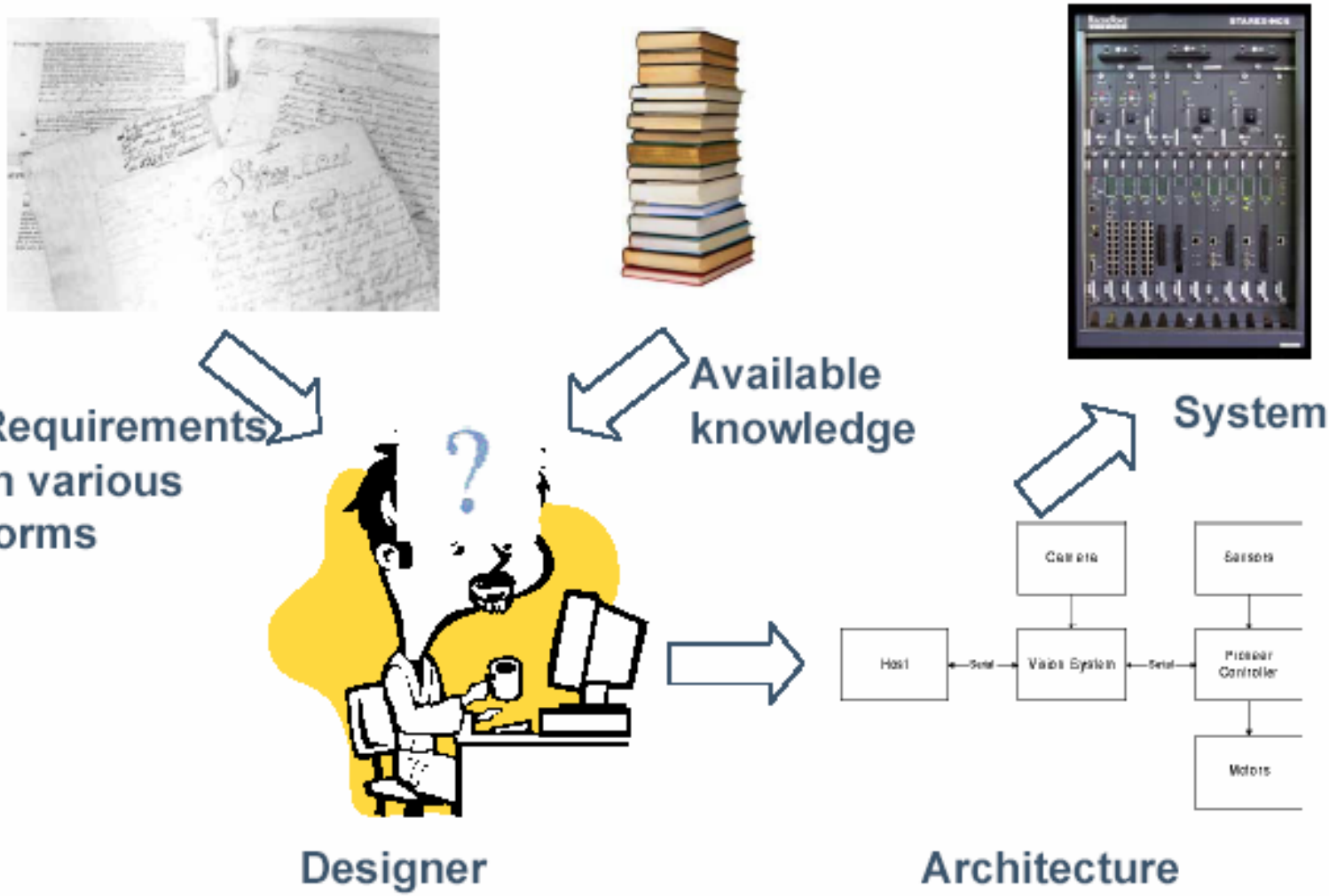
结论：

- 这并不是说某种方案一定比另一种方案好，还需要考虑系统的具体应用环境；
- 实际上，应用系统很少纯粹是一种风格的，往往是多个风格的综合。

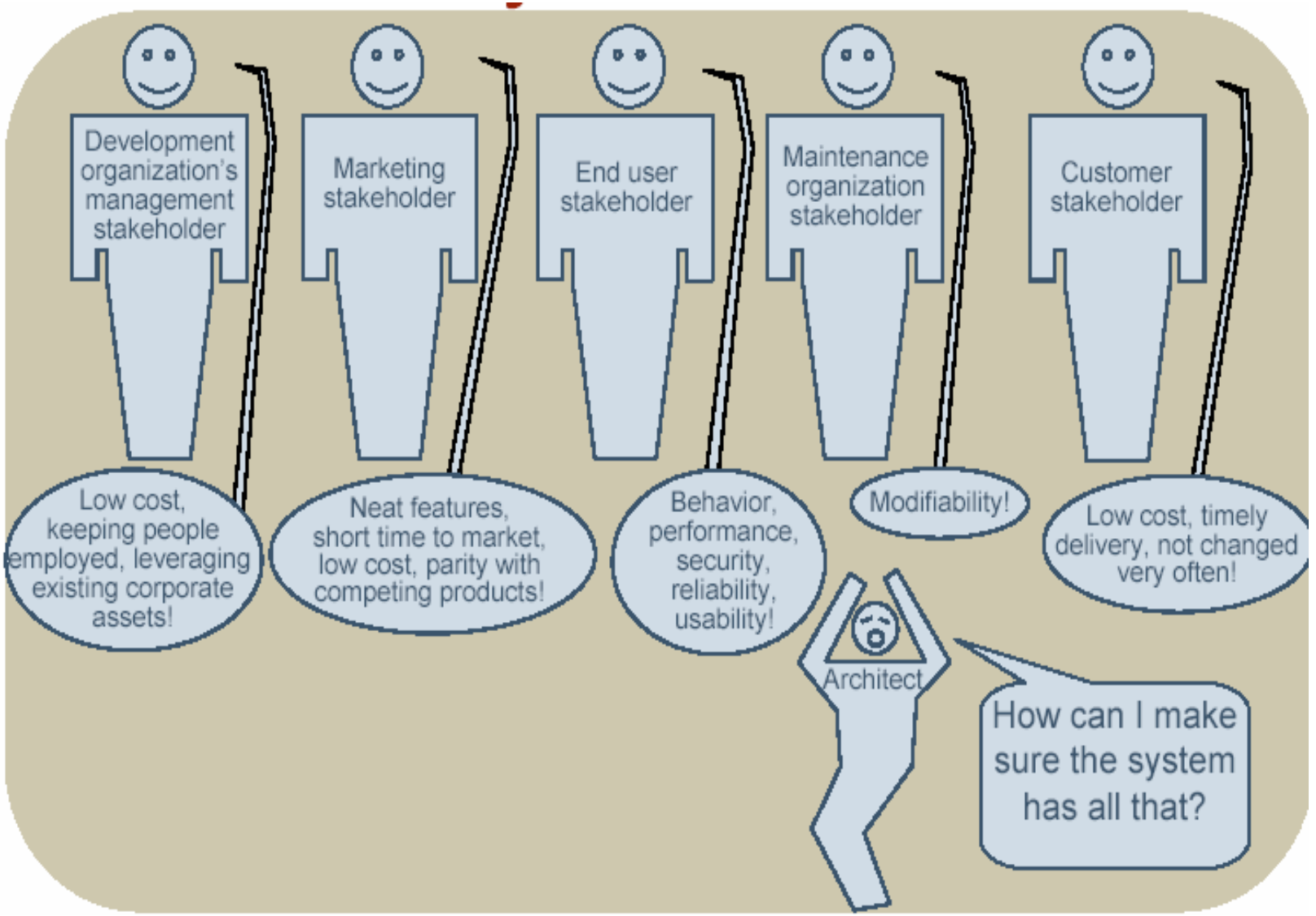
内容

- 软件体系结构概念
  - 定义
  - 重要性
- 特定领域的软件体系结构构造（DSSA)
  - DSSA定义
  - 体系结构风格
    - 体系结构风格定义
    - 典型体系结构风格
  - 质量属性与实现
    - 理解质量属性
    - 质量属性实现通用策略
  - 领域变化性控制机制
    - 变化性维度
    - 变化性控制机制
- 体系结构文档

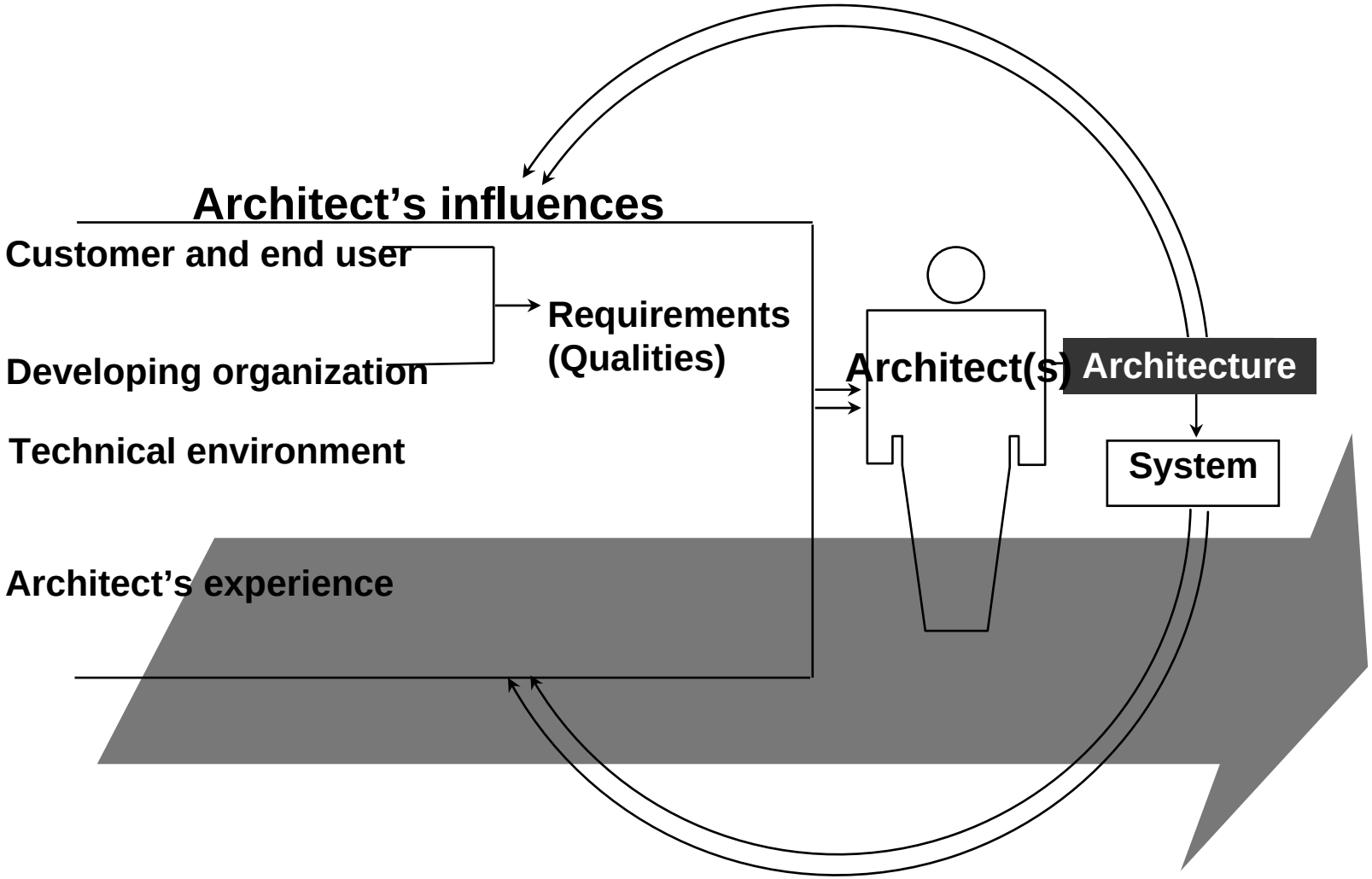
Design


Requirements in various forms
Available knowledge
System
Designer
Architecture
Camera
Sensors
Host
Serial
Vision System
Serial
Pioneer Controller
Motors

系统风险承担者的影响


Development organization's management stakeholder
Marketing stakeholder
End user stakeholder
Maintenance organization stakeholder
Customer stakeholder
Low cost, keeping people employed, leveraging existing corporate assets!
Neat features, short time to market, low cost, parity with competing products!
Behavior, performance, security, reliability, usability!
Modifiability!
Low cost, timely delivery, not changed very often!
Architect
How can I make sure the system has all that?

对体系结构设计师的影响

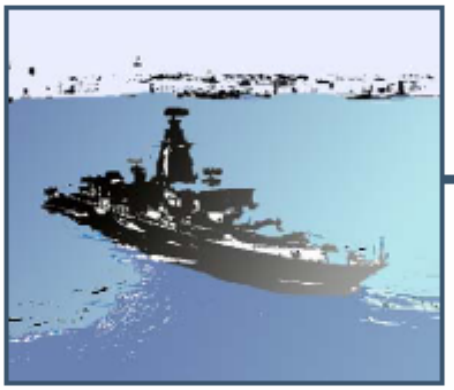


质量

Examples

- Usability
- Modifiability
- Performance
- Security
- Availability
- Testability
- On cost, on schedule
- Cost and benefit
- Integration with legacy systems
- Functionality?

功能性与质量



Functional Software Requirements

If function were all that mattered, any monolithic software would do, ..***but other things matter***…

The important quality attributes and their characterizations are key.

- Modifiability
- Interoperability
- Availability
- Security
- Predictability
- Portability
- :

analysis, design, development

Quality Attribute Drivers → Software Architecture → Software

has these qualities

体系结构与质量属性

- 实现质量属性必须在设计、实现、部署阶段进行考虑
  - 性能，在体系结构方面，取决于构件之间的通信，分配的资源；在非体系结构方面，取决于算法的选择和算法的具体编码
  - 可修改性，体系结构方面：功能性如何分解；非体系结构方面：模块内部的编码技术
  - 使用性：体系结构方面：系统是否提供“undo”能力给用户，用户是否能够复用以前的输入；非体系结构方面：字体，按钮的风格，用户界面的清晰和易用
- 质量属性会之间有消极的影响
  - security and usability
  - performance and modifiability

质量属性种类

- Qualities of the system
  - Availability：系统能够正常运行的时间比例
  - Performance：系统响应能力
  - Modifiability：快速、低成本修改系统的能力
  - Security：阻止非法入侵或拒绝服务的能力
  - Testabiltiy：软件通过测试，易于发现错误的能力
  - Usability：便于用户完成期望的事情的能力
  - Functionality：系统完成预定工作的能力
- Business qualities
  - Time-to-market
  - Cost and benefit
  - Projected lifetime
- Overall architectural qualities
  - Conceptual integrity
  - Correctness and completeness
  - Constructability：在规定的时间里，构建某个系统的难易程度。

Creating An Architecture

- “Beauty is in the eye of the beholder”
  - Booth Tarkington (1838–1918). The Magnificent Ambersons. 1918.
- 体系结构设计师所谈的“质量”是站在谁的角度?
  - 体系结构设计师的角度?
  - 客户的角度?
  - 均不是?
- 我们能不能少些主观多些客观来看待质量?
  - 使用质量属性场景（quality attribute scenarios）

质量属性场景

- 质量属性场景（Quality Attribute Scenario ）是一个具体的质量属性需求，场景就是风险承担者与系统的交互的简短陈述。

包括六个部分

- **刺激源**：可以是一些实体，如人，计算机系统或者其他的刺激源
- **刺激**：到达系统的刺激是一个需要被考虑的情况
- **环境**：刺激在一定的条件下发生
- **制品**：某些制品被激发，可能是整个系统或者其中的一部分
- **反应**：反应是刺激到达后所产生的活动
- **反应测量**：当反应发生的时候，必须可被测量，从而需求也被检测

Example Concrete Availability Scenario

一条没有预料到的外部消息被一个正常运行的进程所获得，进程通知运算器接收这条消息，并继续执行而不停止。

Source:
External
to system

Stimulus:
Unanticipated
Message

Artifact:
Process

Environment:
Normal operation

Response:
Inform operator
continue to
operate

**Response
Measure:**
No Downtime

Sample security scenario

- 一个正确标识的个人试图从外部站点修改系统的数据，系统维护审计跟踪，并在一天内恢复正确的数据。

Source:
Correctly Identified Individual

Stimulus:
Tries to modify information

Artifact:
Data within the system

Environment:
Under normal operations

Response:
System maintains audit trail

Response Measure:
Correct data Is restored within a day

Availability General Scenario Generation

可用性关心的是系统失效及其相关的结果。当系统不再提供与规约一致的服务时就是失效的。

Scenario Portion	Possible Values
Source	Internal to system or external to system
Stimulus	Fault:omission, timing, no response, incorrect response
Artifact	System’s processors, communication channels, persistent storage,processes
Environment	Normal operation; degraded (failsafe) mode
Response	Log the failure, notify users/operators
RespMeasure	Time interval when it must be available, repair time in degraded mode

Modifiability General Scenario Generation

可修改性关心的是变化的代价。关系到什么制品变化了，以及什么时候进行变化，谁来实施变化等方面。

Scenario Portion	Possible Values
Source	End-user, developer, system-administrator
Stimulus	Add/delete/modify functionality or quality attr.
Artifact	System user interface, platform, environment
Environment	At runtime, compile time, build time, design-time
Response	Locate places in architecture for modifying, modify, test modification, deploys modification
RespMeasure	Cost in effort, money, time, extent affects other system functions or qualities

Performance General Scenario Generation

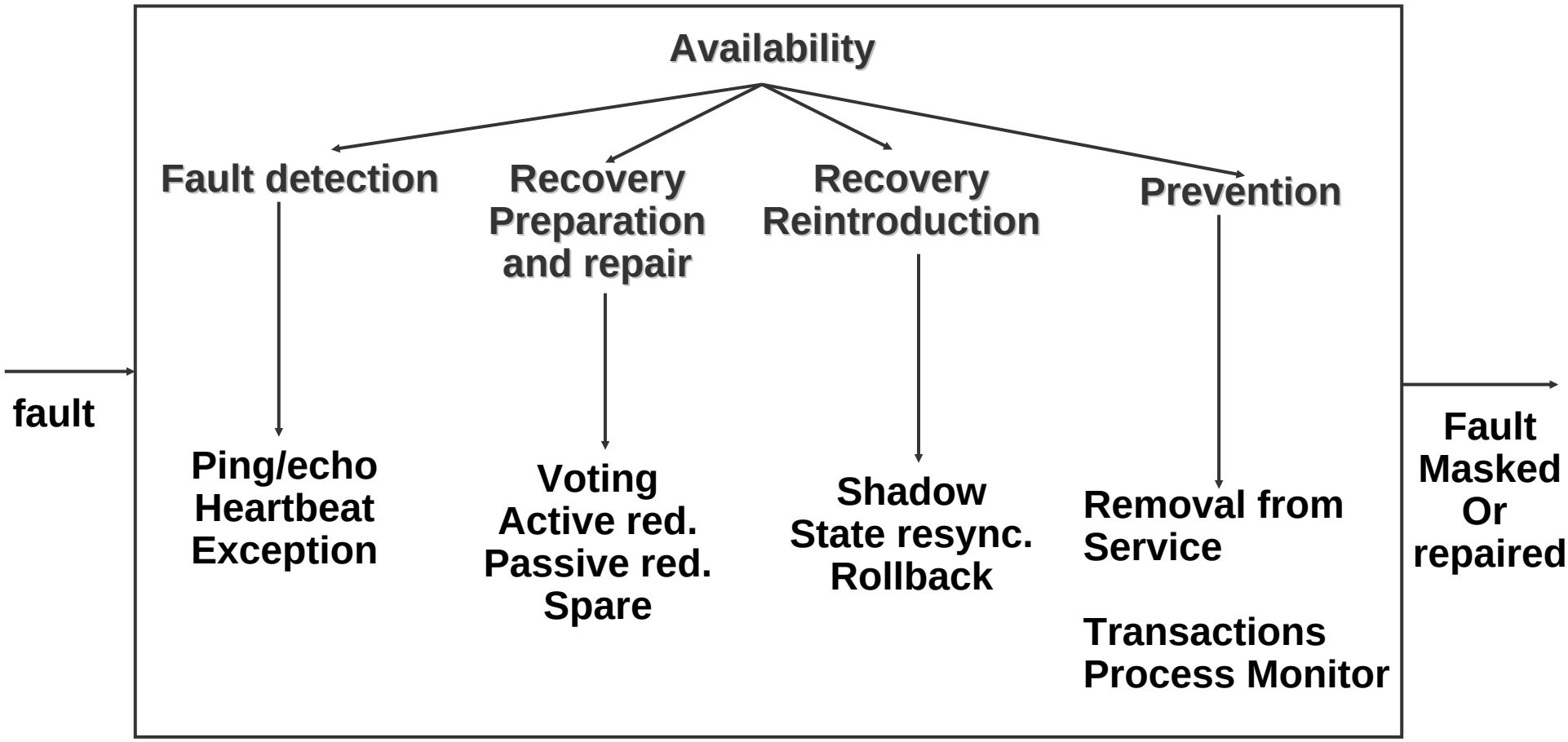
性能与时间相关，关系到事件的相应速度。涉及到事件的数目和事件到达的模式.

Scenario Portion	Possible Values
Source	A number of sources both external and internal
Stimulus	Periodic events, sporadic events, stochastic events
Artifact	System, or possibly a component， user interface..
Environment	Normal mode; overload mode
Response	Process stimuli; change level of service
RespMeasure	Latency, throughput, miss rate, data loss

质量属性策略

- 如何实现质量属性?
- 策略(tactics)是影响质量属性反应的设计决策，并将策略(tactics)的集合称为体系结构策略(strategy),反应了某种体系结构风格。

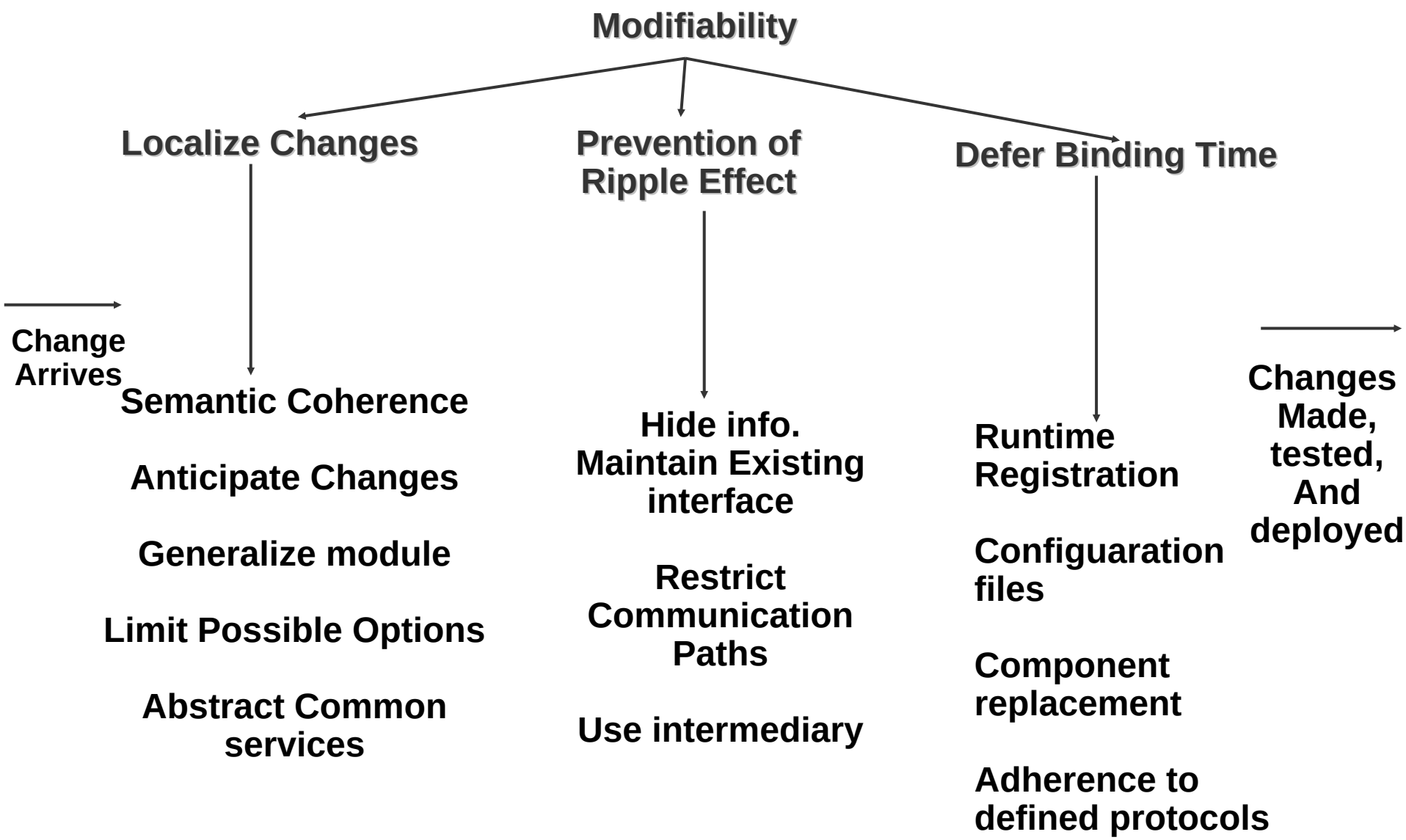
Availability Tactics Hierarchy



Availability Tactics

- 故障检测
  - 询问/回音检测法：一个构件向别的构件发出询问并期望在预定的时间内得到回音
  - 心跳检测法：一个构件定期发出心跳消息，而另外的构件则监听这个心跳消息
  - 异常检测法：当系统有异常的时候，可以作为检测故障的方法。
- 恢复：准备和修复
  - 选举法：在冗余处理器上运行的进程都具有相同的输入，并计算出一个简单的输出，发送给投票者。
  - 主动冗余：所有的冗余构件都对并行的事件做出反应且仅仅只需要使用其中一个冗余构件的结果，而其它的都被忽略。
  - 被动冗余：冗余构件有主从之分，主构件对事件进行反应，并通知其它的构件进行必要的状态更新。
- 恢复：再引入
  - 影子操作：以前失败的构件可以在影子模式下运行一小段时间，确保在恢复前的行为正确。
  - 状态重新同步：在被动冗余策略和主动冗余策略中，要求被恢复的构件的状态再恢复前要更新状态。
  - 检查点/回滚：检查点是周期性或者反应于特定事件而创建的一致性状态的纪录。
- 故障预防
  - 移除：移走一个构件，以防止某些可预见的失败。
  - 事务：事务是一系列的步骤，这些步骤要么全做，要么不做。
  - 进程监视：一旦一个进程中的缺陷被发觉，监视进程能够删除该无用的进程并创建新的实例，并初始化到某合适的状态。

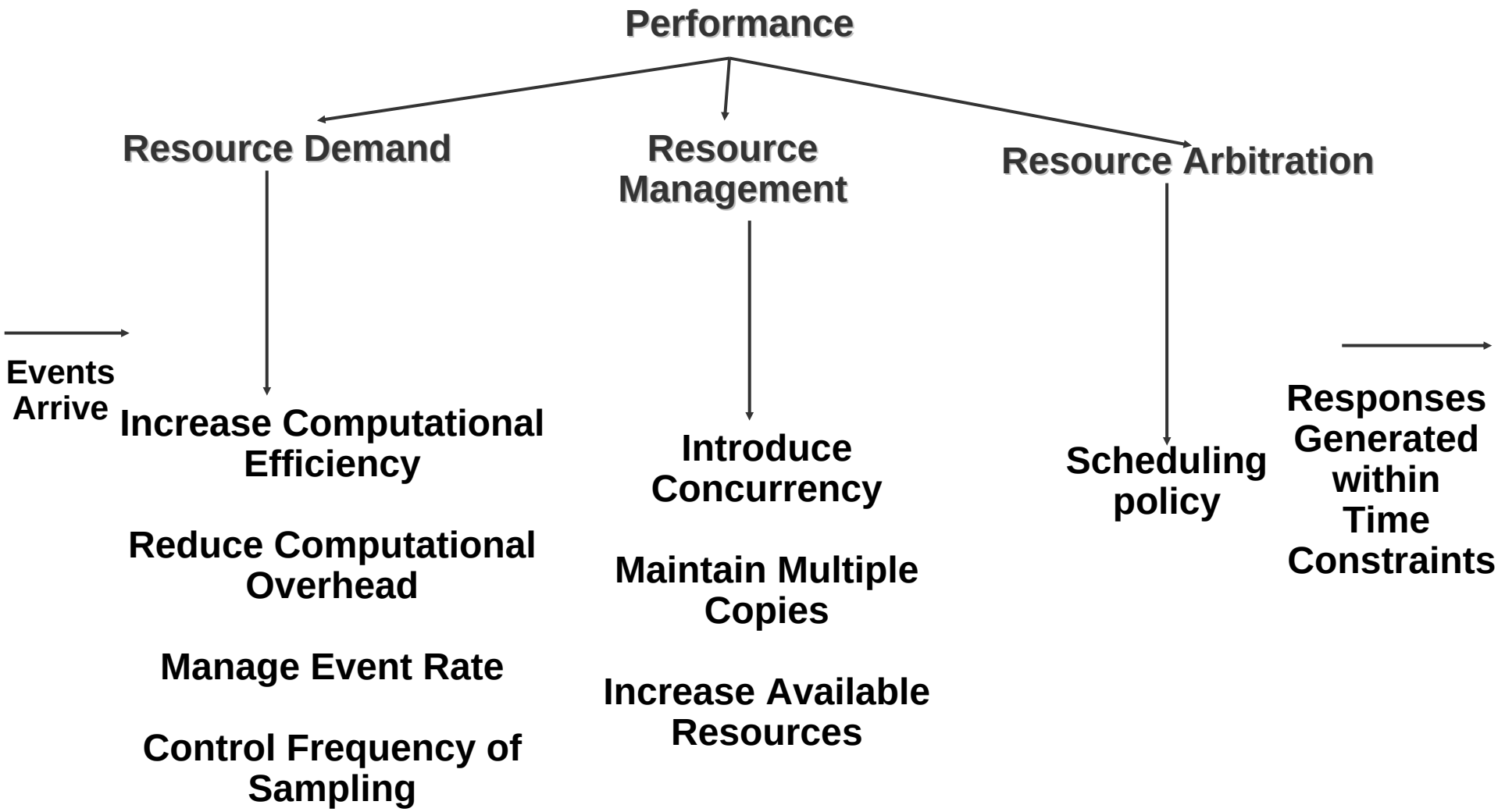
Modifiability Tactics Hierarchy



Modifiability Tactics

- 使变化局部化
  - 维持语义上的一致性：语义一致性指模块里职责之间的关系，目标是在不过分依赖的其它模块的条件下，这些职责协同工作。
  - 模块通用化：使模块更通用，基于输入提供更多的功能。
  - 限制可能的选项，将减少修改带来的影响。
- 防止波动效应：修改引起的波动效应是因为模块修改的间接影响造成的。
  - 信息隐蔽
  - 维护现存接口：添加接口， 添加适配器，提供存根
  - 使用中介，隔离双方
- 延绑定时间
  - 运行时注册
  - 配置文件
  - 多态性
  - 构件置换

Performance Tactics Hierarchy



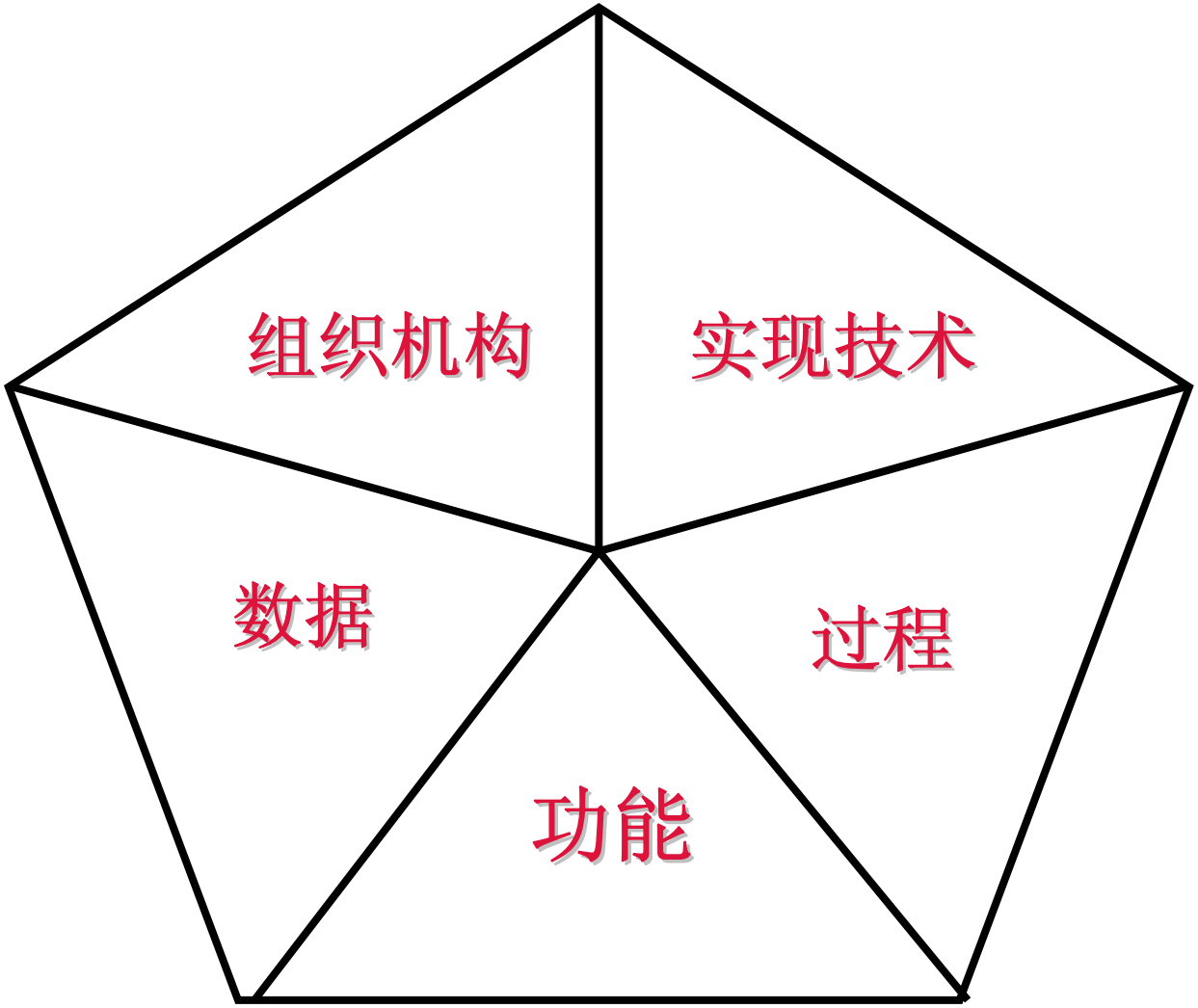
Performance Tactics

- 资源需求
  - 事件流是资源需求的起源。需求的两个特征是：需求的频率和每个需求的资源消耗量。
  - 降低延时的一个策略为降低处理一个事件流所要求的资源：
    - 提高计算效率
    - 减少计算花费
  - 另外一个策略为减少所处理的事件的数量：
    - 管理事件频率
    - 控制采样频率
- 资源管理：管理资源也可以对响应时间产生影响
  - 引入并发
  - 维护数据或者计算的多个备份
  - 增加可用的资源
- 资源仲裁：调度政策从概念上涉及两个部分：优先级分配和分发，通用的调度策略有：
  - 先进先出
  - 固定优先级调度
    - 语义重要性调度；固定最后期限调度；固定周期调度
  - 动态优先级调度
    - 循环调度策略
    - 早到期先调度策略

内容

- 软件体系结构概念
  - 定义
  - 重要性
- 特定领域的软件体系结构构造（DSSA)
  - DSSA定义
  - 体系结构风格
    - 体系结构风格定义
    - 典型体系结构风格
  - 质量属性与实现
    - 理解质量属性
    - 质量属性实现通用策略
  - 领域变化性控制机制
    - 变化性维度
    - 变化性控制机制
- 体系结构文档

领域变化性维度



组织机构

- 变化的原因：
  - 组织机构变动
  - 机构变更
  - 人事变动
  - 运作模式的变动
- 变化性控制机制
  - 引进“角色”的概念，采用基于角色的控制。由于实现了用户与访问权限的逻辑分离，基于角色的控制极大的方便了组织结构的变化 。

数据

- 变化的原因
  - 系统业务功能/对外服务的变化；
  - 实现技术/运行环境的变化；
  - 异构平台/异构系统之间的数据交换；
- 变化性控制机制
  - 表示层、应用逻辑层、数据层，三层体系结构
  - 数据描述和内容分离，元数据与业务数据
  - 参数化的基础数据，应用抽象的原则，提供参数化机制，支持业务数据的变化性
  - 接口与实现分离，数据库中间件，标准的数据库访问接口
  - 标准的数据表示和访问协议
  - 采用公式机制来控制数据加工和数据约束的变化性

功能

- 变化的原因
  - 业务功能/对外服务的变化
- 变化性控制机制
  - 构件配置/定制，通过设置业务构件的参数和属性，来实现构件行为上的变化，从而改变业务功能
  - 插件是应用系统提供的一种扩展机制，用接口与实现分离的原则，提供标准化的应用接口
  - 控制器是一个系统功能集成工具，相当于一个构件容器，向控制器添加构件就是为应用系统添加功能的过程。
  - 构件集成框架，进行构件的动态集成，为应用添加新功能或删除原有功能，支持功能的增加和删除。

过程

- 变化的原因
  - 组织机构、人员职责变化
  - 业务发展的需要，业务过程本身的改变
- 变化性控制机制
  - 过程引擎构件
  - 可以使用构件集成框架提供的动态组装机制，进行运行时刻的集成，动态构造和改变业务过程
  - 也使用脚本语言来编写构件的胶合代码，构造和改变业务过程

实现

- 变化的原因
  - 技术的进步和环境的变化，在应用系统中表现为①系统表现方式的变化，②运行环境的变化。
- 变化性控制机制
  - 三层体系结构
  - 界面定制器，例如提供常用的界面模板方便进行界面定制 。
  - 控制运行环境变化性的两个主要机制是：虚拟机和中间件

DSSA对变化性的支持

- DSSA对变化性的支持可以采取多种形式：
  - 参数化：所有的变化性事先明确，并固化在代码内，在系统构造时对构件、子系统等的形式参数进行赋值。
  - 继承和代理：面向对象系统提供的子类对父类的属性和行为进行改变的机制。
  - 构件替换：在保持接口一致的前提下，可以用功能强大的构件替换以前较弱的构件；此外，还包括构件数目的扩充。

内容

- 软件体系结构概念
  - 定义
  - 重要性
- 特定领域的软件体系结构构造（DSSA)
  - DSSA定义
  - 体系结构风格
    - 体系结构风格定义
    - 典型体系结构风格
  - 质量属性与实现
    - 理解质量属性
    - 质量属性实现通用策略
  - 领域变化性控制机制
    - 变化性维度
    - 变化性控制机制
- 体系结构文档

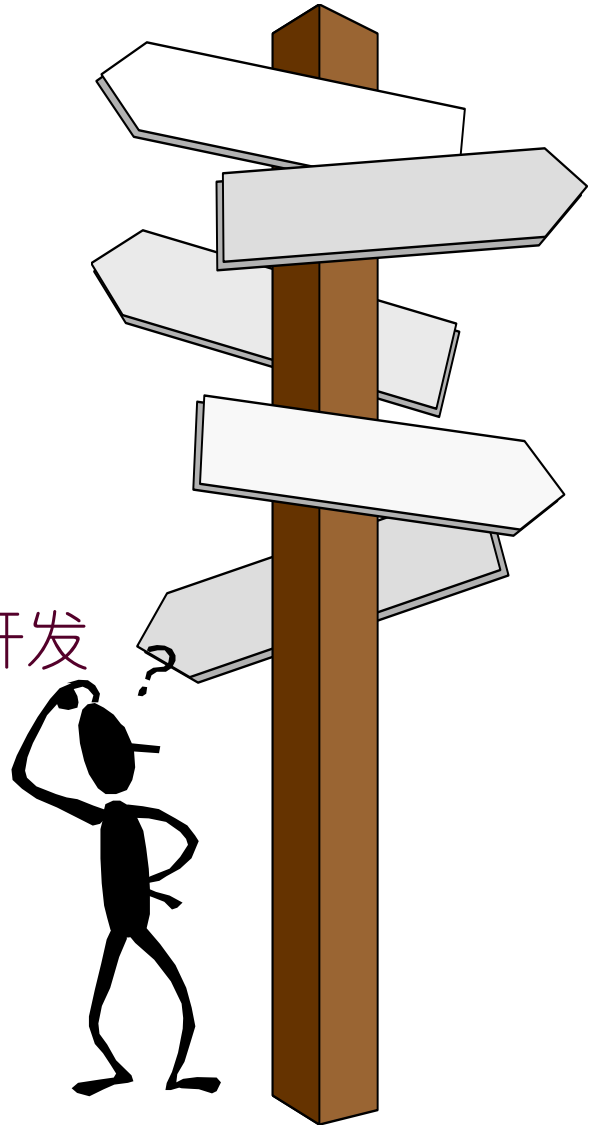
体系结构视图

- Kruchten's 4+1 views
- Siemens Four-Views (Hofmeister, Nord, Soni, *Applied Software Architecture*, 2000):
  - Conceptual view
  - Module interconnection view
  - Execution view
  - Code view
- Herzum & Sims (*Business Component Factory*, 1999):
  - Technical architecture
  - Application architecture
  - Project management architecture
  - Functional architecture

体系结构文档的作用

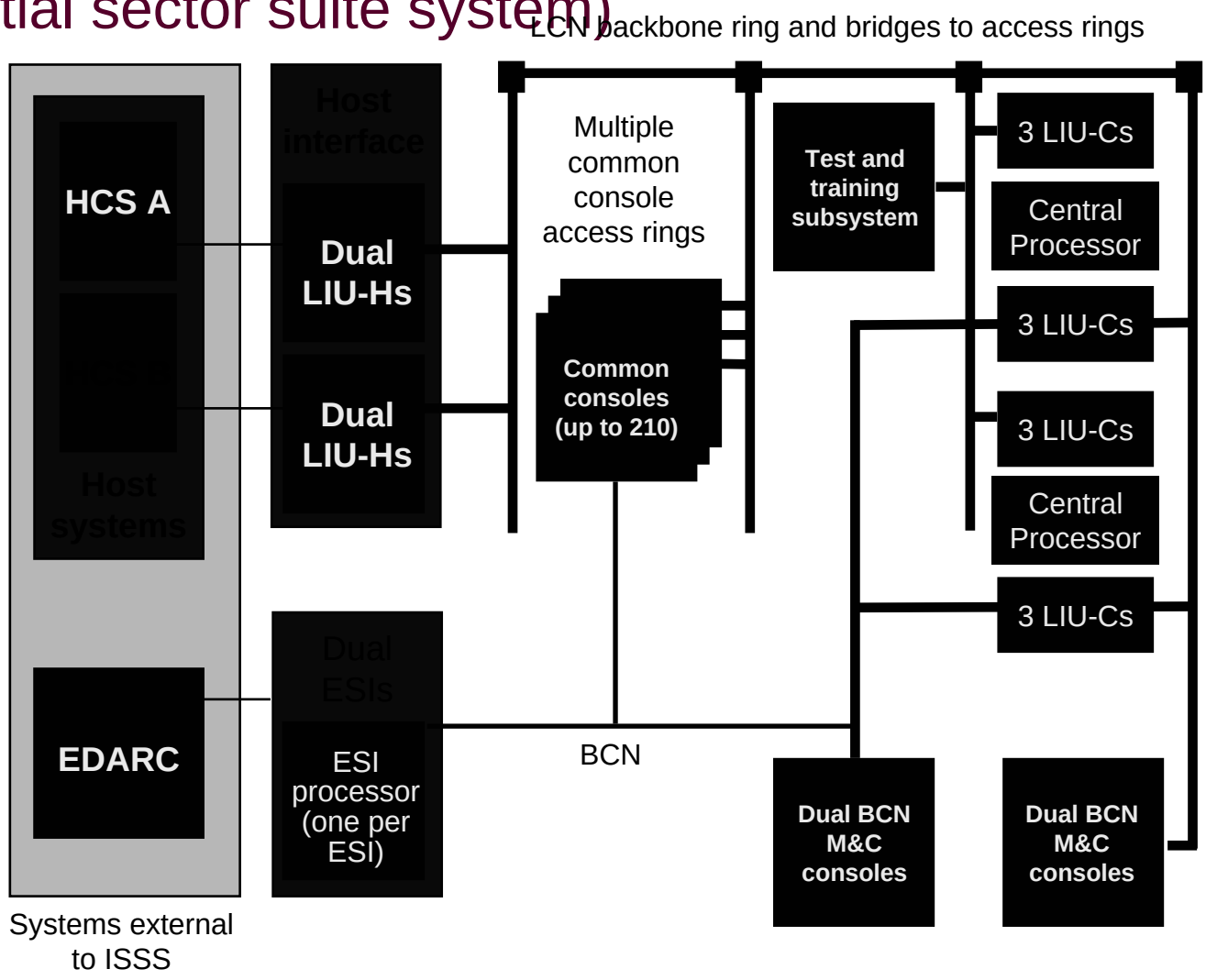
- 哪个试图是我们关心的？依赖于：
  - 谁是风险承担者？
  - 他们将如何使用该文档？

- 体系结构文档三个主要的用途：
  - 指导 – 向人们介绍项目的情况，并指导开发
  - 交流 – 在风险承担者之间
  - 分析 – 保证质量属性



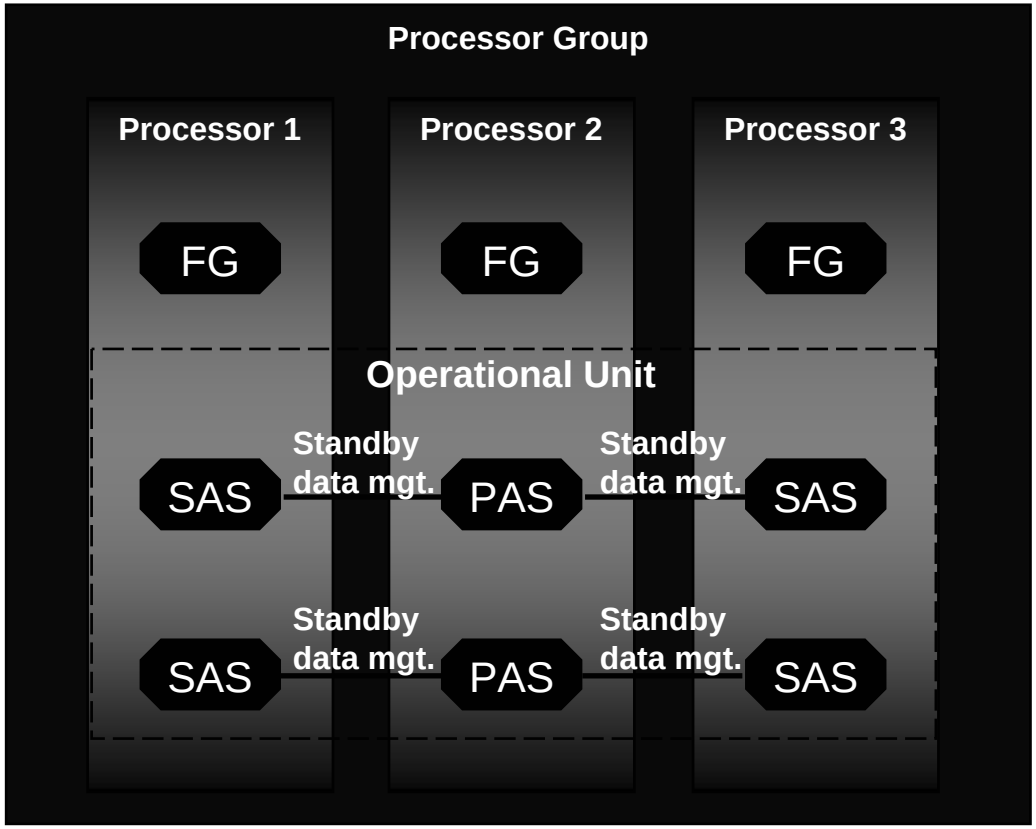
ISSS System Architecture—1

- 美国联邦航空公司空中交通管制系统中初始区段组系统(initial sector suite system)



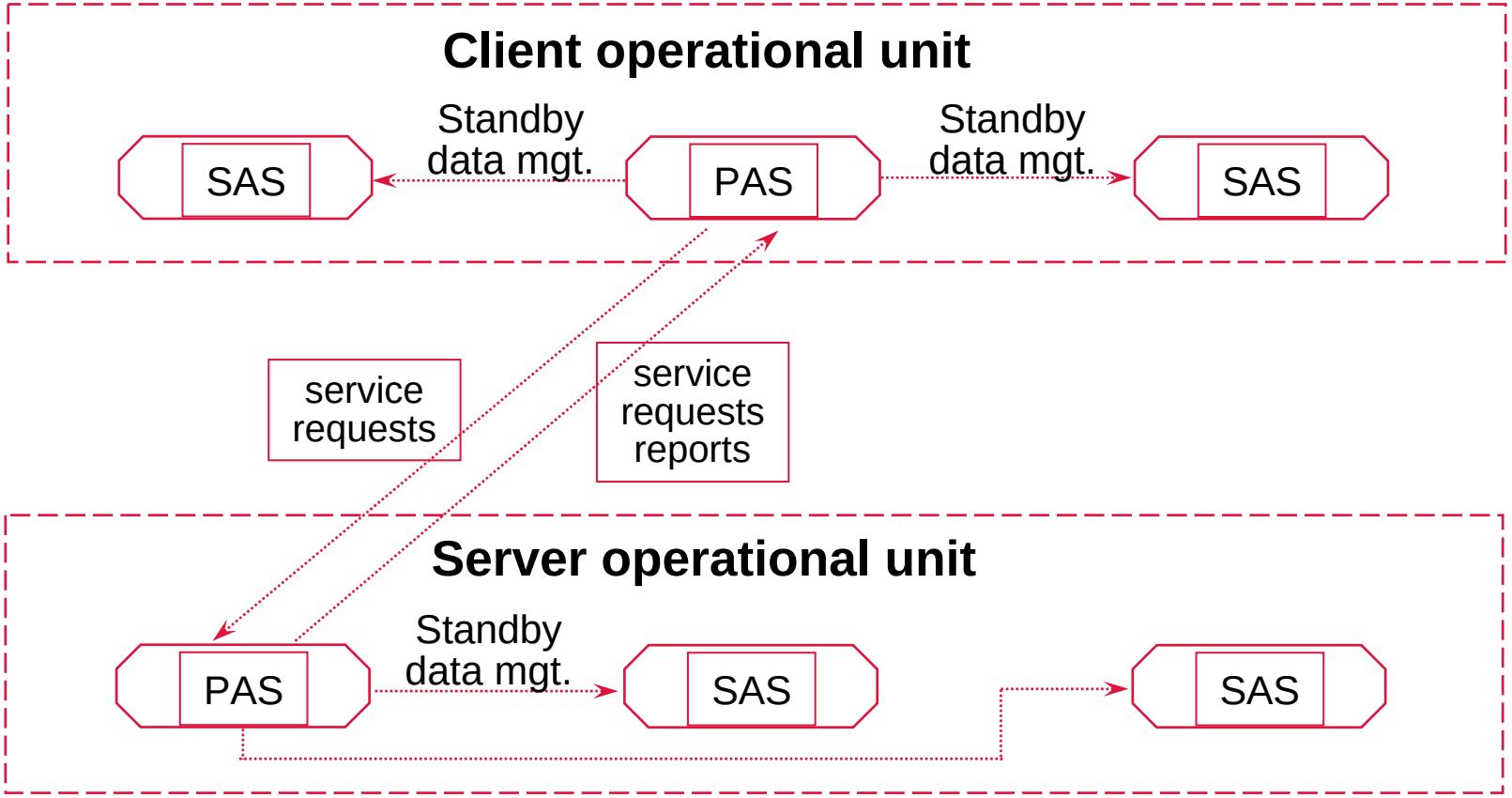
ISSS System Architecture—2

- Fault-Tolerant Software view



ISSS System Architecture—3

- Client-Server



ISSS System Architecture-4

ISSS achieved its quality goals in a variety of ways.

Goal	How achieved
High availability	hardware redundancy (both processor and network) software redundancy (layered fault detection and recovery)
High performance	distributed multiprocessors, front-end schedulability analysis, network modeling
Openness	interface wrapping, layering
Modifiability	templates, table-driven adaptation data
Ability to field subsets	appropriate separation of concerns
Interoperability	client-server division of functionality message-based communications

Documenting a view -1

- 1. 一个初步的表达
  - 通常是图形化的
  - 也可以是文本的（例如表格）
  - 如果是图形的，应该包括对符号集合的关键解释（或者是解释的引用点）
  - 显示元素和元素之间的关系
  - 显示你希望传达的视图索要表达的信息
  - 进行多次分析
- 2. 元素的细目
  - 解释在初步表达中的元素
  - 列出元素和它们的特性（可以参考相关的风格）
  - 解释关系以及一些例外的情况
  - 元素的接口
- 3. 语境图
  - 显示系统与环境之间的关系.

Documenting a view -2

- 4. 变化性指南
  - 显示支持元素变化的变化性机制，以及说明如何实现，何时实用
- 5. 体系结构的背景
  - 所做的设计决策，受到的约束情况
  - 使用系统所需的环境假设
- 6. 其他信息
  - 与需求的关系
  - 与其他视图的关系

谢谢各位！