中国科学技术大学
University of Science and Technology of China

Software Architecture

SSE USTC Qing Ding
dingqing@ustc.edu.cn
http://staff.ustc.edu.cn/~dingqing

Architectures in Context

Fundamental Understanding

- Architecture is a set of principal design decisions about a software system
- Three fundamental understandings of software architecture
  - Every application has an architecture
  - Every application has at least one architect
  - Architecture is not a phase of development

架构不是一个开发阶段

Wrong View: Architecture as a Phase

将架构视为一个阶段否定了它在软件开发中的基本角色

- Treating architecture as a phase denies its foundational role in software development
- More than “high-level design”
- Architecture is also represented, e.g., by object code, source code, …

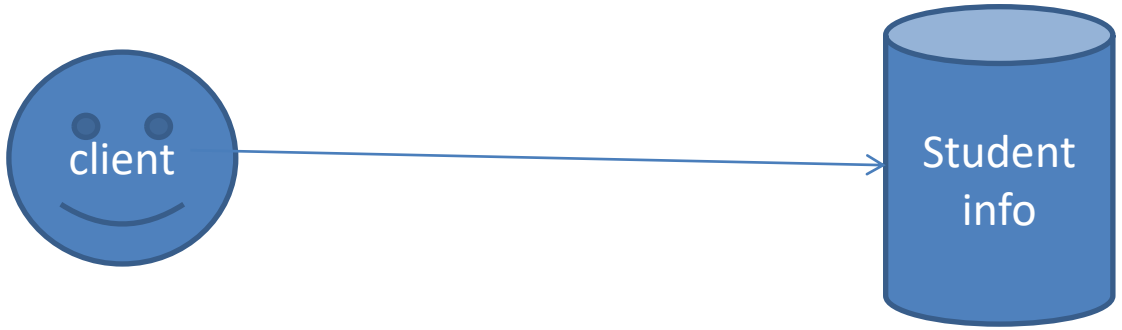
架构也被表现出来

Context of Software Architecture

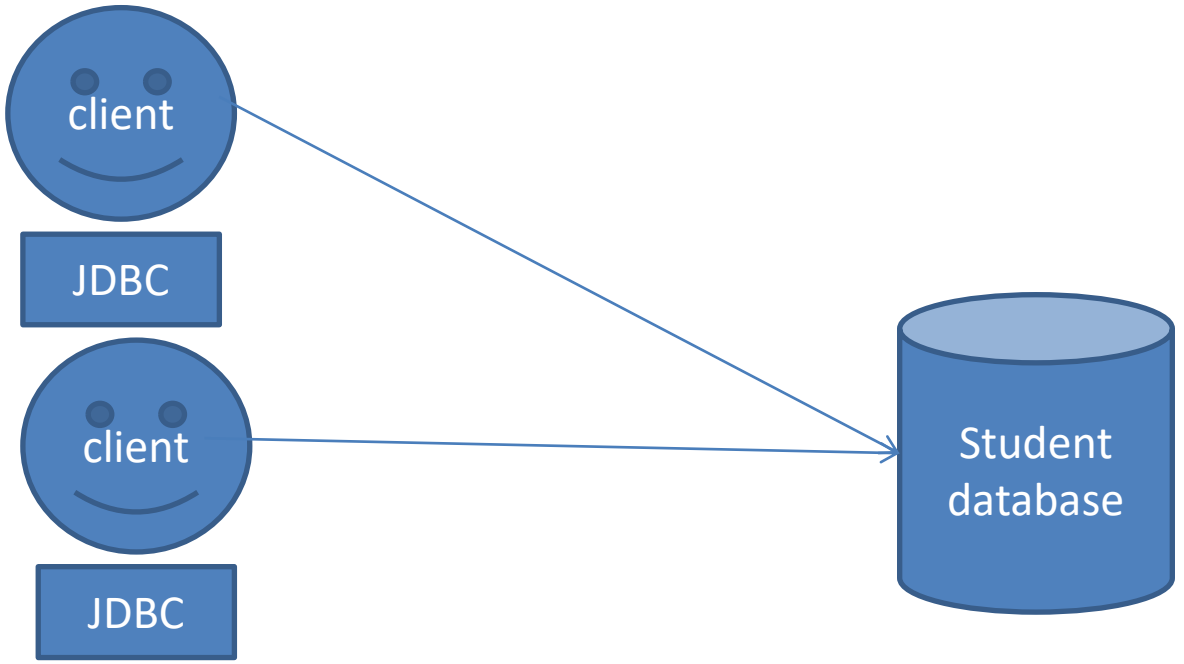
- Requirements
- Design
- Implementation
- Analysis and Testing
- Evolution
- Development Process

Let’s give an example

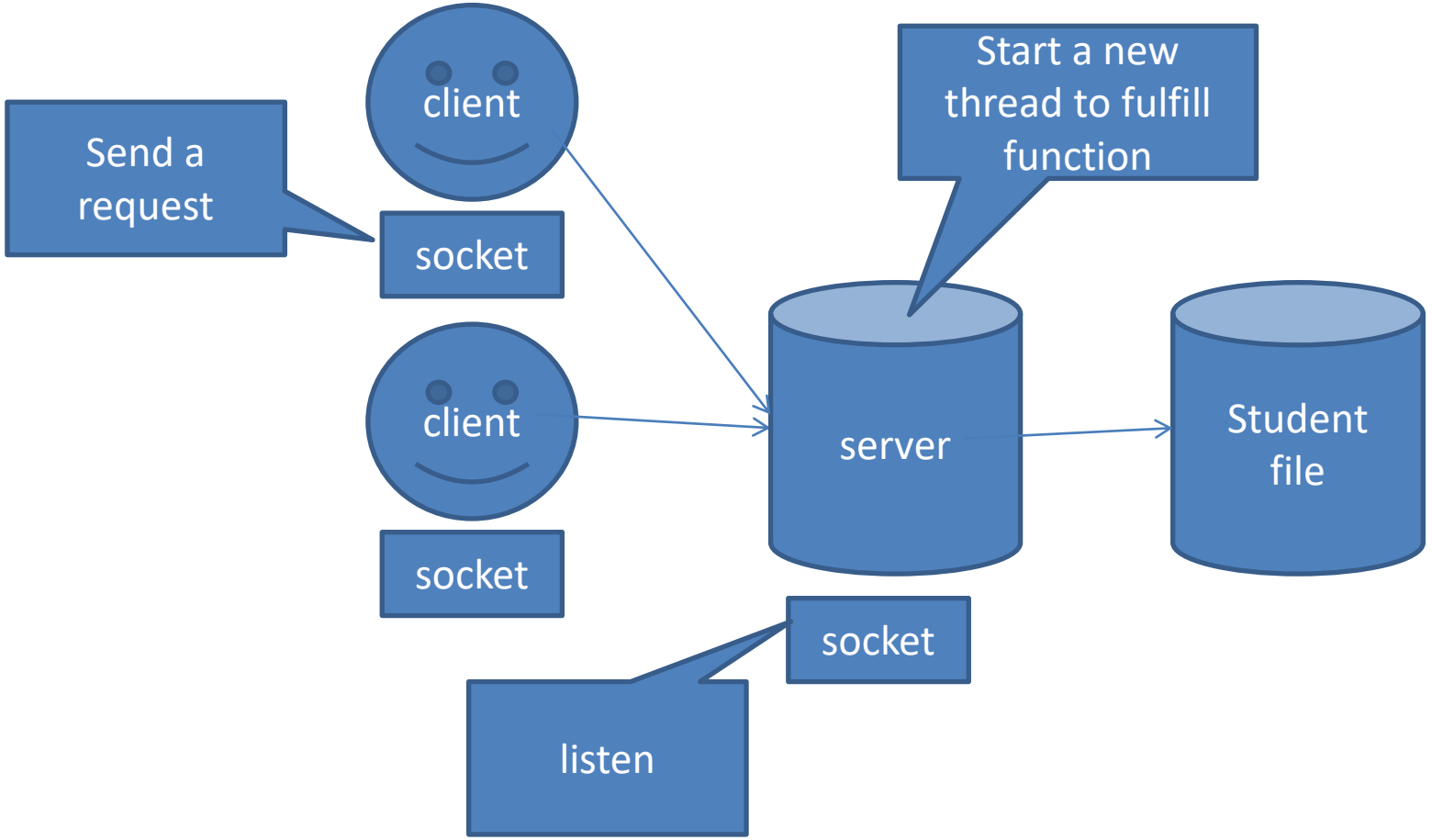
- A student management system
  - Function: query/update/insert/delete student info



You can



You can do


client
Send a request
socket
client
socket
Start a new thread to fulfill function
server
Student file
socket
listen

You can do

中国科学技术大学
University of Science and Technology of China

- Framework?
- Microservice?
- …

Requirements Analysis

传统的SE建议需求分析应该不受任何设计考虑的影响

- Traditional SE suggests requirements analysis should remain unsullied by any consideration for a design
- However, without reference to existing architectures it becomes difficult to assess practicality, schedules, or costs 然而，如果不参考现有的架构，就很难评估实用性、安排或成本
  - In building architecture we talk about specific rooms…
  - …rather than the abstract concept “means for providing shelter” -在建筑学中，我们谈论特定的房间……
    -而不是“提供庇护的手段”这个抽象概念
- In engineering new products come from the observation of existing solution and their limitations

在工程中，新产品来自于对现有解决方案及其局限性的观察

New Perspective on Requirements Analysis

现有的设计和架构提供了解决方案词汇表

- Existing designs and architectures provide the solution vocabulary
- Our understanding of what works now, and how it works, affects our wants and perceived needs 我们对现在什么起作用，以及它如何起作用的理解，影响着我们的需求和感知需求

从我们现有系统的经验中得出的见解

- The insights from our experiences with existing systems
  - 帮助我们想象什么是可行的 helps us imagine what might work and
  - 使我们能够评估开发时间和成本 enables us to assess development time and costs
- → Requirements analysis and consideration of design must be pursued at the same time 需求分析和设计考虑必须同时进行

Non-Functional Properties (NFP)

主要的任务

- NFPs are the result of architectural choices
  NFPs是架构选择的结果
- NFP questions are raised as the result of architectural choices
  NFP问题是架构选择的结果
- Specification of NFP might require an architectural framework to even enable their statement
  NFP的规范可能需要一个架构框架来支持它们的声明
- An architectural framework will be required for assessment of whether the properties are achievable
  需要一个架构框架来评估需求是否可实现

Software Architecture Quality Metrics

- Availability
- Reliability
- Modifiability
- Performance
- Security
- Testability
- Usability
- Supportability

Software Architecture Quality Metrics

商业目标
- Business objectives(T2M,Targeted Market)

开发团队和当地市场团队的技能
- Skill of development team and local market of team

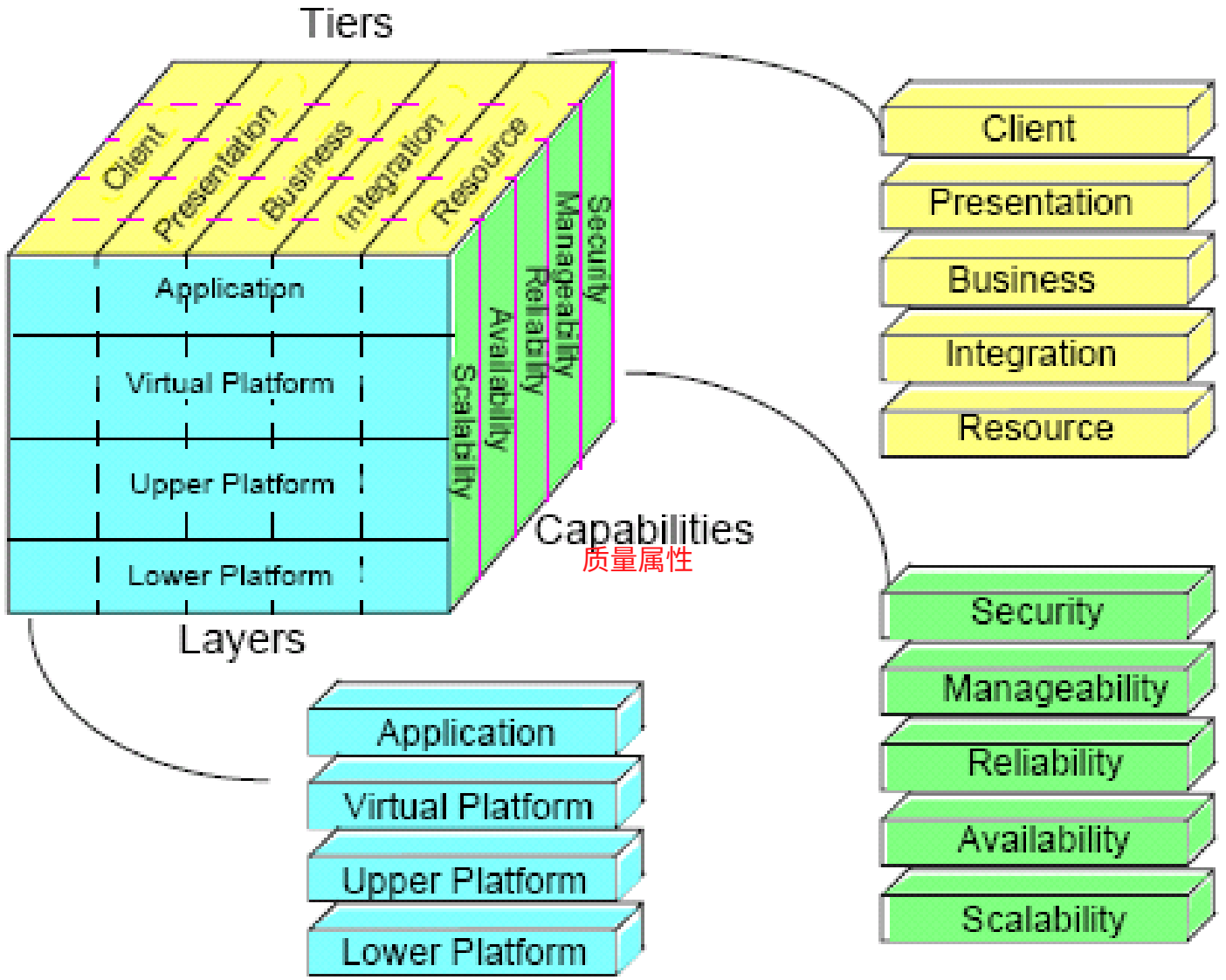
建设和维护成本 vs 效益
- Cost to build and Maintain v.s. Benefit

实质性的技术
- Materiality of Technology

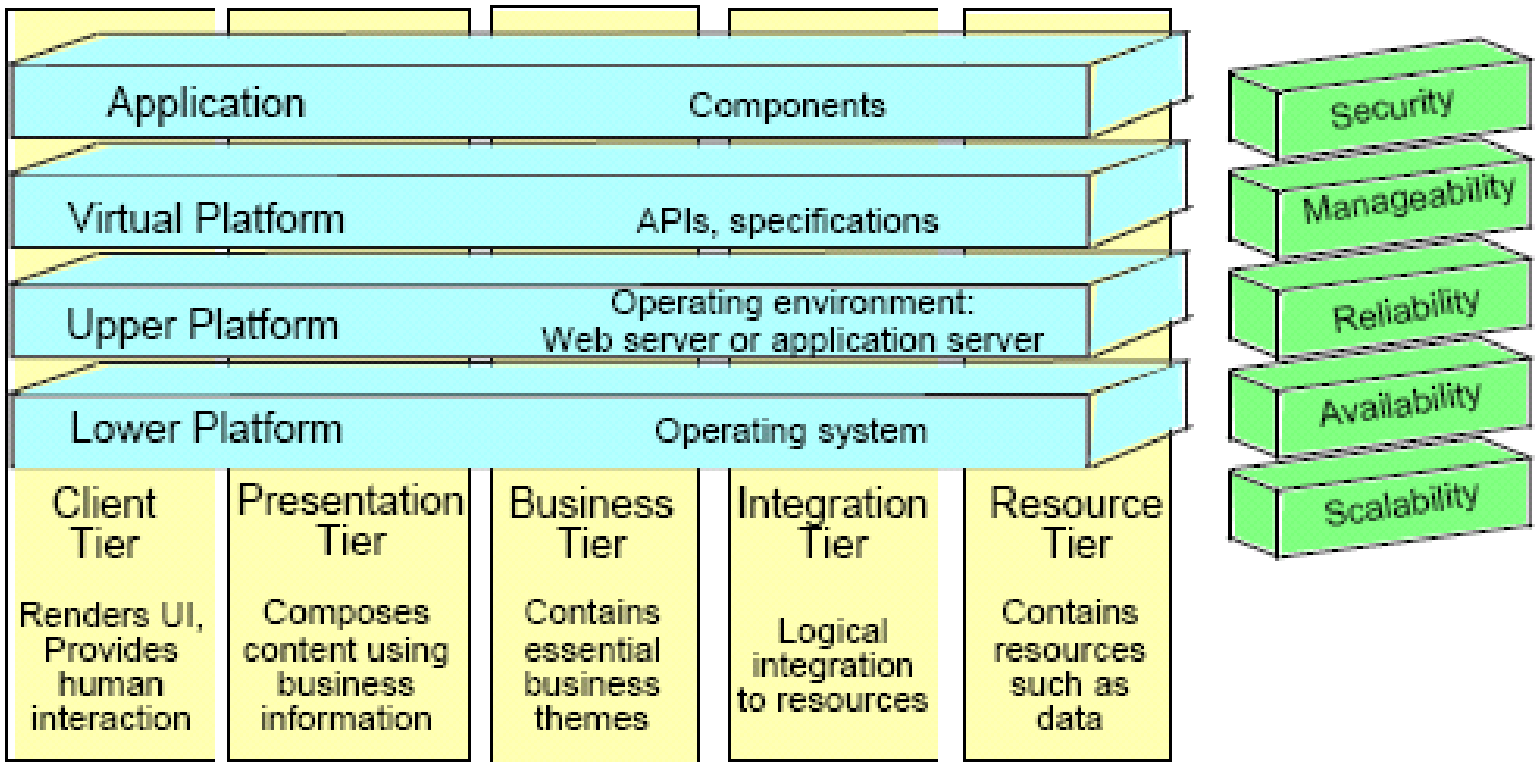
当前系统的约束和完整性
- Current System constrain, integration

迁移,迁移,迁移
- Migration, migration, migration

Architecture and the Cube



Layers and Tiers



Parnas's Law: Every *class*, *object*, *tier*, or *layer* keeps a secret — the secret of how services are presented to the adjacent *class*, *object*, *tier*, or *layer*.

Capabilities

- Nonfunctional, observable system qualities:
  非功能的、可观察的系统质量
  - Do not represent separate specified functions
    不代表单独指定的功能
  - Cannot be satisfied by any one component
    不能被任何一个组成部分满足
- Example:
  - Retail toy store's web site cannot handle the load and crashes:
  - Two weeks before Christmas
  - After a $50 million advertisement campaign
  - Expansion of capacity will take six weeks

Capabilities

System Quality	Definition As Used In This Course
Availability 可用性	The assurance that a service/resource is always accessible. This aspect of a system is often coupled with measures of its performance. 与性能测试有关，满足定义的指标才算可用
Capacity 容量	The ability to run a number of jobs per unit time. 并发能力
Extensibility	The ability to economically modify or add functionality. 关注点：非常经济地修改和增加功能
Flexibility 灵活性	The ability to support architectural and hardware configuration changes. 反映架构的灵活性，可升级性
Manageability 管理性，可维护性	The ability to manage the system to ensure the continued health of a system with respect to the other capabilities. 管理系统以确保系统相对于其他系统质量的持续健康运行的能力
Performance性能	The ability to execute functions fast enough to meet performance goals. 快速执行功能以满足性能目标的能力
Reliability 可靠性	The ability to ensure the integrity and consistency of an application and its transactions. 确保完整性和一致性
Scalability	The ability to support the required quality of service as load increases. 当系统负载增加时仍旧能提供较好的服务质量
Security 安全性	The ability to ensure that information is neither modified nor disclosed except in accordance with the security policy. 确保信息不被修改或披露的能力，除非按照安全策略
Testability 测试性	The ability to determine what the expected results should be. 确定预期结果的能力

Availability and Reliability

Availability 指标是时间 关键性应用追求可用性 采用冗余的方法提高可用性

- The assurance that a service or a resource is highly accessible. (time available) / (time possible) 保证服务或资源是高可访问的
- Example – 201 seconds down/week

Reliability 指标是概率

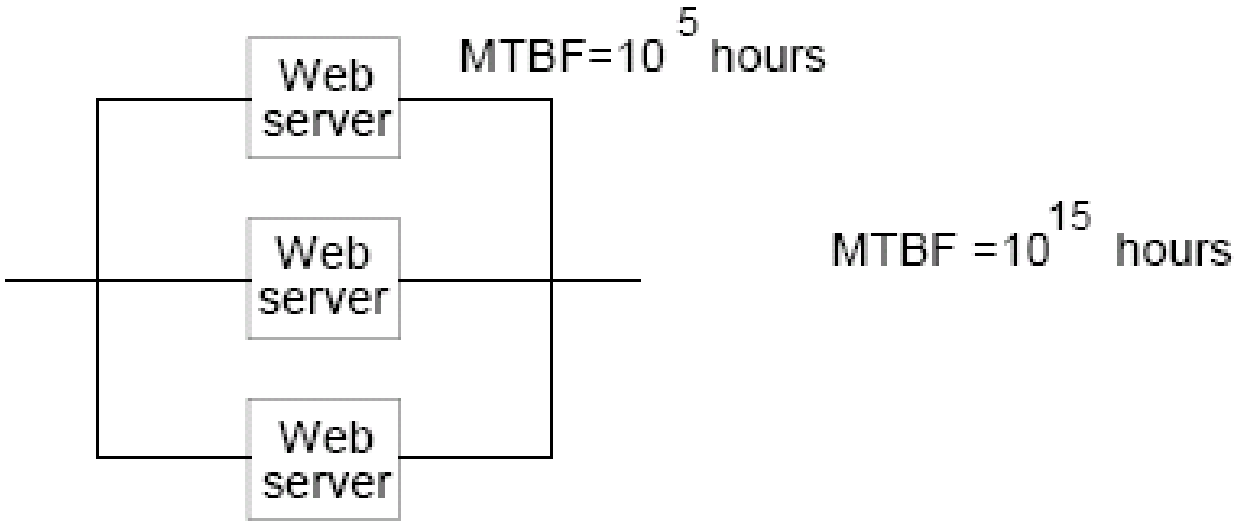
- The ability to ensure the integrity and consistency of the application and all of its transactions. 确保应用程序及其所有事务的完整性和一致性的能力
- Example – Users will experience failed sessions no more than 1 time in 1000 attempts (99.9 percent reliability).

Improving Reliability

MTBF:mean time between failures平均故障间隔

假设web服务器的MTBF为100,000小时

- Assume a web server has a MTBF of 100,000 hours.
- What is the reliability of the following?



Availability increases.

Manageability

采用冗余的方法会降低管理性

管理系统以确保系统相对于其他系统质量的持续健康运行的能力

- The ability to manage the system to ensure the continued health of a system with respect to the other system qualities.

每月执行正常升级的工作小时数

- Metric example – Number of staff hours per month to perform normal upgrades.

Flexibility

快速迭代升级的能力

- Is the ability to change architecture to meet new requirements in a cost-efficient manner

以经济有效的方式改变架构以满足新需求的能力

一个可用的、可靠的、可伸缩的应用程序的关键

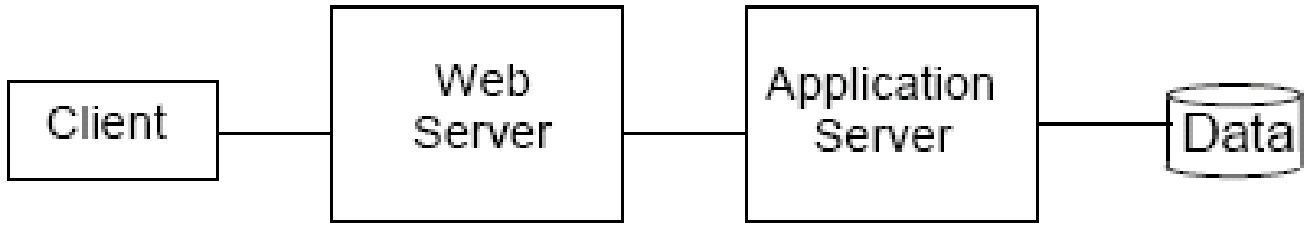
- Is the key to an available, reliable, and scalable application

可以通过应用程序代码的位置独立性进行改进

- Can be improved through location independence of application code

Flexibility

- Example



允许您更改表示语言所影响的内容

- Allows you to change what is affected by changing the presentation language (for example, English to German)?
- What must change to add a “fat client” for intense use?

要增加一个“胖客户端”进行密集使用，必须做哪些改变?

Flexibility Metrics

没有衡量灵活性的标准方法

- No standard way of measuring flexibility

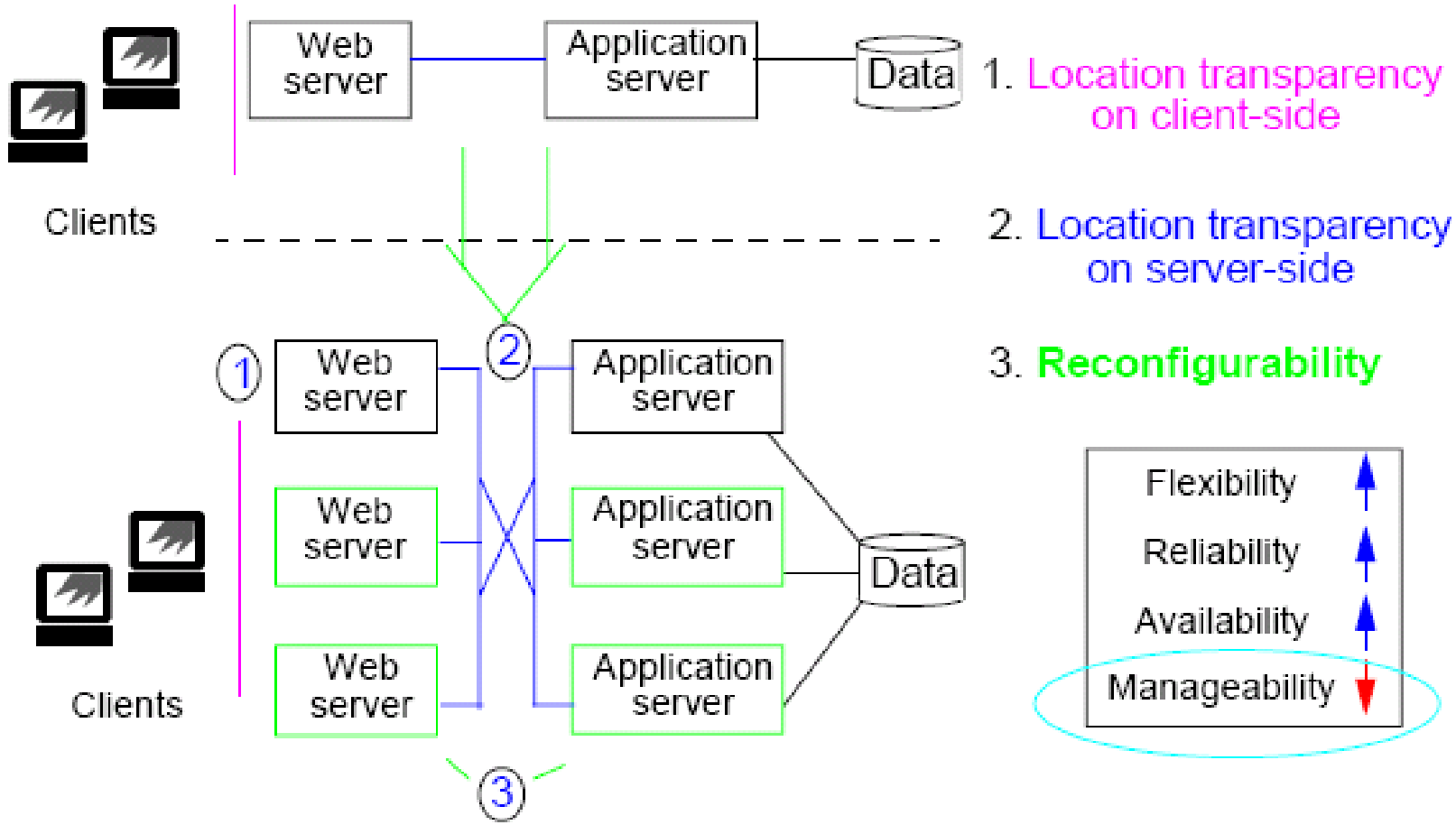
基本的衡量标准是变化的成本，但这取决于什么变化是可能的

- Basic measure is cost of change but this depends on what change is probable.

使用变更案例来评估系统中必须变更的内容

- Use change cases to evaluate what in system must change.

The Effect of Flexibility



Performance

个体：E2E（端到端）延迟
总体：吞吐量

快速执行功能以满足性能目标的能力

- The ability to execute functions fast enough to meet performance goals.

响应时间对应用程序很重要

- Response time is important to an application.

识别和控制昂贵的调用

- Identify and control expensive calls.

在实现之前声明性能目标

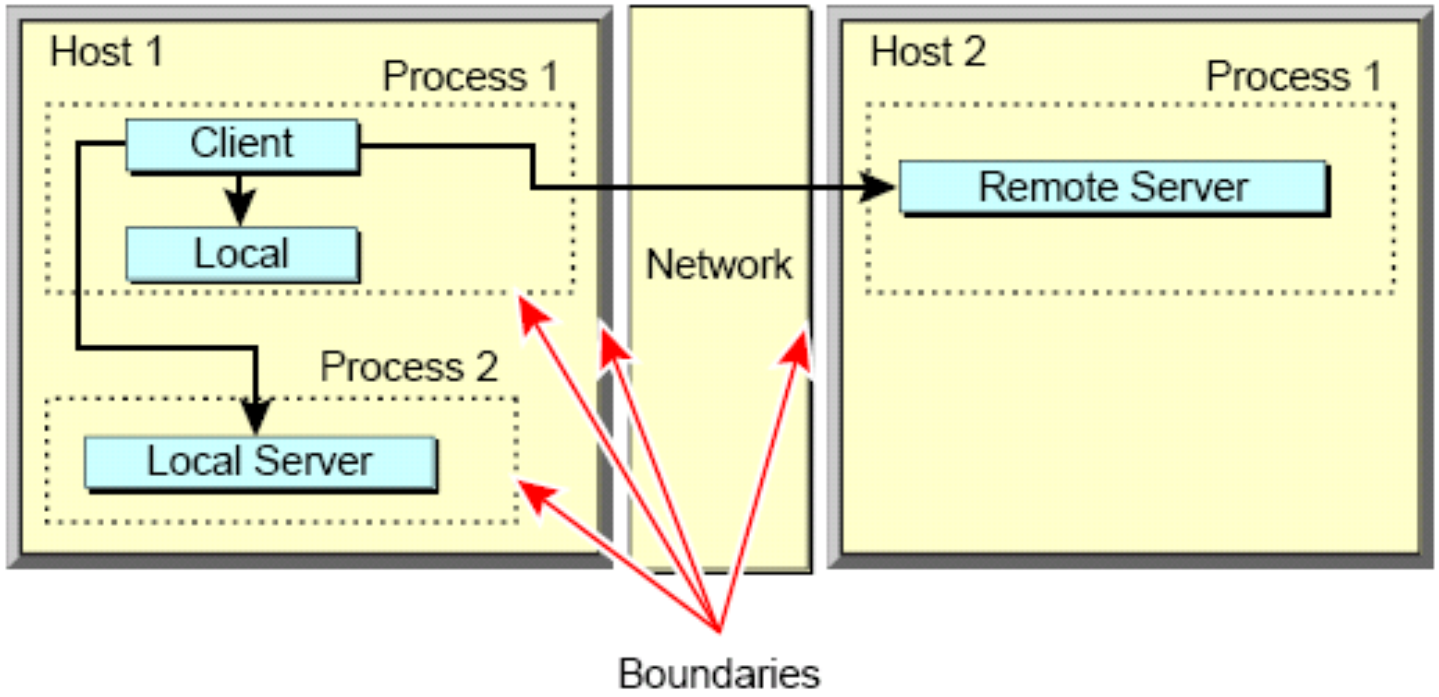
- State performance goals *before* implementing.
- Example—first visible response in browser under maximum specified load occurs in less than 3 seconds, 95 percent of the time. Measurement is made at company external firewall.

在95%的情况下，浏览器在最大指定负载下的第一次可见响应发生在3秒以内。
在公司外部防火墙进行测量

Performance

识别和控制访问昂贵的进程和网络边界

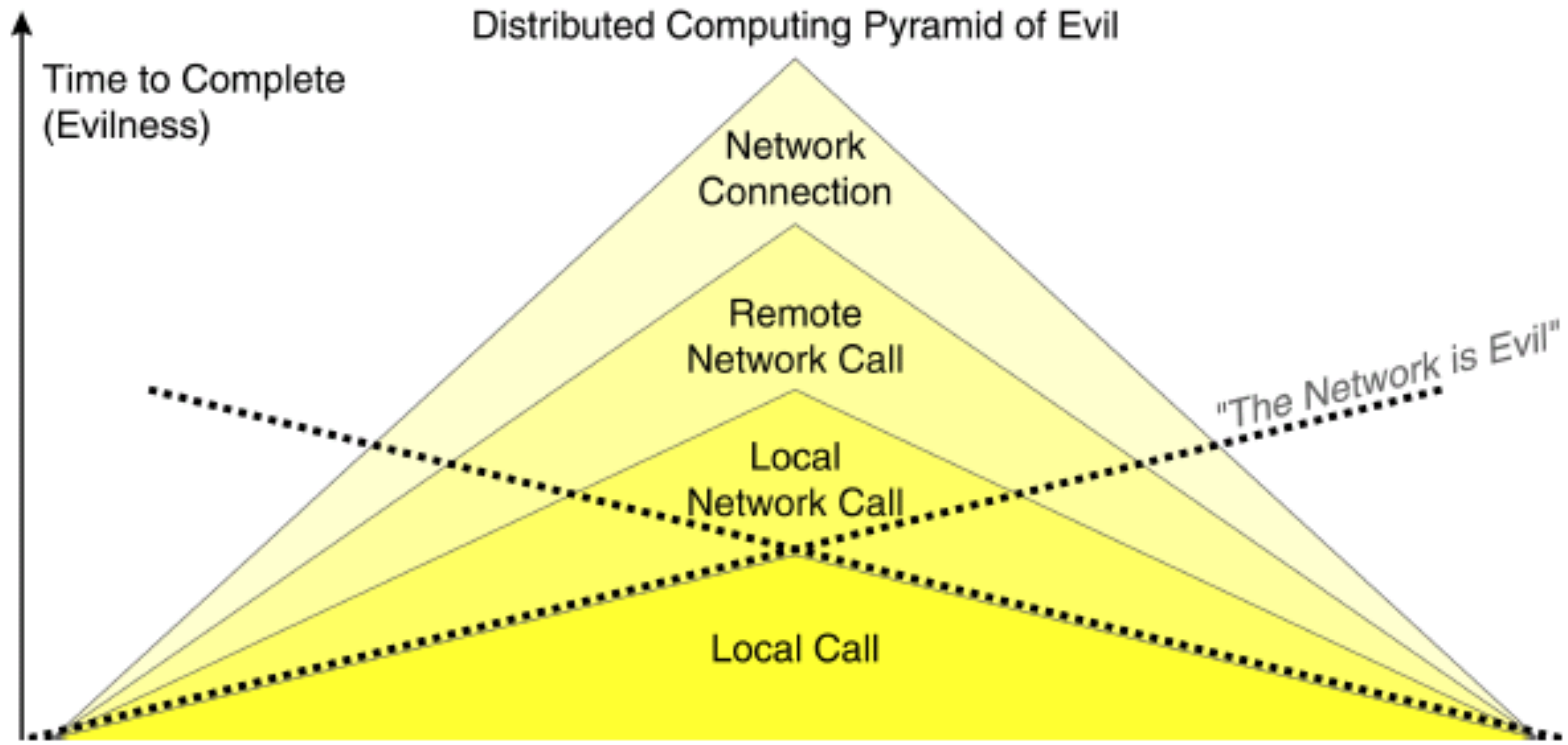
- Identify and control access to expensive process and network boundaries.



Performance

确定昂贵调用的类别

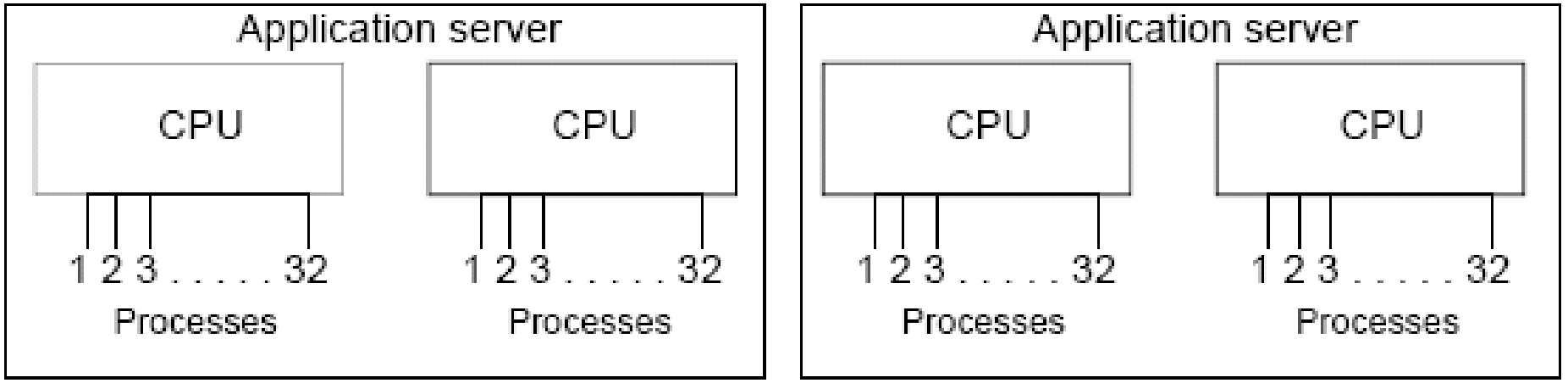
- Identify the class of expensive calls.



Capacity

并发量：单元时间内运行一定数量的工作

- The ability to run a certain number of jobs per unit time. 在单位时间内运行一定数量的作业的能力
- Example



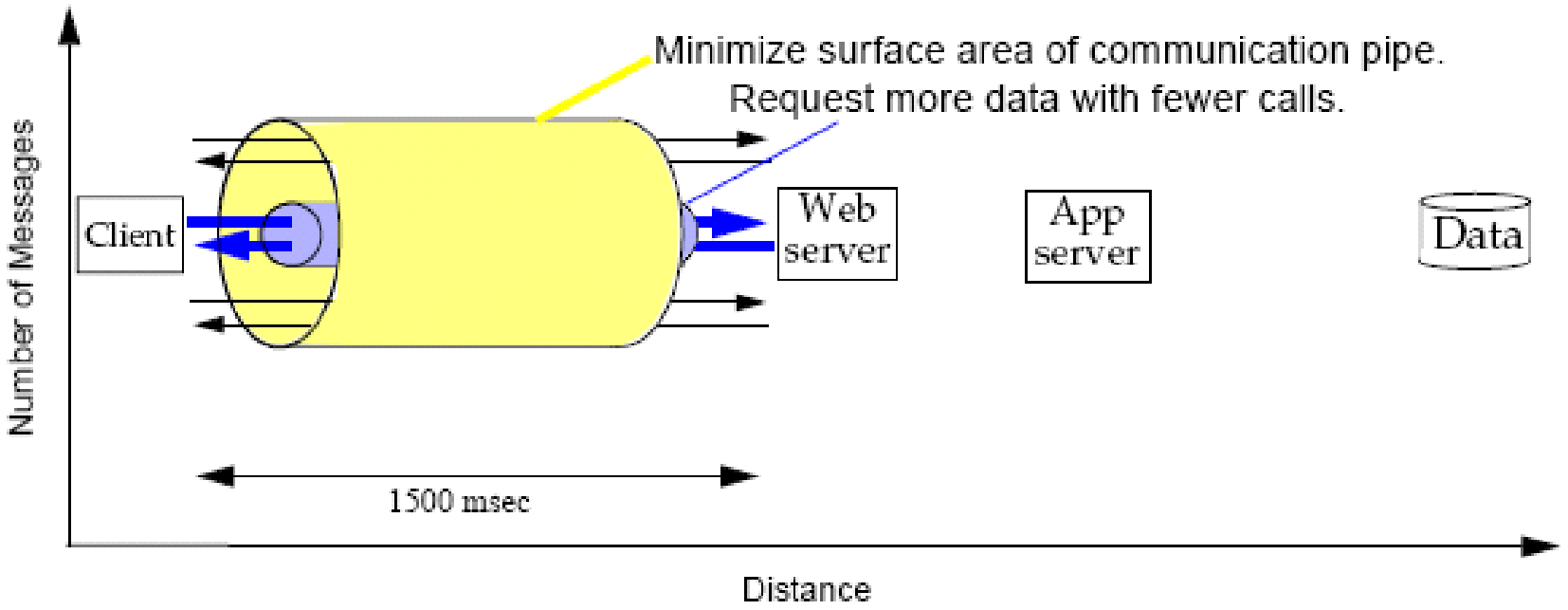
Total is 28*4=112 (total not including overhead)

Latency Space

时延空间

延迟是固有的通信延迟，如ping时间

- Latency is the inherent communication delay, such as ping time.



Scalability

当负载增加时经济地支持所需服务质量的能力

- The ability to economically support the required quality of service as the load increases.

垂直可伸缩性来自于向现有服务器添加容量(内存、cpu)

- *Vertical scalability* comes from adding capacity (memory, CPUs) to existing servers.

对架构的要求更少

  - Makes fewer demands on the architecture

受资源限制

  - Is constrained by resource limits

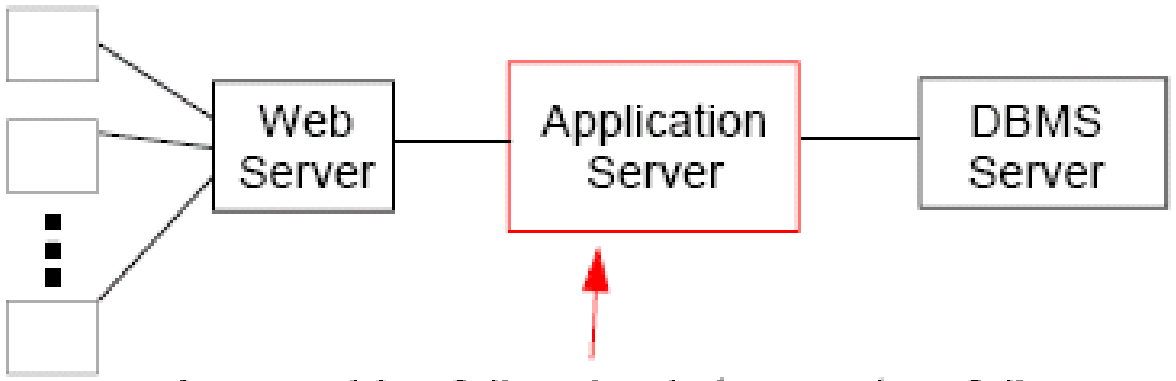
水平可伸缩性来自于添加服务器

- *Horizontal scalability* comes from adding servers.

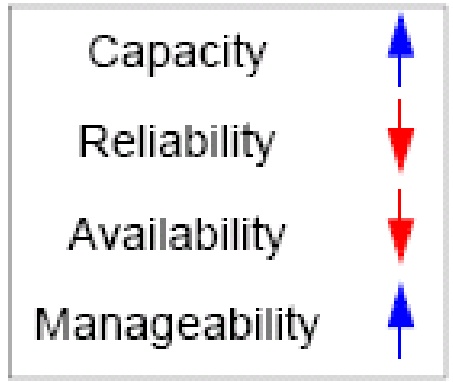
分布式状态，负载均衡

  - Distributed state, load balancing

Vertical Scalability

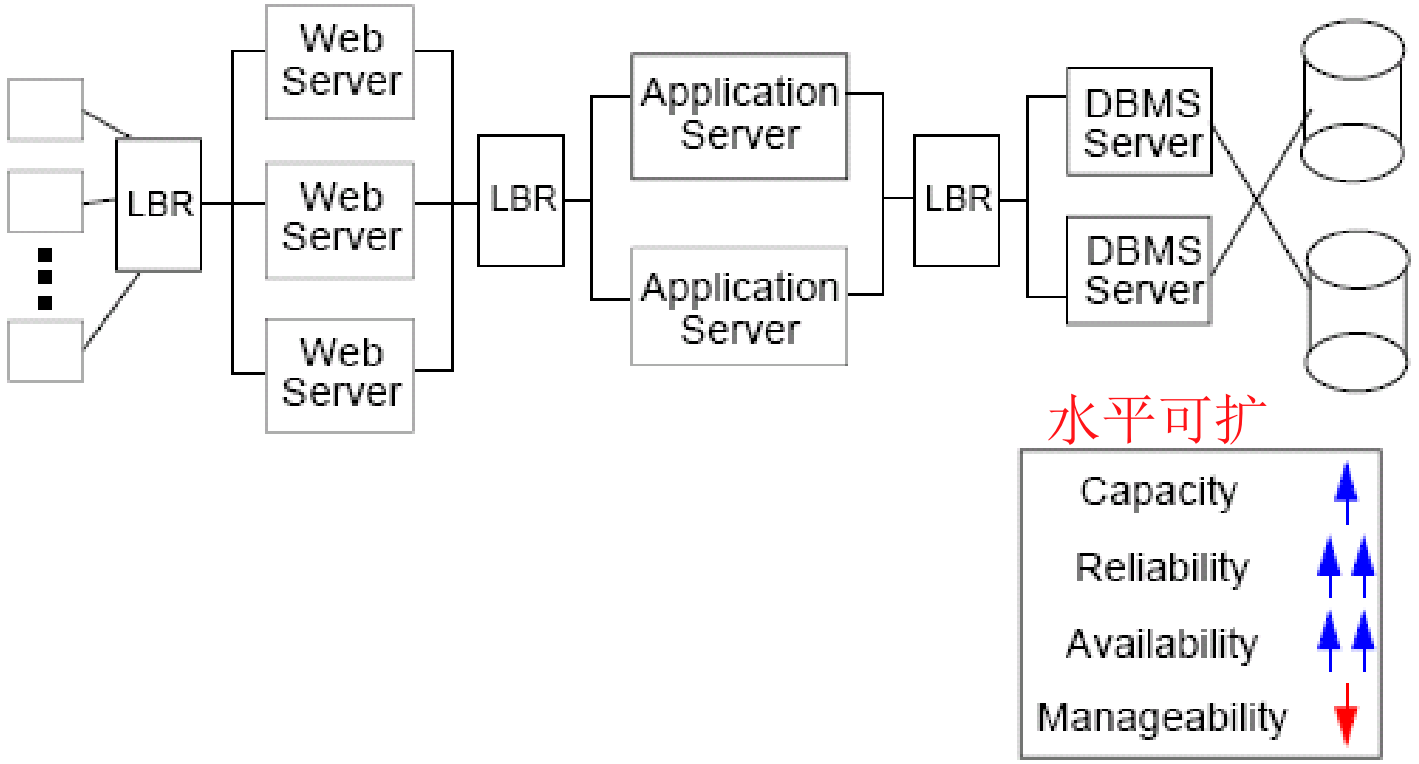


垂直可扩



任何机器故障都会导致系统故障

Horizontal Scalability



- All Web servers or all application servers must fail for a system failure to occur.
  若发生系统故障，则所有Web服务器或所有应用程序服务器都必须故障

The Effect of Scalability

How are other capabilities driven by scalability?

可用性和可靠性是通过可伸缩性获得的

- Availability and reliability are obtained through scalability.
- Capacity is affected by scalability:
  - One machine handles 500 transactions

  or

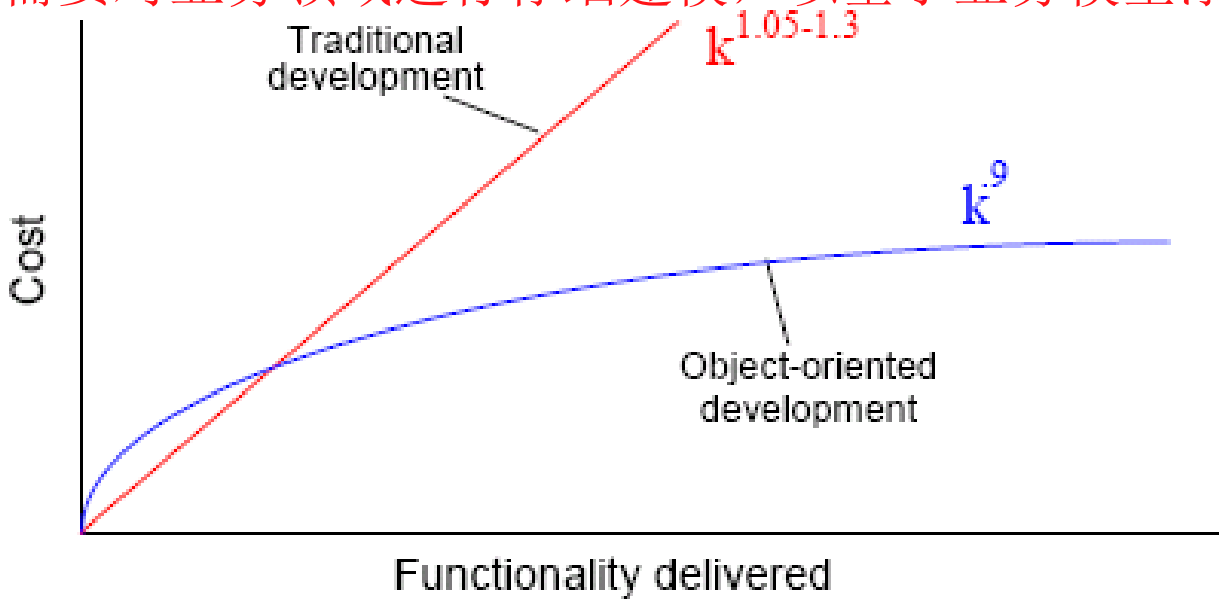
  - 5 machines handle 100 transactions each
  - 50 machines handle 10 transactions each

Extensibility

能够在不影响现有功能的情况下修改或添加功能

- The ability to modify or add functionality without impacting the existing functionality.
- Requires careful modeling of the business domain to add new features based on the business model.

需要对业务领域进行仔细建模，以基于业务模型添加新特性



Extensibility

粗略的指导方针

Rough guidelines:

超过25个顶层类将会导致问题

- More than 25 top-level classes will lead to problems.

每个用例都可以使用域模型方法直接实现

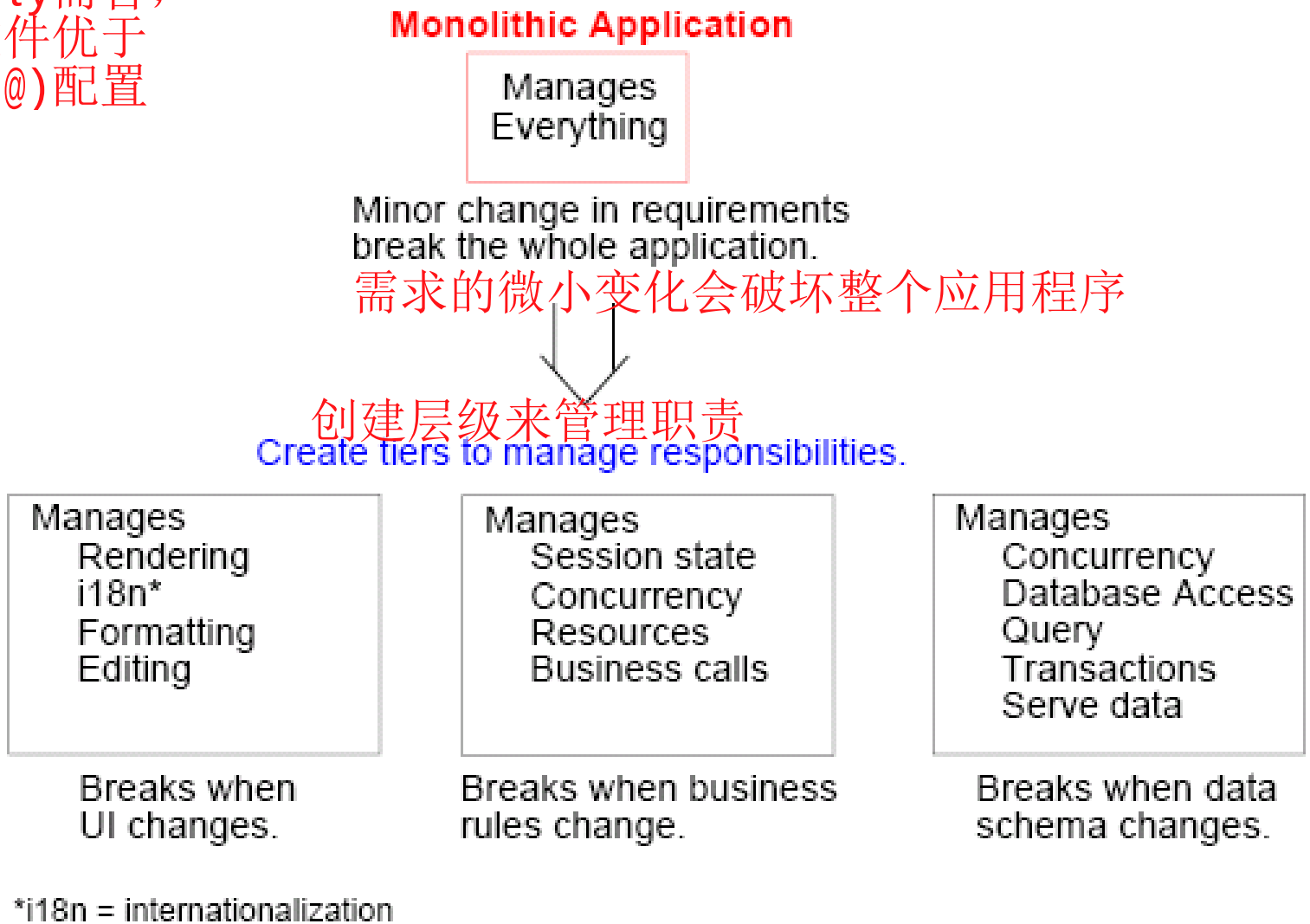
- Every use case can be directly implemented using domain model methods.

平均而言，每次修改大约涉及1.3个类

- On average, every change touches about 1.3 classes.

Extensibility

对于Extensibility而言，
采用xml等配置文件优于
使用annotation(@)配置



Testability

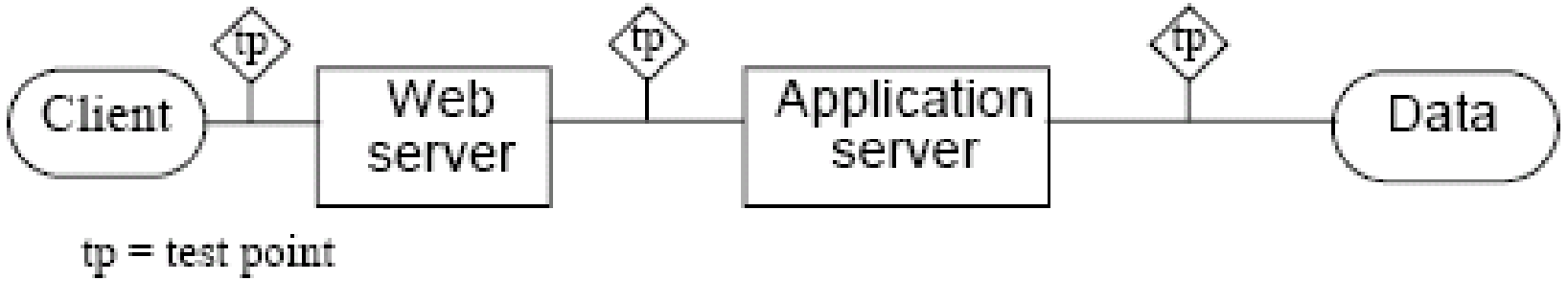
同步通讯可测试性强
异步通讯可测试性差

确定预期结果的能力

- The ability to determine what the expected results should be.

多层体系结构为中间测试和调试提供了许多测试点

- Multi-tier architecture provides for many test points for intermediate testing and debugging.



POJO based programming 纯粹的java对象，可测试较强
component based 基于容器的（component本身不能运行）可测试性较差（测试时需要启动容器，而容器启动较慢）

Security

Principle	Definition
Identity 身份	Trust that the user is correctly identified through an authentication mechanism
Authority权限	Trust that the user can perform only allowed actions
Integrity 完整	Trust that data can be modified only in allowable ways
Privacy 隐私	Trust that data is disclosed to authorized entitites only in authorized ways
Auditability 可审核性	Trust that the system maintains valid records of actions taken for later analysis

编程实现安全
声明式安全（由容器管理安全）（生命周期由容器管理）

Security

正确使用技术

- Proper use of technology.
- More than just technology.
  - Hiring a trusted employee
  - Procedural – intrusion detection
  - Keep server in locked room

不仅仅是技术。
-雇佣值得信赖的员工
-程序-入侵检测
-把服务器锁在房间里

隐私、易用性和费用之间的权衡

- Trade-off between privacy, ease-of-use, and expense

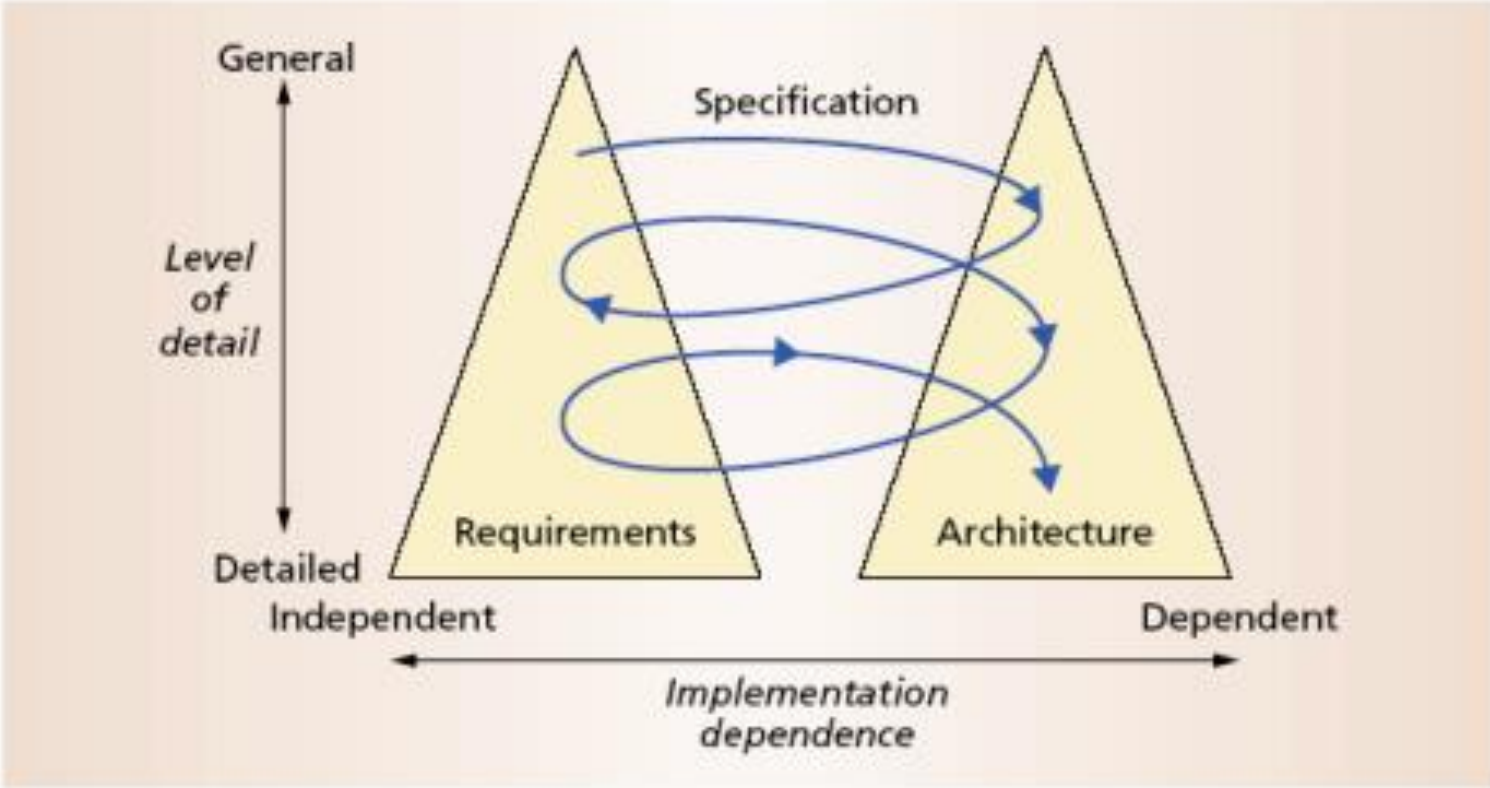
Security Policy

- A security policy should state the acceptable risks and costs.

  It requires at least 30 hours using the best known attacks to compromise one customer account. Three or more successive login failures will be logged.

安全策略应该说明可接受的风险和成本。使用最著名的攻击至少需要30个小时才能攻破一个客户账户。三次或更多连续的登录失败将被记录

The Twin Peaks Model



Design and Architecture

- Design is an activity that pervades software development
- It is an activity that creates part of a system's architecture
- Typically in the traditional Design Phase decisions concern
  - A system's structure
  - Identification of its primary components
  - Their interconnections
- Architecture denotes the set of principal design decisions about a system
  - That is more than just structure

· 设计是软件开发中普遍存在的一种活动
· 它是一种创建部分系统架构的活动
· 通常在传统设计阶段的决策关注；
-系统的结构
-主要部件的识别
-他们的联系
· 架构表示关于系统的一组主要设计决策
-这不仅仅是结构问题

Architecture-Centric Design

传统的设计阶段建议将需求转换成算法，这样程序员就可以实现它们

- Traditional design phase suggests translating the requirements into algorithms, so a programmer can implement them
- Architecture-centric design 以构架为中心的设计
  - stakeholder issues 甲方的问题
  - decision about use of COTS component COTS组件的使用决策
  - overarching style and structure 总体风格和结构
  - package and primary class structure 包和主类结构
  - deployment issues 部署问题
  - post implementation/deployment issues 发布实现/部署问题

Design Techniques

- Basic conceptual tools
  - Separation of concerns
  - Abstraction
  - Modularity

基本的概念性工具
--分离关注事项（SOC：每个文件只做一件事，代码更加纯粹）
--抽象
--模块化

- Two illustrative widely adapted strategies
  - Object-oriented design
  - Domain-specific software architectures (DSSA)

两个说明性的广泛适用的策略
--面向对象设计
--特定领域的软件架构(DSSA)(非常关注可重用性)

Object-Oriented Design (OOD)

- Objects
  - Main abstraction entity in OOD
  - Encapsulations of state with functions for accessing and manipulating that state

-OOD中的主要抽象实体
-用访问和操纵状态的函数封装状态

Pros and Cons of OOD

- Pros
  - UML modeling notation -UML建模符号
  - Design patterns -设计模式
- Cons
  - Provides only -只提供
    - One level of encapsulation (the object) · 一个级别的封装(对象)
    - One notion of interface · 接口的一种概念
    - One type of explicit connector (procedure call) · 一种显式连接器(过程调用)
      - Even message passing is realized via procedure calls -甚至消息传递也是通过过程调用实现的
  - OO programming language might dictate important design decisions -面向对象编程语言可能决定重要的设计决策
  - OOD assumes a shared address space -OOD假设共享地址空间

DSSA(特定领域软件架构)

- Capturing and characterizing the best solutions and best practices from past projects within a domain
  从过去的项目中获取并描述最佳解决方案和最佳实践
- Production of new applications can focus on the points of novel variation
  新应用的生产可以集中在新颖的变化点上
- Reuse applicable parts of the architecture and implementation
  重用架构和实现的可应用部分
- Applicable for product lines
  适用于产品线
  - →Recall the Philips Koala example discussed in the previous lecture
    回想一下上节课讨论的飞利浦考拉的例子

Implementation

目标是创建机器可执行的源代码

- The objective is to create machine-executable source code
  - That code should be faithful to the architecture
    - Alternatively, it may adapt the architecture
    - How much adaptation is allowed?
    - Architecturally-relevant vs. -unimportant adaptations
  - It must fully develop all outstanding details of the application

-代码应该忠实于架构
  •或者，它可以适应架构
  •允许多大程度的适应?
  •与架构相关vs.不重要的调整
-必须全面开发应用的所有未完成细节

Faithful Implementation

在架构中找到的所有结构元素都在源代码中实现

- All of the structural elements found in the architecture are implemented in the source code

源代码不能使用体系结构中没有相应元素的主要新计算元素

- Source code must not utilize major new computational elements that have no corresponding elements in the architecture

源代码不能包含在体系结构中不能找到的、体系结构元素之间的新连接

- Source code must not contain new connections between architectural elements that are not found in the architecture
- Is this realistic?
  Overly constraining?
  What if we deviate from this?

Unfaithful Implementation

- The implementation does have an architecture
  - It is latent, as opposed to what is documented.
- Failure to recognize the distinction between planned and implemented architecture
  - robs one of the ability to reason about the application's architecture in the future
  - misleads all stakeholders regarding what they believe they have as opposed to what they really have
  - makes any development or evolution strategy that is based on the documented (but inaccurate) architecture doomed to failure

实现确实有一个架构
-它是潜在的，而不是记录下来的

未能识别计划的和实现的架构之间的区别

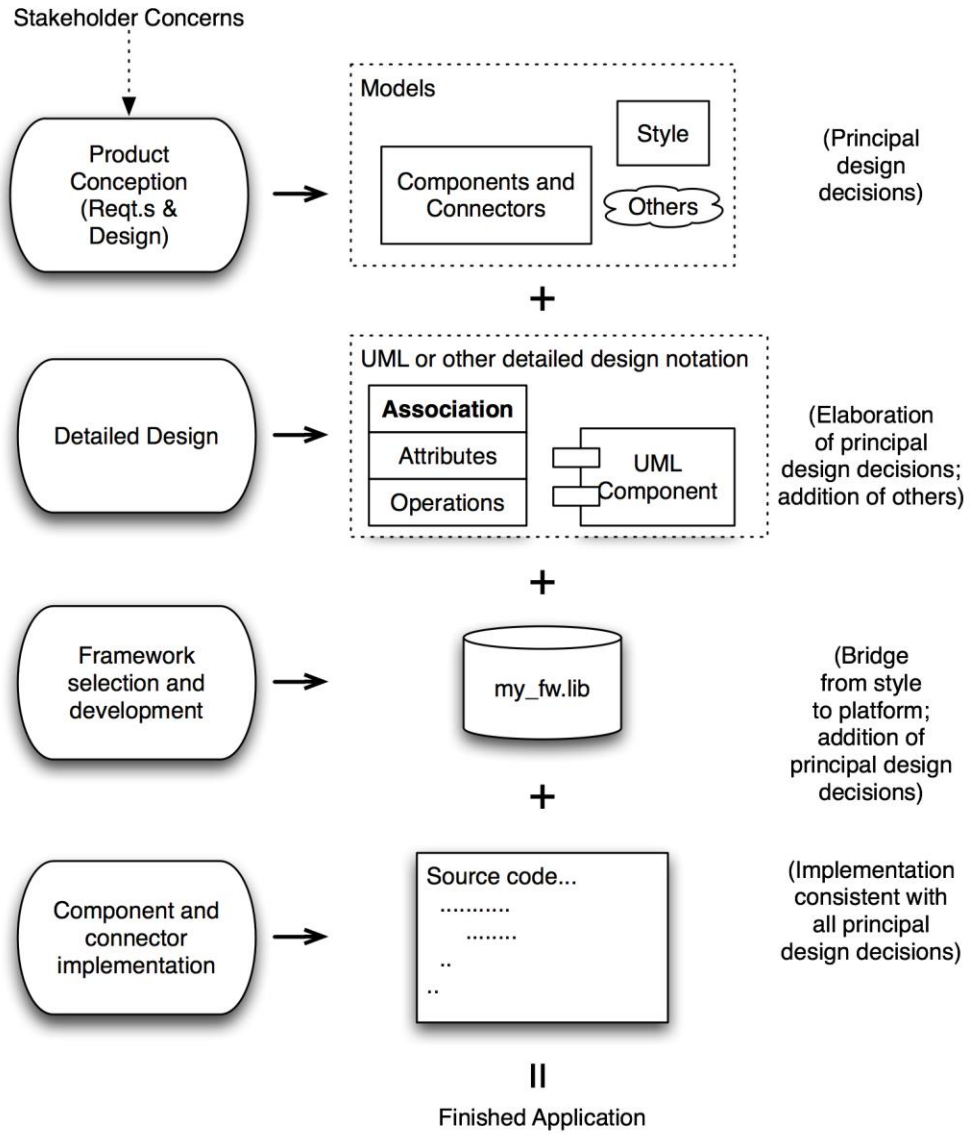
-剥夺了未来分析应用程序体系结构的能力
-误导所有的利益相关者，让他们认为自己拥有什么，而不是真正拥有什么
-使任何基于文档的(但不准确的)架构的开发或演进策略注定失败

Implementation Strategies

APP--specific 功能性
non-specific 非功能性

- Generative techniques
  - e.g. parser generators
    解析生成器
- Frameworks
  - collections of source code with identified places where the engineer must “fill in the blanks”
    源代码的集合，带有工程师必须“填空”的标识位置
- Middleware
  - CORBA, DCOM, RPC, …
- Reuse-based techniques
  - COTS, open-source, in-house
- Writing all code manually
  手工编写所有代码

How It All Fits Together



Analysis and Testing

- Analysis and testing are activities undertaken to assess the qualities of an artifact
  - 分析和测试是评估工件质量的活动
  - 越早发现并纠正错误，总成本越低
  - 分析需要严格的表述，因此可以提出和回答精确的问题
- The earlier an error is detected and corrected the lower the aggregate cost
- Rigorous representations are required for analysis, so precise questions can be asked and answered

Analysis of Architectural Models

- Formal architectural model can be examined for internal consistency and correctness
- An analysis on a formal model can reveal
  - Component mismatch
  - Incomplete specifications
  - Undesired communication patterns
  - Deadlocks
  - Security flaws
- It can be used for size and development time estimations

· 可以检查正式架构模型的内部一致性和正确性
· 对正式模型的分析可以揭示：
--组件不匹配
--不完整的规范
--非理想的沟通模式
--死锁
--安全漏洞
•它可以用于规模和开发时间的估计

Analysis of Architectural Models (cont'd)

- Architectural model 架构模型
  -可检查是否符合要求
  -可用于确定源代码的分析和测试策略
  -可以用来检查一个实现是否忠实
  – may be examined for consistency with requirements
  – may be used in determining analysis and testing strategies for source code
  – may be used to check if an implementation is faithful

Evolution and Maintenance

- All activities that chronologically follow the release of an application
- Software will evolve
  - Regardless of whether one is using an architecture-centric development process or not
- The traditional software engineering approach to maintenance is largely ad hoc
  - Risk of architectural decay and overall quality degradation
- Architecture-centric approach
  - Sustained focus on an explicit, substantive, modifiable, faithful architectural model

•应用程序发布后按时间顺序进行的所有活动
•软件将会进化
  -无论是否使用以架构为中心的开发流程
•传统的软件工程方法在很大程度上是临时性的
  -架构退化和整体质量下降的风险
•以构架为中心的方法
  -持续关注明确的、实质性的、可修改的、忠实的架构模型

Architecture-Centric Evolution Process

- Motivation
- Evaluation or assessment
- Design and choice of approach
- Action
  - includes preparation for the next round of adaptation

•动机
•评估
•方法的设计和选择
•行动
-包括为下一轮适应做准备

Processes

- Traditional software process discussions make the process activities the focal point
- In architecture-centric software engineering the product becomes the focal point
- No single “right” software process for architecture-centric software engineering exists

•传统的软件过程讨论将过程活动作为焦点
•在以架构为中心的软件工程中，产品成为焦点
•对于以架构为中心的软件工程，没有单一的“正确的”软件过程存在

Project Lifecycle Context

- We will consider this in more detail later in the course.
- We know that whatever lifecycle we use we will need to do the following (and these are all done best by talking about architecture:
  - Make a business case for the system
  - Understanding requirements that concern quality attributes
  - Deciding on architecture
  - Documenting architecture
  - Analysing and evaluating architecture
  - Implementing and testing the system based on architecture
  - Ensuring the implementation conforms to the architecture
- These take place in all lifecyles but in different places

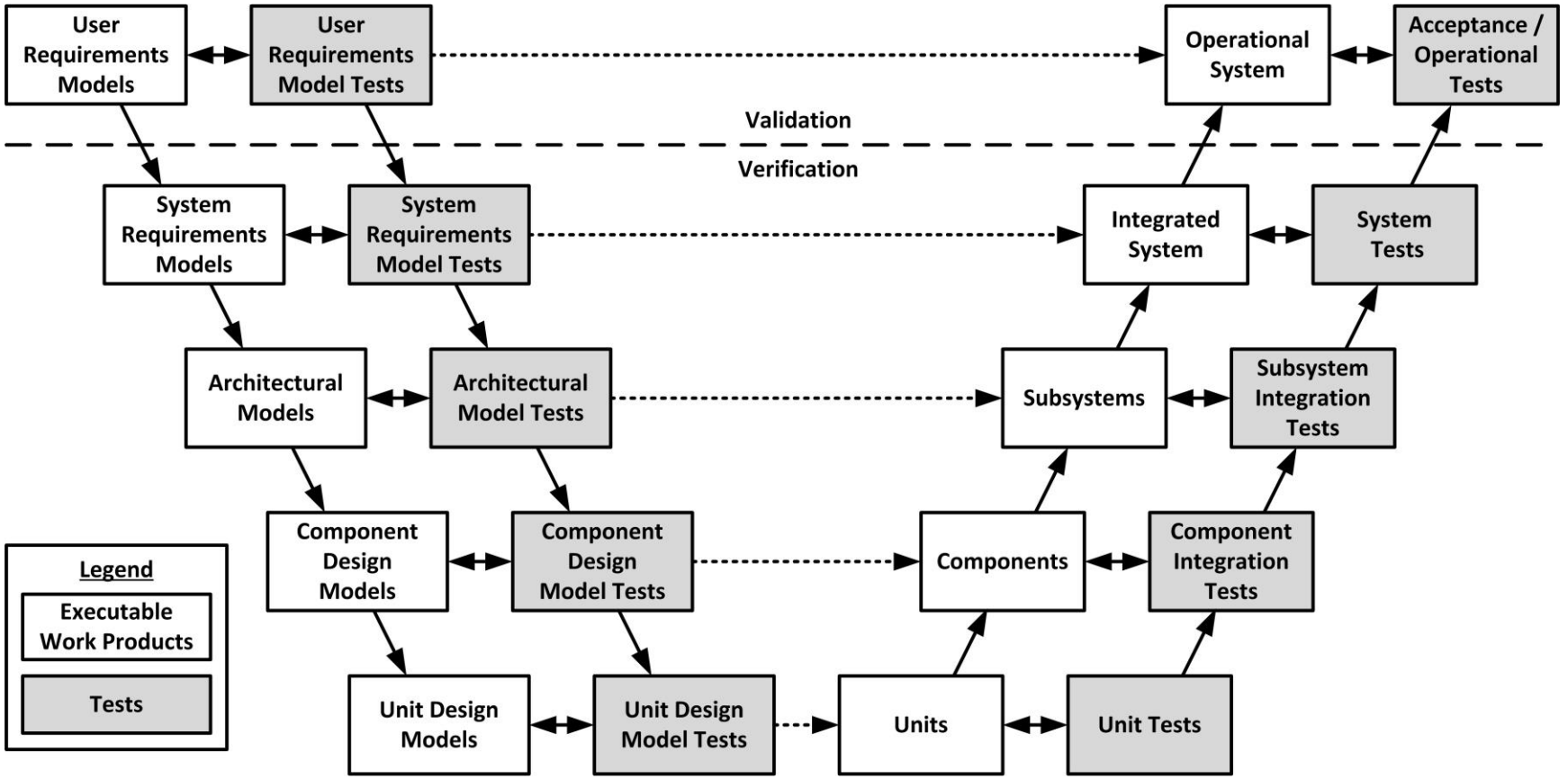
•我们将在稍后的课程中更详细地考虑这个问题。
•我们知道，无论我们使用什么生命周期，我们都需要做以下事情(这些都是通过谈论架构来实现的:
-为系统做一个商业案例
-理解与质量属性相关的要求
-决定架构
-记录架构
-分析和评估架构
-基于架构实现和测试系统
-确保实现符合架构
•这些发生在整个生命周期，但在不同的地方

Lifecycle Models

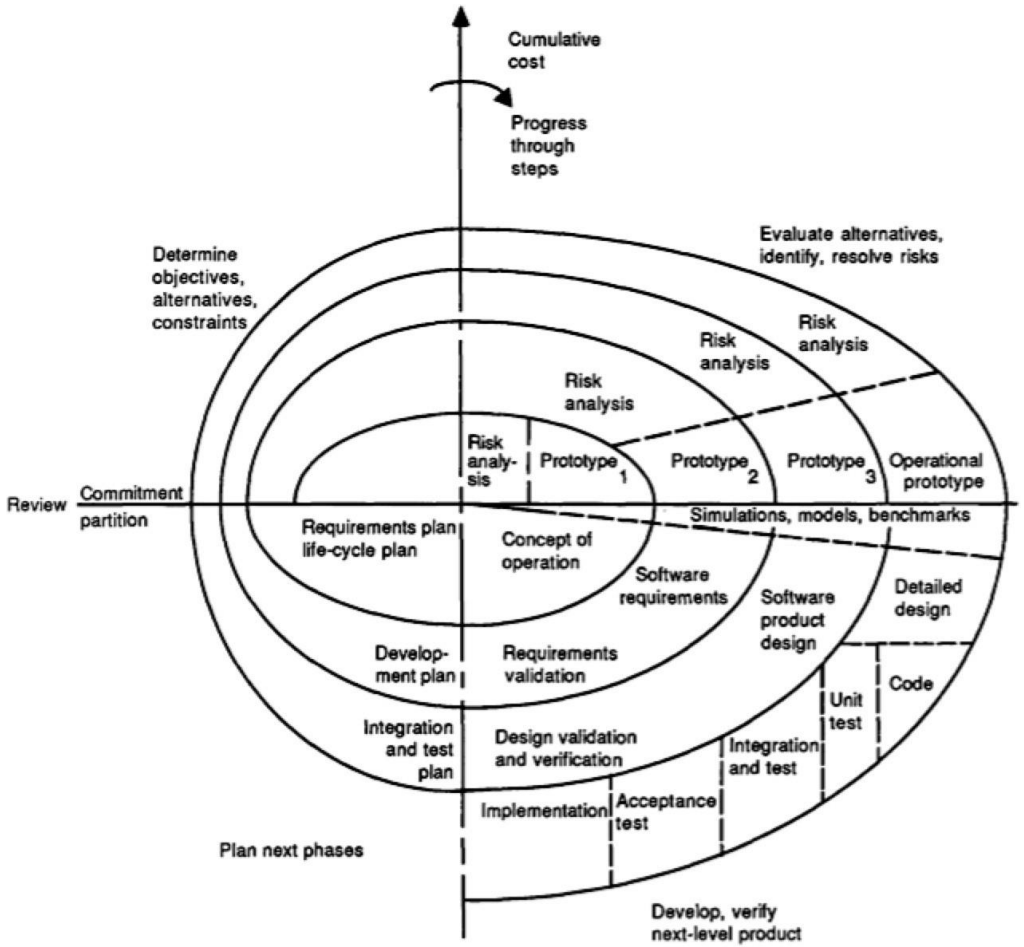
- We will consider three models (or classes of models in future lectures):
  - The V-model which is a development of waterfall and explicitly includes architectural design as a stage.
  - Iterative models as exemplified by Boehm’s spiral model that focusses on project risk management. Here architecture that gives us vision of how development tasks are interrrelated will play a key role as well as ensuring quality requirements are met.
  - Agile – here some decisions are more stable than others and the more stable ones concern architectures.

我们将考虑三种模型(或在未来的课程中考虑模型的类别):
· V模型是瀑布模型的发展，明确地将架构设计作为一个阶段。
· Boehm的螺旋模型强调了项目风险管理的迭代模型。在这里，为我们提供了开发任务是如何相互关联的看法的体系结构将扮演一个关键角色，同时确保质量需求得到满足
· 敏捷——这里有些决策比其他决策更稳定，而更稳定的决策关注架构

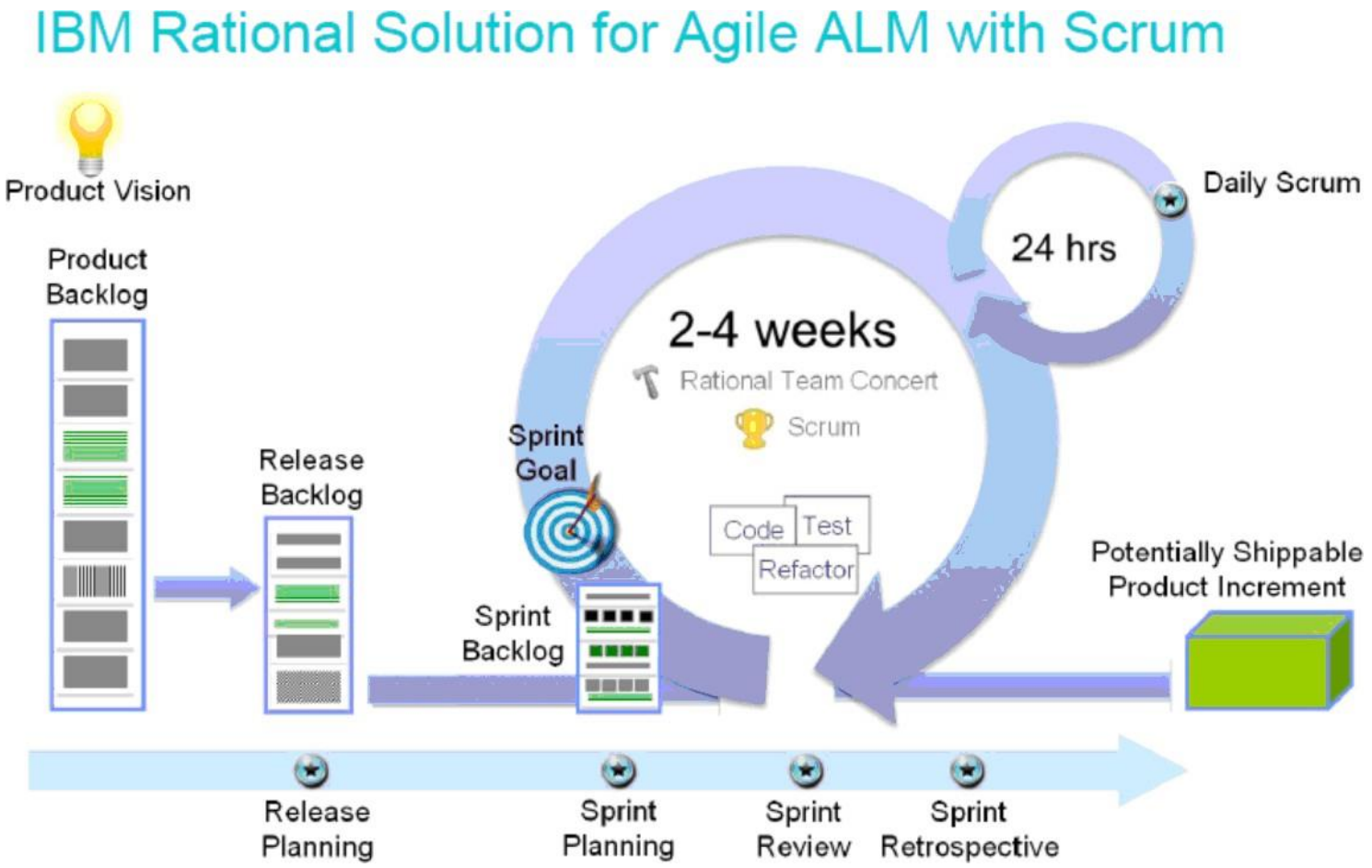
V-model


User Requirements Models
User Requirements Model Tests
Operational System
Acceptance / Operational Tests
Validation
Verification
System Requirements Models
System Requirements Model Tests
Integrated System
System Tests
Architectural Models
Architectural Model Tests
Subsystems
Subsystem Integration Tests
Component Design Models
Component Design Model Tests
Components
Component Integration Tests
Unit Design Models
Unit Design Model Tests
Units
Unit Tests
Legend
Executable Work Products
Tests

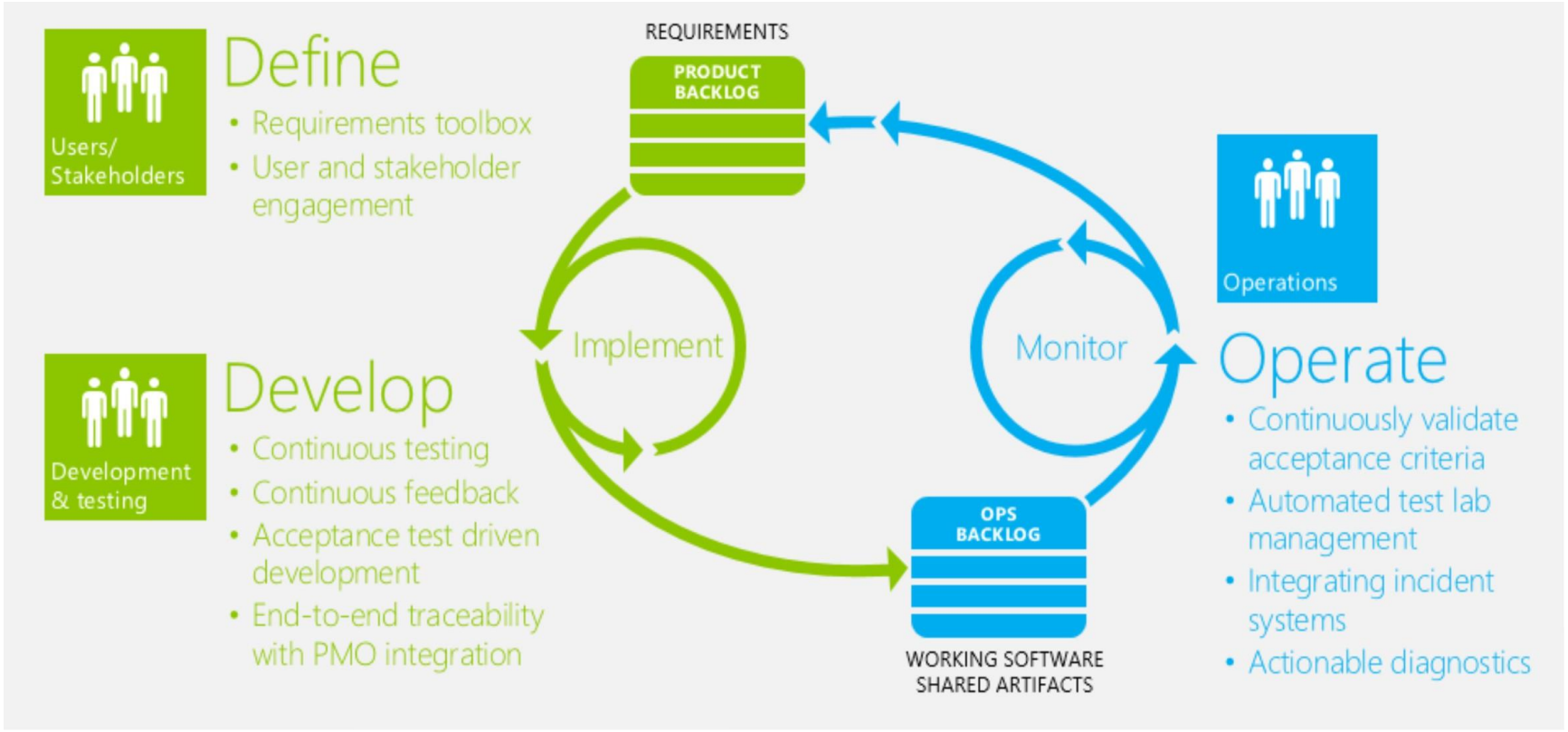
Spiral Model



Agile


IBM Rational Solution for Agile ALM with Scrum
Product Vision
Product Backlog
Release Backlog
Sprint Goal
Sprint Backlog
2-4 weeks
Rational Team Concert
Scrum
Code
Test
Refactor
24 hrs
Daily Scrum
Potentially Shippable Product Increment
Release Planning
Sprint Planning
Sprint Review
Sprint Retrospective

Agile + Devops



hMps://agilepearls.wordpress.com/tag/devops/

Business context

- Here we will consider two aspects in later lectures:
  - How the organisational structure of stakeholders can drive architectural decisions and shapes decision taking around architecture.
  - How architectural expertise drives the structure of development organisations in terms of their functional units and interrelationships.
- This is considered in depth in the SEI report:
  "Rela%ng Business Goals to Architecturally Significant Requirements for So:ware Systems"

•在后面的课程中，我们将考虑两个方面:
  -甲方的组织结构如何驱动架构决策并形成围绕架构的决策
  -就功能单元和相互关系而言，架构专家如何驱动开发组织的结构
•在SEI报告中深入考虑了这一点:将业务目标与So:ware系统在架构上的重要需求联系起来

Professional Context

- The architectural perspective gives you as a professional:
  - A way of describing your expertise.
  - Your skills as an architect will be recognised within organisations you work with.
  - You can use architecture as a way of describing your past experience.
  - You can specialise in particulr classes of architecture e.g. financial architecture.

作为专业人士，架构视角为您提供：
· 描述你专业技能的一种方式。
· 你作为架构师的技能将在你工作的组织中得到认可。
· 您可以使用架构作为描述您过去经验的一种方式。
· 你可以专攻特定的架构课程，例如金融架构。

Managing Attributes

- Enables us to manage the key attributes of a system:
  - If performance is important we need to manage timing behaviour
  - If modifiability is important it is important to try to localise functionality so that modification is localised to change a function.
  - If managing project risk is important it is important to have clear responsibility for delivery of components and good traceability

使我们能够管理系统的主要属性:
-如果性能很重要，我们需要管理时间行为
-如果可修改性很重要，那么尝试将功能本地化是很重要的，这样修改就可以本地化来改变一个功能。
-如果管理项目风险很重要，那么对组件的交付有明确的责任和良好的可追溯性就很重要

Managing Change

- Allows reasoning about and managing change:
  - We can see three levels:
    - Inside an element
    - Between elements maintaining the architecture
    - Requiring architectural change
  - These distinctions help in assessing the cost of change
  - We can identify likely changes and try to architect to minimize architectural change

允许对变化进行推理和管理：
-我们可以看到三个层次：
•元素内部
•维护架构的元素之间
•需要架构变更
-这些区别有助于评估变化的成本
-我们可以识别可能的变更，并尝试最小化架构变更

Prediction of Attributes

- Allows prediction of the key quality attributes
  - Should be able to build models on the basis of the architecture that prdict key attributes
  - E.g. if we want a system to be scalable:
    - the architecture should include:
      - capacity information,
      - dynamic component creation, and
      - dynamic deployment characteristics
    - These should be sufficiently detailed to be able to predict how increases in demand will be serviced

允许预测关键质量属性
-应该能够在预测关键属性的架构的基础上构建模型
-例如，如果我们想要一个系统是可扩展的:
•架构应包括:
-能容纳的信息，
-动态组件创建
-动态部署特性
•这些应足够详细，以便能够预测如何满足需求的增长

Improves Communication

- Allows better communication among stakeholders:
  - User has particular requirements in terms of user experience
  - Customer needs to know about schedule, budget, meeting regulation in their market
  - Project manager needs to know the depndencies in terms of modules and components
- These might be accommodated by different views of the system that are consistent

•促进利益相关者之间更好的沟通:
-用户在用户体验方面有特殊要求
-客户需要了解他们市场的进度、预算、会议规则
-项目经理需要了解模块和组件的依赖程度
•这些可以通过系统的不同观点来适应，而这些观点是一致的

Early Design

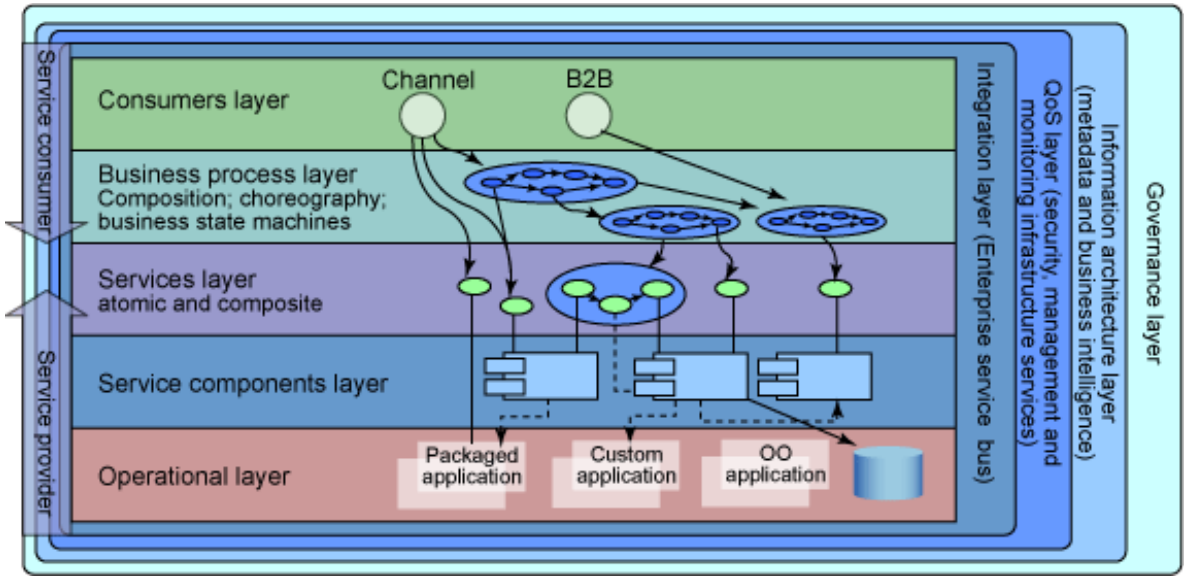
- Carries the earliest (most fundamental) design decisions:
  - What are the key quality attributes?
  - What architecture gives us control over thses attributes?
  - How do we characterise the behaviour of architecture elements?
  - E.g. suppose the attribute is that we want to know whether the system is faulty what does this say about architecture?

携带最早的(最基本的)设计决策:
-主要的质量属性是什么?
-什么样的架构能让我们控制这些属性?
-我们如何描述架构元素的行为?
-例如，假设属性是，我们想知道系统是否有问题，这说明了架构是什么?

Constraints

定义实现的约束：
-架构指定元素及其交互作用
-例如，分层架构通常约束访问在相邻的层之间

- Defines constraints on implementation:
  - Architecture specifies the elements and their interaction
  - For example, layered architecture usually constrain access to be between adjacent layers

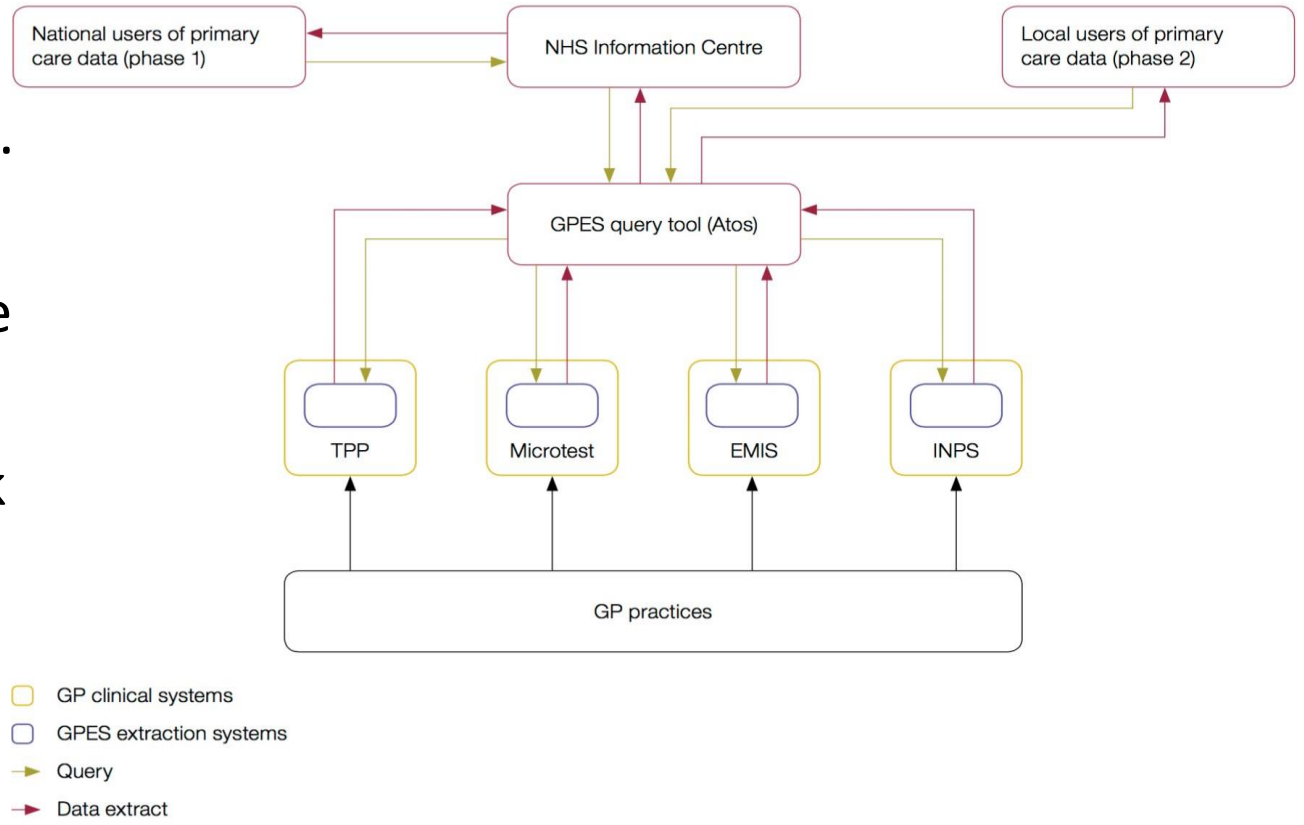


Structure of Organization

- Reflects the structure of an organisation:
  - Of the development organisation.
  - Of the overall organisation
  - The work organisation shapes the architecture
  - The architecture shapes the work organisation.

反映一个组织的结构：
-开发组织
-整体组织
-工作组织塑造架构
-架构塑造工作组织

Figure 2
Design of the General Practice Extraction Service



Source: National Audit Office, based on information in the NHS IC GPES business cases

Evolutionary Prototyping

进化原型

- Provides the basis for evolutionary prototyping:
  - For example:
    - Plug and play – early experience of the base functionality + extensibility
    - Real time architectures – early experience with scheduling – worst case execution times guide design, and deployment

为进化原型设计提供了基础：
--例如：
· 即插即用——基本功能+可扩展性的早期经验
· 实时架构——早期的调度经验——最坏情况下的执行时间指导设计和部署

Cost and Scheduling

- Is the key artifact in reasoning about cost and scheduling:
  - Capture dependencies
  - Estimate required efforts etc
  - Allocate effort to elements
  - Understand how elements influence each other
  - Use architecture to interpret bottom-up estimates from teams working on elements

关于成本和调度的关键工件：
-获取依赖项
-估计需要付出的努力等
-将努力分配给元素
-了解各因素如何相互影响
-使用架构来解释团队对元素进行的自底向上的评估

Product Line

- Can be used as the transferrable, reuseable model at the heart of a product line
  - Elements are assets that can copmpose to give new functionality
  - Architecture provides the means to compose the elements
  - Planned approach to the reuse of architectural elements

在产线的核心可以作为可转移的、可重用的模型
-元素是资产，可以联合给新的功能
-架构提供了组合元素的方法
-重用架构元素的规划方法

Component Level

专注于组件的组装而不是组件的创建

- Focusses on the assembly of components rather than on the creation of the components:

通过精心设计的元素和架构，我们可以组合来自不同生产商的元素，只要它们符合接口标准。带来的好处

  - With well designed elements and architecture we can combine elements from different producers provided they conform to interface standards. Brings benefits:
    - Decrease time to market
    - More reliability
    - Lower cost
    - Flexibility e.g. using multiple or alternate suppliers for a component.

· 缩短上市时间
· 更多的可靠性
· 低成本
· 灵活性，例如，为一个组件使用多个或替代供应商

Channels Development

限制设计替代方案，以协调的方式引导开发人员的工作

- Restricts design alternatives and channels developer effort in a coordinated way
  - 为开发人员提供已定义的上下文
  - Provides a defined context for the developer
  - 定义良好的接口，对所需的功能和质量属性有清晰的概念
  - Well defined interfaces and clear idea of the functionality and the quality attributes required
  - 明确什么是架构决策，什么是开发决策(例如，如果容错很重要，我们不希望在模块中这样做)，我们应该使用架构
  - Clarity on what is an architectural decision and what is a development decision (e.g if fault tolerance is important we don’t want to do that inside a module) we should use architecture.

Training Resource

- Provides the basis for training new team members:
  - Multiple views provide different stakeholder perspectives
  - Records interactions between elements
  - Abstracts from detail to provide an overview

为培训新团队成员提供基础:
-多视图提供不同的利益相关者视角
-记录元素之间的相互作用
-从细节中摘录以提供概述

Contexts Shape Design

- The technical context identifies features we want to control and packages a range of other properties.
- Standard architecture (parterns, domain specific architectures etc.) package these.
- The other context we consider also help to shape the choice of architecture.
- Design uses pre-decided structures and then alters and extends structure as necessary.

· 技术上下文标识了我们想要控制的特性，并打包了一系列其他属性。
· 标准架构(模式，领域特定架构等)将这些打包。
· 我们考虑的另一个背景也有助于塑造架构的选择。
· 设计使用预先决定的结构，然后在必要时改变和扩展结构。

Architectural Design Goals

为了实现功能，架构师创建了具有以下目标的体系结构

- To achieve capabilities, an architect creates an architecture with the following goals in mind.
  - Modularity
  - Protection and exposure
  - Component extensibility
  - Roles and responsibilities
  - Contracts
  - Pluggable behavior

-模块化
-保护和暴露
–组件可扩展性
-角色及职责
-合同
-可插入的行为

Modularity

将代码分离为自给自足、高内聚、低耦合的部分

- Separate the code into self-sufficient, highly cohesive, low-coupling pieces.

模块化创建了良好的代码抽象

  - Modularity creates good code abstractions.

模块化隐藏细节

  - Modularity hides details.

模块化代码就像一个黑盒子

  - Modular code is like a black box.

Protection and Exposure

保护昂贵的资源，防止不当使用

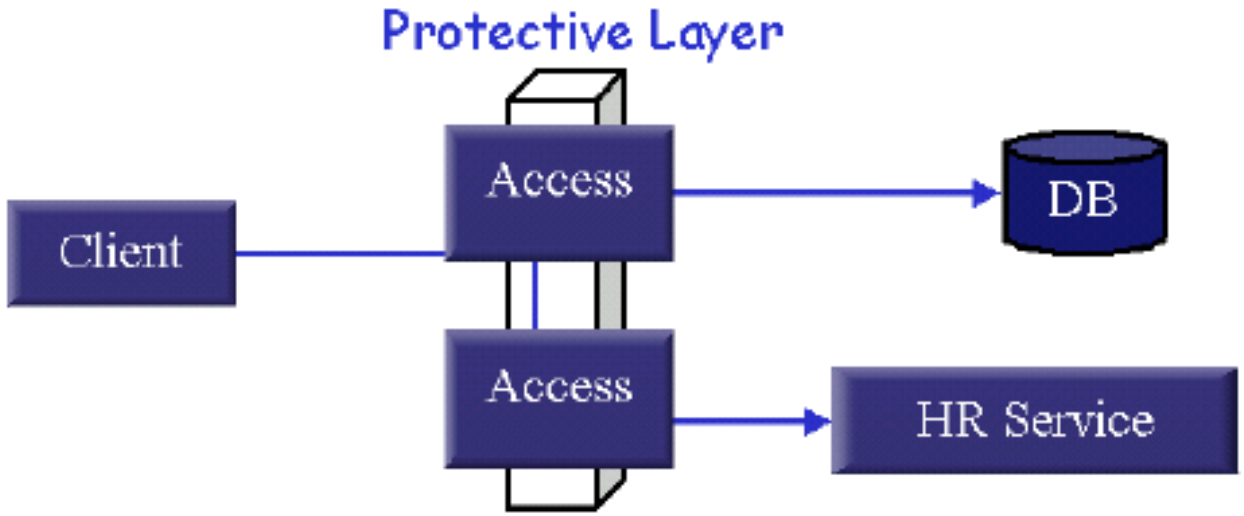
- Guard expensive resources to prevent inappropriate use.

共享资源的访问成本通常很高

- Shared resources are usually expensive to access.

保护要求使用可见性访问类控制公开

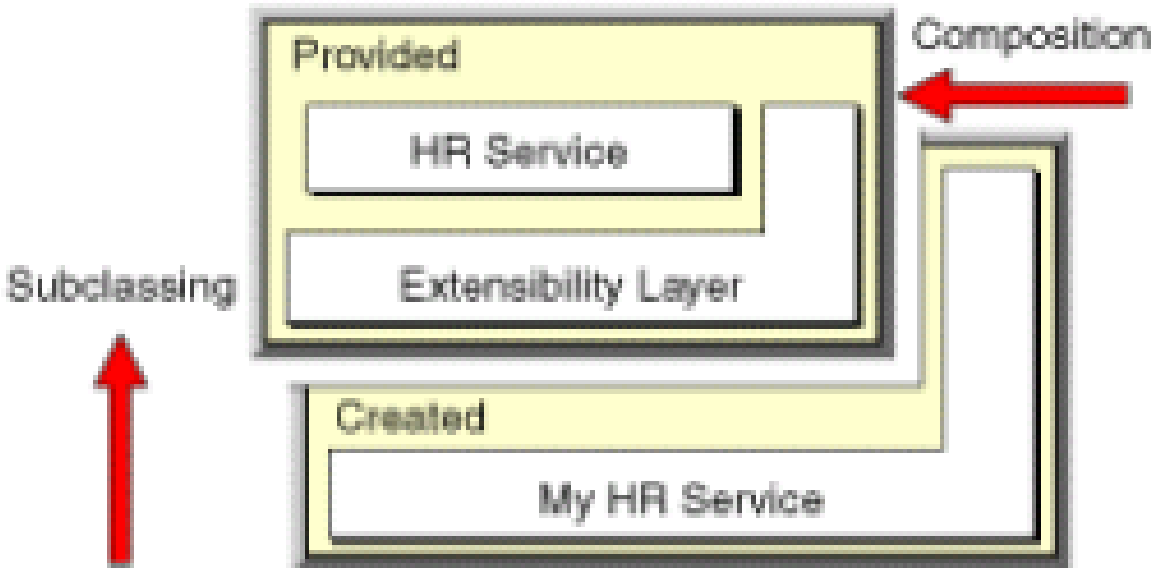
- Protection mandates controlled exposure using *visibility access* classes.



Component Extensibility

在不影响现有功能的情况下修改或添加功能

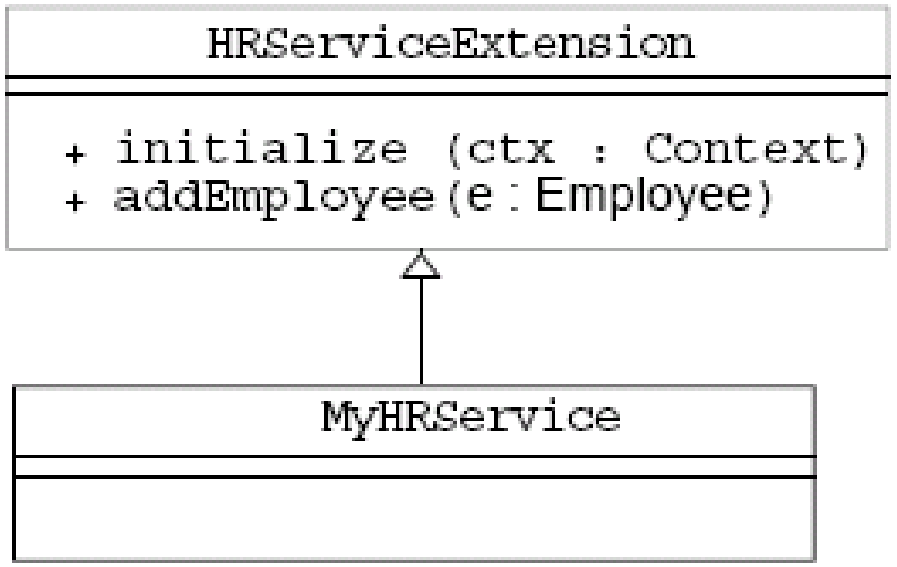
- Modify or add functionality without impacting the existing functionality.
- Subclass
- Composition



Component Extensibility Using Subclassing

通过子类化扩展创建了强大的专门化机制

- Extending by subclassing creates powerful specialization mechanisms.



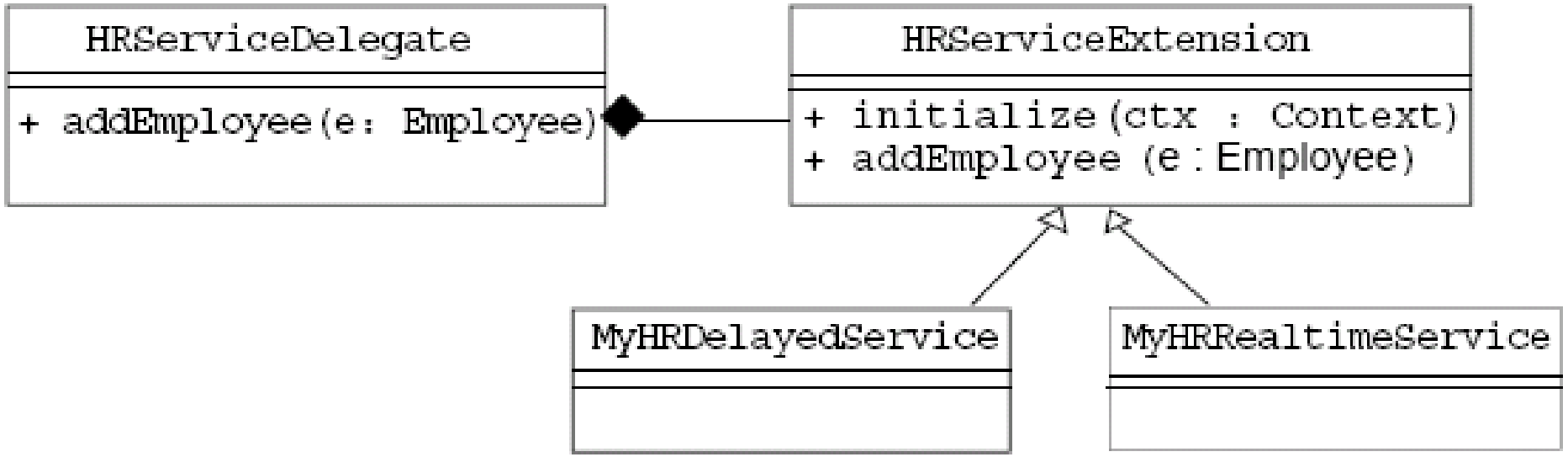
Component Extensibility Using Composition

组合提供了一种强大的动态扩展机制

- Composition provides a powerful and dynamic extension mechanism.

提供了一个更“可插拔”的架构

- Provides for a more “pluggable” architecture.



Roles and Responsibilities

要创建清晰的抽象，您必须确定角色和职责

- To create clear abstractions, you must identify roles and responsibilities.

Role: Dynamic presentation creator

Role: Account data formatter

Role: Control account access

确定抽象的角色和职责

- Identify roles and responsibilities for the abstraction.

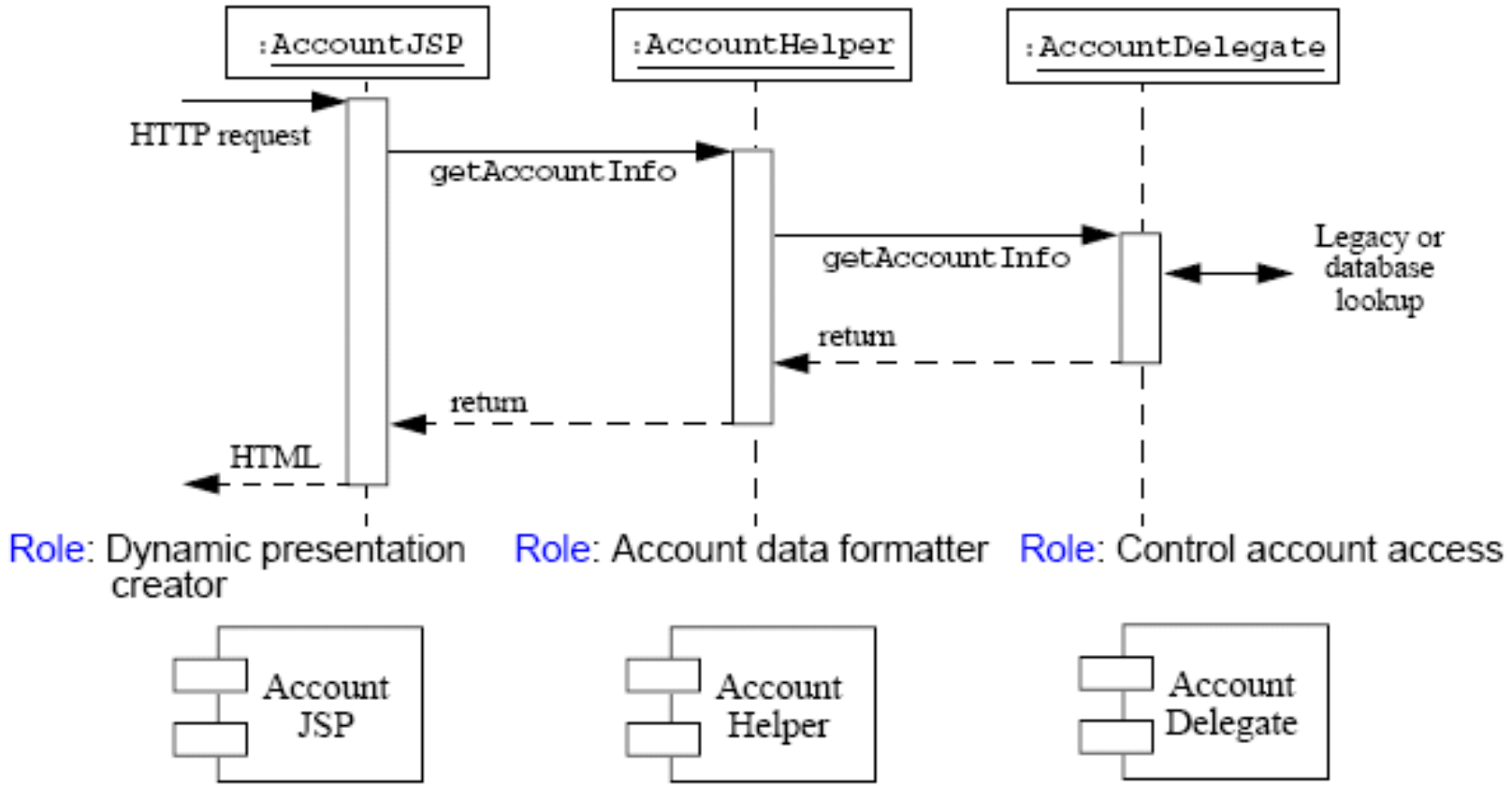
Role: Dynamic presentation creator

- ☑ Should: Create presentation
- ☑ Should Not: Execute business logic or make remote calls

Roles and Responsibilities

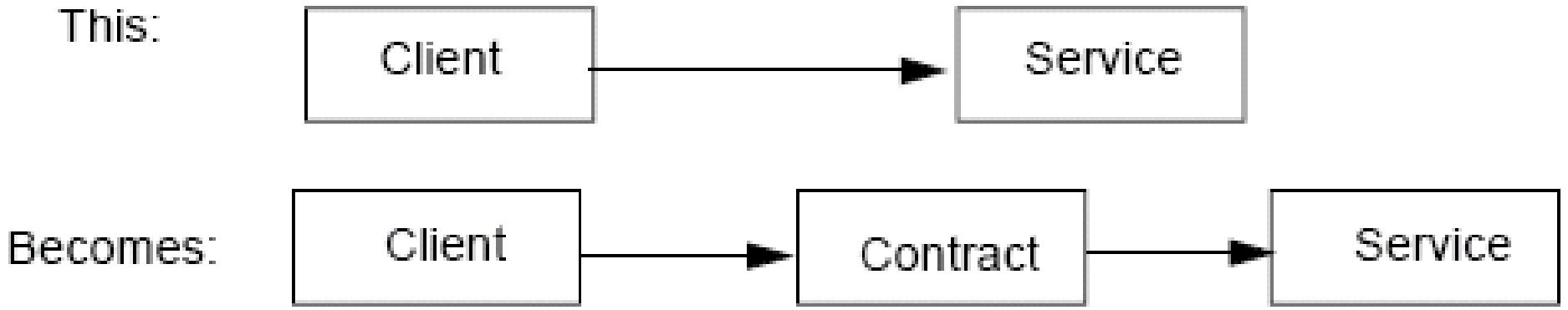
定义角色来标识有效的数据输入和输出

- Define roles to identify valid data input and output



Contracts

- 契约是客户端可见的API
- Contracts are the client-visible API.
- 也称为“接口编程”
- Also known as “programming to the interface.”
- 契约在体系结构上表示为抽象
- Contracts are architecturally represented as abstractions.
- 契约规定了业务访问和数据类型
- Contracts mandate business access and data type.
- 识别“前”和“后”条件
- Identify “pre” and “post” conditions.



Types of Contracts

- There are two types of contracts:
  - Contract-based – Action and data are explicitly defined in the contract.
  - Message-based – Action and data are packaged in the request. The contents of the message are parsed to identify the requested function.

基于合同的--在合同中明确定义了动作和数据

基于消息的--操作和数据打包在请求中。消息的内容将被解析以识别所请求的函数

Pluggable Behavior

通常，您希望通过简单地传递参数来使用方法

- In general, you expect to use a method by simply passing parameters.

```
aTextArea.insertText(“This is inserted”, 42);
```

您还可以将复杂的行为放入参数对象中，例如字体、段落布局，等等在aStyledDocument中

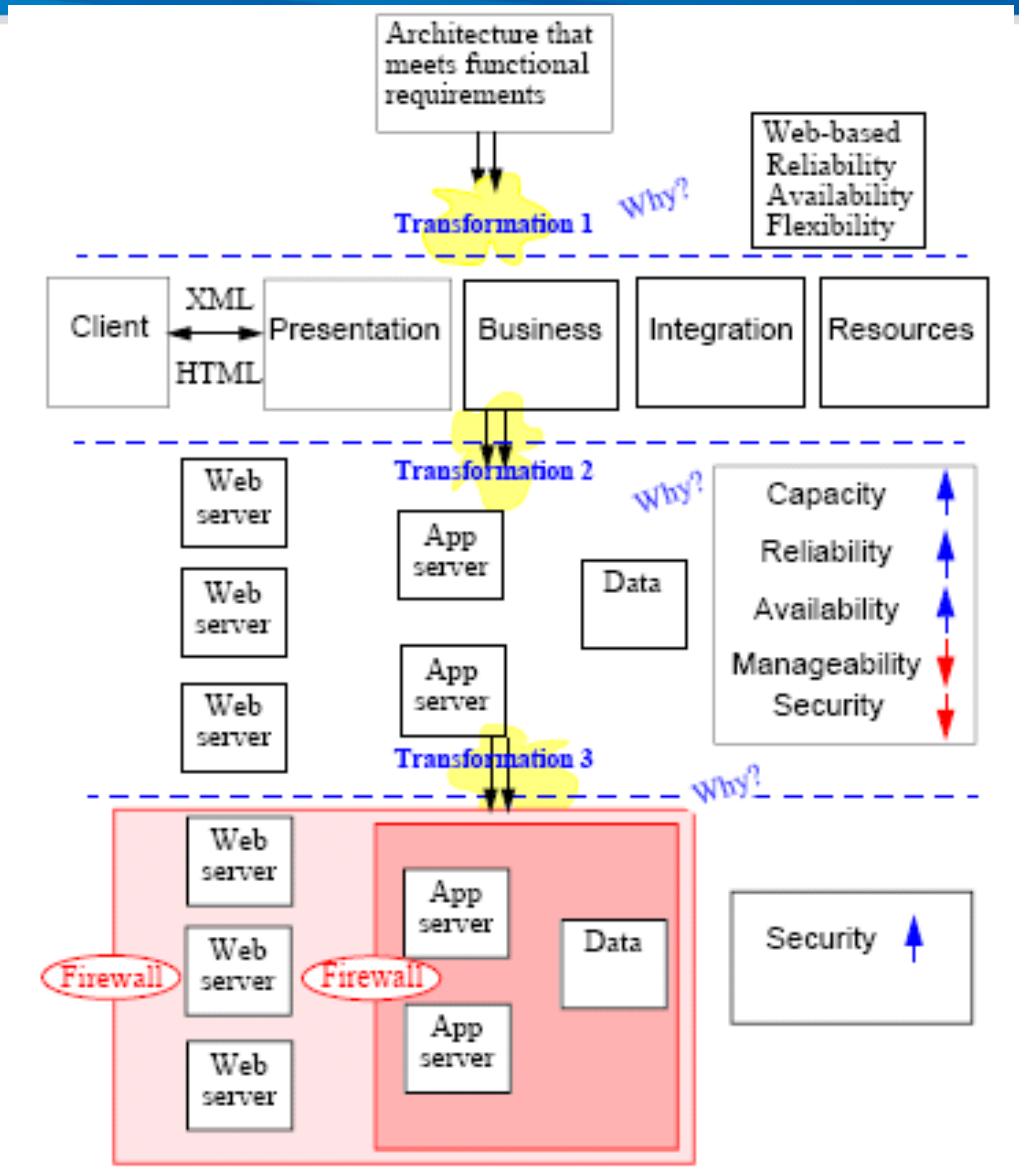
- You can also put complicated behavior into a parameter object, such as fonts, paragraph layouts, and so on in aStyledDocument.

```
aJTextPane.setStyledDocument(aStyledDocument);
```

传递复杂对象的能力使系统更加灵活

- The ability to pass complex objects makes a system more flexible.

Providing System Requirements



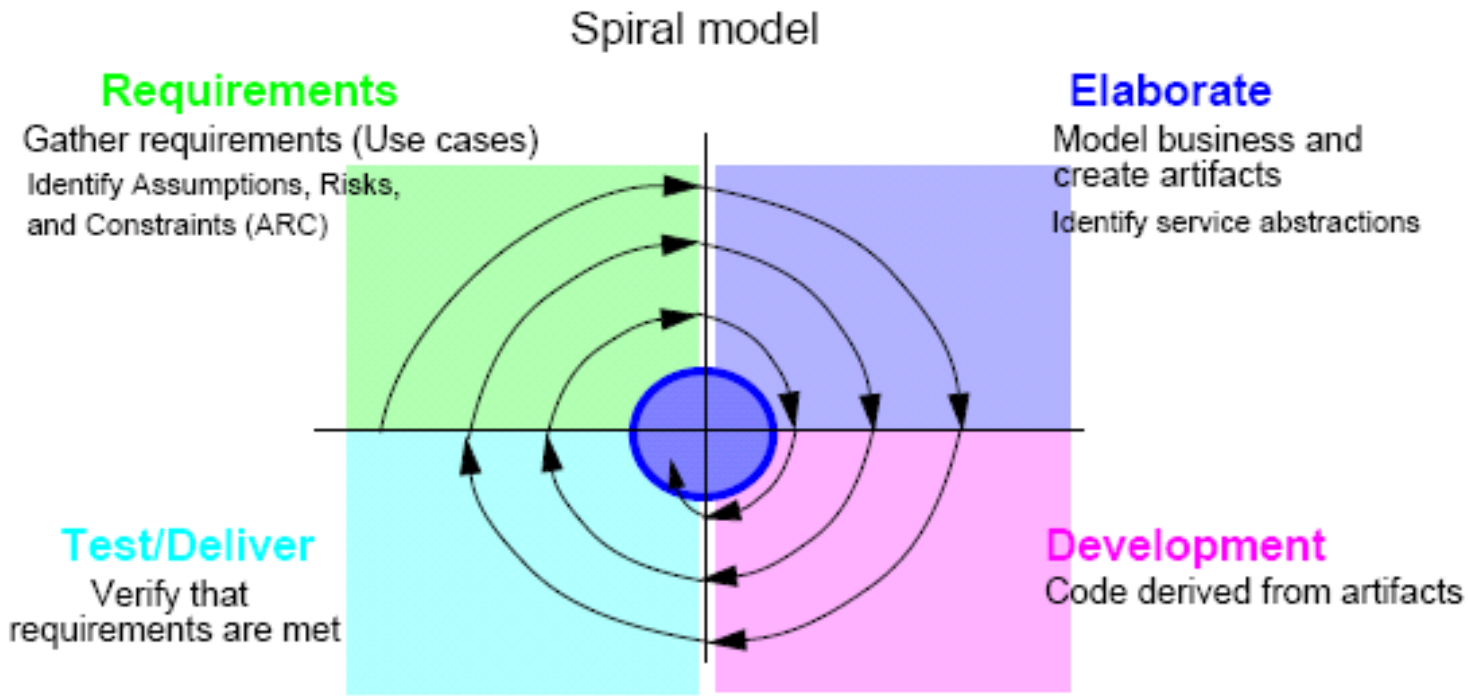
Process

架构师使用流程来构建满足需求的系统

- Architects use processes to build a system that meets the requirements.

流程应该是迭代的，并由用例驱动

- A process should be iterative and driven by use-cases.



Assumptions, Risks, and Constraints (ARC)

必须确定假设、风险和限制.

- Imperative to identify assumptions, risks and constraints.

Term	Definition	Examples
Assumptions	Refers to preconditions, the absence of which can affect the architecture of a system	Only one content language Fewer than 50 page views/ second 95% reliability is adequate
Risks	The cost (in probability of remediation) of making specific assumptions	Localize text content for easy translation Scalable architecture Translation and i18n
Constraints	Rules or imposed assumptions to which the system must adhere.	Static content is out sourced SPARC/Solaris implementation Pure HTML at client

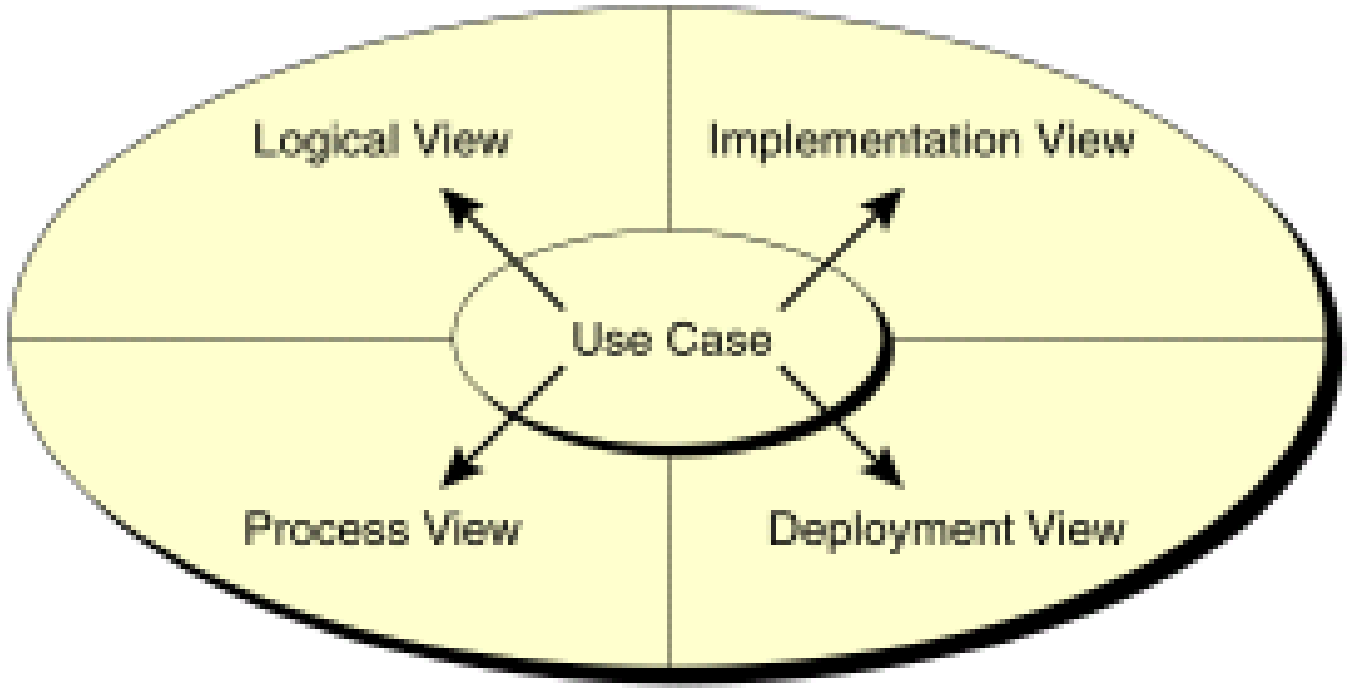
Model the Business

用用例为业务建模

- Model the business with use cases.

用例在每个视图中驱动架构

- Use cases drive the architecture at every view.



Check Your Progress

列出并定义关键的架构功能
- List and define the key architectural capabilities

列出并定义关键的架构设计目标
- List and define the key architectural design goals

列出并描述架构决策所产生的权衡
- List and describe the trade-offs that result from architectural decisions.

Think Beyond

J2EE技术如何与体系结构的原则，特别是功能一起工作

- How does J2EE technology work with the principles of architecture, specifically capabilities?

Summary (1)

对软件架构的正确看法影响着经典软件工程活动的每一个方面

- A proper view of software architecture affects every aspect of the classical software engineering activities

需求活动是与设计活动同等重要的伙伴

- The requirements activity is a co-equal partner with design activities

利用以前产品开发中获得的知识技术丰富了设计活动

- The design activity is enriched by techniques that exploit knowledge gained in previous product developments
- The implementation activity

实现活动
-以创建一个忠实的架构实现为中心
-利用各种技术以划算的方式达到这一目的

  - is centered on creating a faithful implementation of the architecture
  - utilizes a variety of techniques to achieve this in a cost-effective manner

Summary (2)

分析和测试活动可以集中在体系结构上并受其指导

- Analysis and testing activities can be focused on and guided by the architecture

演进活动围绕着产品的架构展开

- Evolution activities revolve around the product’s architecture.

对过程和产品的同等关注源于对软件架构角色的正确理解

- An equal focus on process and product results from a proper understanding of the role of software architecture