第二部分
创建软件体系结构

张莉 教授
北京航空航天大学
软件工程研究所
lily@buaa.edu.cn

第二部分　创建软件体系结构

- 理解和实现质量属性
  - 软件系统的质量属性
  - 软件体系结构与质量属性
  - 软件体系结构设计的原子操作
  - 软件体系结构战术
  - 软件体系结构模式和风格
- 软件体系结构案例分析
  - KWIC System
  - WWW
  - CORBA

实现质量属性

- 软件体系结构设计的原子操作
- 软件体系结构战术
- 设计模式
- 软件体系结构风格

第二部分　创建软件体系结构

- 理解和实现质量属性
  - 软件系统的质量属性
  - 软件体系结构与质量属性
  - 软件体系结构设计的原子操作
  - 软件体系结构战术
  - 软件体系结构模式和风格
- 软件体系结构案例分析
  - KWIC System（针对体系结构风格）
  - WWW
  - CORBA

单元操作

设计软件体系结构的基本操作

怎样在一组原则的指导下创建一个新的软件体系结构和模式

1. 单元操作简介

- 单元操作原始性更强，更为抽象
- 单元操作
  - 分离
  - 抽象
  - 压缩
  - 资源共享
- 质量属性之间的相互作用

单元操作 (1)

- 分离
  - 性能，易创建性，任务分配，可修改性，可移植性
  - 统一分离
    - 可以简化组件的集成,有利于系统整体的扩充
    - 部分－整体
    - 是－一个
  - 复制
    - 增强可靠性，提高性能
    - 运行时间复制
    - 静态复制

单元操作 (2)

- 抽象——创建一个虚拟机的操作
  - 简化创建和维护工作
  - 虚拟机：模拟功能片断
  - 分层系统
  - 公共接口

分离和抽象

单元操作 (3)

- 压缩
  - 和分离相反
  - 主要目的：
    - 提高系统性能
    - 在分层不能提供所需的服务时规避层次划分
    - 加快系统的开发
  - 常用技术：
    - 宏
    - 内嵌过程

单元操作 (4)

- 资源共享
  - 创建时一般要付出较高的代价
  - 增强系统的可集成性、可移植性和可更改性。
  - 例子
    - 数据库
    - 黑板
    - 集成软件工程工具环境
    - 服务器

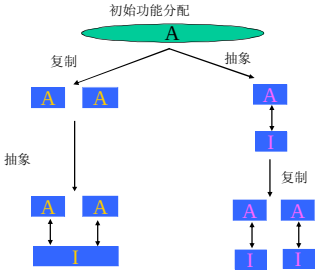
质量属性之间的互相作用

操作

	抽象	压缩	部分－整体分离	是－一个分离	复制	资源共享
可扩展性					+	
系统可修改性	+	-	+	+		
可集成性	+	-	+	+		+
可移植性	+	-				
串行性能	--	+		-	-	-
并行性能		-	+	+	+	
容错能力	+				+	
系统创建的简便性	+	-		+		+
组件可修改性	+	-				-
组件创建的简便性	+			+		+
可重用性	+	-	+			

质量属性之间的互相作用：单元操作的运作

- 示例：
  - 可修改性---抽象
  - 容错能力---复制



2. 将单元操作运用于用户界面软件

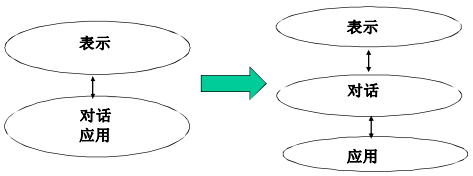
- 人机交互界面(HCI)
  - 单一构架
  - HCI的SEEHEIM模型
  - 模型视图控制器模型(MVC)
  - SEEHEIM 和MVC构架的比较
  - 下一代开发:
    - Part 1: SEEHEIM模型演化成ARCH/SLINKY模型
    - Part 2: MVC模型演化成PAC模型
    - PAC模型和ARCH/SLINKY模型的统一: PAC-AMODEUS模型
  - HCI软件构架的未来研究方向

将单元操作运用于用户界面软件(1)

- 单一构架

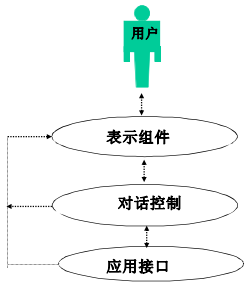
将单元操作运用于用户界面软件(2)

- HCI软件的的Seeheim模型
  - 可更改性,可移植性:抽象
  - 对单一模型进行部分一整体分离操作



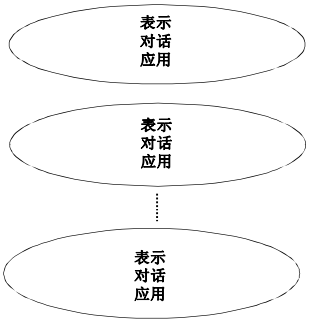
将单元操作运用于用户界面软件(3)

- Seeheim的逻辑模型



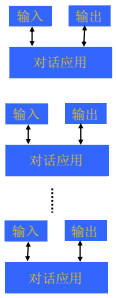
将单元操作运用于用户界面软件(4)

- MVC



将单元操作运用于用户界面软件(5)

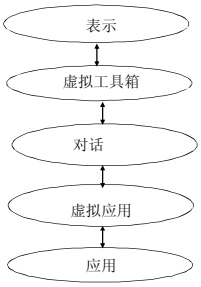
- 对表示进行分离



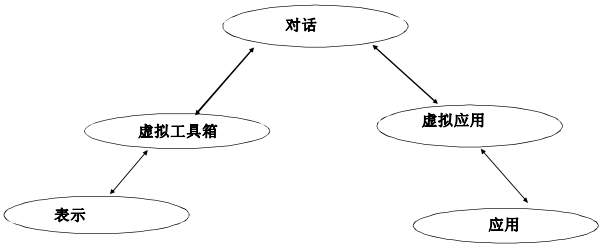
将单元操作运用于用户界面软件(6)

– 下一代开发

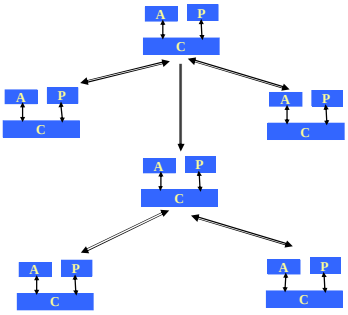
• 之一: SEEHEIM模型演化成ARCH/SLINKY模型



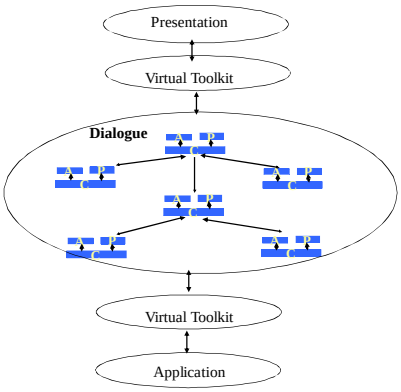
ARCH/SLINKY



之二: MVC 模型演化程PAC模型



PAC模型与Arch/Slinky模型的统一：PAC-AMODEUS模型



第二部分　创建软件体系结构

– **理解和实现质量属性**
  - **软件系统的质量属性**
  - **软件体系结构与质量属性**
  - **软件体系结构设计的原子操作**
  - **软件体系结构战术**
  - **软件体系结构模式和风格**
– **软件体系结构案例分析**
  - **KWIC System（针对体系结构风格）**
  - **WWW**
  - **CORBA**

软件体系结构战术

- **主要讨论：设计师使用设计模式、体系结构模式或体系结构策略创建设计的“战术”。**
- **概念:**
  – 战术：是影响质量属性响应控制的设计决策;
  – 战术的集合，称为“体系结构策略”
- **战术与体系结构模式**
  – 设计师通常会选择一个精心设计的模式或模式集合，来实现一个战术;
  – 同时，每个模式都实现了多个战术，不管是否是所期望的。
- **战术与体系结构风格**
  – 战术是设计的基本“构建块”

（一）可用性战术

- [可用性]：系统正常运行的时间比例。

$$\alpha = \frac{\text{mean-time-to-failure}}{\text{mean-time-to-failure} + \text{mean-time-to-repair}}$$

- 系统不再提供与其规范一致的服务时，故障就发生了；系统的用户可以观察到故障。
- 错误（或错误的组合）可能会导致故障的发生。
- 恢复或修复是可用性的重要方面

（一）可用性战术（续）

- 可用性战术的目标：
  - 阻止错误发展为故障，至少把错误限制在一定范围内，从而使修复成为可能。

- 维持可用性的方法：
  - 某种类型的冗余、用来检测故障的某种类型的健康监视、以及当检测到故障时恢复。
- 分别讨论错误检测、错误恢复、错误预防。

（一）可用性战术（续）

- 错误检测：识别错误
  - 命令/响应（ping/echo)
  - 心跳（dead man计时器）
  - 异常
    - 当识别到前面（可用性场景）所讨论的错误类中的某一个时，就出现了异常。
    - 异常处理程序通常将错误在语义上转换为可以被处理的形式。

（一）可用性战术（续）

- 错误恢复：包括准备恢复和修复系统两个部分
  - 表决
  - 主动冗余（热启动）
    - 所有的冗余组件都以并行的方式对事件做出响应。因此，它们构处于相同的状态。
    - 仅使用一个组件的响应（通常是做出响应的第一个组件），丢弃其余组件的响应。
    - 错误发生时，使用该战术的系统停机时间通常是几毫秒－－因为恢复时间就是切换时间。
    - 通常用在客户机/服务器配置中（如数据库管理系统）。

（一）可用性战术（续）

- 被动冗余（暖启动/双冗余/三冗余）
  - 一个组件（主要的）对事件做出响应，并通知其它组件（备用的）必须进行的状态更新。
  - 当错误发生时，在继续提供服务前，系统必须确保备用状态是最新的。
- 备件
  - 备用件是计算平台配置用于更换各种不同的故障组件。
  - 出现故障时，必须将其重新启动为适当的软件配置，并对其状态进行初始化。
  - 该战术的停机事件通常是几分钟。

（一）可用性战术（续）

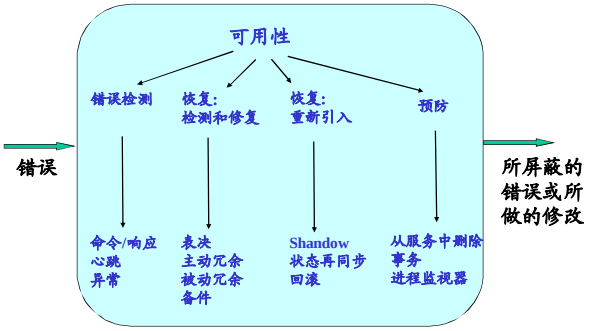
有一些依赖于组件重新引入的修复战术。当冗余组件失败时，可以在纠正该组件后将其再次引入。相关战术有：

- Shadow操作：以前出现故障的组件可以在短时间内以“shadow模式”运行，以确保在恢复组件之前。模仿工作组件的行为。
- 状态再同步：主动和被动冗余战术要求所恢复的组件在重新提供服务前更新其状态。更新方法取决于可以承受的停机事件、更新的规模以及更新所要求的信息的数量。
- 检查点/回滚：检查点就是记录所创建的一致状态，或定期进行，或对具体事件做出响应。故障时，可检测到状态不一致，这时，应使用上一个一致状态检测点和当时的事务日志来恢复系统。

（一）可用性战术（续）

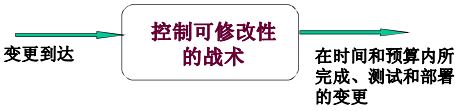
- 错误预防
  - 从服务中删除
  - 事务
  - 进程监视器

可用性战术总结



（二）可修改性战术

- 目标:
  - 控制实现、测试和部署变更的时间和成本。



（二）可修改性战术（续）

将可修改性战术根据目标分类:

- 局部化修改:
  - 目标：减少由某个变更直接影响的模块的数量
- 防止连锁
  - 目标:
- 延迟绑定
  - 目标:

前两个战术的区别:
有直接受变更影响的模块
（那些调整其职责来完成变更的模块）
和间接受变更影响的模块
（那些其职责保持不变，但必须改变
其实现来适应直接影响的模块）

（二）可修改性战术（续）

- 局部化修改
  - 目标：在设计阶段为模块分配职责，以把预期的变更限制在一定的范围内。
  - 战术:
    - 维持语义的一致性：明确模块中职责的关系，确保所有职责能协同工作，不需要依赖于其他模块。
    - 预期期望的变更
    - 泛化该模块
    - 限制可能的选择

（二）可修改性战术（续）

- 防止连锁反应
  - 修改所产生的一个连锁反应就是需要改变该修改没有直接影响到的模块。
  - 例如：如果修改了模块A以完成某个特定的修改，就必须修改模块B。称B依赖于模块A。
- 8中类型的依赖

8种类型的依赖性

- （1）语法
  - **数据。**要使B正确编译（或执行），由A产生并由B使用的数据的类型（或格式）必须与B所假定的数据的类型（或格式）一致。
  - **服务。**要使B正确编译和执行，由A提供并由B调用的服务的签名必须与B所假定的一致。
- （2）语义
  - **数据。**要使B正确执行，由A产生并由B使用的数据的语义必须与B所假定的一致。
  - **服务。**要使B正确执行，由A产生并由B调用的服务的语义必须与B所假定的一致。
- （3）顺序
  - **数据。**要使B正确执行，它必须以一个固定的顺序接受由A产生的数据。
  - **控制。**要使B正确执行，A必须在一定的时间限制内执行。

8种类型的依赖性

- (4)A的一个接口的身份
  - A可以有多个接口。要使B正确编译和执行，该接口的名称或句柄必须与B的假定一致。
- (5)A的位置（运行时）
  - 要使B正确执行，A的运行时位置必须和B的假定一致。
- (6)A提供的服务/数据的质量
  - 要使B正确执行，涉及A所提供的数据或服务的质量的一些属性必须和B的假定一致。
- (7)A的存在
  - 要使B正常执行，A必须存在
- (8)A的资源行为
  - 要使B正确执行，A的资源行为必须和B的假定一致。

可修改性战术（续）

- 防止连锁反应的战术
  - **信息隐藏:**
    - 责任分解，定义public的接口。
    - 目的是将变更隔离在一个模块内，防止变更扩散。
    - 与“希望的变更”有很大关系。
  - **限制通信路径**
    - 限制与一个给定模块共享数据的模块。即减少使用由该模块所产生数据的模块的数量，以及产生由该模块所使用数据的模块的数量。

可修改性战术（续）

- 防止连锁反应的战术（续）
  - **维持现有的接口**
    - 如果B依赖于A的一个接口，则维持该接口及其语法能够使B保持不变。
    - 不能解决语义以来问题。也很难屏蔽对服务质量、数据质量、资源使用和资源拥有的依赖性。
    - 通过将接口与实现分离可实现接口的稳定性。如创建屏蔽变化的走向接口
    - **实现该战术的模式有:**
      - 添加接口
      - 添加适配器
      - 提供一个占位程序A

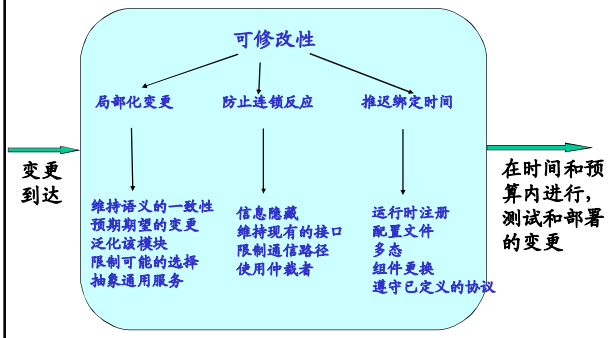
可修改性战术（续）

- 防止连锁反应的战术（续）
  - **仲裁者的使用**
    - 如果B对A具有非语义的任何类型的依赖，则可以在A和B之间插入一个仲裁者，以管理与该依赖相关的活动。**仲裁者可以是:**
      - **数据（语法）**：存储库充当数据生产者和使用者之间的仲裁。可以把A产生的数据转换为符合B的语法。
      - **服务（语法）:** 把服务的语法从一种形式转换为另一种形式的仲裁者，防止A的变化扩散到B。如桥、调停者、策略、代理、工厂模式等。
      - **A的接口的身份**：使用经纪人模式屏蔽一个接口的身份变化。
      - **A的位置（运行时）**：名称服务器可以使A的位置发生变化，且不影响B。
      - **A的资源行为或由A控制的资源:** 资源管理器，条件是A将对该资源的控制转让给资源管理器。
      - **A的存在**：如工厂模式。

可修改性战术（续）

- 推迟绑定时间:
  - 分为开发人员修改和运行时修改。前者会增加系统的不可用时间。
  - 许多战术的目的是在载入时或运行时产生影响:
    - **运行时注册**：即插即用操作，但需要管理注册的额外开销。
    - **配置文件**：在启动时设置参数。
    - **多态**：允许方法调用的后期绑定。
    - **组件更换**：允许载入时间绑定。
    - **遵守已定义的协议**：允许独立进程的运行时绑定。

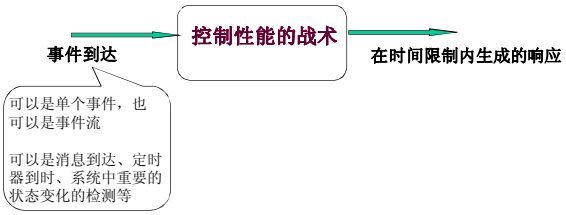
（二）可修改性战术总结



北京航空航天大学软件工程研究所

（三）性能战术

- 性能战术的目标：
  - 对在一定时间限制内到达系统的事件生成一个响应



北京航空航天大学软件工程研究所

性能战术（续）

- 影响响应时间的两个基本因素
  - 资源消耗：
    - 包括CPU、数据存储、网络通信带宽和内存，以及设计中系统特定的实体。
  - 闭锁时间
    - 资源争用：
      - 对同一个资源的争夺，会增加等待时间。
      - 取决于如何仲裁争用以及仲裁机制如何处理单个请求
    - 资源的可用性
      - 资源离线、组件故障等会导致资源不可用－－增加等待时间
    - 对其他计算的依赖性
      - 可能导致等待
- 三类战术：资源需求、资源管理、资源仲裁

北京航空航天大学软件工程研究所

性能战术（续）

- 资源需求：
  - 事件流时资源需求的源。需求的两个特性：
    - 资源流中的事件之间的时间（在事件流中多长时间进行一次请求）
    - 每个请求所消耗的资源数量
  - 减少等待时间的战术之一：减少处理一个事件流所需要的资源。方法：
    - 提高计算效率：关键算法
    - 减少计算开销。如删除仲裁者－可修改性/性能的权衡问题
  - 减少等待时间的战术之二：减少所处理事件的数量。方法：
    - 管理事件率：如取样频率
    - 控制采样频率
  - 用于减少或管理需求的其他战术包括控制资源的使用
    - 限制执行时间
    - 限制队列大小

北京航空航天大学软件工程研究所

性能战术（续）

- 资源管理：如果不能降低对资源的需求，有效的资源管理，也可以影响响应时间。
  - 引入并发：
    - 如果可以并行处理请求，可减少闭锁时间
    - 适当地将线程分配给资源（负载平衡），尽可能利用并发
  - 维持数据或计算的多个副本
    - 客户机-服务器中客户机是计算的副本，减少在中央服务器上进行所有计算时出现的争用；
    - 高速缓存是在不同速度的存储库或单独的存储库上复制数据的战术，目的减少争用。需要解决副本的一致和同步问题。
  - 添加可用资源
    - 进行成本/性能的权衡分析

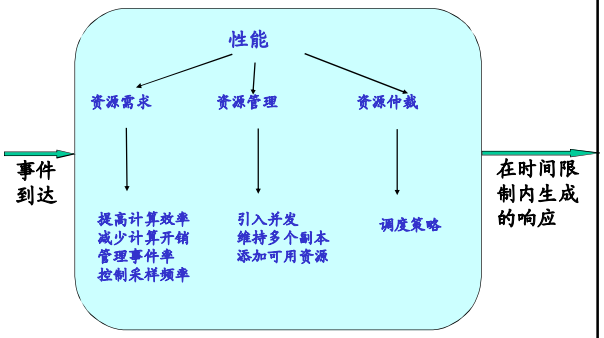
北京航空航天大学软件工程研究所

性能战术（续）

- 资源仲裁：理解资源特性，合理调度资源
  - 调度策略包括：优先级分配和分派
  - 调度的竞争标准：
    - 最佳资源分配、请求重要性、最小化所使用资源的数量、使等待时间最少、使吞吐量最大、防止资源缺乏以保证合理性、
  - 常见调度策略：
    - 先进/先出：同等对待，排队模型。常见、缺点。
    - 固定优先级调用：静态分配优先级、根据优先级顺序分配资源。3个常见的优先级策略：语义重要性、时限时间单调、速率单调。
    - 动态优先级调用：轮转；时限时间最早优先。
    - 静态调度：离线确定先占点和资源分配顺序。

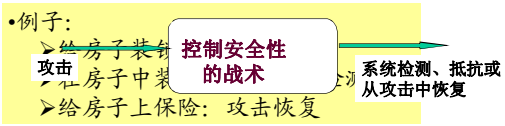
北京航空航天大学软件工程研究所

性能战术总结



（四）安全性战术

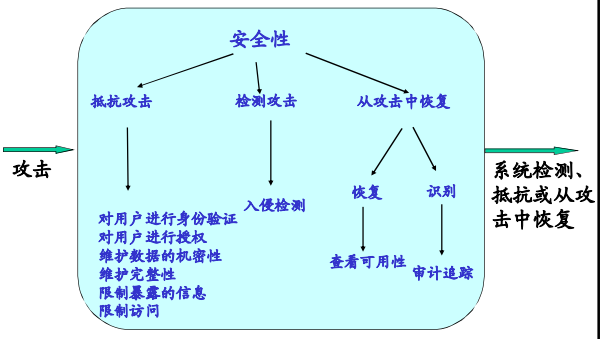
- 安全性战术的目标:
  - 从系统检测、抵抗或从攻击中恢复过来
- 三类安全性战术
  - 与抵抗攻击有关的战术
  - 与检测攻击有关的战术
  - 从攻击中恢复有关的战术



安全性战术（续）

- 抵抗攻击
  - 对用户进行身份验证
  - 对用户进行授权
  - 维护数据的机密性：对数据和通信链路进行加密
  - 维护完整性
  - 限制暴露的信息
  - 限制访问：如防火墙
- 检测攻击：入侵检测系统
- 从攻击中恢复
  - 与恢复状态相关的战术：如可用性战术中的恢复战术
  - 与识别攻击者相关的战术：审计追踪（预防性或惩罚性目的）

安全性战术总结



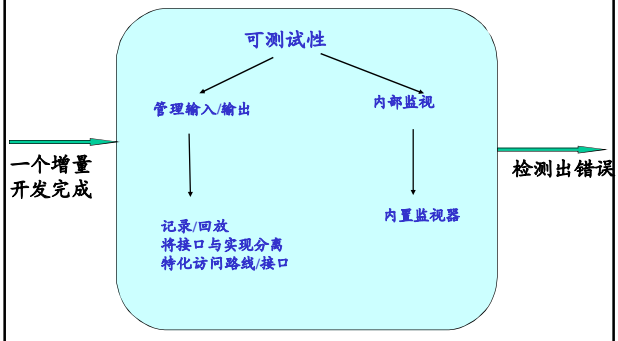
（五）可测试性战术

- 目标:
  - 允许在完成软件开发的一个增量后，较轻松地对软件进行测试

（五）可测试性战术

- 提供输入并捕获输出
  - **记录/回放**：捕获跨接口的信息，作为测试专用软件的输入，用于比较测试输出。
  - **将接口与实现分离**：允许实现替代，支持各种测试目的。如占位测试其他部分等。
  - **特化访问路线/接口**：具有特化的测试接口允许通过测试工具开独立于其正常操作，来捕获或指定组件的变量值。
- 内部监视
  - 内置监视器：维持状态、性能负载、容量、安全性或其他可通过接口访问的信息。会增加测试的工作量。

可测试性战术总结



（六）易用性战术

- 易用性与用户完成期望任务的容易程度以及系统为用户提供的支持种类有关。
- 两类战术：
  - 运行时：在系统运行期间支持用户的战术
  - 基于用户接口设计的迭代特性，在设计时支持接口开发人员

（六）易用性战术（续）

- 运行时战术：系统和用户的交互
  - “**用户主动**”：设计师设计响应，和实现其他功能一样
  - “**系统主动**”：依赖于关于用户的信息（一个模型），如用户执行的任务或直通本身的状态。是确定系统用来预测其自身行为或用户意图的模型的战术。
    - **维持任务的一个模型**：维护关于任务的信息。用于确定上下文，以使系统了解用户试图做什么，并提供各种协助。如句子的第一个字母大写。
    - **维持用户的一个模型**：维护关于用户的信息。如用户在期望的响应时间方面的行为，或特定用户的某些方面。
    - **维持系统的一个模型**：维护关于系统的信息。确定了希望的系统行为，以便为用户提供适当的反馈。
  - “**混合主动**”

（六）易用性战术（续）

- 设计时战术：测试和集成中对接口的修改。支持接口开发人员
  - 将用户接口与应用的其余部分分离开，常用的体系结构模式：
    - 模型－视图－控制器
    - 表示－抽象－控制
    - Seeheim
    - Arch/Slinky

第二部分　创建软件体系结构

- 理解和实现质量属性
  - 软件系统的质量属性
  - 软件体系结构与质量属性
  - 软件体系结构设计的原子操作
  - 软件体系结构战术
  - ☞ 软件体系结构模式和风格
- 软件体系结构案例分析
  - KWIC System
  - WWW
  - CORBA