第一章：概论

一、软件危机

**什么是软件危机**?

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

**第一次：危机表现**：1、软件生产效率低下，成本高；开发成本和进度的估计常常很不准确。

2、软件产品质量差：用户对“已完成的”软件系统不满意的现象经常发生。

软件产品难以（不能）维护：软件通常没有适当的文档资料。

**危机根源**：1、人员知识结构单一。

2、手工作坊：code and fix。

3、理论体系不成熟【软件开发】：语言、工具、过程、方法、质量保证……

4、软件开发管理不完善：规划、预算、可行性分析、风险预测和控制。

**第二次：危机表现**：1、项目经常开发出有错误的产品。

2、项目经常开发出低质量的产品。

3、项目经常延迟。

4、程序员工作时间很长。

5、项目组经常违约。

6、开发过程没有乐趣。

**危机根源**：假设在开发之前可以充分理解需求。与用户的交流就会仅仅在开发开始和结束的时候发生。但不幸的是这个假设几乎从不成立

**第三次：危机表现**：1、变更/演化/重用（质量、效率、成本）

2、知识产权问题、安全问题

**危机根源**：1、大量遗产系统的挖掘和重用

2、大量开源软件/代码的使用

3、普遍存在的动态/不确定需求

二、软件体系结构的兴起和发展

**概念**：软件体系结构由三种基本元素组成：构件、连接件、配置/规约。

构件： 构件是计算或数据存储的单位，对应于常规编程语言的编译单元和其他用户级对象。它可以是简单型或复合型，例如一个系统就可以看成是复合型构件。构件作为一个封装的实体，只能通过其接口与外部交互，构件的接口由一组端口组成，每个端口表示了构件与外部的交互点。

连接件：连接件是构件之间的交互及控制这些交互的规则。它们有时是简单交互，如过程调用，共享变量访问；有的则是复杂的和寓意为丰富的交互，，如C/S协议等。连接件通常不能与编译单元各自相对应。他们的表现形式可以是表项、缓冲区、动态数据结构、调用、初始化参数等。

配置：可以看做是“约束/拓扑结构”。软件体系结构的拓扑结构反映了该结构的基本准则。

意义： （1）体系结构是风险承担者进行交流的手段，代表了系统的公共的高层次的抽象。

（2）体系结构是早期设计决策的体现：明确了对系统实现的约束条件、决定了开发和维护组织的组织结构、制约系统的质量属性、可以通过体系结构预测软件质量、使推理和控制更改更简单、有助于顺序渐进的原型设计、可以作为培训的基础。

（3）体系结构是可传递和可重用的模型：体现了一个相对来说比较小又可理解的模型、重用意味着决策在具有相似需求的多个系统中发生影响，比代码级的重用要有更大好处。

三、软件重用

需求： （1）更快：软件必须满足市场需求，其他竞争对手的存在导致你必须有更快的开发速度。

（2）更好：软件必须完成它服务的进程的需求并且在服务过程中必须只允许很少的失败。

（3）更便宜：生产和获取软件的成本都要变得更小。

**层次**：（1）代码重用：包括目标代码和源代码的重用。目标代码重用级别最低，源代码略高。

（2）设计重用：设计结果比源程序的抽象级别更高，因此它的重用受实现环境的影响较少，从而使可重用组件被重用的机会更多，并且所需的修改更少。

（3）需求重用：比设计结果更高级别的重用，可重用的分析组件是针对问题域的某些事物或某些问题的抽象程度更高的解法，受设计技术及实现条件的影响很少，所以可重用的机会更大。

**效益**：使用软件重用技术可以减少软件开发活动中大量的重复性工作，这样就能提高软件生产率，降低开发成本，缩短开发周期。同时，由于软构件大都经过严格的质量认证，并在实际运行环境中得到校验，因此，重用软构件有助于改善软件质量。此外，大量使用软构件，软件的灵活性和标准化程度也可望得到提高。

第二章：软件体系结构模型

一、软件体系结构建模概述

软件模型的类型：

结构模型：用组件、连接件和其他概念来刻画结构，力图通过结构来反映系统的重要语义内容

框架模型：与结构模型类似，但更侧重于整体的结构。

动态模型：是对结构模型和框架模型的补充，研究系统的“大颗粒”的行为性质。动态可以指系统总体结构的配置、建立或扯出通信通道或计算的过程。

过程模型：研究构造系统的步骤和过程，因而结构是遵循某些过程脚本的结果。

功能模型：该模型认为软件体系结构是由一组功能组件按层次组成，下层向上层提供服务。

二、4+1视图模型

逻辑视图：**视角**：主要是系统功能需求，即系统提供最终用户的服务。

**特点**：系统分解成一系列的功能抽象。风格为面向对象的风格，其主要的设计准则是是试图在整个系统中保持单一的、连贯的对象模型。

画法：在面向对象技术中，通过抽象、封装和继承，可以用对象模型来代表逻辑视图，用类图来描述逻辑视图。标记方法：Booch标记法

开发视图：**视角**：关注软件开发环境下实际模块的组织。

**特点**：侧重于软件模块的组织和管理。风格使用层次式风格。每层均有良好的职责、设计规则，即某层子系统依靠同层次的或更低层次的子系统，从而减少了具有复杂模块依赖关系的网路开发量，得到层次式的简单发布策略。对于各个层次来说，层次越低，通用性越强，以保证需求变更时，所做的改动最小。

画法：可以用模块和子系统图来表达。

进程视图：**视角**：侧重于系统的运行特性，关注一些非功能性的需求，例如系统的性能和可用性。

**特点**：强调并发性、分布性、系统集成性和容错能力。

物理视图：**视角**：主要考虑如何把软件映射到硬件上。

**特点**：通常要考虑到系统性能、规模、可靠性等；解决系统拓扑结构、系统安装、通信等问题。

场景：可以看做重要系统活动的抽象，使4个视图有机联系起来。可以帮助架构师找到体系结构的构架和他们之间的作用关系。有两点作用：1、在SA设计过程中，作为一个驱动来发现SA元素 2、在SA设计结束时，从书面上作为对SA原型测试的起点。

画法：场景的设计用对象场景图和对象交互图来表示。

4+1视图模型的作用：软件体系结构的5种模型各有所长，4+1视图模型将5种模型有机地统一在一起，形成一个完整的模型来刻画软件体系结构

三、软件体系结构的核心模型

**SA核心模型的五种元素：组件、连接件、配置、端口、角色。**

组件：具有某种功能的可重用的软件模块单元，表示了系统中主要的计算单元和数据存储。组件有两种，复合组件和原子组件。复合组件由其他复合组件和原子组件通过连接而成。

连接件：表示了组建之间的交互，简单的连接件有：管道、过程调用、事件广播等。复杂的连接件有：客户－服务器通信协议，数据库和应用之间SQL连接等。

配置：表示了组件和连接件的拓扑逻辑和约束。

端口：组件作为一个封装的实体，只能通过其接口与外部交互，组件的接口由一组端口组成，每个端口表示了组件和外部环境的交汇点。通过不同的端口类型，一个组件可以提供多重接口。

角色：连接件作为建模软件体系结构的主要实体，同样也有接口，连接件的接口由一组角色组成，每个角色定义了改连接件表示的交互的参与者。

四、**软件体系结构的生命周期**

1、软件体系结构非形式化描述：自然语言描述；创造性和开拓性。

2、软件体系结构的规范化描述和分析：形式化数学理论模型描述，分析SA性质。

3、软件体系结构的求精及其验证：抽象到具体，逐步求精：验证判断具体SA与抽象SA语义一致性，并实现抽象SA。

4、软件体系结构的实施：将精化的SA实施于系统的设计中，并将SA的组件和连接件等有机的组织在一起，形成系统设计的框架，以便据此实施于软件设计和构造中。

5、软件体系结构的演化和扩展：在实施SA时，根据系统的需求（常常是非功能的需求，如性能、容错、安全性、互操作性、自适应性等非功能性质）来扩展和改动SA，这称为SA的演化。

6、软件体系结构的提供、评价和度量：定性评价和定量度量，以利于对SA的重用。

7、软件体系结构的中介：SA进行多次演化、修改变得难以理解，不能适应系统的发展。

第三章：软件体系结构风格

一、软件体系结构风格概述

软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用范型。

主要内涵：1、定义了一个系统家族，即一个体系结构定义一个词汇表和一组约束。

2、词汇表中包含一些构件和连接件类型。

3、约束指出系统是如何将这些构件和连接件组合起来的。

体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效的组织成一个完整的系统。

（1）提供一个词汇表

（2）定义一套配置规则

（3）定义一套语义解释原则

（4）定义对基于这种风格的系统所进行的分析

二、**经典风格**

1、管道-过滤器风格

每个功能模块都有一组输入和输出，功能模块称作过滤器，功能模块间的连接可以看作输入、输出数据流之间的通路，所以称作管道。

特点：过滤器的相对独立性：完成自身功能，相互之间无需进行状态交互。（自身无状态也不和非相邻的过滤器共享状态，对相邻的过滤器不施加任何限制）

结果的正确性不依赖于各个过滤器运行的先后次序。

优点：

软构件具有隐蔽性、高内聚、低耦合

设计者可以将整个系统的输入、输出特性简单地理解为各个过滤器功能的合成。（将问题分解，化繁为简）

支持功能模块的复用。（任何两个过滤器只要传送的数据遵守相同的规约就可以相连接）

具有较强的可维护性和可扩展性。（旧过滤器的替代、为增强功能添加新过滤器都很方便）

支持特定的分析，比如吞吐量计算和死锁检测。

具有并发性。（每个过滤器只要具备输入就可以与其他过滤器并发地执行）

缺点：

交互式处理能力弱。（整个系统没有共享的数据区，导致用户操作某项数据困难）

改进方法：对每个过滤器增加相应的用户控制接口。

导致进程成为批处理的结构

系统性能下降，且增加编写过滤器的复杂性

适用范围：

适用于数据流的处理和交换，不适合为与用户交互频繁的系统建模

2、数据抽象和面向对象风格

将逻辑上的实体映射为对象，实体之间的 关系映射为对象之间的应用关系。

对象利用应用关系来访问对方公开的接口， 完成某个特定任务;一组对象之间相互协 作，完成总体目标。

优点：

高度模块性。（数据与其相关操作被组织为对象）

封装功能。（一组功能和其实现细节被封装在一个对象中）

代码共享。（对象的相对独立性可被反复重用）

灵活性。（对象在组织过程中相互关系可以任意变化）

易维护性。（对象接近于人对问题的思维方式，易于理解和修改）

缺点：

最大的不足在于如果一个对象需要调用另一个对象，其必须知道该对象的标识，这样会增强对象之间的依赖关系。当修改一个对象时将不得不修改所有与他有调用关系的对象。

适用范围：

面向对象的体系结构模式适用于数据和功能分离的系统中，同样也适合于问题域模型比较明显，或需要人机交互界面的系统。大多数应用事件驱动风格的系统也常常应用了面向对象风格。

3、事件驱动风格

组件不直接调用一 个过程，而是触发或广播一个或多个事件。

特点：

系统由若干子系统或元素所组成

系统有一定的目标，各子系统在某一种消息机制的控制下为了这个目标而协调行动。

在一个系统的若干子系统中，必定有一个起着主导作用，其他处于从属

任一系统和系统内的任一元素都有一个事件收集机制和一个事件处理机制，通过这种机制与周围环境发生作用和联系。

事件的触发者并不知道哪些组件会被这些事件影响。

优点：

非常适合于描述系统族，属于同一族的任何系统中的高级管理子系统的描述类似，便于重用
容易实现并发处理和多任务操作
具有良好的可扩展性，只需为某个对象注册一个事件处理接口就可以将该对象引入整个系统

缺点：

组件放弃了对系统计算的控制，不能确定其他组件是否会响应它，也不能保证过程调用顺序
数据交换存在问题。有时可以靠事件传递，但有时需要共享的资源库。
过程的语义必须依赖于事件的上下文约束，导致正确性的推理存在问题。

适用案例：

在编辑器中支持语法检查——触发报警

4、层次系统风格

每一层都要为上层提供服务同时作为下层的客户调用下层提供的功能函数。

优点：

支持系统设计过程中的逐级抽象，允许将复杂问题分解成一个增量步骤序列。
具有较好的可扩展性
支持软件复用

缺点:

一些系统不易被分层，即使是层次化的系统，也可能因为性能等因素而把低级和高级的功能综合起来
很难找到一个合适的、正确的层次抽象方法

5、仓库风格

两个组件，中央数据结构说明当前状态，独立组件在中央数据存贮上执行
两类控制策略，传统数据库，黑板

优点：

解决问题的多方法性
具有可更改性和可维护性（知识源相互独立，通信通过黑板来完成）
有可重用的知识源
支持容错性和健壮性（所有结果都是假设的，只有被数据和其他假设强烈支持的才能生存）

缺点：

测试困难（系统的执行没有确定的顺序，结果的可再现性比较差）
不能保证有好的求解方案（提供给客户的往往是解决问题的一个百分比而不是最优解）
效率低
开发成本高（需要用几年的时间来进化）
缺少对不并行机的支持（黑板模式要求中心数据同步并发访问）

6、C/S风格

定义了工作站如何与服务器相连，以实现数据和应用分布到多个处理机上。
三个主要组成部分：数据库服务器、客户应用程序、网络。

优点：

强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受
对硬件和软件的变化显示出极大的适应性和灵活性，易于对系统进行扩充和缩小。
功能构件充分隔离，节约大量成本

缺点：

二层C/S结构是单一服务器所以难以扩展至大型企业广域网或Internet

软硬件的组合及集成能力有限。

客户端程序设计复杂，客户机负荷过重

数据安全性不好

软件维护或升级时，每个客户端都要升级。

7、三层C/S风格

分为表示层、功能层和数据层。表示层为应用的用户接口部分。负责应用与用户的交互。功能层相当于应用的本题，将具体的业务处理逻辑编入。数据层负责管理数据库。

优点：

允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，提高系统和软件的可维护性和可扩展性。

允许更灵活有效的选用相应的平台和硬件系统，使之在处理负荷能力上与处理特性上分别适应于结构清晰的三层;并且这些平台和各个组成部分可以具有良好的可升级性和开放性。

各层可以并行开发，节约时间，各自选择最适合的开发语言。

用功能层隔开表示层和数据层使得数据更加安全

缺点：

对各层间的通信效率要求较高，否则会影响性能。

对设计要求较高，必须慎重考虑三层间的通信方法、通信频度及数据量。

8、B/S风格

浏览器/服务器风格为三层C/S结构的一种实现方式，其具体结构为：浏览器/Web服务器/数据库服务器。

优点：

系统安装、修改和维护全在服务器端解决，用户仅需要一个浏览器作为“客户端”。

提供异种机、异种网、异种应用服务的联机、联网、统一服务的最现实的开放性基础

缺点：

系统扩展能力差，安全性难以控制。

在数据查询等响应速度上远远低于C/S体系结构。

B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理(OLTP)应用。

9、B/S、C/S混合软件体系结构

内外有别模型：

企业内部用户通过局域网直接访问数据库服务器，软件系统采取C/S体系结构，企业外部用户通过Internet访问Web服务器，通过Web服务器再访问数据库，采用B/S结构。

优点：

外部用户不直接访问数据库服务器保证了安全性，内部用户由于交互操作较多采用直接访问保证响应速度较快。

缺点：

外部用户修改和维护数据，速度较慢，较繁琐。

查改有别模型：

需要执行修改和维护数据操作就使用C/S体系结构，需要查询浏览操作就使用B/S结构。

优点：此结构拥有C/S和B/S的共同优点

缺点：

企业数据容易暴露给外部用户，数据安全有一定威胁。

三、**SOA架构**（面向服务的体系结构）

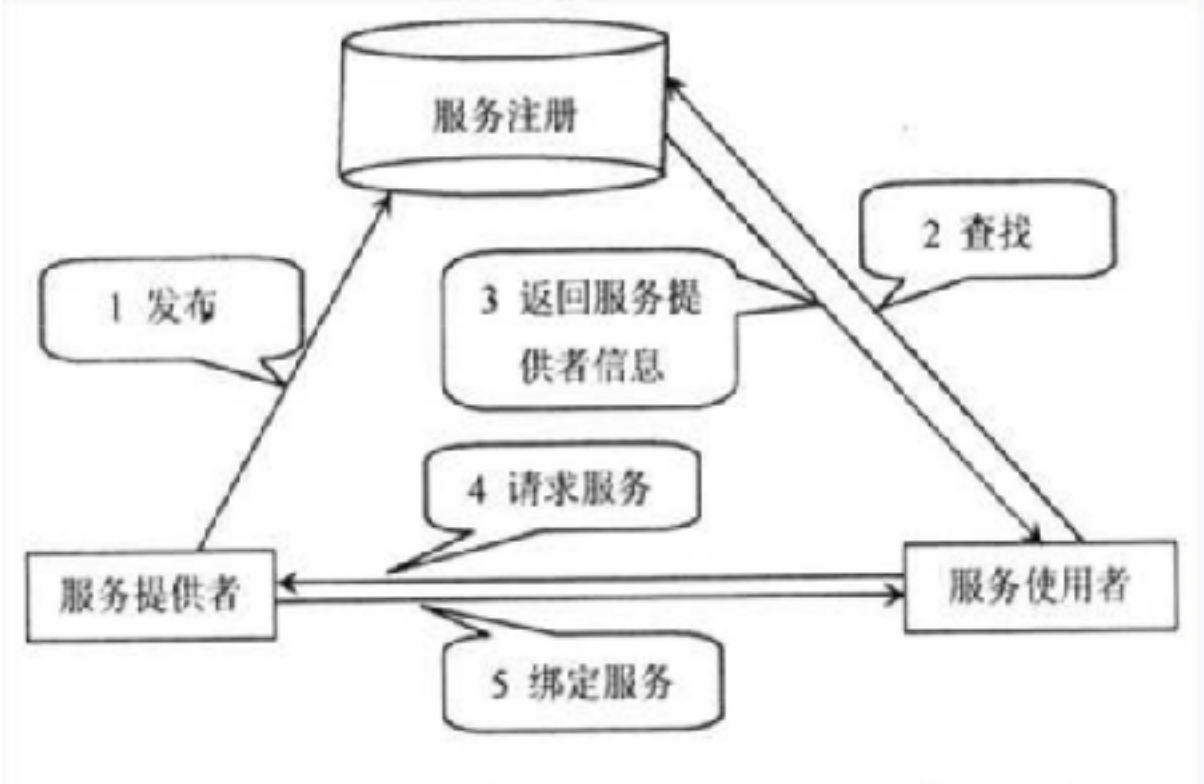
角色：

服务提供者：发布自己的服务并对服务请求进行相应。

服务注册中心：注册已经发布的web service，对其进行分类，并提供搜索服务。

服务请求者：利用服务中心查找所需要的服务，然后使用该服务。

工作流程：




操作：

发布：为了使服务可访问，需要发布服务描述以使服务使用者可以发现它。

查找：服务请求者定位服务，方法是查询服务注册中心来找到满足其标准的服务。

绑定：检索到服务描述之后，服务使用者根据服务描述中的信息来调用服务。

协议：

SOAP：简单对象访问协议，作为传输层用来在消费者和服务者之间传送消息。

WSDL：Web服务描述语言，用来描述服务

UUDI：统一描述、发现和集成，用来注册和查找服务。

核心思想：

标准化封装：SOA通过标准的、支持Internet、与操作系统无关的SOAP协议实现连接互操作。

复用：复用对象包括子程序，组件，企业对象组件，服务

耦合性：低耦合?

第四章：基于SA的开发

一、设计模式

作用：1、模式是前人经验的总结，它使人们可以方便地复用成功的设计和体系结构

2、遇到特定类型的问题，采用模式可以降低分析、设计和实现的难度，另外可以使系统具有更好的可复用性和灵活性。

3、一个软件体系结构的模式描述了一个出现在特定设计语境中的特殊的再现设计问题，并为它的解决方案提供了一个经过充分验证的通用图示。

MVC模式：

将建模、显示和操作分为三种独立的类：

模型：用于管理应用程序域的行为和数据，并响应请求。

视图：管理信息的显示。

控制器：解释用户的键盘和鼠标输入，通知模型和视图进行相应的更改。

优点：支持多视图、适应更改。

缺点：复杂性、频繁更新导致成本增加

模式和体系结构：

模式作为体系结构构造块（使用特定的面向问题技术对体系结构技术进行有效补充）

构造异构体系结构（将多个模式组织成模式系统，完成完整的体系结构构造）

方法（模式提供了解决一个问题的方案）

在设计层次模式只需要适当的抽象机制，因此可以使用任意的变成语言来实现模式

二、中间件技术

功能：

负责客户机和服务器之间的连接和通信，以及客户机和应用层之间的高效率通信机制

提供应用层不同服务之间的互操作机制，以及应用层和数据库之间的连接和控制机制

提供一个多层体系结构的应用开发和运行的平台，支持模块化应用开发

屏蔽硬件、操作系统、网络和数据库差异

提供应用的负载均衡和高可用性、安全机制和管理功能

提供一组通用的服务，避免重复的工作和使应用之间可以协作

应用：

提供了应用系统基本的运行环境，也为应用系统提供了更多的高级服务功能

三、基于体系结构的软件设计方法

步骤：

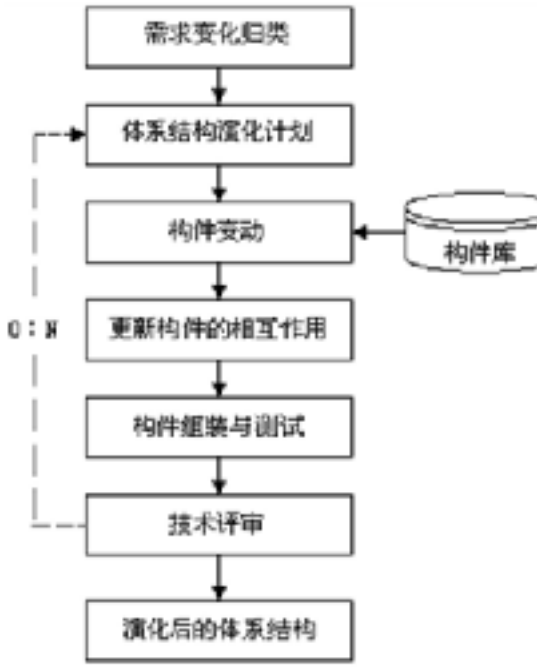
1、功能分解

2、选择体系结构风格

3、为风格分配功能（为产生的构件类型分配功能）

4、细化模板
5、功能校验
6、创建并发视图
7、创建配置视图
8、验证质量场景
9、验证约束

设计和演化过程：
实验原型阶段：重点是获得对问题域的理解，为此需构建一系列原型。
第一个开发周期：没有具体、明确的目标。结束时形成两个版本：图形用户界面的初始设计和问题域模型。
第二个开发周期：设计一个正交软件体系结构
演化开发阶段：重点是最终产品的开发，也就是构件的精确化
1、需求变更归类
2、制订体系结构演化计划
3、修改、增加或删除构件
4、更新构件的相互作用
5、产生演化后的体系结构
6、迭代
7、对以上步骤进行确认，进行阶段性技术评审
8、对所做的标记进行处理




第五章：软件体系结构评估
一、软件体系结构评估概念
**相关概念**：
敏感点：一个或多个构件的特征，可以使设计师搞清楚实现质量目标时应该注意什么
权衡点：影响多个质量属性的特征，是多个质量属性的敏感点，需要进行权衡
风险承担者：即涉众、牵涉到的人。包括体系结构设计师、开发人员、维护人员、集成人员、测试人员、标准专家、性能工程师等。
场景：从风险承担者的角度对与系统的交互的简短描述。在体系结构评估中，一般从刺激（怎样引发交互）、环境（发生时的状态）、响应（系统的反应）三方面来描述

**评估的作用**：
1、可以减少后期的测试和纠错的开销
2、评估是挖掘隐形需求并将其补充到设计的最后机会

评估方法种类：
基于调查问卷或检查表的评估方式：这一评估方式比较自由灵活，可评估多种质量属性，也可以在软件体系结构设计的多个阶段进行

基于场景的评估方式：分析对场景也就是对系统的使用或修改活动的支持程度

基于度量的评估方式：度量是指为软件产品的某一属性所赋予的数值，如代码行数、方法调用层数、构件个数等

| 评估方式 | 调查问卷或检查表 | | 场景 | 度量 |
|---|---|---|---|---|
| | 调查问卷 | 检查表 | | |
| 通用性 | 通用 | 特定领域 | 特定系统 | 通用或特定领域 |
| 评估者对体系结构的了解程度 | 粗略了解 | 无限制 | 中等了解 | 精确了解 |
| 实施阶段 | 早 | 中 | 中 | 中 |
| 客观性 | 主观 | 主观 | 较主观 | 较客观 |

二、ATAM评估方法（体系结构权衡分析方法）

步骤：

1、陈述

①ATAM方法（评估负责人）：评估负责人向参加会议的相关人员介绍ATAM方法。在这 一步，要对每个人解释参与的过程，使每个人都知道要 收集哪些信息，如何描述信息，将向谁报告等等。

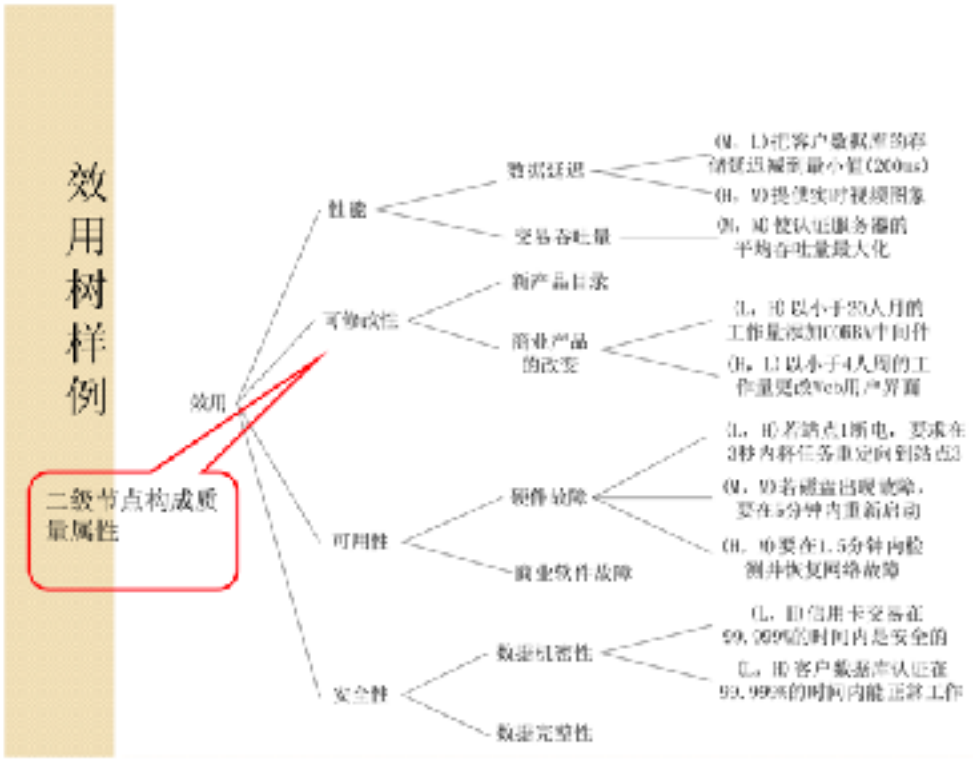
②商业动机（项目经理或系统客户）：项目经理从业务角度，向相关人员介绍系统概况

③SA（系统设计人员）：架构师在适合的细节层次上描述体系结构。这些体系结构信息直接影响可能的分析及分析的质量。

2、调查与分析

④确定体系结构方法：体系结构方法定义了系统的重要结构，描述了系统演化、对更改的响应、对攻击的防范以及与其他系统的集成等

⑤生成质量属性效用树：评估小组与项目决策者合作，共同确定出该系统的最重要的质量属性目标，并设置优先级，进行进一步的细化，该步指导其他的分析



⑥分析体系结构方法：针对划分了优先级的质量需求(第5步)和采用的体系结构方法(第4步)，评估它们的匹配情况
⑦集体讨论并确定场景优先级：测试所理解的的体系结构，场景被用作测试的主要手段，第五步主要是从体系结构设计人员的角度看质量属性需求，这一步从相关人员的角度讨论场景
⑧分析体系结构方法：对新增的场景，分析其体系结构方法;对不变的场景，进行 检查
⑨陈述结果：把在ATAM分析中所得到的各种信息进行归纳总结，并呈现给相关人员

三、SAAM评估方法
1、场景的形成：集体讨论形成场景，全面捕捉系统的主要用途、系统用户类型、系统将来可能的变更、系统在当前及可预见未来必须满足的质量属性
2、体系结构的表述：描述系统给的静态特征和动态特征，第一步和第二步之间反复促进，反复迭代。
3、场景的分类和优先级的确定：分为直接场景（不需对体系结构进行修改）和间接场景，划分优先级，考察重要场景
4、对场景的单个评估：场景与体系结构描述对应起来。对直接场景，说明体系结构如何执行；对间接场景，说明更改及其代价。
5、场景相互作用的评估：相互作用（多个间接场景要去更改同一个构件），体现体系结构设计中的功能分配，暴露体系结构文档未充分说明的结构的分解
6、形成总体评估：为场景设置权值，考察体系结构支持的直接场景数量