一、什么是软件系统结构

软件体系结构也称为软件构架（有时简称构架），是系统的一个或多个结构，它包括：软件的组成元素（组件），这些元素（组件）的外部可见特性，以及这些元素（组件）之间的相互关系。

含义：

(1) 系统由一个或多个结构组成，其中任何一个结构并不能与构架等同。

(2) 每个系统都有一个体系结构。

(3) 软件体系结构是系统的抽象。

(4) 构架定义了软件元素以及各元素间的交互关系。

(5) 以往作为体系结构传递的线框图，事实上并等同于体系结构。

二、构架商业周期 (ABC)

1.构架由什么决定 ？

构架是否由系统需求决定？×

软件构架是技术、商业和社会因素共同作用的结果。

2. 构架从哪里来？(影响构架的因素)

影响构架的因素主要包括：

- ❑ 系统涉众（stakeholder）、主要有：

  管理者：成本要低，人人都得干活

  营销人员：特性突出、投放市场快、成本低、可与同类产品相匹敌。

  终端用户：行为、性能、安全性、可靠性、易用性。

  维护人员：可修改性强。

  客户：成本低、及时交付、不要频繁修改。
- ❑ 开发组织
  - ・组织内对现存构架的重用
  - ・对某个基础设施进行长期的商业投资以实现某些战略目标
  - ・开发组织本身的机构也会影响构架的形成
- ❑ 构架师的素质和经验

  构架师先前的一些经验、教育、培训以及所接触到过的成功构架模式都会影响到他们对某种构架的选择。
- ❑ 技术环境

  当前技术发展水平代表了某个时代的构架师的普遍素质和经验，对架构有很大的影响力。
- ❑ 其它因素

  其它如社会、法律、人文环境等都会对构架产生影响。

3.构架的反影响力

- ・构架会影响开发组织的结构
- ・构架会影响开发组织的目标
- ・构架会影响客户对下一个系统的要求
- ・构建系统的过程丰富了整个开发团队的经验，从而将影响设计师对后继系统的设计
- ・一些系统会影响并实际改变软件工程的环境，也就是系统开发人员学习或实践的技术环境。

4.构架的商业周期

软件构架是技术、商业和社会等诸多因素作用的结果，而软件构架的存在反过来又会影响技术、商业和社会环境，从而影响未来的软件构架。我们把这种相互影响的周期--从环境到软件构架又返回到环境--称作软件构架商业周期。

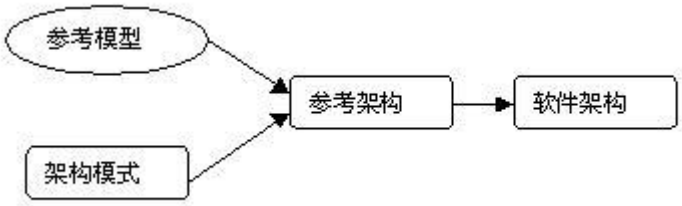
三、架构模式、参考模型、参考架构

1、架构模式是对元素和关系类型以及一组对其使用方式的限制的描述。

2、参考模型是一种考虑数据流的功能划分。

3、参考架构是映射到软件元素（它们相互协作，共同实现在参考模型中定义的功能）及元素之间数据流上的参考模型。

4、软件架构、架构模式、参考模型、参考架构之间的关系



5、软件架构的重要性

(1)、架构是涉众进行交流的手段。

绝大多数系统涉众都借助软件体系结构来进行彼此理解、协商、达成共识或者相互沟通。

(2)、架构是早期设计决策的体现。

构架设计是在所开发系统的最早时间点，明确对系统实现的约束条件、决定开发组织的组织结构、影响质量属性的实现等。是系统最早期设计决策的体现，它们对软件系统的后续开发、部署和维护具有相当重要的影响。

(3)、架构是可传递、可重用的模型。

软件构架是关于系统构造以及系统各个元素工作机制的相对较小、却又能够突出反映问题的模型。这种模型可以在多个系统之间传递，特别是可以应用到具有相似质量属性和功能需求的系统中，并能够促进大规模软件的系统级复用。

四、架构的结构

架构定义中指出系统由多种结构构成的，下面列出一些常见的结构。

软件结构		关系	适用环境
模块结构	分解	是一个子模块；与之共享秘密	资源分配、项目结构化和规划；信息隐藏、封装；配置控制
	使用	模块之间的调用	设计子集；设计扩展
	分层	只允许相邻两层之间调用模块、使用服务、提供服务等	增量式开发；在“虚拟机”可移植性之上实现系统
	类	特化：由类创建对象或子类继承基类	在面向对象的设计系统中，从一个公共的模版中产生快速的、相近的

		泛化：从许多对象中抽取共同特征和行为，构成类	实现

软件结构		关系	适用环境
组件—连接器结构	客户机-服务器	与之通信；依赖于	分布式操作；关注点分离；性能分析；负载平衡
	进程	与之并发运行、可能会与之并发运行；排除；优先于等	调度分析；性能分析
	并发	在相同的逻辑线程上运行	确定存在资源争用，线程可以交叉、连接、被创建或被杀死的位置
	共享数据	产生数据；使用数据	性能；数据完整性；可修改性
分配结构	部署	分配给；移植到	性能、可用性、安全性分析
	实现	存储在	配置控制、集成、测试活动
	工作分配	分配到	项目管理、最佳利用专业技术、管理通用性

五、软件体系结构几种建模方法

1. 结构模型

这是一个最直观、最普遍的建模方法。这种方法以体系结构的构件、连接件和其他概念来刻画结构，并力图通过结构来反映系统的重要语义内容，包括系统的配置、约束、隐含的假设条件、风格、性质等。

研究结构模型的核心是体系结构描述语言。

2. 框架模型

框架模型与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。

框架模型主要以一些特殊的问题为目标建立只针对和适应该问题的结构。

3. 功能模型

功能模型认为体系结构是由一组功能构件按层次组成，下层向上层提供服务。

功能模型可以看作是一种特殊的框架模型。

4. 动态模型

动态模型是对结构或框架模型的补充，研究系统的“大颗粒”的行为性质。

例如，描述系统的重新配置或演化。动态可以指系统总体结构的配置、建立或拆除通信通道或计算的过程。

5. 过程模型

过程模型研究构造系统的步骤和过程。

六 “4+1”模型

示意图：

含义：“4+1”视图模型从 5 个不同的视角－逻辑视图、进程视图、物理视图、开发视图和场景视图来描述软件体系结构。每一个视图只关心系统的一个侧面，5 个视图结合在一起才能反映系统的软件体系结构的全部内容。

1. 逻辑视图

逻辑视图主要支持系统的功能需求，即系统提供给最终用户的服务。在逻辑视图中，系统分解成一系列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素。

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。

逻辑视图中使用的风格为面向对象的风格，逻辑视图设计中要注意的主要问题是要保持一个单一的、内聚的对象模型贯穿整个系统。

2. 开发视图

开发视图也称模块视图，主要侧重于软件模块的组织和管理。

开发视图要考虑软件内部的需求，如软件开发的容易性、软件的重用和软件的通用性，要充分考虑由于具体开发工具的不同而带来的局限性。

开发视图通过系统输入输出关系的模型图和子系统图来描述。

在开发视图中，最好采用 4-6 层子系统，而且每个子系统仅仅能与同层或更低层的子系统通讯，这样可以使每个层次的接口既完备又精练， 避免了各个模块之间很复杂的依赖关系。

设计时要充分考虑，对于各个层次，层次越低，通用性越强，这样，可以保证应用程序的需求发生改变时，所做的改动最小。开发视图所用的风格通常是层次结构风格。

3. 进程视图

进程视图侧重于系统的运行特性，主要关注一些非功能性的需求。

进程视图强调并发性、分布性、系统集成性和容错能力，以及从逻辑视图中的主要抽象如何适合进程结构。它也定义逻辑视图中的各个类的操作具体是在哪一个线程中被执行的。

进程视图可以描述成多层抽象，每个级别分别关注不同的方面。在最高层抽象中，进程结构可以看作是构成一个执行单元的一组任务。它可看成一系列独立的，通过逻辑网络相互通信的程序。它们是分布的，通过总线或局域网、广域网等硬件资源连接起来。

4. 物理视图

物理视图主要考虑如何把软件映射到硬件上，它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、通讯等问题。

当软件运行于不同的节点上时，各视图中的构件都直接或间接地对应于系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其他视图的影响最小。

大型系统的物理视图可能会变得十分混乱，因此可以与进程视图的映射一道，以多种形式出现，也可单独出现。

5. 场景

场景可以看作是那些重要系统活动的抽象，它使四个视图有机联系起来，从某种意义上说场景是最重要的需求抽象。在开发体系结构时，它可以帮助设计者找到体系结构的构件和它们之间的作用关系。同时，也可以用场景来分析一个特定的视图，或描述不同视图构件间是如何相互作用的场景可以用文本表示，也可以用图形表示。

七、软件体系结构风格定义、含义

1. 定义与含义：

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模式。

体系结构风格定义了一个系统家族，即定义一个词汇表和一组约束。词汇表中包含一些构件和连接件类型，而这组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地组织成一个完整的系统。

2. 经典的体系结构风格的分类

数据流风格：批处理序列；管道/过滤器。

调用/返回风格：主程序/子程序；面向对象风格；层次结构。

独立构件风格：进程通讯；事件系统。

虚拟机风格：解释器；基于规则的系统。

仓库风格：数据库系统；超文本系统；黑板系统。

3. 几种体系结构风格

（1）管道/过滤器

含义：每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。

这里的构件被称为过滤器，这种风格的连接件就象是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。

组成：构件（过滤器）和连接件（管道）

优点：

- 使得软构件具有良好的隐蔽性和高内聚、低耦合的特点；
- 允许设计者将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；
- 支持软件重用。只要提供适合在两个过滤器之间传送的数据，任何两个过滤器都可被连接起来；
- 系统维护和增强系统性能简单。新的过滤器可以添加到现有系统中来；旧的可以被改进的过滤器替换掉；
- 允许对一些如吞吐量、死锁等属性的分析；
- 支持并行执行。每个过滤器是作为一个单独的任务完成，因此可与其它任务并行执行。

缺点：

- 通常导致进程成为批处理的结构。
- 不适合处理交互的应用。
- 因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性。

例子：

Unix，DOS 中的重定向： (dir | sort)，perl 脚本语言，编译器

（2）数据抽象和面向对象组织

含义：这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和它们的相应操作封装在一个抽象数据类型或对象中。这种风格的构件是对象，或者说是抽象数据类型的实例。对象是一种被称作管理者的构件，因为它负责保持资源的完整性。对象是通过函数和过程的调用来交互的。

组成：构件（对象）

结构：对象抽象数据类型过程调用

优点：

- 因为对象对其它对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其它的对象；
- 设计者可将一些数据存取操作的问题分解成一些交互的代理程序的集合。

缺点：

- 为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。只要一个对象的标识改变了，就必须修改所有其他明确调用它的对象；
- 必须修改所有显式调用它的其它对象，并消除由此带来的一些副作用。例如，如果 A 使用了对象 B，C 也使用了对象 B，那么，C 对 B 的使用所造成的对 A 的影响可能是料想不到的。

（3）基于事件的隐式调用

含义：构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其它构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用在这个事件中注册的所有过程，这样，一个事件的触发就导致了另一模块中的过程的调用。这种风格的构件是一些模块，模块既可以是一些过程，又可以是一些事件的集合。过程可以用通用的方式调用，也可以在系统事件中注册一些过程，当发生这些事件时，过程被调用。这种风格的主要特点是事件的触发者并不知道哪些构件会被这些事件影响。这样不能假定构件的处理顺序，甚至不知道哪些过程会被调用，因此，许多隐式调用的系统也包含显式调用作为构件交互的补充形式。

组成：构件（一些模块，模块既可以是一些过程，又可以是一些事件的集合）

优点：

- 为软件重用提供了强大的支持。当需要将一个构件加入现存系统中时，只需将它注册到系统的事件中。
- 为改进系统带来了方便。当用一个构件代替另一个构件时，不会影响到其它构件的接口。

缺点：

- 构件放弃了对系统计算的控制。一个构件触发一个事件时，不能确定其它构件是否会响应它。而且即使它知道事件注册了哪些构件的构成，它也不能保证这些过程被调用的顺序。

- 数据交换效率的问题。有时数据可被一个事件传递，但另一些情况下，基于事件的系统必须依靠一个共享的仓库进行交互。在这些情况下，全局性能和资源管理便成了问题。
- 既然过程的语义必须依赖于被触发事件的上下文约束，关于正确性的推理存在问题。

（4）分层系统

含义：层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层只对相邻的层可见。这样的系统中构件在一些层实现了虚拟机（在另一些层次系统中层是部分不透明的）。连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。这种风格支持基于可增加抽象层的设计。允许将一个复杂问题分解成一个增量步骤序列的实现。由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件重用提供了强大的支持。

组成：构件，连接件

- 优点：
- 支持基于抽象程度递增的系统设计，使设计者可以把一个复杂系统按递增的步骤进行分解；
- 支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层；
- 支持重用。只要提供的服务接口定义不变，同一层的不同实现可以交换使用。这样，就可以定义一组标准的接口，而允许各种不同的实现方法。

缺点：

- 并不是每个系统都可以很容易地划分为分层的模式，甚至即使一个系统的逻辑结构是层次化的，出于对系统性能的考虑，系统设计师不得不把一些低级或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）仓库系统及知识库

含义：有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存贮上执行。控制原则的选取产生两个主要的子类。若输入流中某类时间触发进程执行的选择，则仓库是一传统型数据库；另一方面，若中央数据结构的当前状态触发进程执行的选择，则仓库是一黑板系统。

组成：构件（中央数据结构，独立构件）

例子：黑板系统的传统应用是信号处理领域，如语音和模式识别。另一个应用是松耦合代理数据共享存取。视频会议系统中有一个共享的白板可以用来交流。

（6）C2 风格

含义：通过连接件绑定在一起的按照一组规则运作的并行构件网络。C2 风格中的系统组织规则如下：

- 系统中的构件和连接件都有一个顶部和一个底部；
- 构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；
- 一个连接件可以和任意数目的其它构件和连接件连接；
- 当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部。

组成：构件、连接件

特点：

- 系统中的构件可实现应用需求，并能将任意复杂度的功能封装在一起；
- 所有构件之间的通讯是通过以连接件为中介的异步消息交换机制来实现的；
- 构件相对独立，构件之间依赖性较少。系统中不存在某些构件将在同一地址空间内执行，或某些构件共享特定控制线程之类的相关性假设。

（7）C/S

含义：C/S 软件体系结构是基于资源不对等，且为实现共享而提出来的，是 20 世纪 90 年代成熟起来的技术，C/S 体系结构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。

C/S 体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络。

组成：数据库服务器、客户应用程序和网络

优点：

- 模型思想简单，易于人们理解和接受。
- 灵活、易维护与扩充：系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。
- 资源可以进行合理配路：在 C/S 体系结构中，系统中的功能构件充分隔离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个 DBMS 进行编码。将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

缺点：

- 开发成本较高
- 客户端程序设计复杂

（8）B/S

含义：浏览器/服务器（B/S）风格就是三层应用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。B/S 体系结构主要是利用不断成熟的 WWW 浏览器技术，结合浏览器的多种脚本语言，用通用浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。从某种程度上来说，B/S 结构是一种全新的软件体系结构。

组成：浏览器/Web 服务器/数据库服务器

优点：

- 基于 B/S 体系结构的软件，系统安装、修改和维护全在服务器端解决。用户在使用系统时，仅仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。
- B/S 体系结构还提供了异种机、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：

- 没有集成有效的数据库处理功能，对数据处理功能不强。
- 安全性难以控制。
- 采用 B/S 体系结构的应用系统，在数据查询等响应速度上，要远远地低于 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

为什么要使用 B/S 与 C/S 混合体系结构：

- B/S 与 C/S 混合是一种典型的异构体系结构。
- 传统的 C/S 体系结构并非一无是处，而新兴的 B/S 结构也并非十全十美。C/S 结构和 B/S 结构还将长期共存。

(9) CORBA 体系结构

CORBA 是由对象管理组织 OMG 制定的一个工业标准，主要目标是提供一种机制，使得对象可以透明的发出请求和获得应答，从而建立起一个异质的分布式应用环境。

1991 年，OMG 基于面向对象技术，给出了对象请求代理为中心的对象管理结构。

CORBA 技术规范

接口定义语言（IDL）

接口池（IR）

动态调用接口（DII）

对象适配器（OA）

特点

◎引入中间件作为事务代理，完成客户机向服务对象方（Server）提出的业务请求。

◎ 实现客户与服务对象的完全分开，客户不需要了解服务对象的实现过程以及具体位置。

◎ 提供软总线机制，使得在任何环境下、采用任何语言开发的软件只要符合接口规范的定义，均能够集成到分布式系统中。

◎ 采用面向对象的软件实现方法开发应用系统，实现对象内部细节的完整封装，保留对象方法的对外接口定义。

(10)正交软件体系结构

含义：

正交软件体系结构由组织层和线索的构件构成。

层是由一组具有相同抽象级别的构件构成。

每条线索完成系统中相对独立的一部分功能,由完成不同层次功能的构件组成。

每条线索的实现与其他线索的实现无关或关联很少。

特征

◎ 系结构由不同功能的 n（n ＞ 1）个线索（子系统）组成；

◎ 系统具有 m（m ＞ 1）个不同抽象级别的层；

◎ 线索之间是相互独立的（正交的）；

◎ 同一层的构件不能相互调用；

◎ 系统有一个公共驱动层（一般为最高层）和公共数据结构（一般为最低层）。

优点：

◎结构清晰，易于理解。线索功能相互独立，不进行互相调用，结构简单、清晰。构件的位置就能说明它的抽象级别和担负功能。

◎易修改，可维护性强。线索之间相互独立，对一个线索的修改不会影响到其他线索。系统功能的增加或减少，只需相应的增删线索构件族，而不影响整个正交体系结构。

◎可移植性强，重用粒度大。正交结构可以为一个领域内的所有应用程序所共享，这些软件有着相同或类似的层次和线索，可以实现体系结构级的重用。

(11)基于层次消息总线的体系结构(Hierarchy Message Bus, HMB)

含义：

◎基于 HMB 的风格，支持构件的分布和并发，构件之间通过消息总线进行通信。

◎消息总线是系统的连接件，负责消息的分派，传递和过滤以及处理结果的返回。

◎ 各个构件挂在消息总线上，向总线登记感兴趣的消息类型;

◎ 构件根据需要发出消息，由消息总线负责把该消息分派到系统中所有对此消息感兴趣的构件;

◎ 构件收到消息后，根据自身状态对消息进行响应，并通过总线返回处理的结果。

◎ 消息是构件间唯一的通信方式

◎ 复杂构件可分解为比较低层的子构件，这些构件通过局部消息总线进行连接，这种复杂的构件成为复合构件。

(12)为什么要使用异构风格

不同的结构有不同的处理能力的强项和弱点，一个系统的体系结构应该根据实际需要进行选择，以解决实际问题。

关于软件包、框架、通信以及其他一些体系结构上的问题，目前存在多种标准。即使在某段时间内某一种标准占统治地位，但变动最终是绝对的。

一些遗留下来的代码，它们仍有效用，但是却与新系统有某种程度上的不协调。然而在许多场合，将技术与经济综合进行考虑时，总是决定不再重写它们。

八、软件体系结构的描述的几种方法

（1）图形表达工具

矩形框和有向线段组合而成。

矩形框代表抽象构件，框内标注的文字为抽象构件的名称。

有向线段代表辅助各构件进行通讯、控制或关联的连接件。

优点：简洁、易懂、使用广泛。

缺点：图形在术语和表达语义上存在着一些不规范和不精确，不同系统和不同文档之间有着不一致甚至矛盾。

（2）模块内连接语言 **MIL（Module Interconnection Language）**

将一种或几种传统程序设计语言的模块连接起来的语言

优点：程序设计语言和模块内连接语言具有严格的语义基础，因此能支持对较大的软件单元进行描述。诸如定义/使用和扇入/扇出等操作。例如， Ada 语言采用 use 实现包的重用，Pascal 语言采用过程（函数）模块的交互等。

MIL 方式对模块化的程序设计和分段编译等程序设计与开发技术确实发挥了很大的作用。

缺点：但是由于这些语言处理和描述的软件设计开发层次过于依赖程序设计语言，因此限制了它们处理和描述比程序设计语言元素更为抽象的高层次软件体系结构元素的能力。

（3）基于软构件的系统描述语言

基于软构件的系统描述语言将软件系统描述成一种是由许多以特定形式相互作用的特殊软件实体构造组成的组织或系统。是一种以构件为单位的软件系统描述方法。

优点：

可以用来在一个较高的抽象层次上对系统的体系结构建模。如 Darwin 最初用作设计和构造复杂分布式系统的配路说明语言，因具有动态特性，也可用来描述动态体系结构。

缺点：但是他们所面向和针对的系统元素仍然是一些层次较低的以程序设计为基础的通信协作软件实体单元，而且这些语言所描述和表达的系统一般而言都是面向特定应用的特殊系统，这些特性使得基于软构件的系统描述仍然不是十分适合软件体系结构的描述和表达。

（4）体系结构描述语言 ADL（Architecture Description Language）

ADL 针对软件体系结构的整体性和抽象性特点，定义和确定适合软件体系结构表达与描述的有关抽象元素。

ADL 是当前软件开发和设计方法学中发展很快的软件体系结构描述方法。

目前，已有几十种常见的 ADL，如：ACME，Darwin，Aesop、MetaH、C2、Rapide、SADL、Unicon，Weaves 和 Wright 等。

现状：这些 ADL 强调了体系结构不同的侧面，有不同的特点，对体系结构的研究和应用起到了重要的作用。但每一种 ADL 都以独立的形式存在，描述语法不同且互不兼容。

ADL 与其他语言的比较，具有下列特点：

◎ 构造能力：使用较小的独立体系结构元素来建造大型软件系统；

◎ 抽象能力：构件和连接件描述可以只关注它们的抽象特性，而不管其具体的实现细节；

◎ 重用能力：构件、连接件甚至是软件体系结构都成为软件系统开发和设计的可重用部件；

◎ 组合能力：ADL 支持软件系统的动态变化组合；

◎ 异构能力：允许多个不同的体系结构描述关联存在；

◎ 分析和推理能力：允许对其描述的体系结构进行多种不同的性能和功能上的多种推理分析。

（5）基于 UML 对体系结构描述

UML 用一系列视图来描述软件体系结构。主要的视图有：

- 用例图　捕获用户能够看到的系统功能
- 类图　描述一类对象的共同的属性和行为
- 对象图　对象是类的一个实例，是有具体属性和行为的具体事物
- 顺序图　是基于时间的动态交互。捕获系统的动态行为(面向时间的)
- 协作图　系统功能由各组成元素相互协作完成。捕获系统的动态行为(面向消息的)
- 状态图　记录系统中各组成元素的状态及变化。捕获系统的动态行为(面向事件的)
- 活动图 活动图类似于流程图，用于描述事件流结构。捕获系统的动态行为(面向活动的)

・构件图 是软件系统的一个物理单元，捕获实现的物理结构。
・部署图 捕获系统硬件的拓扑结构。
・其他 包、注释、构造型等

优点：用视图描述，非常直观，容易理解。

缺点：UML1.x 缺乏对体系结构下述的元素进和相应的描述与应用能力

体系结构风格

显式的体系结构连接器

体系结构约束

缺少形式化的描述语言，不能表达体系结构中的语义，不能描述体系结构的相关模型。

UML2.0 中增加了一种扩展机制来进行形式化描述，有所改进，成为了目前流行的体系结构描述工具。

（6）基于 XML 对体系结构描述

HTML(Hypertext Marked Language，超文本标记语言)用于描述网页的格式设计和 WWW 上其他网页的链接信息，简单易用，比较适合 Web 页面的开发，缺点是标记相对较少,不能支持特定领域的标记语言,如数学、化学、音乐等领域。

XML(Extensible Markup Language，扩展标记语言) 是一种自描述性的语言，是对 HTML 的扩展，用户可以自己创建标记来描述内容，描述能；和非常强。因此可以用来描述软件体系结构。

基于 XML 的软件体系结构描述语言有：

XADL2.0：加州大学欧文分校和卡内基梅隆大学合作创建。

XBA：复旦大学计算机系的赵文耘教授。

XCOBA：湖南师范大学的张友生教授。

七、质量属性及战术

・质量属性分类

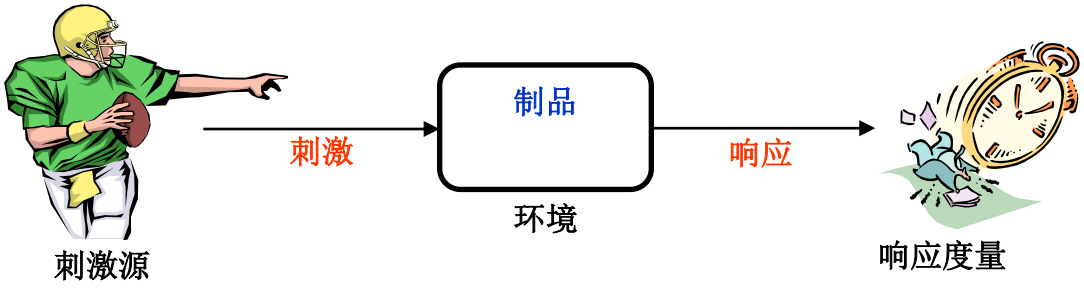
系统从设计、实现到部署的整个过程中要考虑质量属性的实现。质量属性包括下列三类：

（1）、系统的质量属性。（可用性、可修改性、性能、安全性、可测试性和易用性）

（2）、受架构影响的商业属性。（上市时间、成本和收益、所希望的系统生命期的长短、目标市场、推出计划、与老系统的集成）

（3）、与架构本身相关的一些质量属性。（概念完整性、正确性与完整性、可构建性）

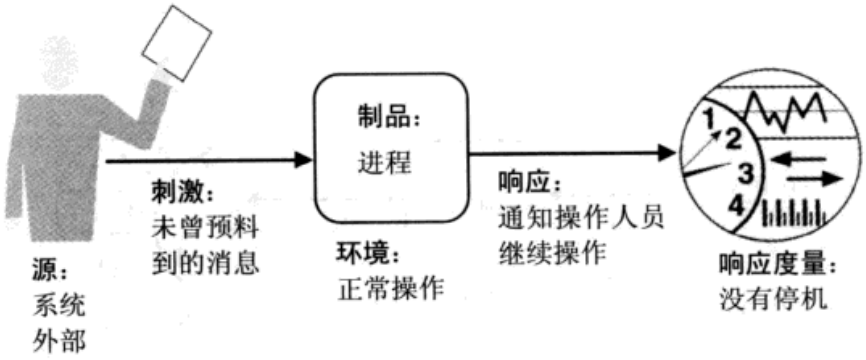
・质量属性场景的组成



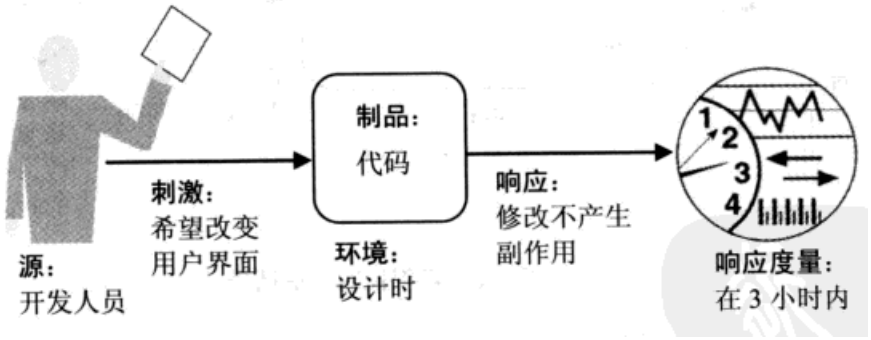
质量属性场景的6个组成部分

刺激源：产生事件活动的源，如用户、计算机系统等。
刺激：可以看作是产生的具体事件。
环境：系统当前的状态。
制品：系统中对事件作出反应的部分，可以是整个系统或系统的某一部分。
反应：事件到达后系统的相关行为。
反应度量：对反应结果提供某种形式的衡量。
・六个质量属性的场景样例

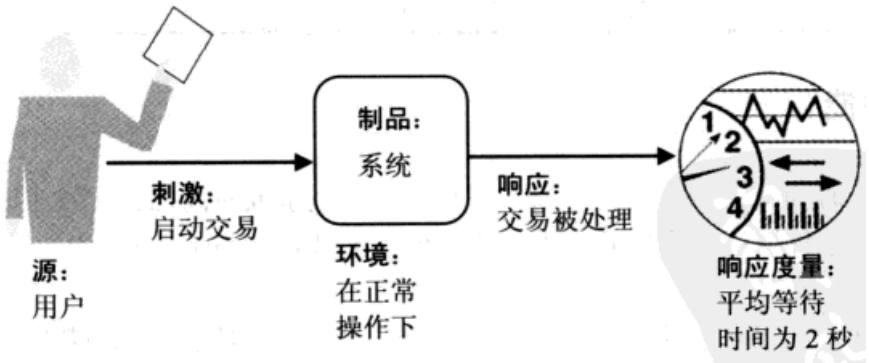
可用性



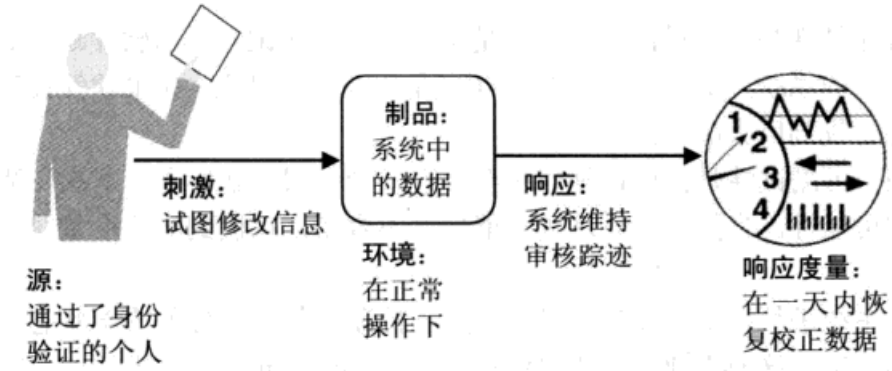
可修改性



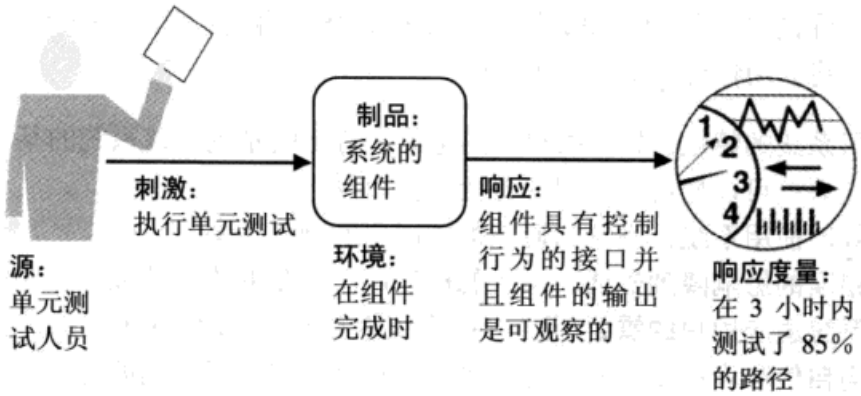
性能



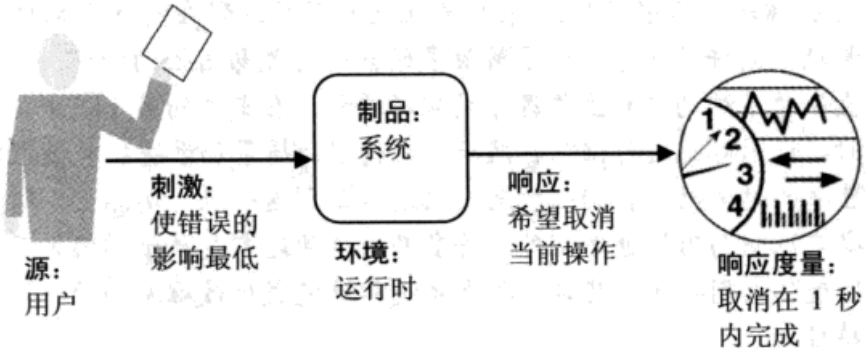
安全性



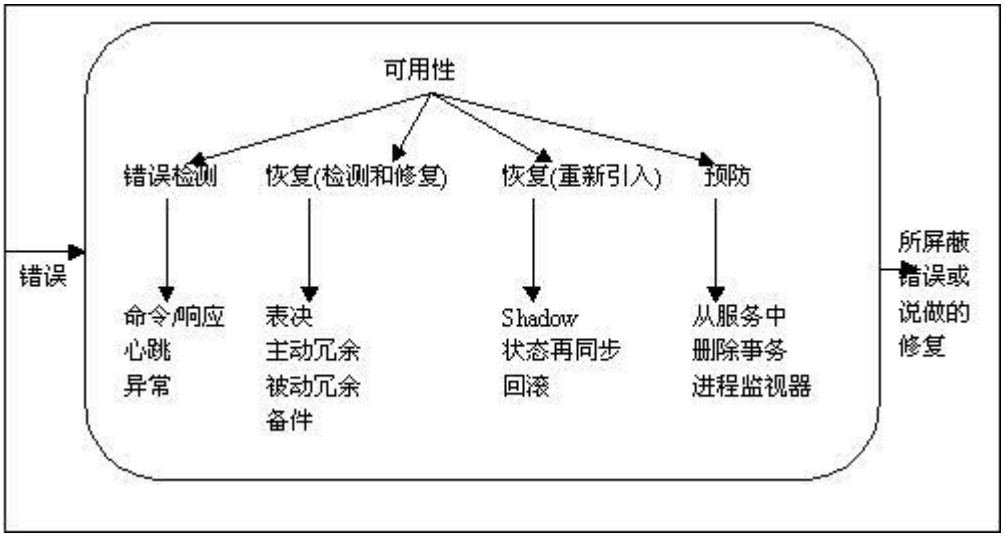
可测试性

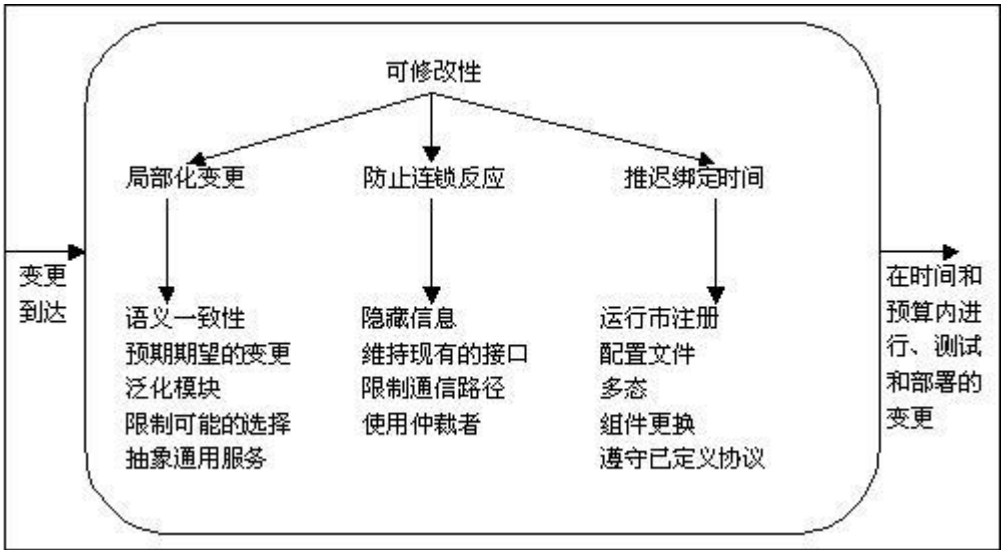


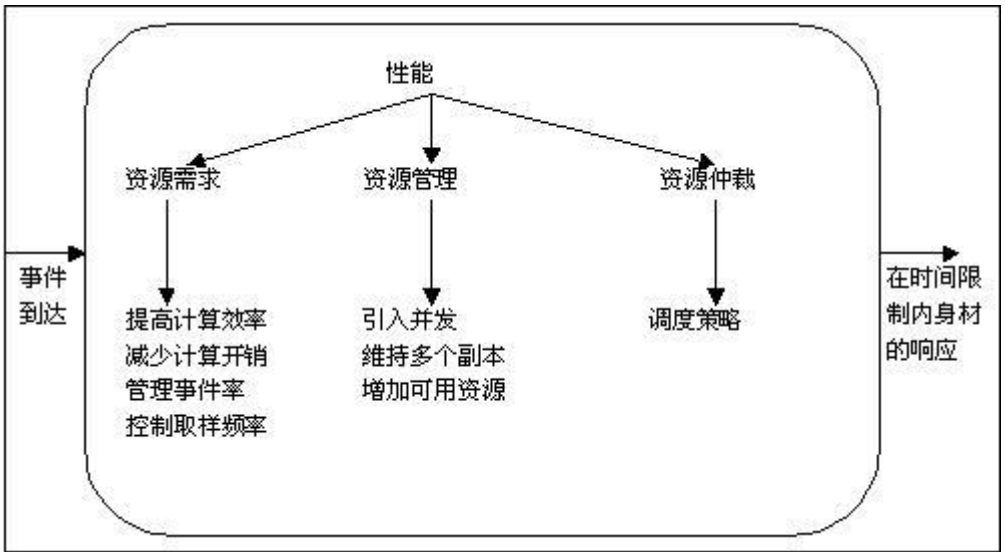
易用性

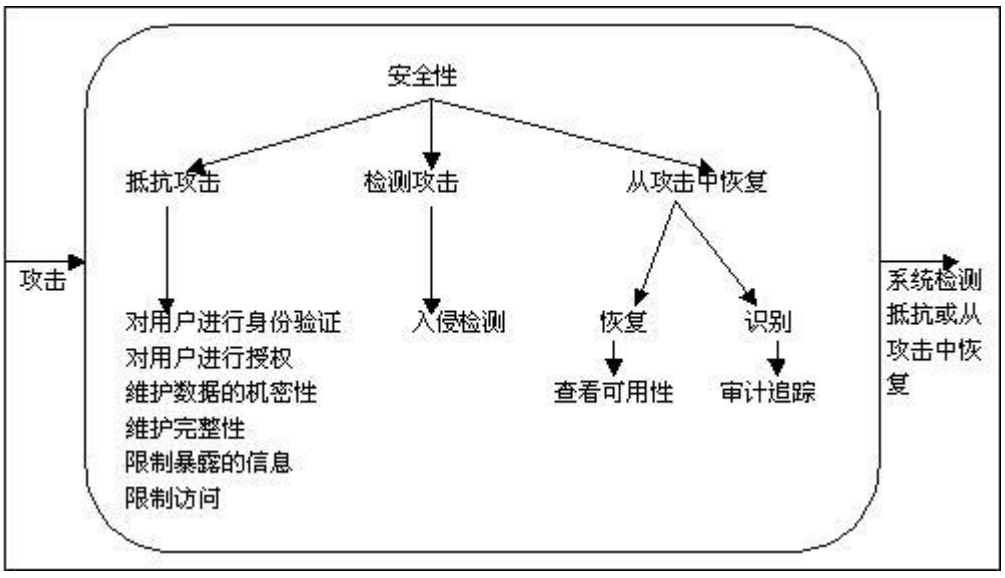


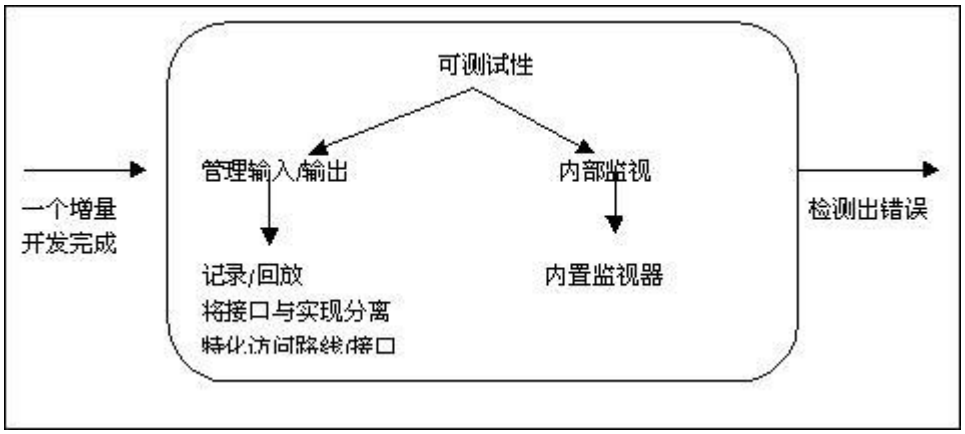
·六个质量属性的战术列表：

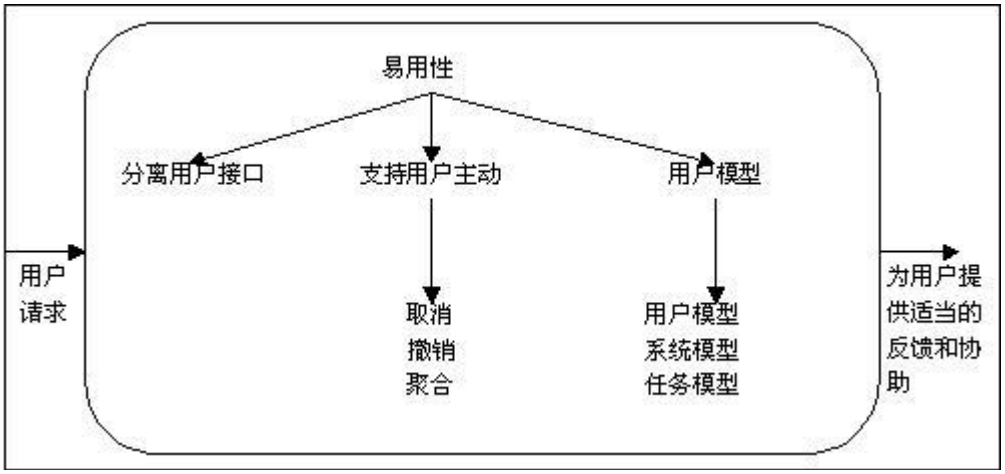



可修改性
局部化变更
防止连锁反应
推迟绑定时间
变更
到达
语义一致性
预期期望的变更
泛化模块
限制可能的选择
抽象通用服务
隐藏信息
维持现有的接口
限制通信路径
使用仲裁者
运行市注册
配置文件
多态
组件更换
遵守已定义协议
在时间和
预算内进
行、测试
和部署的
变更


性能
资源需求
资源管理
资源仲裁
事件
到达
提高计算效率
减少计算开销
管理事件率
控制取样频率
引入并发
维持多个副本
增加可用资源
调度策略
在时间限
制内身材
的响应


安全性
抵抗攻击
检测攻击
从攻击中恢复
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
维护完整性
限制暴露的信息
限制访问
入侵检测
恢复
识别
查看可用性
审计追踪
系统检测
抵抗或从
攻击中恢
复





八、设计架构

1. 设计模式定义、作用

设计模式定义:

对通用设计问题的重复解决方案
对真实世界问题的实践的/具体的解决方案
面向特定的问题环境
权衡利弊之后得到的“最佳”解决方案
领域专家和设计老手的“杀手锏”
用文档的方式记录的最佳实践
在讨论问题的解决方案时，一种可交流的词汇
在使用（重用）、共享、构造软件系统中，一种有效地使用已有的智慧/经验/专家技术的方式

设计模式的作用：利用设计模式可方便地重用成功的设计和结构。把已经证实的技术表示为设计模式，使他们更加容易被新系统的开发者所接受。设计模式帮助设计师选择可使系统重用的设计方案，避免选择危害到可重用性的方案。设计模式还提供了类和对象接口的明确的说明书和这些接口的潜在意义，来改进现有系统的记录和维护。

2.几种常见设计模式

创建性模式（5 种），处理的是对象的创建过程,

Factory Method（工厂模式）

Abstract Factory（抽象工厂模式）

Builder（建造模式）

Prototype（原型模式）

Singleton（单例模式）

结构性模式（7 种）,处理的是对象/类的组合

Adapter （适配器模式）

Bridge（桥接模式）

Composite （组合模式）

Decorator （装饰模式）

Facade（门面模式）

Flyweight （享元模式）

Proxy（代理模式）

行为性模式（11 种），处理的是类和对象间的交互方式和任务分布

Chain of Responsibility（反应链）

Command（命令模式）

Interpreter （解释器模式）

Iterator（迭代器模式）

Mediator（中介者模式）

Memento（备忘录模式）

Observer（观察者模式）

State（状态模式）

Strategy（策略模式）

Template Method（模板模式）

Visitor（访问者模式）

3. 设计架构

属性驱动的设计(ADD)是一个用于设计构架以满足质量需求和功能需求的方法。ADD 把一组质量属性场景作为输入，并使用对质量属性实现和构架之间的关系的了解，对构架进行设计。

ADD 构架设计的步骤如下（共 4 步）：

1．样本输入。系统要满足的功能、质量及受到的限制。

2．选择要分解的模块。首先要分解的就是系统本身最大的等待分解的模块。

3．根据下列步骤对模块进行求精：

a. 从具体的质量场景和功能需求集合中选择构架驱动因素。先找到比较重要的功能、重要的质量场景及重要的限制条件，个数不能太多。

b. 选择满足构架驱动因素的构架模式。
根据体系结构风格、质量战术来选择满足构架驱动因素的构架模式

c. 实例化模块并根据用例分配功能，使用多个视图进行表示。

d. 定义子模块的接口。

e. 验证用例和质量场景（是否得到满足）并对其进行求精，使它们成为子模块的限制（这样可以使子模块得到进一步划分）。

4．对需要进一步分解的每个模块重复上述步骤。这样递归的过程一般不超过 2～3 步

九、构架编档

让不同的风险承担者都能快速找到和理解他们所需要的信息。

构架编档的基本顺序:

(1) 产生结构列表

- 产生一个候选结构列表
- 组合结构
- 对结构划分优先级

(2)将相关结构编档

- 展示结构中的组件和组件之间关系的主要表示，常用图形方式。
- 组件目录至少祥述在主要表示中提到的组件和组件之间的相互关系，还包括组件的接口和行为。
- 系统与其环境相关的上下文图。
- 可变性指南，包括：
  要在其中做出选择的选项
  做出选择的时间
- 解释构架设计的背景，包括：
  基本原理
  分析结果
  设计中所反映的假定
- 结构中所选择的术语表。
- 其他信息。

(3)对行为进行编档

对系统行为进行推断，提供元素间的交互顺序、并发机会以及交互的时间依赖性的信息。

(4)对接口进行编档

接口就是两个独立的实体相遇并进行交互或通信的边界。组件接口就是其他组件可对该组件所做的假设。对接口进行编档的模板包括：

- 接口身份
- 所提供的资源
- 数据类型定义
- 异常定义
- 该接口提供的可变性
- 接口的质量属性特征
- 基本原理和设计问题
- 使用指南

(5)对跨结构信息编档

结构文档看起来不应是孤立存在的，而要形成一个整体，这包括构架概述、构架的基本原理和如何安排与组织文档。

构架概述包括：

- 系统概述
- 结构之间的映射

- 组件列表
- 项目词汇

构架基本原理包括：

- 设计决策
- 预计可能的修改对构架的影响
- 在实现解决方案中对开发人员的限制
- 拒绝采用的决策方案

十、软件构架重构

从现有系统中，利用实用的工具，提取构架方面的有用的信息，重新构建系统的软件构架的活动，称为软件构架重构。它是一种解释、交互和迭代的过程，涉及许多活动；它需要反向工程专家和设计师具备相关技能并投入精力。

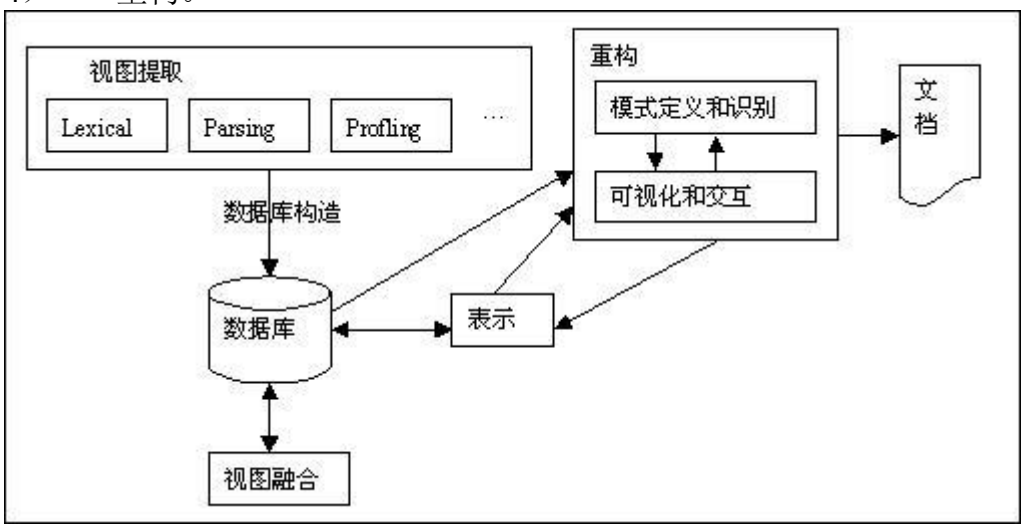
软件构架重构由以下活动组成，这些活动以迭代的方式进行：

（1） 信息提取。

（2） 数据库构造。

（3） 视图融合。

（4） 重构。



十一、分析构架

1．软件体系结构评估的主要方式 ：

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。但是由于评估的结果很大程度上来自评估人员的主观推断，因此不同的评估人员可能会产生不同甚至截然相反的结果，而且评估人员对领域的熟悉程度、是否具有丰富的相关经验也成为评估结构是否正确的主要因素。

尽管基于调查问卷与检查表的评估方式相对比较主观，但由于系统相关的人员的经验和知识是评估软件体系结构的重要信息来源，因而它仍然是进行软件体系结构评估的重要途径之一。

基于场景的评估方式

主要有体系结构权衡分析方法（ATAM）和软件体系结构分析方法（SAAM）。

这种软件体系结构评估方式分析软件体系结构对场景也就是对系统的使用或修改活动的支持程度，从而判断该体系结构对这一场景所代表的质量需求的满足程度。

基于场景的评估方式涉及到的基本活动包括确定应用领域的功能和软件体系结构的结构之间的映射，设计用于体现待评估质量属性的场景以及分析软件体系结构对场景的支持程度。

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。在软件体系结构评估里，可以把度量作为评判质量的重要依据。这类评估方式称作基于度量的评估方式。

主要有基于成本的分析方法(CBAM)等。

2．ATAM 评估

在该方法中，项目决策者和涉众要清晰地阐述一个准确的质量属性需求列表（以场景的方式），并说明与实现每个高优先场景相关的构架决策。然后，把这些决策确定为有风险决策或无风险决策，以找到构架中任何存在问题的地方。

ATAM 不是一个准确的手段，但它识别了构架中可能存在风险的区域。这些风险包含在敏感点和权衡中。

ATAM 活动的 4 个阶段：

在第 0 阶段（合作关系和准备）确定细节：人员名单，时间，地点；评估小组获取资料并进行初步了解分析。

第 1 阶段，评估阶段，决策者参与，小组开始信息收集与分析；耗时约 1 周。1～2 周中断期，评估小组进一步以非正式方式了解构架。

第 2 阶段，评估阶段，涉众参与，分析继续；约 2 天。

第 3 阶段，后续阶段，生成最终报告，进行评估活动总结；1 周。

评估阶段的步骤：

第 1 步：ATAM 方法的表述。评估负责人向决策者表述 ATAM 方法，使大家理解其过程，了解角色布局。

第 2 步：商业动机的表述。决策者介绍系统商业动机、重要功能、各种限制（任何相关的技术、管理、经济和政治限制）、商业目标和上下文、主要的涉众、驱动因素等。

第 3 步：构架的表述。首席设计师或架构小组介绍构架，技术限制、所用模式等。

第 4 步：对构架方法进行分类。评估小组利用所有已知信息对构架方法进行分类。

第 5 步：生成质量属性效用树。生成质量属性效用树，捕获详细的需求信息，为每个场景分配一个级别，如（高，中），前者为重要度，后者为实现难易度，重点放在（高，高）的场景；此处场景具备刺激、环境、响应三要素就可以了。

第 6 步：分析构架方法。评估小组分析所有重要场景，设计师解释如何支持该场景，检查所用构架方法，分析风险点、权衡点、敏感点。

经过一段中断期，第 2 阶段开始，此时涉众开始参与；首先仍然需要一个对 ATAM 方法的介绍，并使涉众了解已有的成果。

第 7 步：集体讨论并确定场景的优先级。集体讨论并分析场景的优先级，以了解更广泛的涉众的想法；该过程可能产生新的场景；使用“有限票数法”投票确定每个场景的优先级——此处不考虑实现难度。

第 8 步：分析构架方法。分析新的高优先级的场景，构架师解释构架是怎么满足各场景的。

第 9 步：结果表述。总结评估结果，评估负责人展示该结果。

十二、软件产品线

1．定义：

产品线是一个产品集合，这些产品共享一个公共的、可管理的特征集，这个特征集能满足选定的市场或任务领域的特定需求。这些系统遵循一个预描述的方式，在公共的核心资源(core assets)基础上开发的。

2．公共核心资产库和 COTS（商业构件）

公共核心资产库：其中保存可重用资产，这些资产可被应用到多个系统中。是产品线的基础，是管理支持产品开发的可复用资源的机制。

COTS（商业构件）：是一种开架出售的构件。构件经过对某个领域中的分析，这个领域里面经常会用的，共性的，完成一定功能的部分，做成一个构件。

3．SEI 的产品线开发模型中的基本活动

核心资产开发活动

产品开发活动

管理