极客时间 训练营

系统架构设计师

第7章 系统架构设计基础知识

授课：王建平

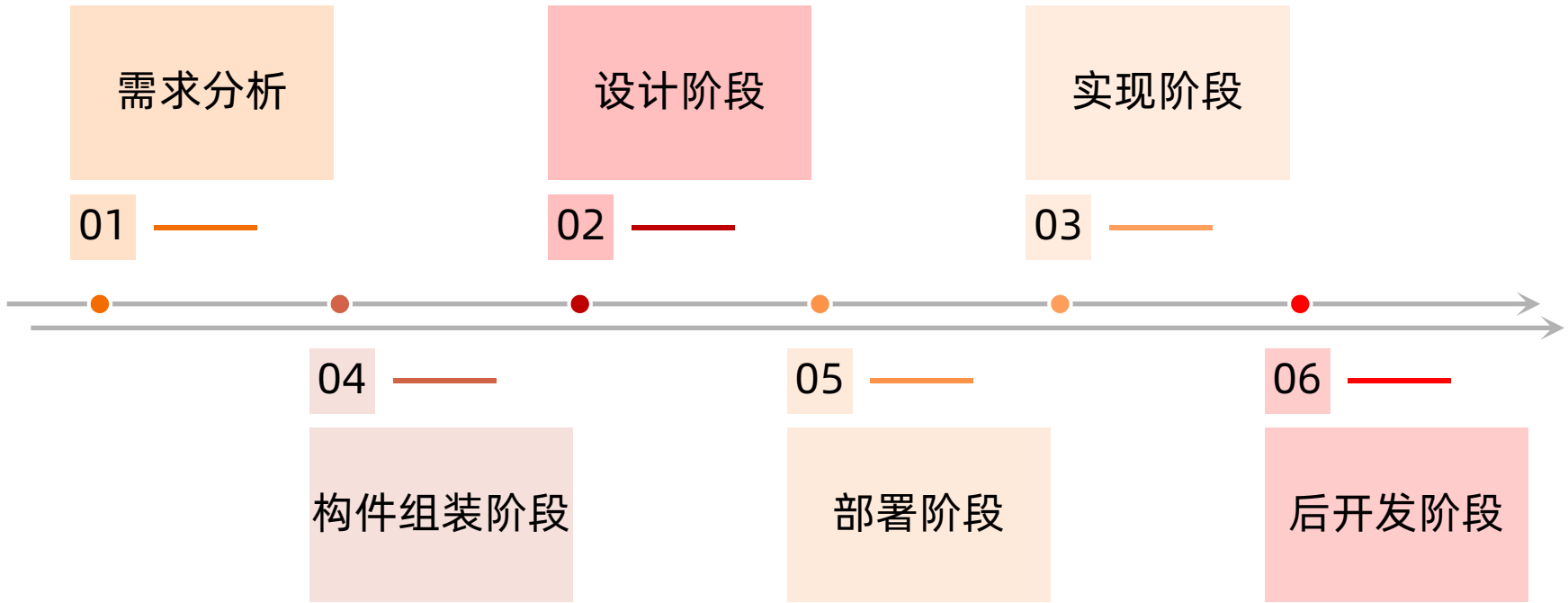
目录

目录

软件架构设计概念及生命周期

◆概念：软件架构设计是指通过一系列的设计活动，获得满足系统功能性需求，符合一定非功能性需求与质量属性有相似含义的软件系统框架模型。在软件体系结构设计过程中，主要考虑系统的非功能性需求。软件体系结构设计经验的总结与重用是软件工程的重要目标之一，所采用的手段主要包括体系结构风格、（特定领域的架构）DSSA等技术。

◆软件架构设计与生命周期：（★★★）



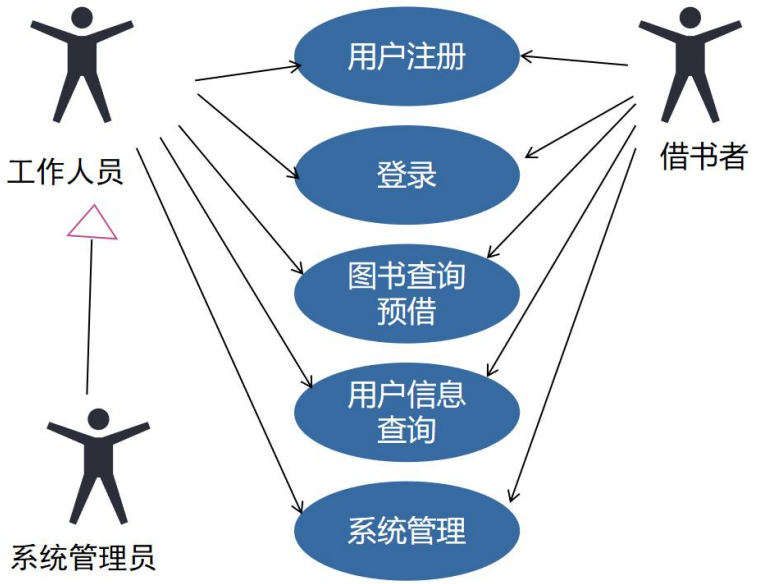
软件架构设计生命周期

◆需求阶段软件架构研究还处于起步阶段。在本质上，需求和软件架构设计面临的是不同的对象：一个是问题空间；另一个是解空间。保持二者的可追踪性和转换，一直是软件工程领域追求的目标。

◆从软件需求模型向 软件架构模型的转换主要关注两个问题：

① 如何根据需求模型构建软件架构模型。

② 如何保证模型转换的可追踪性。



某图书管理系统的顶层需求模型--用例图

系统架构的定义

◆设计阶段：这一阶段的研究主要包括：软件架构模型的描述、软件架构模型的设计与分析方法，以及对软件架构设计经验的总结与复用等。

◆有关软件架构模型描述的研究分为三个层次：（★）

1. SA的基本概念，即SA模型由哪些元素组成，这些组成元素之间按照何种原则组织。传统的设计概念只包括构件（软件系统中相对独立的有机组成部分，最初称为模块）以及一些基本的模块互联机制。随着研究的深入，构件间的互联机制逐渐独立出来，成为与构件同等级别的实体，称为连接子。
2. 体系结构描述语言（Architecture Description Language，ADL），支持构件、连接子及其配置的描述语言就是如今所说的体系结构描述语言。
3. 软件架构模型的多视图表示，从不同的视角描述特定系统的体系结构，从而得到多个视图，并将这些视图组织起来以描述整体的软件架构模型。系统的每一个不同侧面的视图反映了一组系统相关人员所关注的系统的特定方面，多视图体现了关注点分离的思想。典型的包括4+1视图。

软件架构设计生命周期

◆实现阶段的体系结构研究在以下几个方面：

1. 研究基于软件架构的开发过程支持，如项目组织结构、配置管理等。
2. 寻求从软件架构向实现过渡的途径，如将程序设计语言元素引入软件架构阶段、模型映射、构件组装、复用中间件平台等。
3. 研究基于软件架构的测试技术。

◆构件组装阶段研究内容包括：（★★★）

1. 如何支持可复用构件的互联，即对 SA 设计模型中规约的连接子的实现提供支持。
2. 组装过程中，如何检测并消除体系结构失配问题。这些问题主要包括：

a) 由构件引起的失配，包括由于系统对构件基础设施、构件控制模型和构件数据模型的假设存在冲突引起的失配。

b) 由连接子引起的失配，包括由于系统对构件交互协议、连接子数据模型的假设存在冲突引起的失配。

c) 由于系统成分对全局体系结构的假设存在冲突引起的失配等。要解决失配问题，首先需要检测出失配问题，并在此基础上通过适当的手段消除检测出的失配问题。

软件架构设计生命周期

◆部署阶段

部署阶段的软件架构对软件部署作用如下：

1. 提供高层的体系结构视图描述部署阶段的软硬件模型。
2. 基于软件架构模型可以分析部署方案的质量属性，从而选择合理的部署方案。

◆后开发阶段

后开发是指软件部署安装之后的阶段。这一阶段的软件架构研究主要围绕维护、演化、复用等方面来进行。典型的研究方向包括动态软件体系结构、体系结构恢复与重建等。

目录

基于架构的软件开发方法

◆基于体系结构（架构）的软件设计(Architecture-Based Software Design, ABSD)方法。ABSD方法是体系结构驱动，即指构成体系结构的商业、质量和功能需求的组合驱动的。在基于体系结构的软件设计方法中，采用视角与视图来描述软件架构，采用用例来描述功能需求，采用质量场景来描述质量需求。（★★★）

◆ABSD方法是一个自顶向下，递归细化的方法，软件系统的体系结构通过该方法得到细化，直到能产生软件构件和类。（★★★）

◆ABSD方法有三个基础：（★★★）

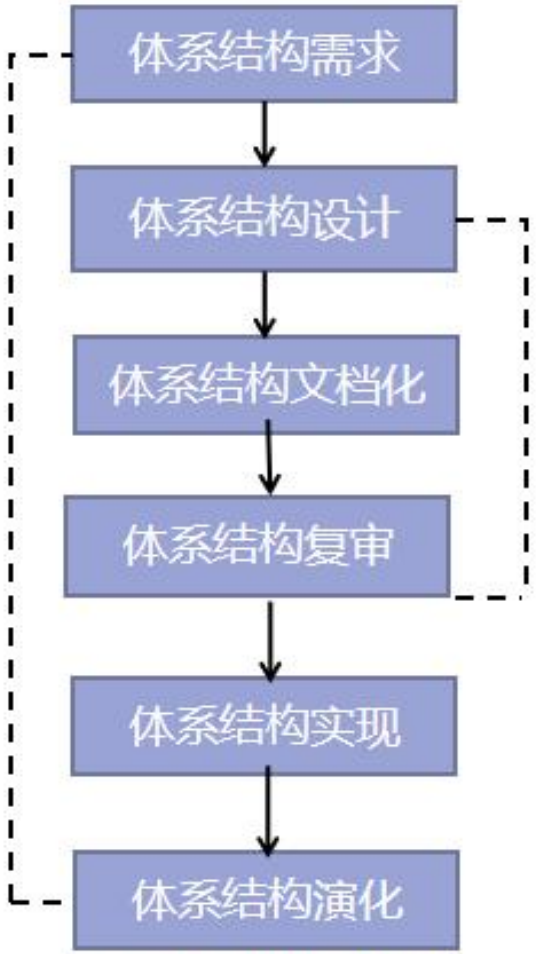
① 第一个基础是功能的分解。在功能分解中，ABSD方法使用已有的基于模块的内聚和耦合技术。

② 第二个基础是通过选择体系结构风格来实现质量和商业需求。

③ 第三个基础是软件模板的使用。软件模板利用了一些软件系统的结构。

基于架构的软件开发方法

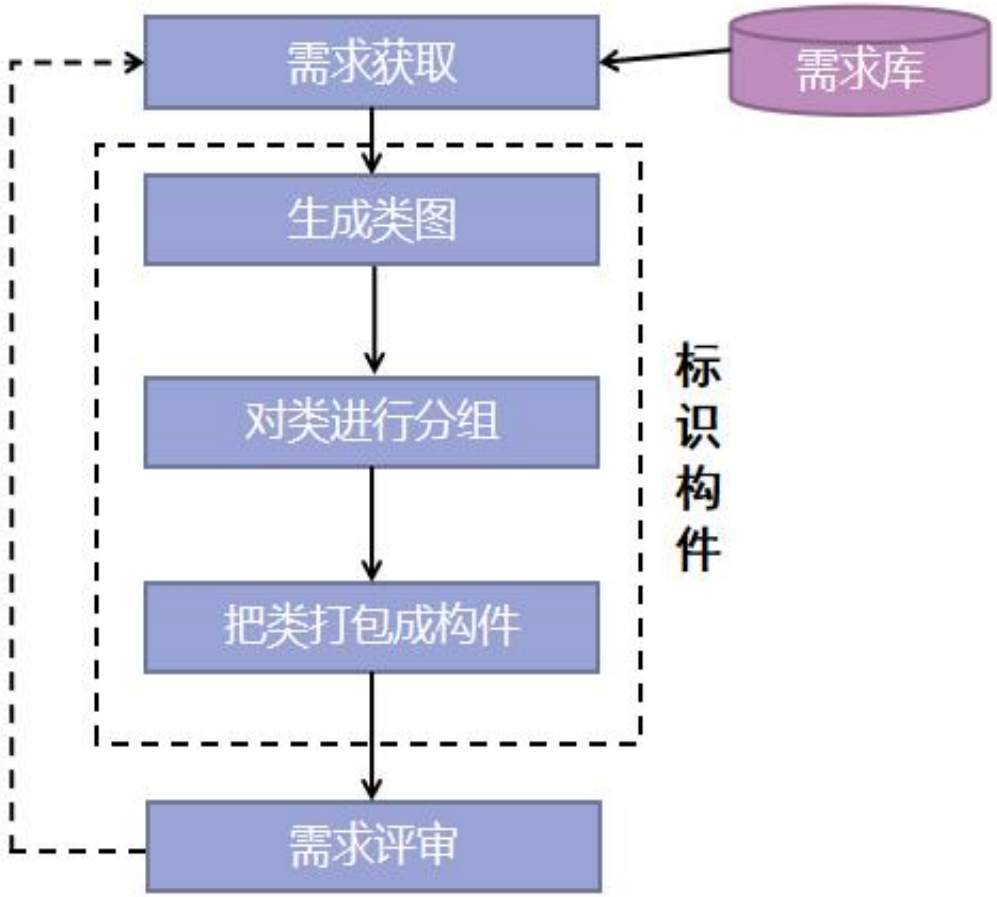
◆ABSDM模型把整个基于体体系结构的软件过程划分为体系结构需求、设计、文档化、复审、实现和演化等6个子过程，得到细化，直到能产生软件构件和类。（★★★）



体系结构开发模型

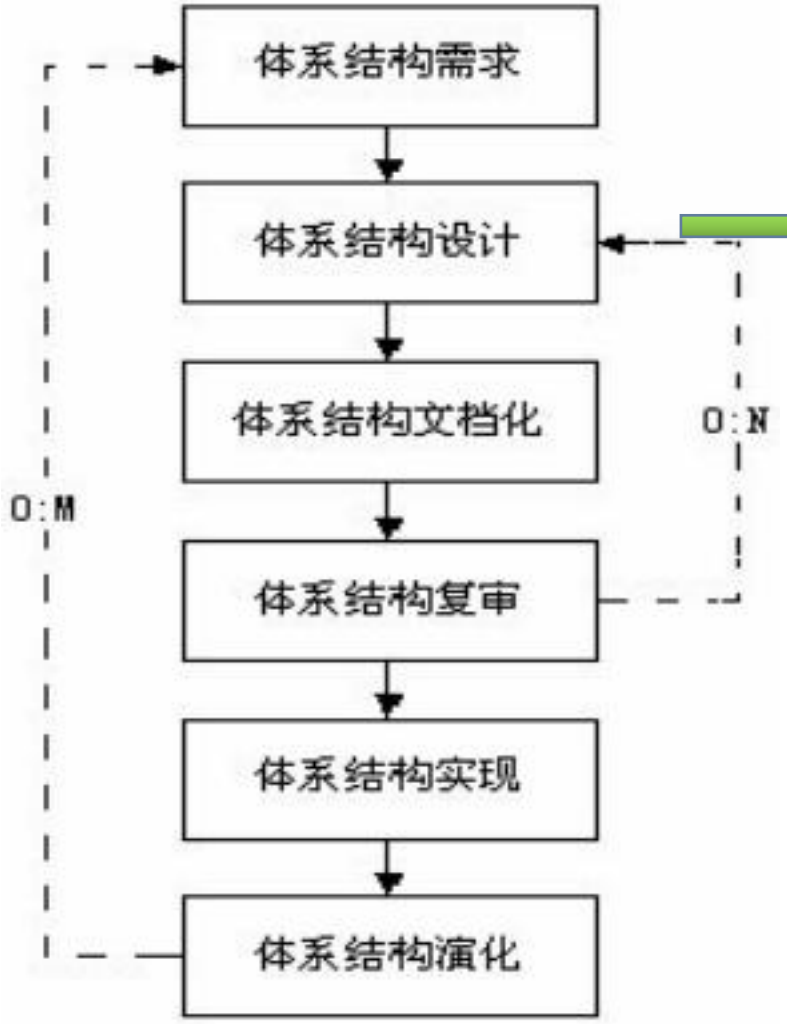
基于架构的软件开发方法

◆体系结构需求受技术环境和体系结构设计师的经验影响，需求过程主要是获取用户需求，标识系统中所要用到的构件。如果以前有类似的系统体系结构的需求，我们可以从需求库中取出，加以利用和修改，以节省需求获取的时间，减少重复劳动，提高开发效率。（★）



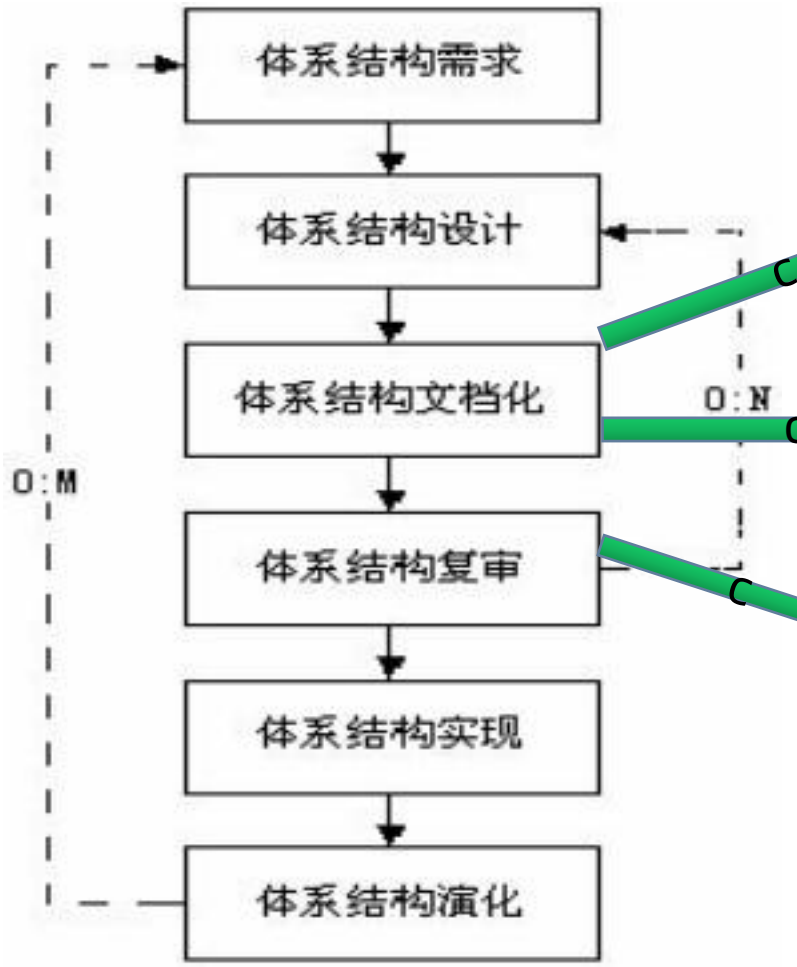
体系结构需求过程

基于架构的软件开发方法



◆体系结构设计
（★★★）
1、提出架构模型
2、映射构件
3、分析构件相互作用
4、产生架构
5、设计评审

基于架构的软件开发方法

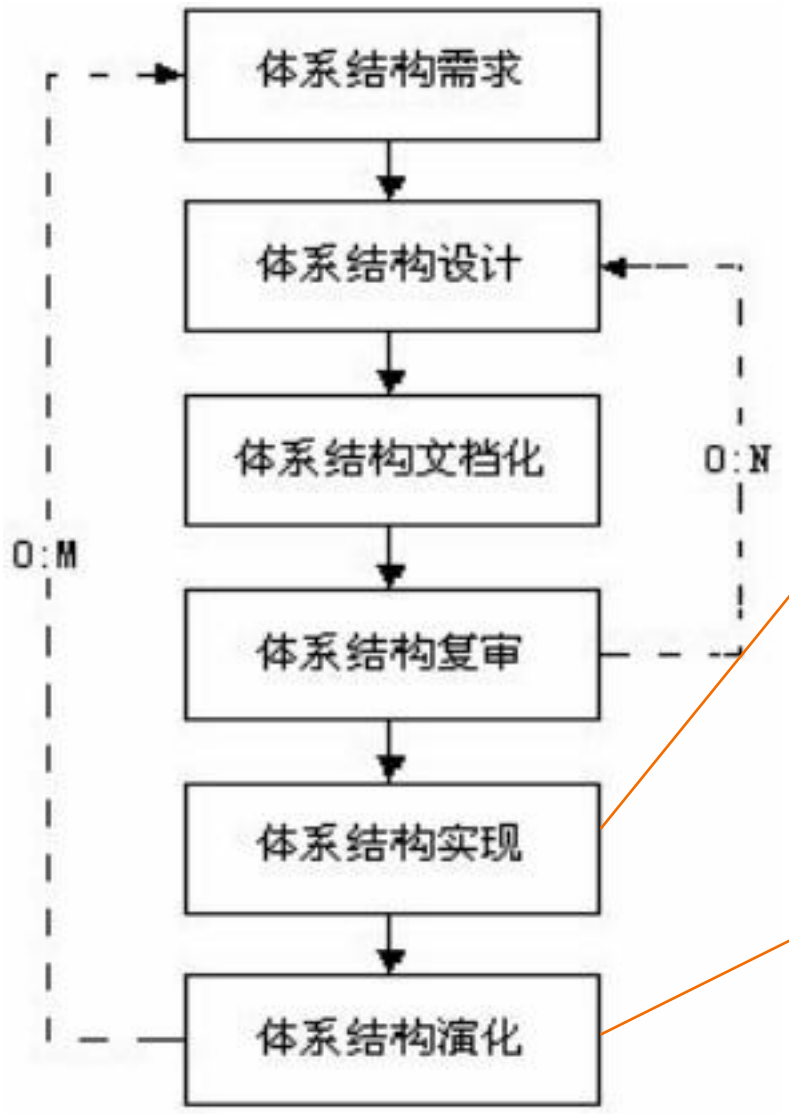


◆主要产出两种文档（★★★）
1、架构（体系结构）规格说明
2、测试架构（体系结构）需求的质量设计说明书。文档是至关重要的，是所有人员通信的手段，关系开发的成败。

◆文档的注意事项（★★★）
1、文档必须从使用者的角度编写
2、必须分发给所有与系统有关的开发人员
3、且保持开发者手上的文档是最新的

◆架构复审目的是标识**潜在的风险**，**发现架构中的缺陷和错误**。（★★★）
◆由**外部人员**（独立于开发组织之外的人，如用户代表和领域专家等）参加的复审，复审架构是否满足需求，质量问题，构件划分合理性等。若复审不过，则返回架构设计阶段进行重新设计、文档化，再复审。

基于架构的软件开发方法



◆体系结构实现（★★★）

用实体来显示出架构。实现构件，构件组装成系统复审后的文档化架构

1、分析与设计
2、构件实现
3、构件组装
4、系统测试

需要构件库的支持

◆体系结构演化（★★★）

对架构进行改变，按需求增删构件，使架构可复用

1、需求变化分类
2、架构演化计划
3、构件变动
4、更新构件的相互作用
5、构件组装与测试
6、技术评审
7、演化后的架构

典型真题

在基于体系结构的软件设计方法中，采用（ ）来描述软件架构，采用（ ）来描述功能需求，采 用（ ）来描述质量需求。

A．类图和序列图 B.视角与视图 C.构件和类图 D.构件与功能

A．类图 B.视角 C.用例 D.质量场景

A．连接件 B.用例 C.质量场景 D.质量属性

答案：BCC

体系结构文档化有助于辅助系统分析人员和程序员去实现体系结构。体系结构文档化过程的主要输出包括（ ）。

A.体系结构规格说明、测试体系结构需求的质量设计说明书

B.质量属性说明书、体系结构描述

C.体系结构规格说明、软件功能需求说明

D.多视图体系结构模型、体系结构验证说明

答案：A

典型真题

基于架构的软件设计(Archiecture-Based Software Design，ABSD)方法是架构驱动的方法，该方法是一个 ( ) 的方法，软件系统的架构通过该方法得到细化，直到能产生( )。

A.自顶向下　　B.自底向上　　C.原型　　D.自顶向下和自底向上结合

A.软件质量属性　B.软件连接性　C.软件构件或模块　　D.软件接口

答案：A C

目录

软件架构风格

◆软件架构的概念
软件体系结构（架构）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。（★★★）
① 词汇表中包含一些构件和连接件类型。
② 约束指出系统是如何将这些构件和连接件组合起来的。
◆软件架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。

◆通用架构风格的分类如下：（★★★）
(1)数据流风格：批处理序列；管道/过滤器。
(2)调用/返回风格：主程序/子程序；面向对象风格；层次结构；客户端/服务器。
(3)独立构件风格：进程通信；事件系统。
(4)虚拟机风格：解释器；基于规则的系统。
(5)仓库风格：数据库系统；超文本系统；黑板系统。

软件架构风格

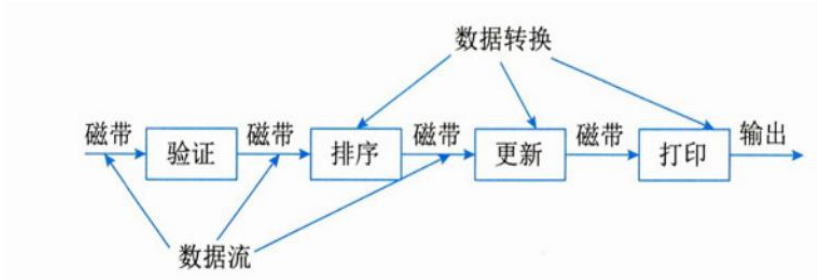
数据流风格包括：批处理序列；管道/过滤器（★★★）

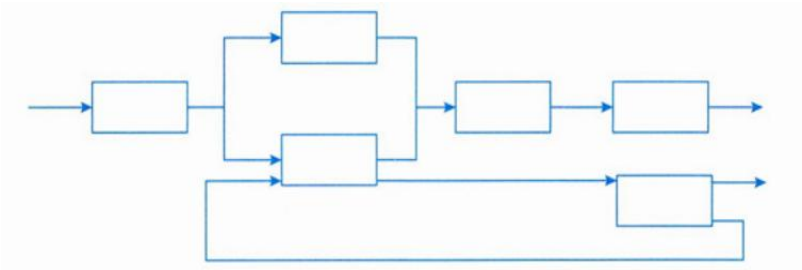
◆批处理体系结构风格

在批处理风格的软件体系结构中，每个处理步骤是一个单独的程序，每一步必须在前一步结束后才能开始，并且数据必须是完整的，以整体的方式传递。它的基本构件是独立的应用程序，连接件是某种类型的媒介。连接件定义了相应的数据流图，表达拓扑结构。

◆管道-过滤器体系结构风格

当数据源源不断地产生，系统就需要对这些数据进行若干处理(分析、计算、转换等)。解决方案是把系统分解为几个序贯的处理步骤，步骤之间通过数据流连接，一个步骤的输出是另一个步骤的输入。每个处理步骤由一个过滤器(Fiter)实现，处理步骤间的数据传输由管道(Pipe)负责。每个处理步骤(过滤器)都有一组输入和输出，过滤器从管道中读取输入的数据流，经过内部处理，然后产生输出数据流并写入管道中。管道-过滤器风格的基本构件是过滤器。典型的管道/过滤器体系结构的例子：①UNIX Shell编写的程序②传统的编译器





软件架构风格

优点	缺点	典型实例
1、高内聚、低耦合 2、良好的重用性/可维护性 3、可扩展性(标准接口适配） 4、良好的隐蔽性 5、支持并行	1交互性差 2、复杂性较高 3、性能差（每个过滤器都需要解析与合成数据）	传统编译器 网络报文处理

软件架构风格

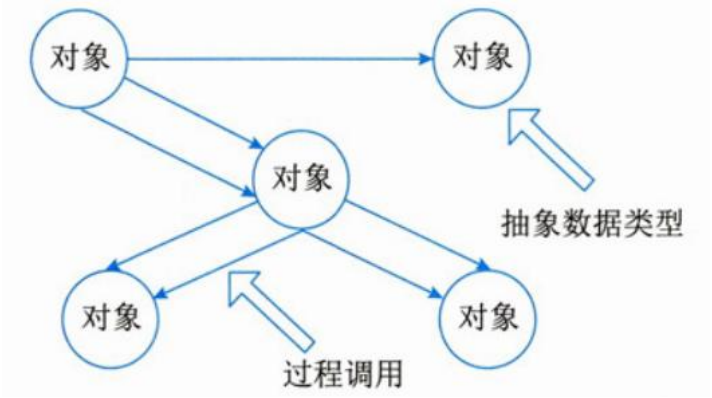
◆调用/返回风格是指在系统中采用了调用与返回机制。利用调用-返回实际上是一种分而治之的策略，主要思想是将一个复杂的大系统分解为若干子系统，以便降低复杂度，并且增可修改性。调用/返回体系结构风格主要包括：主程序/子程序风格、面向对象风格、层次风格、客户端/服务器风格

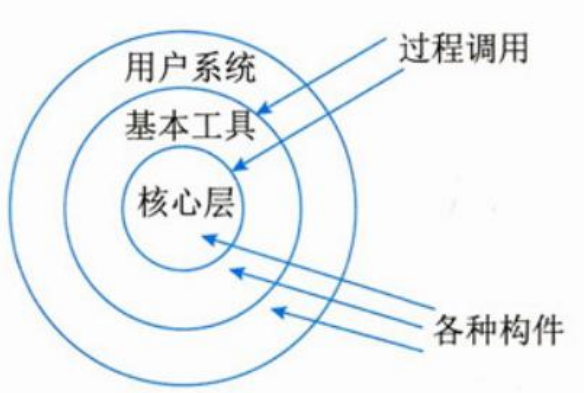
◆主程序/子程序风格

主程序/子程序风格一般采用单线程控制，把问题划分为若干处理步骤，构件即为主程序和子程序。子程序通常可合成为模块。过程调用作为交互机制，即充当连接件。调用关系具有层次性，其语义逻辑表现为主程序的正确性取决于它调用的子程序的正确性。

◆面向对象风格

面向对象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。

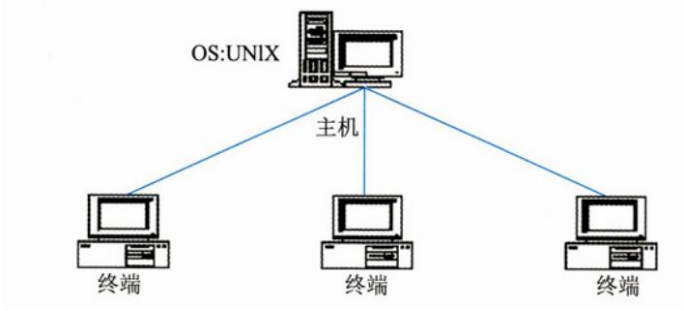


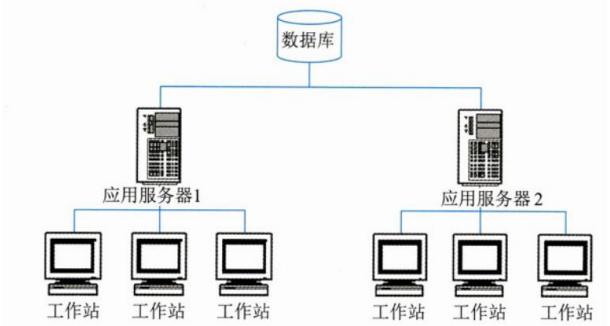


软件架构风格

◆层次型体系结构风格（★★★）
层次系统组成一个层次结构，每一层为上层提供服务，并作为下层的客户。在一些层次系统中，除了一些精心挑选的输出函数外，内部的层接口只对相邻的层可见。由于每一层最多只影响两层，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，这同样为软件重用提供了强大的支持。这样的系统中构件在层上实现了虚拟机。连接件由通过决定层间如何交互的协议来定义，拓扑约束包括相邻层间交互的约束。

◆客户端/服务器体系结构风格
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的，两层 C/S 体系结构有3 个主要组成部分：数据库服务器、客户应用程序和网络。服务器 (后台)负责数据管理，客户机(前台)完成与用户的交互任务，称为"胖客户机，瘦服务器"
三层C/S 结构增加了应用服务器。整个应用逻辑驻留在应用服务器上，只有表示层位于客户机上，故称为"瘦客户机"。应用功能分为表示层、功能层和数据层三层。

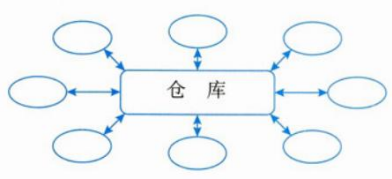




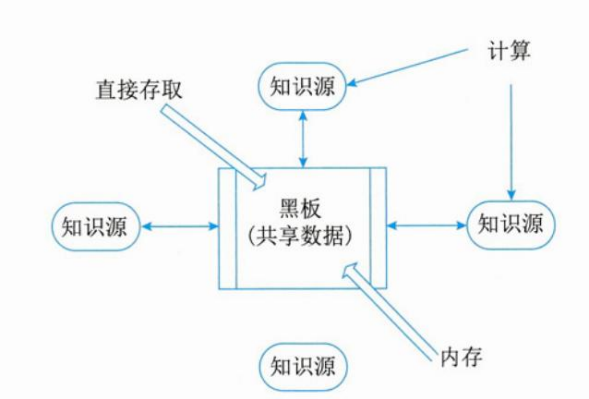
软件架构风格

优点	缺点	典型实例
1、良好的重用性，只要接口不变可用在其它处。 2、可维护性好。 3、可扩展性好，支持递增设计。	1、并不是每个系统都方便分层。 2、很难找到一个合适的、正确的层次抽象方法。 3、不同层次之间耦合性高的系统很难实现。	1.每个层次的组件形成不同功能级别的虚拟机； 2.多层相互协同工作，而且实现透明性。

软件架构风格



◆仓库体系结构风格
仓库(Repository)是存储和维护数据的中心场所。在仓库风格中，有两种不同的构件：中央数据结构(仓库)，说明当前数据的状态。独立构件，它对中央数据进行操作。
◆数据库系统：构件主要有两大类，一类是中央共享数据源，保存当前系统的数据状态；另一类是多个独立处理单元，处理单元对数据元素进行操作。场景：ORACLE数据库开发的系统
◆黑板体系结构风格（★★★）
黑板体系结构风格适用于解决复杂的非结构化的问题，能在求解过程中综合运用多种不同知识源，使得问题的表达、组织和求解变得比较容易。黑板系统是一种问题求解模型，是组织推理步骤、控制状态数据和问题求解之领域知识的概念框架。黑板系统的传统应用是信号处理领域，如语音和模式识别。另一应用是松耦合代理数据共享存取。
◆黑板系统主要由三部分组成：
(1)知识源。知识源中包含独立的、与应用程序相关的知识，知识源之间不直接进行通讯，它们之间的交互只能通过黑板来完成。
(2)黑板数据结构。黑板数据是按照与应用程序相关的层次来组织的解决问题的数据，知识源通过不断地改变黑板数据来解决问题。
(3)控制。控制完全由黑板的状态驱动，黑板状态的改变决定使用的特定知识。



3、超文本系统

软件架构风格

◆虚拟机体系结构风格的基本思想是人为构建一个运行环境，在这个环境之上，可以解析与运行自定义的一些语言，这样来增加架构的灵活性。虚拟机体系结构风格主要包括解释器风格和规则系统风格。（★★★）

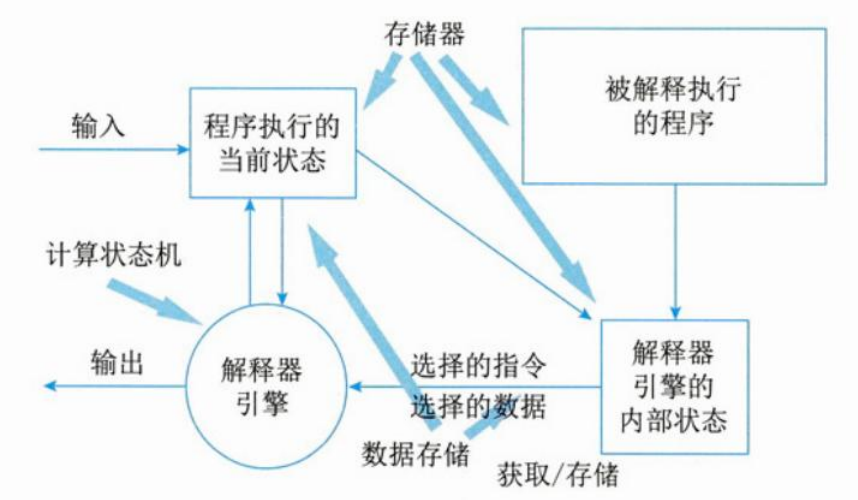
◆解释器体系结构风格（★★★）

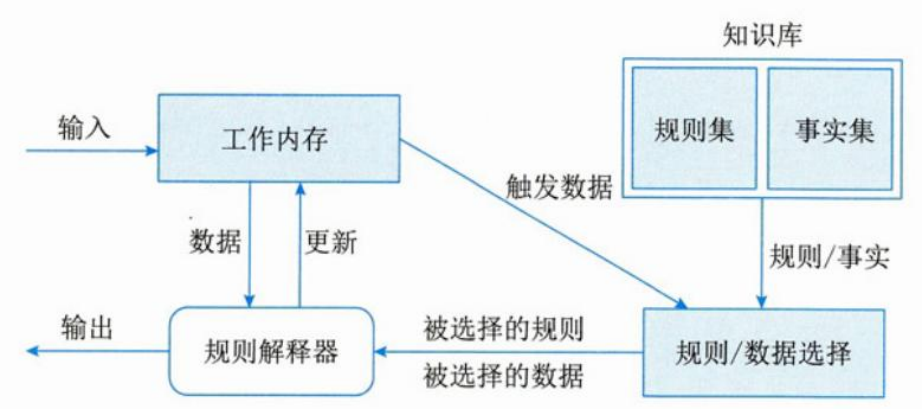
一个解释器通常包括完成解释工作的解释引擎，一个包含将被解释的代码的存储区，一个记录解释引擎当前工作状态的数据结构，以及一个记录源代码被解释执行进度的数据结构。

解释器通常被用来建立一种虚拟机以弥合程序语义与硬件语义之间的差异。其缺点是执行效率较低。典型的例子是专家系统。

◆规则系统体系结构风格（★★★）

基于规则的系统包括规则集、规则解释器、规则/数据选择器及工作内存。





软件架构风格

优点	缺点	要点
可以灵活应对自定义场景	复杂度较高	1、解释器：适用于需要“自定义规则”的场合。 2、规则为中心：在解释器的基础上增加经验规则。

软件架构风格

◆独立构件风格主要强调系统中的每个构件都是相对独立的个体，它们之间不直接通信，以降低耦合度，提升灵活性。独立构件风格主要包括进程通信和事件系统风格。

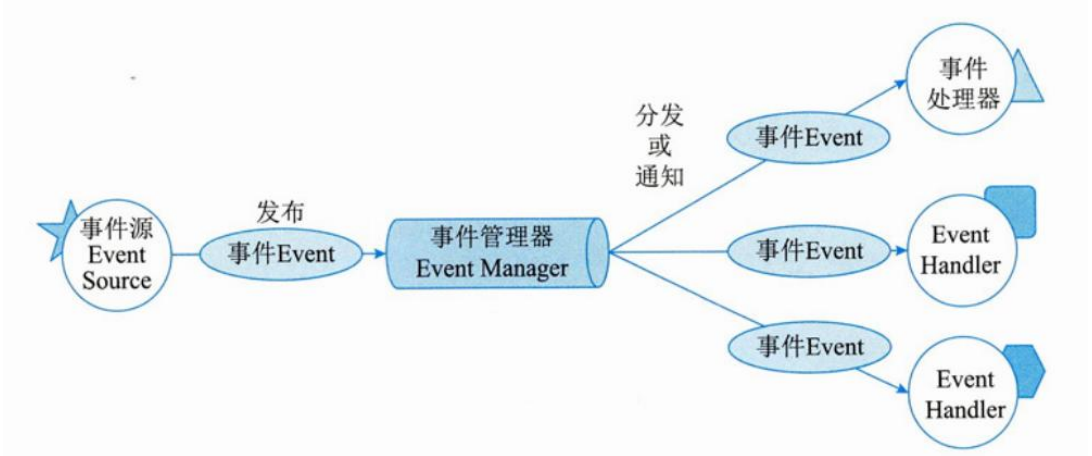
◆进程通信体系结构风格

在进程通信结构体系结构风格中，构件是独立的过程，连接件是消息传递。这种风格的特点是构件通常是命名过程，消息传递的方式可以是点到点、异步或同步方式及远程过程调用等。

◆事件系统体系结构风格（★★★）

基于事件的隐式调用风格的思想是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。系统中的其他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。

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，这使得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。



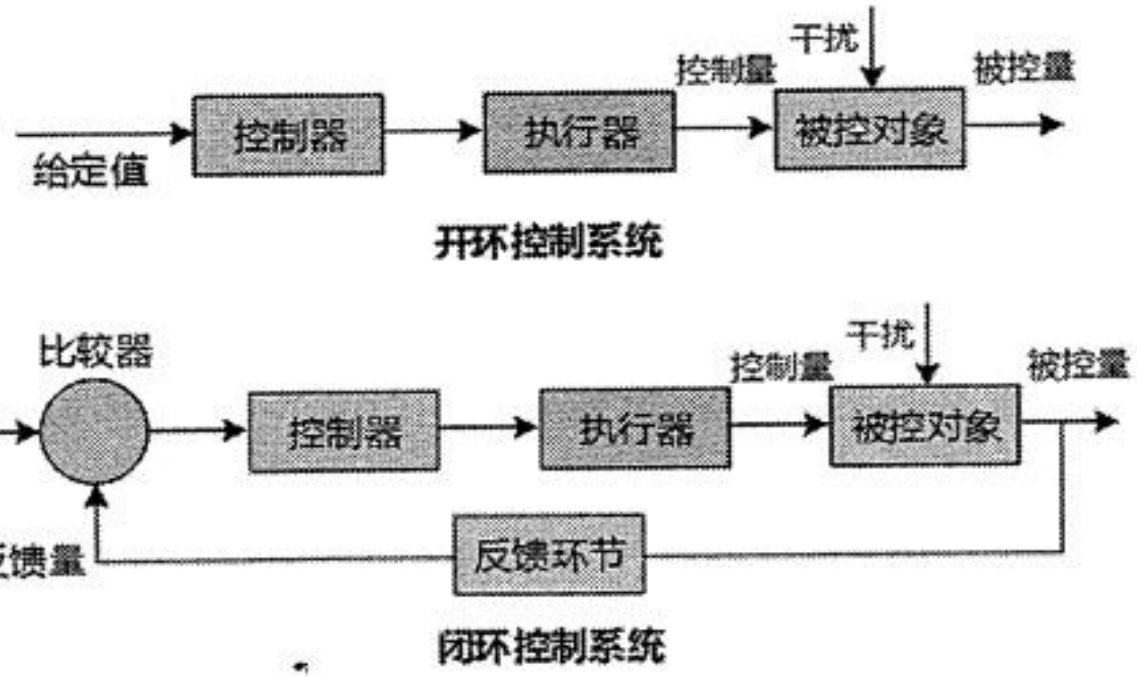
软件架构风格

优点	缺点
1、松耦合。 2、良好的复用性/可修改性/可扩展性。	1.构件放弃了对系统计算的控制。2.数据交换问题； 3.过程的语义必须依赖于被触发事件的上下文约束，正确性的推理存在问题

软件架构风格

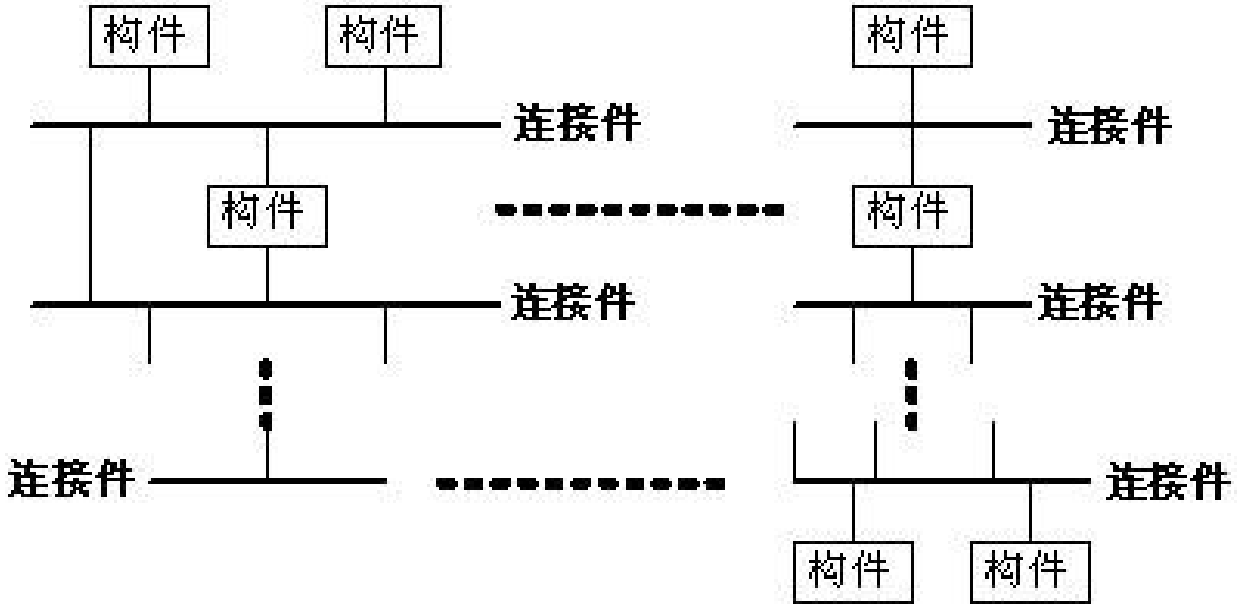
闭环控制

◆当软件被用来操作一个物理系统时，软件与硬件之间可以粗略的表示为一个反馈循环，这个反馈循环通过接受一定的输入，确定一系列的输出，最终使环境达到一个新的状态，适合于嵌入式系统，涉及连续的动作与状态。
◆场景：空调控温恒温机制。



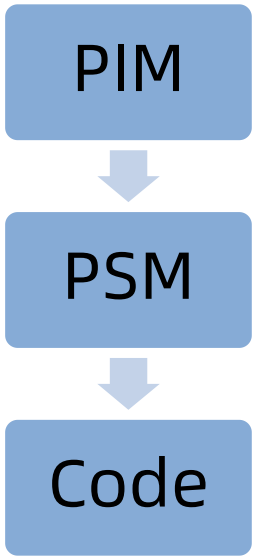
软件架构风格

◆C2体系结构风格可以概括为：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：
(1)系统中的构件和连接件都有一个顶部和一个底部；
(2)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；
(3)一个连接件可以和任意数目的其它构件和连接件连接；
(4)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。



软件架构风格

1、平台独立模型（PIM）：具有较高抽象层次，独立于任何实现技术的模型。
2、平台相关模型（PSM）：为某种特定实现技术量身定做。可用的实现构造来描述系统的模型。PIM会被变换成一个或多个PSM。
3、代码：用源代码对系统进行描述。每个PSM都被将变换成代码。



Mappings（映射 ）：PIM到PSM的映射。

软件架构风格

架构风格名	常考关键字及实例	简介
数据流-批处理	传统编译器，每个阶段产生的结果作为下一个阶段的输入，区别在于整体。	一个接一个，以整体为单位。
数据流-管道-过滤器		一个接一个，前一个输出是后一个输入。
调用/返回-主程序/子程序		显示调用，主程序直接调用子程序。
调用/返回-面向对象		对象是构件，通过对象调用封装的方法和属性。
调用/返回-层次结构		分层，每层最多影响其上下两层，有调用关系。
独立构件-进程通信		进程间独立的消息传递，同步异步。
独立构件-事件驱动（隐式调用）	事件触发推动动作，如程序语言的语法高亮、语法错误提示	不直接调用，通过事件驱动。
虚拟机-解释器	自定义流程，按流程执行，规则随时改变，灵活定义，业务灵活组合 机器人。	解释自定义的规则，解释引擎、存储区、数据结构。
虚拟机-规则系统		规则集、规则解释器、选择器和工作内存，用于 DSS 和人工智能、专家系统。
仓库-数据库	现代编译器的集成开发环境 IDE，以数据为中心。 又称为数据共享风格	中央共享数据源，独立处理单元。
仓库-超文本		网状链接，多用于互联网。
仓库-黑板		语音识别、知识推理等问题复杂、解空间很大、求解过程不确定的这一类软件系统，黑板、知识源、控制。
闭环-过程控制	汽车巡航定速，空调温度调节，设定参数，并不断调整。	发出控制命令并接受反馈，循环往复达到平衡。
C2 风格	构件和连接件、顶部和底部	通过连接件绑定在一起按照一组规则运作的并行构件网络

典型真题

软件架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。架构风格反映领域中众多系统所共有的结构和（ ），强调对架构（ ）的重用。

A．语义特性．　B.功能需求　　C.质量属性　　D.业务规则

A．分析　　B.设计．　　C.实现　　D.评估

答案：A　　B

以下关于软件架构风格与系统性能的关系叙述中，错误的是（ ）。

A．对于采用层次化架构风格的系统，划分的层次越多，系统的性能越差

B.对于采用隐式调用架构风格的系统，可以通过处理函数的并发调用提高系统处理性能

C.采用面向对象架构风格的系统，可以通过引入对象管理层提高系统性能

D.对于采用解释器架构风格的系统，可以通过部分解释代码预先编译的方式提高系统性能

答案：C

解析：引入对象管理层不但不能提高性能，反而会降低系统性能。这个道理与分层模型中增加层次是一样的。

典型真题

某公司拟为某种新型可编程机器人开发相应的编译器。该编译过程包括词法分析、语法分析、语义 分析和代码生成四个阶段，每个阶段产生的结果作为下一个阶段的输入，且需独立存储。针对上述 描述，该集成开发环境应采用（ ）架构风格最为合适。

A．管道—过滤器.　　B.数据仓储　　C.主程序—子程序　　D.解释器

答案：A

解析：“每个阶段产生的结果作为下一个阶段的输入”是典型的数据流架构风格的特点，选项中，仅有管 道-过滤器属于这种风格。

某企业内部现有的主要业务功能已封装成为Web服务。为了拓展业务范围，需要将现有的业务功能 进行多种组合，形成新的业务功能。针对业务灵活组合这一要求，采用（）架构风格最为合适。

A．规则系统　　B.面向对象　　C.黑板　　D.解释器.

答案：D

解析：根据题意，要求对业务功能灵活组合形成新的业务功能，就是有自定义类型的业务。自定义的业务 能正常执行，需要有虚拟机架构的支撑。 目前备选答案中A与D都是虚拟机风格。而A主要适合于专家系统，所以应选D。

典型真题

某公司拟开发一个VIP管理系统，系统需要根据不同商场活动，不定期更新VIP会员的审核标准和VIP 折扣系统。针对上述需求，采用（　）架构风格最为合适。
A．规则系统.　　B.过程控制　　C.分层　D.管道-过滤器
答案：A
解析：根据题目的意思，拟开发的VIP管理系统中VIP会员审核标准要能随时改变，灵活定义。在这方面虚 拟机风格最为擅长，而属于虚拟机风格的只有A选项。

（ ）架构风格可以概括为通过连接件绑定在一起按照一组规则运作的并行构件。
A．C2.　B.黑板系统　　C.规则系统　　D.虚拟机
答案：A

典型真题

某公司拟开发一个语音识别系统，其语音识别的主要过程包括分割原始语音信号、识别音素、产生候选词、判定语法片断、提供语义解释等，每个过程都需要进行基于先验知识的条件判断并进行相应的识别动作。针对该系统的特点，采用( )架构风格最为合适。
A．解释器　B．面向对象　C．黑板．　D．隐式调用
参考答案：C

某公司为其研发的硬件产品设计实现了一种特定的编程语言，为了方便开发者进行软件开发，公司拟开发一套针对该编程语言的集成开发环境，包括代码编辑、语法高亮、代码编译、运行调试等功能。针对上述描述，该集成开发环境应采用( )架构风格最为合适。
A．管道—过滤器　B．仓库风格．C．主程序-子程序　D．解释器
参考答案：B

试题分析

编程语言的集成开发环境需要提供代码编辑、语法高亮、代码编译、运行调试等功能，这些功能的特点是以软件代码为中心进行对应的编译处理与辅助操作。根据常见架构风格的特点和适用环境，可以知道最合适的架构设计风格应该是数据仓库风格。

目录

软件架构复用

◆软件架构复用的定义及分类
软件复用是一种系统化的软件开发过程，通过识别、开发、分类、获取和修改软件实体，以便在不同的软件开发过程中重复的使用它们。早期的软件复用主要是代码级复用，被复用的专指程序，后来扩大到包括领域知识、开发经验、体系结构、需求、设计、测试、代码和文档、过程方法和工具等一切有关方面。

◆软件架构复用的类型包括机会复用和系统复用。（★★★）
① 机会复用是指开发过程中，只要发现有可复用的资产，就对其进行复用。
② 系统复用是指在开发之前，就要进行规划，以决定哪些需要复用。

◆软件架构复用的原因：软件架构复用可以减少开发工作、减少开发时间以及降低开发成本，提高生产力。还可以提高产品质量使其具有更好的互操作性。同时，软件架构复用会使产品维护变得更加简单。

软件架构复用

◆软件架构复用过程（★★★）

1、构造/获取可复用的软件资产

首先需要构造恰当的、可复用的资产，并且这些资产必须是可靠的、可被广泛使用的、易于理解和修改的。

2、管理可复用资产

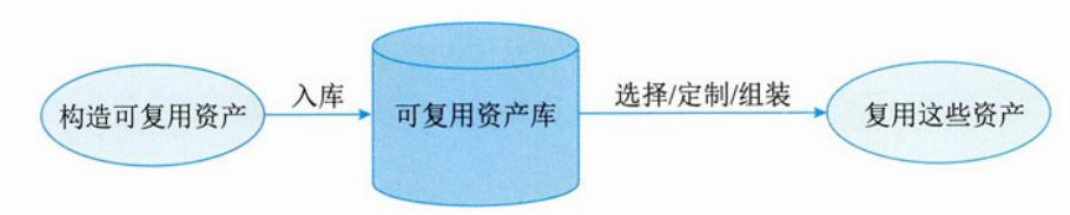
通过构件库对可复用构件进行存储和管理，构件库应提供的主要功能包括构件的存储、管理、检索以及库的浏览与维护等，以及支持使用者有效地、准确地发现所需的可复用构件。在这个过程中，存在两个关键问题：

① 构件分类，是指将数量众多的构件按照某种特定方式组织起来。

② 构件检索，是指给定几个查询需求，能够快速准确地找到相关构件。

3、使用可复用的资产

在最后阶段，通过获取需求，检索复用资产库，获取可复用资产，并定制这些可复用资产：修改、扩展、配置等，最后将它们组装与集成，形成最终系统。



典型真题

软件架构复用的类型包括机会复用和（），（）开发过程中，只要发现有可复用的资产，就对其进行复用.

A．代码复用　B．系统复用　C．结构复用　D．成本复用

A．代码复用　B．系统复用　C．结构复用　D．机会复用

参考答案：BD

目录

特定领域软件架构

◆特定领域软件架构DSSA（★）
特定领域软件架构是在一个特定领域中为一组应用提供组织结构参考的标准软件框架。
其目标是支持在一个特定领域中多个应用的生成。

◆DSSA中领域的含义：（★）
(1)垂直域。定义一个特定的系统族，包含整个系统族内的多个系统，结果是在该领域中可作为系统的可行解决方案的一个通用软件架构。---相同领域，深入
(2)水平域。定义了在多个系统和多个系统族中功能区域的共有部分，在子系统级上涵盖多个系统族的特定部分功能，无法为系统提供完整的通用架构。---不同领域，平移

◆DSSA的基本活动（★★★）
(1)领域分析。主要目的是获得领域模型。领域模型描述领域中系统之间的共同的需求，所描述的需求为领域需求。
(2)领域设计。主要目的是获得DSSA。DDSA描述在领域模型中表示需求的解决方案，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。
(3)领域实现。主要目的是依据领域模型及DSSA开发和组织可重用信息。

特定领域软件架构

◆参与DSSA的人员（★★★）

① 领域专家：有经验的用户，从事该领域中系统的需求分析、设计、实现以及项目管理的有经验的软件工程师。
② 领域分析师：具有知识工程背景的有经验的系统分析员来担任。
③ 领域设计人员：由有经验的软件设计人员来担任。
④ 领域实现人员：由有经验的程序设计人员来担任。

◆DSSA的建立过程（★★★）

DSSA的建立过程分为5个阶段，每个阶段又可分为一些步骤和子阶段，是并发的、递归的、反复的，是螺旋的。

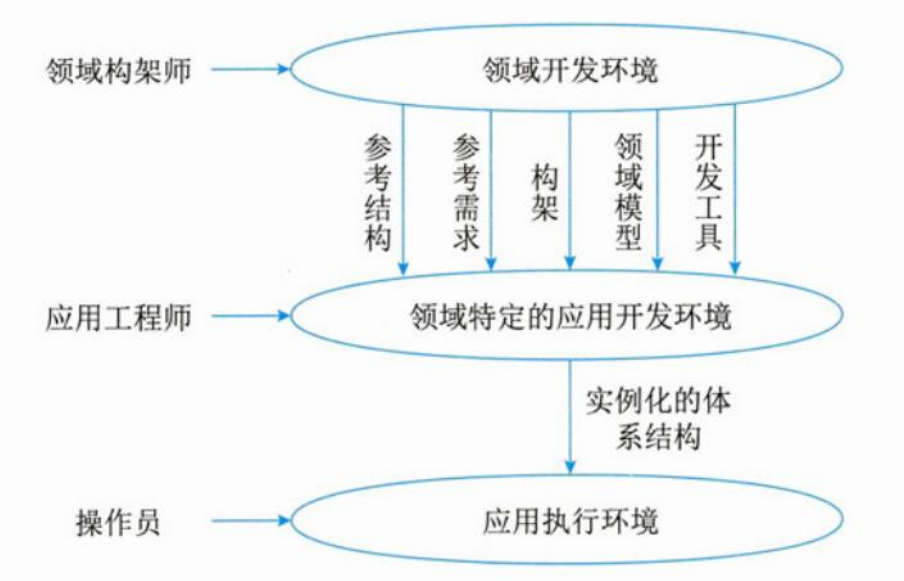
(1)定义领域范围。主要输出是领域中的应用需要满足一系列用户的需求。
(2)定义领域特定的元素。编译领域字典和领域术语的同义词词典。
(3)定义领域特定的设计和实现需求约束。
(4)定义领域模型和架构。
(5)产生、搜集可重用的产品单元。

特定领域软件架构

DSSA包含两个过程：（★）

◆领域工程：为一组相近或相似的应用建立基本能力与必备基础的过程，它覆盖了建立可重用软件元素的所有活动。

◆应用工程：通过重用软件资源，以领域通用体系结构为框架，开发出满足用户需求的一系列应用软件的过程 。



典型真题

特定领域软件架构(Domam Specifie Sottware Architecture.DSSA)是指特定应用领域中为一组应用提供组织结构参考的标准软件架构。从功能覆盖的范围角度，(1) 定义了一个特定的系统族，包含整个系统族内的多个系统，可作为该领城系统的可行解决方案的一个通用软件架构；(2) 定义了在多个系统和多个系统族中功能区域的共有部分，在子系统级上涵盖多个系统族的特定部分功能。

(1)A.垂直域. B.水平域　C.功统域 D.属性域

(2)A.垂直域　B.水平域. C.功统域 D.属性域

答案：A B

特定领域软件架构（Domain Specific Software Architecture, DSSA）的基本活动包括领域分析、领域设计和领 域实现。其中，领域分析的主要目的是获得领域模型。领域设计的主要目标是获得（1）。领域实现是为了（2）。

（1）A.特定领域软件需求　B.特定领域软件架构. C.特定领域软件设计模型　　D.特定领域软件重用模型

（2）A.评估多种软件架构

B.验证领域模型

C.开发和组织可重用信息，对基础软件架构进行实现.

D.特定领域软件重用模型

答案：（1）B（2）C

本章重点回顾

1、架构生命周期
2、ABSDM过程与注意要点
3、架构风格
4、特定领域架构要点

THANKS