软件工程基础 —— 第13章 构件级设计

计算机学院 孟宇龙

目录

关键概念

- 内聚性
- 构件适应性、分类、组合、合格性
- 基于构件的开发
- 内容设计
- 耦合
- 依赖倒置原则
- 设计重用
- 设计准则
- 领域工程

构件是软件体系结构的一部分

- 把设计模型转化为运行软件
  - 设计模型的抽象层次相对较高
  - 程序的抽象层次相对较低
- 构件级设计定义了数据结构、算法、接口特征和分配给每个软件构件的通信机制
- **数据、体系结构和接口的设计表示构成了构件级设计的基础**

13.1 什么是构件？

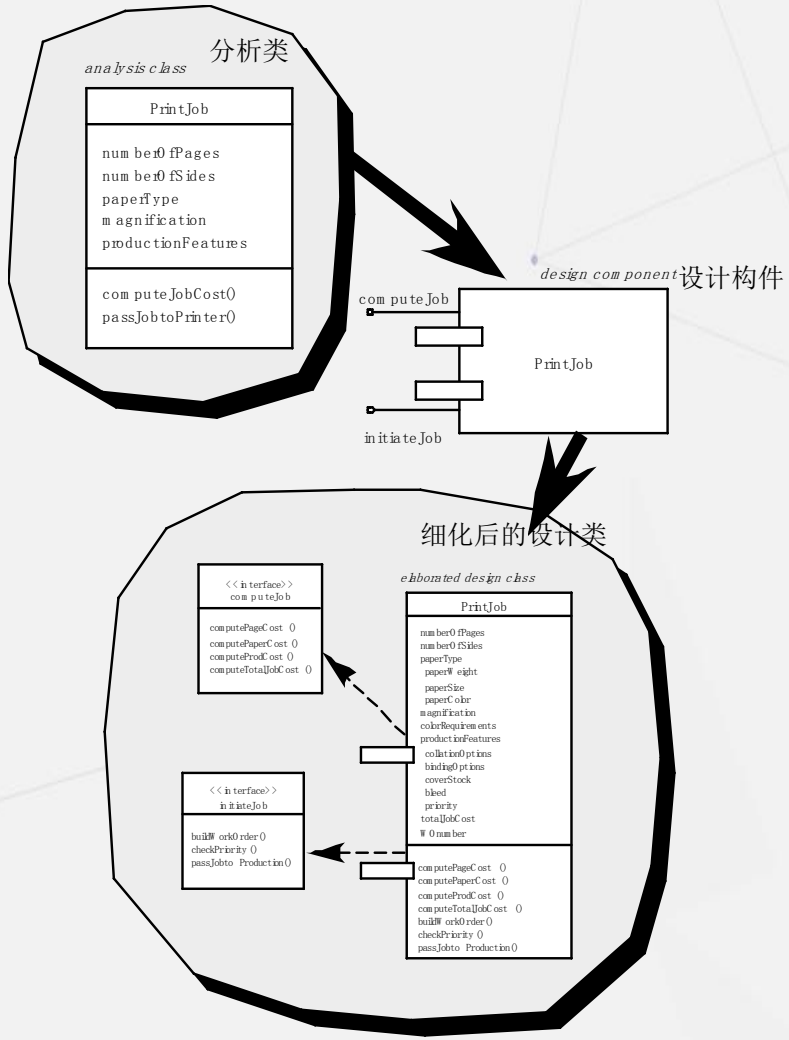
- OMG 统一建模语言规范[OMG01] 是这样定义构件的：

  “系统模块化的、可部署的和可替换的部件，该部件封装了实现并对外提供一组接口。”

- OO 观点：构件包含一组协作类的集合
- 传统观点：一个构件包含处理逻辑，实现处理逻辑所需的内部数据结构以及能保证构件被调用和实现数据传递的接口。

13.1.1 OO 构件

面向对象中，构件是一个协作类集合



- PrintJob被定义为软件体系结构的一个构件
- 它有两个接口：computeJob 和 initiateJob
- 细化后的设计类PrintJob包含更多的属性信息和构件实现所需要的更广泛的操作描述

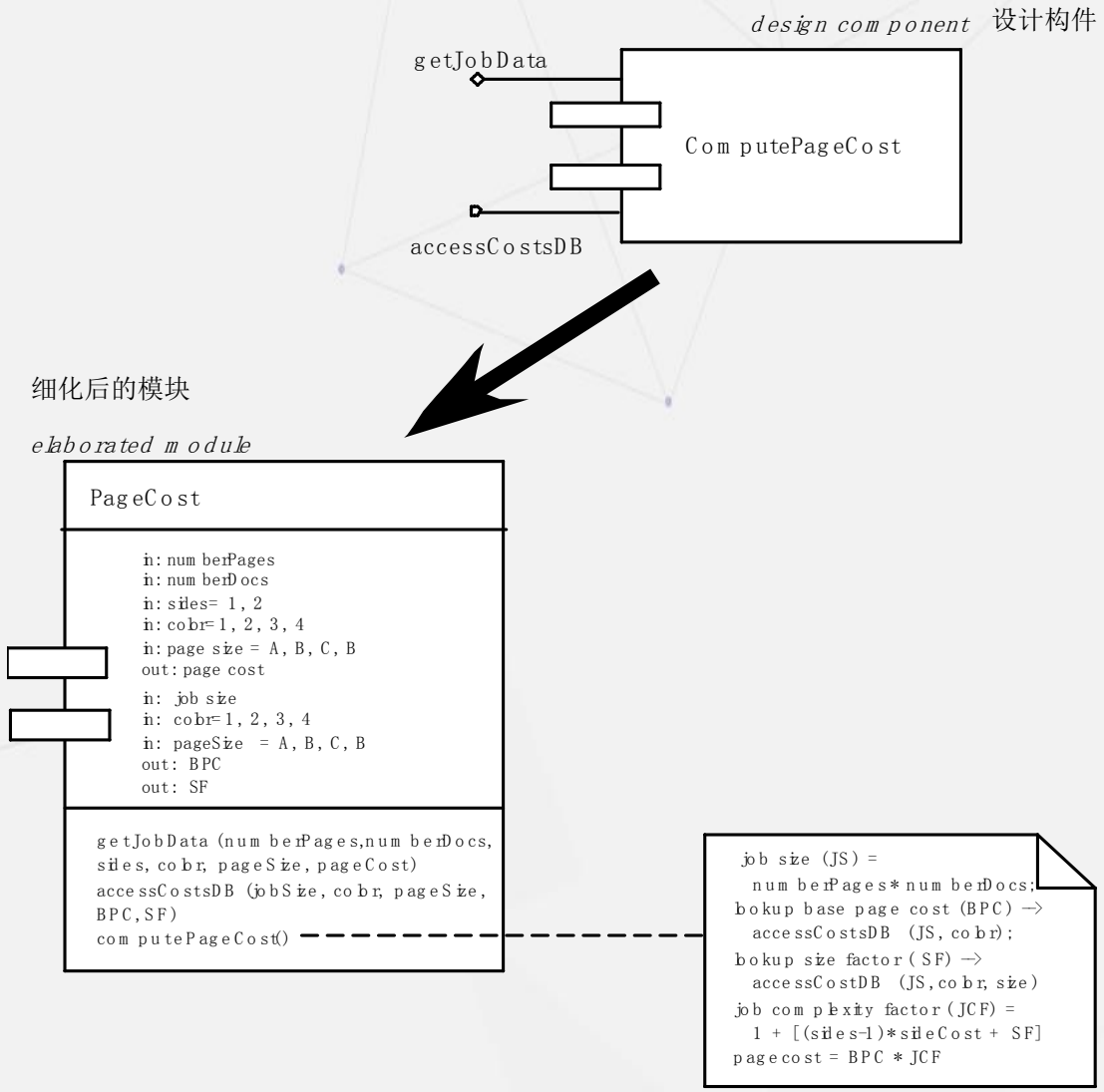
13.1.2 传统的观点

复杂系统都是由简单系统演化而来

传统控件也称为**模块**，作为软件体系结构的一部分，承担以下的3个角色之一：

- 控制构件，协调问题域中所有其他控件的调用；
- 问题域构件，实现客户需要的全部或者部分功能
- 基础设施构件，负责完成问题域中所需支持处理的功能

13.1.2 传统的观点


design component 设计构件
getJobData
ComputePageCost
accessCostsDB
细化后的模块
elaborated module
PageCost
in: numberPages
in: numberDocs
in: sides= 1, 2
in: cobr=1, 2, 3, 4
in: page size = A, B, C, B
out: page cost
in: job size
in: cobr=1, 2, 3, 4
in: pageSize = A, B, C, B
out: BPC
out: SF
getJobData (numberPages,numberDocs, sides, cobr, pageSize, pageCost)
accessCostsDB (jobSize, cobr, pageSize, BPC, SF)
computePageCost()
job size (JS) = numberPages*numberDocs;
lookup base page cost (BPC) -> accessCostsDB (JS, cobr);
lookup size factor (SF) -> accessCostDB (JS, cobr, size)
job complexity factor (JCF) = 1 + [(sides-1)*sideCost + SF]
pagecost = BPC * JCF

13.1.3 过程相关

- 从头开始设计
- 或使用已经验证过的设计或代码级构件目录：接口，功能，通信协作等

13.2.1 基本设计原则

- 开闭原则(OCP)。“模块[构件]应该对外延具有开放性，对修改具有封闭性”。
- Liskov 替换原则(LSP)。“子类可以替换它们的基类”。
- 依赖倒置原则(DIP)。“依赖于抽象，而非具体实现”。
- 接口分离原则(ISP)。“多个客户专用接口比一个通用接口要好”。
- 发布复用等价性原则(REP)。“复用的粒度就是发布的粒度”。
- 共同封装原则(CCP)。“一同变更的类应该合在一起”。
- 共同复用原则(CRP)..。“不能一起复用的类不能被分到一组”。

13.2.2 构件级设计指导方针

- 构件
  - 对那些已经被确定为体系结构模型一部分的构件应该建立命名约定，并对其做进一步的细化和精华，使其成为构件级模型的一部分。
- 接口
  - 接口提供关于通信和协作的重要信息(也可以帮助我们实现OPC原则)。
- 依赖与继承
  - 这种方法很好，模型依赖关系是自左向右，继承关系是自底向上(基类)。

13.2.3 内聚性

- 传统观点：
  - 模块的专一性
- OO 观点: :
  - 内聚性意味着构件或者类只封装相互关联密切，以及与构件或类自身有密切关系的属性和操作。
- 内聚分类
  - 功能的，当一个模块只完成某一组特定操作并返回结果时
  - 分层的，由包、构件和类来实现。高可以访问低，反之不可
  - 通信的，访问相同数据的所有操作在一个类中

13.2.4 耦合性

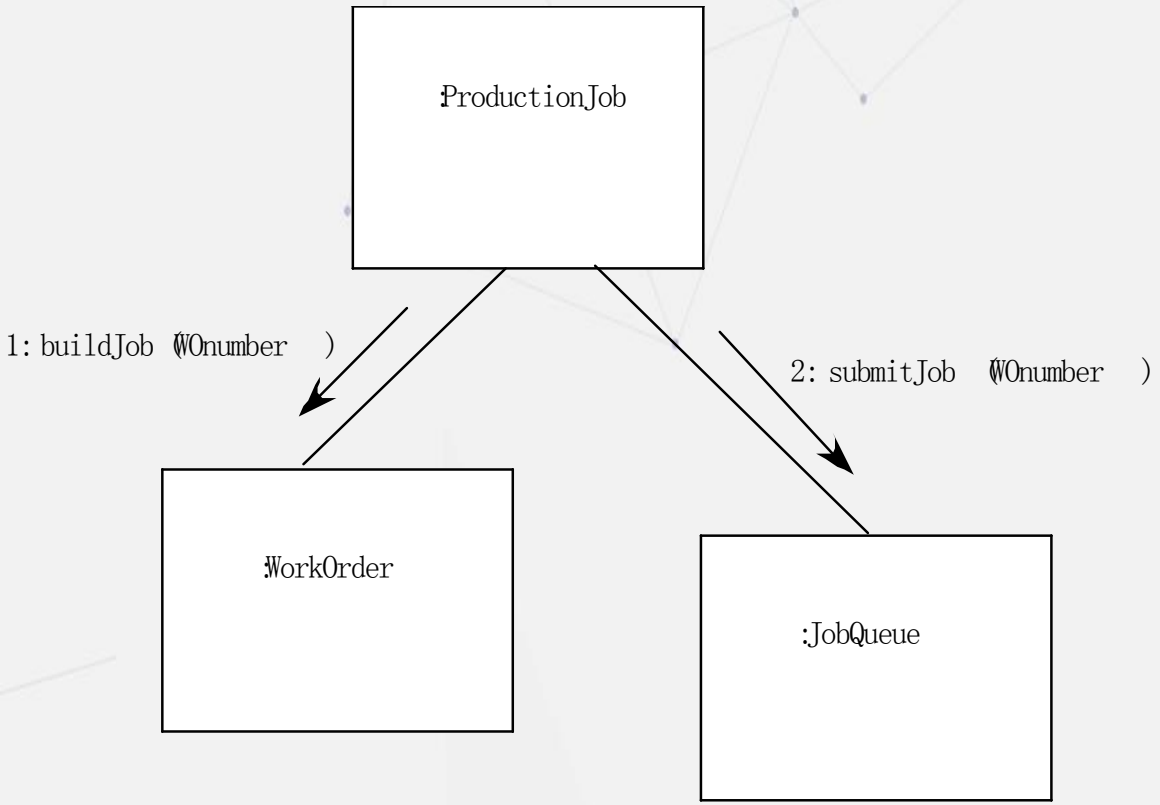
尽量保持低耦合

- 传统观点
  - 耦合是构件之间彼此联系、构件和外部世界联系程度的一种度量
- OO观点：
  - 类之间彼此联系程度的一种定性度量
- 耦合分类
  - 内容，一个构件暗中修改其他构件的内部结构
  - 共用，当大量的构件都要使用同一个全局变量时
  - 控制，当操作A调用操作B，并且向B传递控制标记时
  - 标记，当类B被声明为类A某一操作中的一个参数类型时
  - 数据，当操作需要传递长串的数据参数时
  - 历程调用，当一个操作调用另外一个操作时
  - 类型使用，当构件A使用了在构件B定义的一个数据类型时
  - 包含或者导入，当构件A引入或者包含一个构件B的包或者内容时
  - 外部，当一个构件和基础设施构件进行通信和协作时

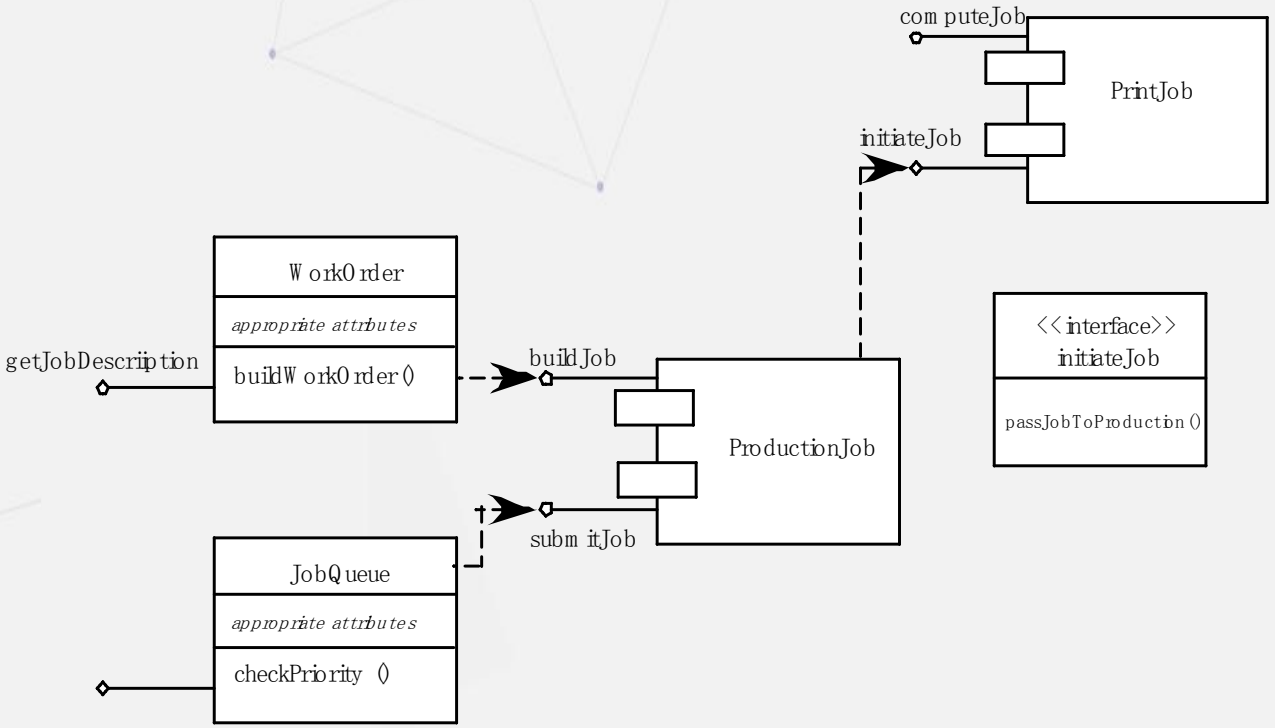
13.3 实施构件级设计

对需求模型和构件模型的一种设计表示

- 步骤1：标识出所有与问题域相对应的设计类。
- 步骤2：确定所有与基础设施域相对应的设计类。
- 步骤3：细化所有不需要作为可复用构件的设计类。
  - 步骤3a：在类或构件协作时说明消息的细节。
  - 步骤3b：为每个构件确定适当的接口。
  - 步骤3c： 细化属性，并且定义实现属性所需要的数据类型和数据结构。
  - 步骤3d：详细描述每个操作的处理流。
- 步骤4：说明持久数据源(数据库和文件)并确定管理数据源所需要的类。
- 步骤5：开发并且细化类或构件的行为表示。
- 步骤6：细化部署图以提供额外的实现细节。
- 步骤7：考虑每个构件级设计表示，并且时刻考虑其他可选方案。

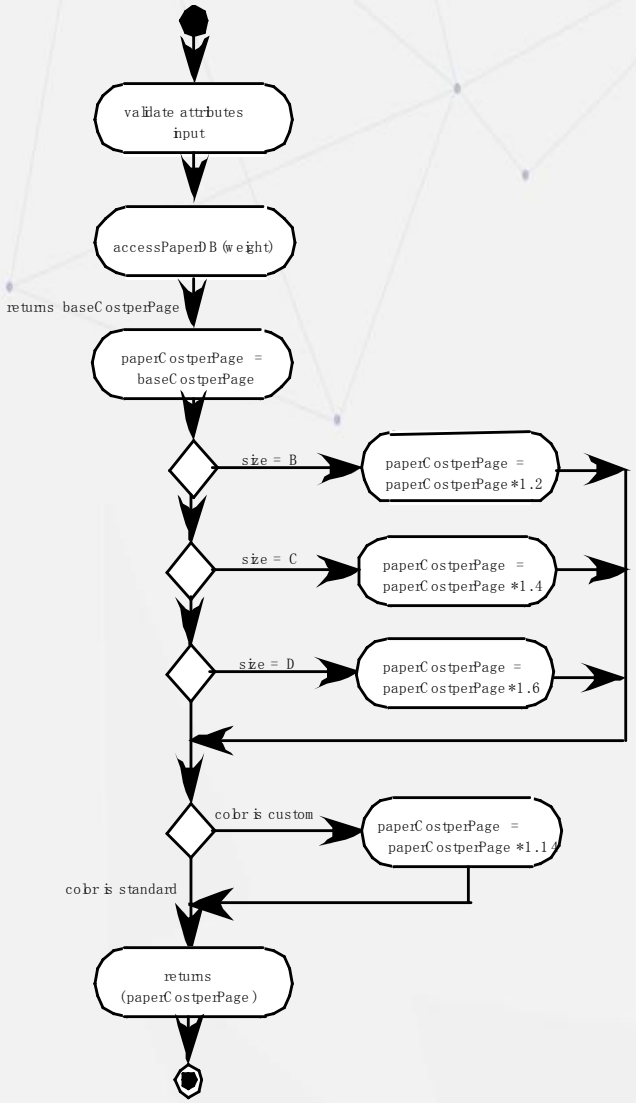

:ProductionJob
1: buildJob (WOnumber)
2: submitJob (WOnumber)
:WorkOrder
:JobQueue

带消息的协作图

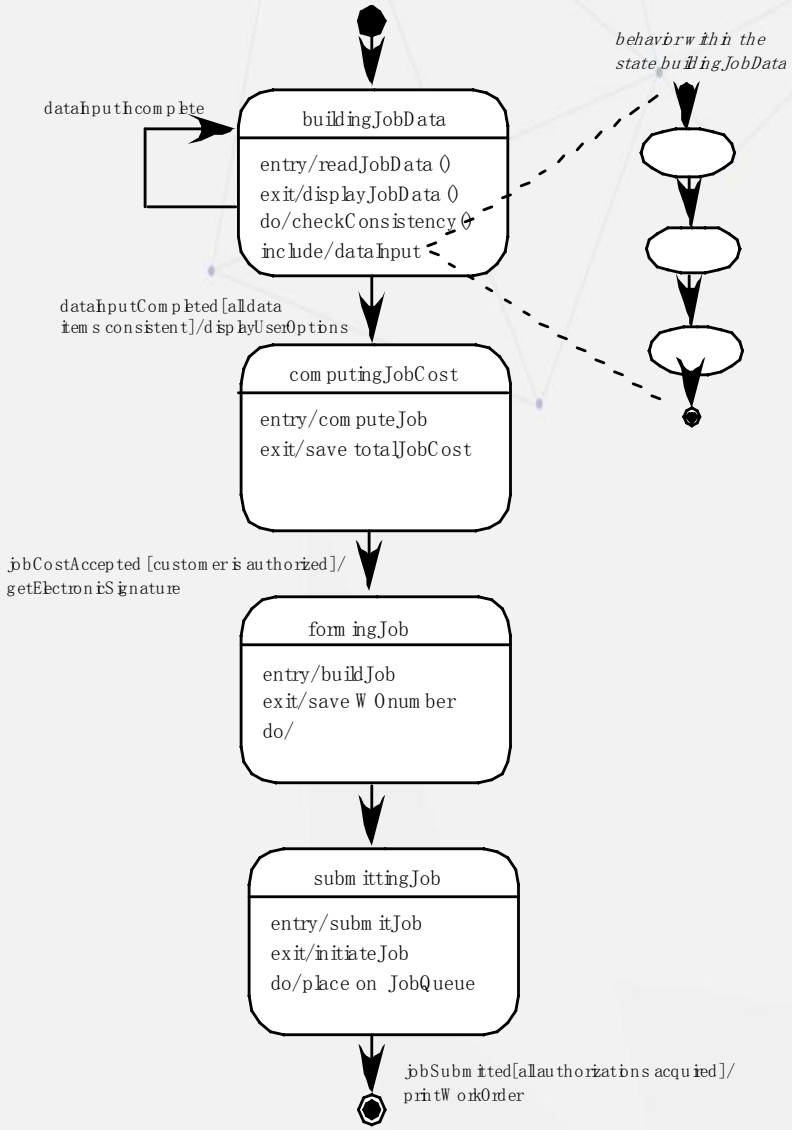

computeJob
PrintJob
initiateJob
WorkOrder
appropriate attributes
buildWorkOrder()
getJobDescription
buildJob
ProductionJob
<<interface>>
initiateJob
passJobToProduction()
submitJob
JobQueue
appropriate attributes
checkPriority ()

重构接口和类定义

活动图


validate attributes input
accessPaperDB(weight)
returns baseCostperPage
paperCostperPage = baseCostperPage
size = B
paperCostperPage = paperCostperPage*1.2
size = C
paperCostperPage = paperCostperPage*1.4
size = D
paperCostperPage = paperCostperPage*1.6
color is custom
paperCostperPage = paperCostperPage*1.1
color is standard
returns (paperCostperPage)

状态图


behavior within the state buildingJobData
dataInputIncomplete
buildingJobData
entry/readJobData()
exit/displayJobData()
do/checkConsistency()
include/dataInput
dataInputCompleted[alldata items consistent]/displayUserOptions
computingJobCost
entry/computeJob
exit/save totalJobCost
jobCostAccepted[customer is authorized]/ getElectronicSignature
formingJob
entry/buildJob
exit/saveWOnumber
do/
submittingJob
entry/submitJob
exit/initiateJob
do/place on JobQueue
jobSubmitted[allauthorizations acquired]/ printWorkOrder

13.4 WebApp的构件级设计

- WebApp 构件是

  (1) 定义良好的聚合功能，为最终用户处理内容，或提供计算或数据处理。

  (2) 内容和功能的聚合包，提供最终用户所需的功能。

因此，WebApp构件级设计通常包括内容设计元素和功能设计元素。

13.4.1 构件级内容设计

- 关注于内容对象，以及包装展示给WebApp最终用户的方式
- 以**SafeHomeAssured.com**的基于网络视频监控功能为例
  - 可以为视频监控器功能定义很多潜在的内容构件：
    - (1) 内容对象表示具有传感器与摄像头等特殊图标位置的空间布局（平面图）
    - (2) 采集到的极小的视频缩略图（每个都是一个独立的数据对象）
    - (3) 专用摄像头的视频流窗口
  - 可以对每种构架单独命名并作为一个包进行操作

13.4.2 构件级功能设计

- 现代Web应用系统提供了更加成熟的处理功能：
  - (1) 执行本地化处理，从而动态地产生内容和导航功能；
  - (2) 提供适合于WebApp业务领域的计算或数据处理；
  - (3) 提供高级的数据库查询和访问；
  - (4) 建立与外部系统的数据接口。
- 为了实现这些（及许多其他）能力，Web工程师必须设计和创建WebApp功能构件，这些构件在形式上类似于传统的软件构件。

移动App的构件级设计

- 基于Web的瘦客户端
  - 设备上仅有接口层
  - 使用web或云服务实现业务层和数据层。
- 胖客户端
  - 所有3层（接口层、业务层、数据层）均在设备上实现
  - 受到移动设备的限制。

13.5 设计传统构件

- **处理逻辑的设计**是由算法设计的基本原则和结构化编程支配
- **数据结构的设计**由系统开发的数据模型定义
- **接口的设计**是由协作支配的，协作受构件的影响

13.6 基于构件的开发

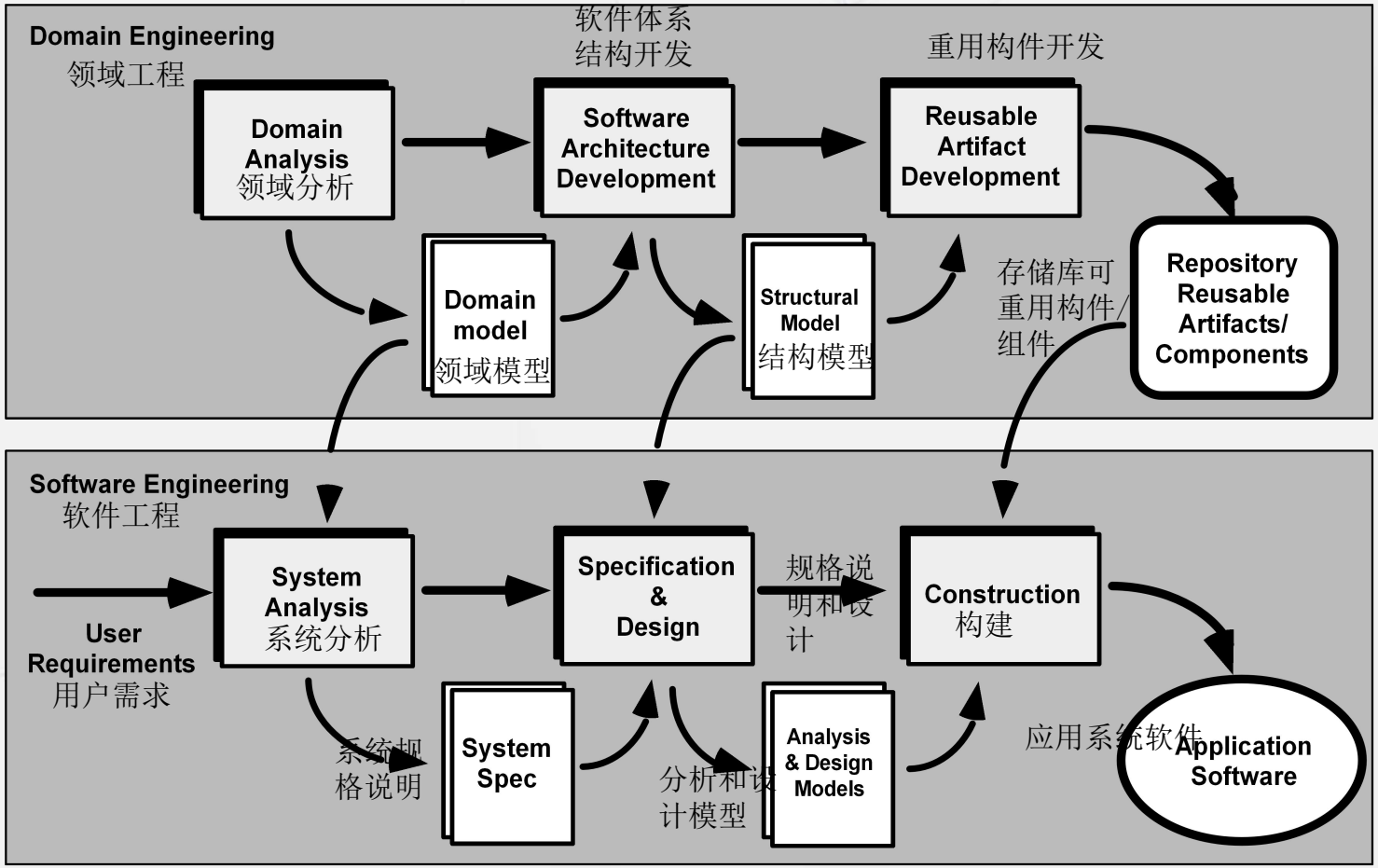
强调复用

- 当面对重用的可能，软件团队会问：
  - 商业成品构件（COTS）对实现需求是可用的吗？
  - 内部开发的可复用构件可实现需求吗？
  - 可复用构件的接口兼容待建系统的体系结构吗？
- 与此同时，重用时他们会遇到以下障碍……

复用困难

- 很少有企业和组织有任何甚至略微相似的一个全面的软件可重用计划。
- 虽然越来越多的软件厂商目前销售的工具或构件为软件重用提供直接援助，但大多软件开发者都不用他们。
- 相对较少的训练是可以帮助软件工程师和管理者理解并应用重用。
- 许多软件从业者仍然认为，“相比其价值，重用的麻烦更多。”
- 许多公司继续鼓励软件开发方法，而不促进重用。
- 很少有公司提供生产可重用的程序组件的激励措施。

CBSE过程


Domain Engineering
领域工程
软件体系
结构开发
重用构件开发
Domain Analysis
领域分析
Software Architecture Development
Reusable Artifact Development
Domain model
领域模型
Structural Model
结构模型
存储库可
重用构件/
组件
Repository Reusable Artifacts/ Components
Software Engineering
软件工程
User Requirements
用户需求
System Analysis
系统分析
Specification & Design
规格说
明和设
计
Construction
构建
系统规
格说明
System Spec
分析和设
计模型
Analysis & Design Models
应用系统软件
Application Software

13.6.1 领域工程

最终目的：复用

为软件工程师建立一种机制来分享这些构件

1. 定义待研究的领域。
2. 把从领域中提取的项进行分类。
3. 收集领域中有代表性的应用系统样本。
4. 分析样本中的每个应用系统。
5. 为这些对象开发需求模型。

确定可复用构件

- 组件功能需要在未来实现吗？
- 领域内组件的功能有多普遍？
- 领域内组件功能有重复的吗？
- 组件依赖于硬件吗?
- 在实现时，硬件保持不变吗？
- 硬件细节能被远离另一个组件吗？
- 最优设计对接下来的实现足够吗？
- 我们能参数化一个不可重用的组件使其变为可重用的吗？
- 在许多仅有一点改变的实现中，组件可重用吗？
- 通过修改使重用可行吗?
- 一个不可重用的组件能被分解生成可重用的组件吗？
- 组件如何被有效分解成可重用的？

13.6.2 构件的合格性检验、修改与组合

领域工程提供了基于构件的SE所需的可复用的构件库：

- 库中的构件必须是可供使用的
- 构件应该有一致的结构
- 存在标准，例如：
  - OMG/CORBA
  - Microsoft COM
  - Sun JavaBeans

13.6.2 检验、修改与组合

- 构件合格检验
- 构件适应性修改
- 构件组装
- 构件更新

13.6.2 检验、修改与组合

在使用构件之前，必须考虑到：

- 应用系统的编程接口(API)
- 构件所需的开发工具与集成工具
- 运行时需求，包括资源使用(例如：内存或外存储器)、时间或速度以及网络协议
- 服务需求，包括操作系统接口和其他构件的支持
- 安全特性，包括访问控制和身份验证协议
- 嵌入式设计假定，包括特定的数值或非数值算法的使用
- 异常处理

13.6.2 检验、修改与组合

“容易集成”的含义是：

(1) 对于库中的所有构件，都已经实现了一致的资源管理方法；

(2) 所有的构件都存在诸如数据管理等公共活动

(3) 已经以一致的方式实现了体系结构的内部接口与外部环境的接口

13.6.2 检验、修改与组合

- 必须建立一个基础设施以将构件绑定到一起
- 体系结构组装成分包括：
  1. 数据交换模型
  2. 自动化
  3. 结构存储
  4. 基础对象模型

Microsoft COM

- 构件对象模型(COM) 提供了在Windows操作系统上运行的单个应用系统内使用构件的规格说明，这些构件可以是不同厂商生产的。
- COM 包含两个元素：
  - COM 接口(作为COM 对象实现)
  - 为在COM接口间注册并传递消息的一组机制

Sun JavaBeans

- JavaBeans 构件系统是一个可移植的、平台独立的CBSE基础设施，是使用Java程序设计语言开发的。
- JavaBeans 构件系统包括一组工具，称为Bean开发工具箱(BDK)， 它允许开发者做以下工作：
  - 分析现有的Bean（构件）如何工作
  - 定制它们的行为和外观
  - 建立协作及通信机制
  - 开发在特定应用中使用的定制Bean
  - 测试和评估Bean的行为。

13.6.3 体系结构不匹配

- 抽象、隐蔽、功能独立、细化、结构化程序设计、面向对象、测试、SQA、正确性验证等可有助于创建可复用的软件构件

13.6.4 复用的分析与设计

- 如果不能复用，则需要创建新的构件
- 创建新的构件是要考虑到复用，则需要良好的软件设计概念和规则
    - 建立标准数据、接口和程序模板

13.6.5 构件分类与检索

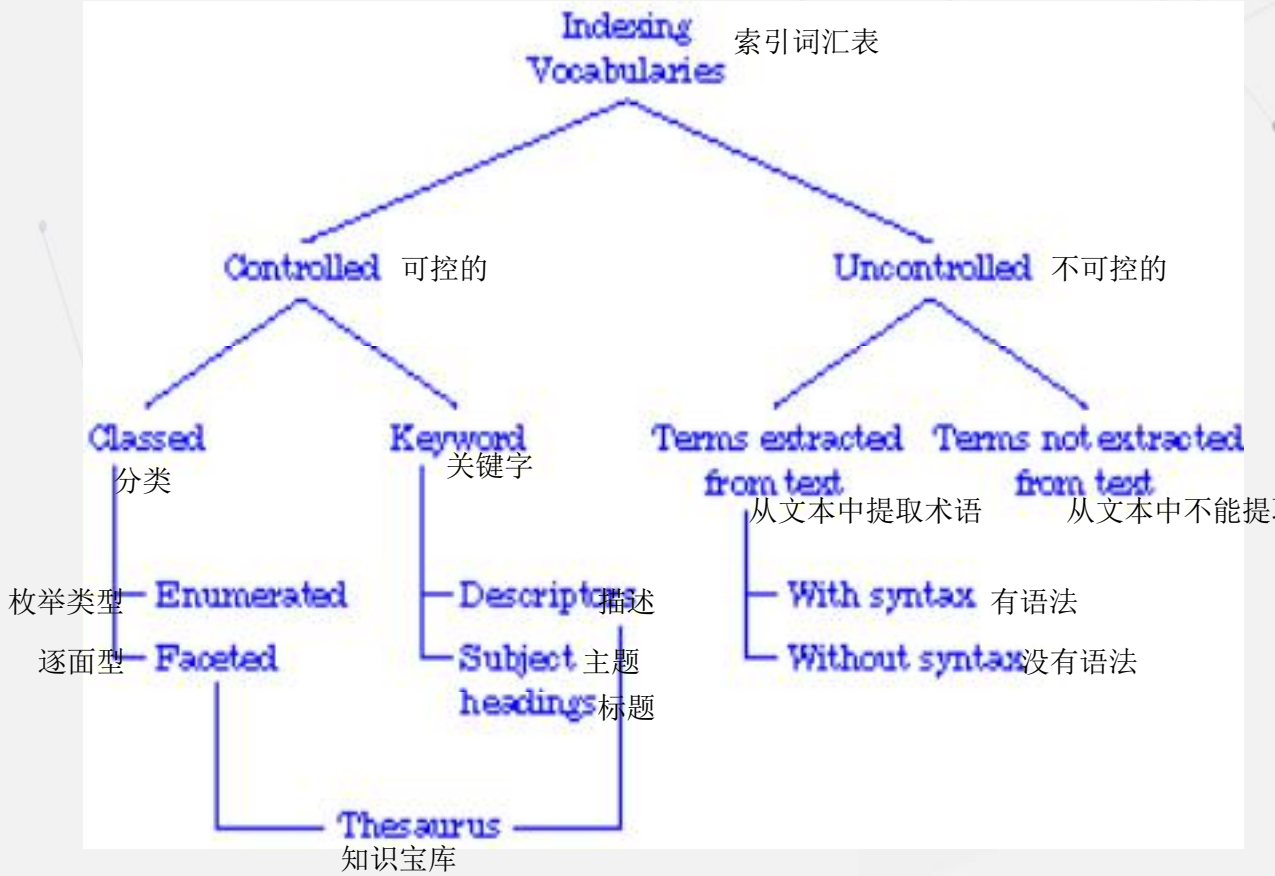
可复用软件构件的理想描述包括Tracz提出的3C模型：

概念（concept）——构件是做什么的，用来传达构件的意图。

内容（content）——描述概念如何被实现，对一般用户隐蔽信息。

环境（context）——将可复用的软件构件放到其应用领域中。通过描述概念的、操作的和实现的特征，环境使得软件工程师能够发现满足应用需求的合适构件。

检索



- 枚举类型——通过定义类中的分层结构及定义软件构件子类等级来描述构件。
- 逐面分类法——分析领域范围，并确定一组基本特征的描述。
- 属性-值分类——为所有在领域范围内的构件定义一组属性。

可复用环境特点：

- 能够存储软件构件和检索所需分类信息的构件数据库。
- 提供访问数据库的库管理系统。
- 软件构件检索系统（例如，对象请求代理），允许客户应用系统从构件库服务器中检索构件和服务。
- CBSE工具，支持将复用构件集成到新的设计或实现中。