软件详细设计

学时：48，授课24，上机24

授课内容：以案例为驱动，介绍如何应用软件工程理论、方法和技术完成软件开发。

上机内容：确定软件开发题目，完成调研、软件需求分析和软件详细设计，提交三份报告。（3人一题目）。

考核方式：

方式说明	考核方式				
	课堂表现	作业	随堂测验	上机操作	期末考试
占总成绩比例	5%	5%	10%	30%	50%
工作方式	个人	个人	个人	团队/个人	个人
成果形式	课堂验收	作业本	试卷	设计报告	试卷

参考教材

[英] 伊恩•萨默维尔 . 软件工程 第10版，机械工业出版社

张海藩，牟永敏. 软件工程导论 第6版 ， 清华大学出版社

软件详细设计报告要求

一、调研报告内容

- 收集5-10篇参考文献，和题目相关，非教材一类工具书。
- 项目目的和意义、问题定义、可行性分析

二、系统分析报告

1、引言：描述系统的需要。其中简要描述系统的功能，解释系统将如何与其他系统一起工作。

2、用户需求定义：描述为用户提供的服务。包括非功能性系统需求。使用客户可以理解的自然语言、图形或者其他表示法。描述必须遵循的产品和过程标准 。

3、系统体系结构：描述所预计的系统体系结构的高层概览，显示各个系统模块上的功能分布。

4、系统需求规格说明：更详细地描述功能性需求和非功能性需求。还可以定义与其他系统的接口。

5、系统模型：包括图形化的系统模型，显示系统构件之间以及系统及其环境之间的关系。可能的模型包括对象模型、数据流模型（数据流图）。

软件详细设计报告要求

三、软件详细设计报告

– 使用UML的面向对象设计，主要内容包括

1、理解并定义上下文以及与系统的外部交互；

– 确定系统边界，确定哪些属于、哪些不属于新系统。考虑新旧系统功能重叠，并决定新功能应当在哪里实现。

– UML活动图显示过程中活动及从一活动到另一活动控制流。

– 用例建模，建模系统与外部主体（人或其他系统）间交互；

– 或顺序图，建模系统构件之间的交互，也可包含外部主体。

2、设计系统体系结构；

3、识别系统中的主要对象；

– 对象类识别，应用领域知识来识别其他对象、属性和服务。

4、开发设计模型；使用UML应当开发以下两类设计模型。

– 结构模型，静态类及其关系描述静态结构。要描述的重要关系类型：泛化(继承)关系、使用/被使用关系、组合关系。

– 动态模型，描述系统的动态结构并展示所期望的系统对象间的运行时交互。服务请求序列及对象交互触发状态变化。

软件详细设计报告要求

- 3种UML模型对于增加用例和体系结构模型的细节特别有用。
  - 子系统模型，展示了如何对对象进行逻辑分组以构成内聚的子系统。可以使用一种类图来表示、其中每个子系统表示为一个包含对象的包。子系统模型是结构模型。
  - 顺序模型，动态的。对象的交互序列。顺序图或协作图。
  - 状态机模型，动态模型，UML状态图，表示对象如何在事件响应中改变自己状态。

5、刻画接口。

  - 接口设计关注刻画一个对象或一组对象的接口的细节。这意味着要定义由一个对象或者一组对象所提供服务的型构（signature）和语义。接口在UML中可以用类图一样的表示法来刻画。然而，接口没有属性部分，而名称部分应当包含UML构造型（stereotype）«interface» 接口的语义可以使用对象约束语言（Object Constraint Language，OCL）来定义。

第1章 软件工程概述

- 目标：介绍软件工程概念，为理解其他部分内容提供框架。
  - 理解软件工程是什么，为什么它很重要；
  - 理解开发不同类型软件系统可能需要不同的软件工程技术；
  - 理解对于软件工程师很重要的道德和职业问题；
  - 了解4个不同类型的软件系统，后续都将以它们为例。
- 软件是抽象、不可触摸的，不受物质材料限制，也不受物理定律或加工过程的制约：一方面使软件工程得以简化，因为软件的潜能不受物理因素的限制；另一方面，由于缺乏自然约束，软件系统也就很容易变得极为复杂，理解它会很困难，改变它价格高昂。不同类型的软件需要不同的方法。
- 软件失效很多都源于以下两方面的因素：
  - 不断增长的系统复杂性。
  - 未有效采用软件工程方法。

1.1专业化软件开发

- 大多数软件开发都是专业化的活动，是为满足某种业务目标，嵌入其他设备中，或作为软件产品（如信息系统、CAD等）。
- 专业化开发与个人化开发的关键区别在于，专业化软件除了开发者之外还有其他用户会使用，而且专业化软件通常都是由团队而非个人开发的。
- 专业化软件在其生命周期内要不断维护和修改。
- 软件工程的目的是支持专业化的软件开发，而非个人编程。
- 软件工程包括支持软件规格说明、设计和维护的相关技术，而这些通常都与个人化的软件开发无关。
- 一些常见的问题及其回答说明软件工程（见下表）

1.1专业化软件开发

问题	答案
什么是软件?	程序+文档。软件可以针对特定客户开发，也可以面向通用的市场开发。
好的软件具有哪些特性?	好的软件应当向用户提供所需功能与性能，好的可维护性、可依赖性和可用性。
什么是软件工程?	软件工程是工程学科，涵盖了软件生产各个方面，从初始的构想到运行和维护。
基本的软件工程活动有哪些?	软件规格说明、软件开发、软件确认和软件维护。
软件工程和计算机科学有什么区别?	计算机科学关注理论和基础，而软件工程则关注开发和交付有用的软件的实践。
软件工程和系统工程有什么区别?	系统工程关注基于计算机的系统开发的所有方面，包括硬件、软件和过程工程。软件工程是这个更加泛化的过程的一部分。
软件工程面临的关键挑战是什么?	应对不断增长的多样性、缩短交付时间以及开发可信软件的要求。
软件工程的成本有哪些?	软件开发占总成本60%，测试占40%，定制化软件，维护成本常超过开发成本。
最好的软件工程技术和方法是什么?	虽然所有的软件项目都必须进行专业化的管理和开发，但适合于最好的软件工程技术和方法是不同类型的系统的技术各不相同。如游戏开发总是需要使用系列的原型，而安全关键的控制系统开发则要求开发一个完整并且可分析的规格说明。没有任何方法和技术适用于所有系统。
互联网给软件工程带来了哪些不同?	互联网不仅带来了大规模、高度分布式、基于服务的系统的开发而且在互联网的支持下创造了改变软件经济模式的移动App产业。

1.1专业化软件开发

- 讨论软件工程时，软件不仅包括程序，而且还包括所有使程序能够正常使用的相关文档、库、支持网站、配置数据等。
- 系统可能包含多个程序以及用于设置这些程序的配置文件。系统还可能包括描述系统结构的系统文档、解释如何使用该系统的用户文档，及告知用户下载最新产品信息的Web网站。
- 软件产品有以下两类。
  - 通用软件产品，软件组织开发，市场上销售，独立使用。
  - 定制化软件产品，特定客户委托，软件承包商专门为客户设计和实现。
- 区别：通用，软件规格说明由开发者自己确定，开发过程中遇到问题，可以重新思考所要开发的东西；定制化软件的软件规格说明是由客户给出，开发者必须按客户要求进行开发。
- 界限越来越模糊，通用基础+定制，如ERP

1.1专业化软件开发

- 专业化软件系统的基本特性。

产品特性	描述
可接受性	软件对于目标类型的用户而言必须是可接受。这意味着软件必须可理解、有用，并且与用户使用的其他系统相兼容
可依赖性和安全信息性	软件可依赖性包括一系列特性，如可靠性、信息安全性、安全性。可依赖的软件即使在系统失效时也不应当导致物理或经济上的破坏。软件必须保证信息全性安全，使得恶意用户无法访问或破坏系统
效率	软件不应当浪费系统资源，如存储和处理器周期。因此，效率包括响应性、处理时间、资源利用情况等
可维护性	软件应当能够通过维护满足客户变化的需求。是一关键属性，因为软件变更是一个变化的业务环境不可避免的要求

1.1.1 软件工程

- 软件工程是工程学科，涉及软件生产的各个方面，从最初的系统规格说明直到投入使用的系统维护，都属于其学科范畴。
  - 工程学科。工程师让事物运转起来。适当的地方应用理论、方法和工具。
  - 软件生产的各个方面。软件工程不仅仅关注软件开发的技术过程，它也包括其他一些活动，例如软件项目管理以及支持软件开发的工具、方法和理论的开发。
- 软件工程中所使用的系统化方法称为“软件过程”。是指实现软件产品开发的活动序列。包含以下4项基本活动：
  - 软件规格说明，客户和工程师定义软件及其运行的约束；
  - 软件开发，对软件进行设计和编程实现；
  - 软件确认，对软件进行检查以确保它是客户所需要的；
  - 软件维护，对软件进行修改以反映客户和市场需求的变化。

1.1.1 软件工程

- 软件工程与计算机科学和系统工程都相关。
  - 计算机科学关注支撑计算机和软件系统的基础理论和方法，而软件工程则关注软件开发过程中的实际问题。
  - 系统工程关注复杂系统的开发和维护的各个方面，在这类系统中软件扮演着重要的角色。
- 不同类型软件。没有放之四海而皆准的软件工程方法和技术。
- 4个相关的问题对许多不同类型的软件都有影响。
  - 异构性。跨网络运行的分布式系统。
  - 企业和社会的变革。企业和社会变革。要能够快速地修改现有的软件同时开发新的软件。
  - 信息安全与信任。软件能够取得人们的信任是很重要的。
  - 规模。软件开发必须能在很大范围内支持不同规模的系统。

1.1.2 软件工程的多样性

- 软件工程是生产软件的系统化的方法，它考虑了现实的成本、进度、可靠性等问题，以及软件客户和开发者的需要。
- 特定的方法、工具和技术取决于开发软件的组织、软件的类型以及开发过程中所涉及的人。
- 没有一个通用的软件工程方法和技术适合于所有系统和公司。
- 因为软件不同特点，每类系统需要专门软件工程技术。
  - 汽车上的嵌入式控制系统安全性十分重要，安装时是烧录到ROM 中。修改昂贵。全面的验证和确认，以确保销售之后被召回以修复软件问题可能性最小。用户交互少，因此不需要使用一个依赖于用户界面原型的开发过程。
  - 对于交互式的基于Web的系统或移动应用，迭代式开发和交付是最好的方法，其中系统由可复用的构件组成。

1.1.3互联网软件工程

- Web浏览器的发展使得浏览器可以运行小程序并且做些本地处理，这导致了业务和组织软件的变化。
- 软件部署在Web服务器上。修改和升级软件变得更加便宜。这也降低了成本，因为用户界面开发非常昂贵。
- 公司会选择将软件系统移动到基于Web的交互模式上。
- 软件即服务21世纪初提出（SaaS）。现已经成为基于Web的系统产品（Google移动应用、Office 365）交付的标准方法。
- 越来越多软件运行在远端“云”上非本地服务器。
- Web的出现使业务软件的组织方式发生了剧烈的改变。
- 现在，软件是高度分布式的，有时会跨越整个世界。业务应用不再是从头开始编写程序，而是包含了对构件和程序的大规模复用。

1.1.3互联网软件工程

- Web的出现使**业务软件的组织方式**发生了剧烈的改变
- 这种变化已经对基于Web系统的软件工程造成巨大影响。
  - 软件复用已经成为构建基于Web的系统的主流方法。
  - 人们普遍承认提前确定这些系统的所有需求是不切实际的。基于Web的系统总是增量开发和交付的。
  - 软件可以使用面向服务的软件工程来实现，其中软件构件是独立的Web服务。
  - AJAX和HTMLS等界面开发技术出现，支持Web浏览器中的富客户端界面的创建。
- 软件工程的基本思想同样适用于基于Web的软件。
- 基于Web的系统正变得越来越大，因此应对规模和复杂性的软件工程技术与这些系统相关。

1.2 软件工程职业道德

- 软件工程是在一个社会和法律框架中进行的，这个框架限制了在这个领域中工作的人们的自由。
- 作为一个软件工程师，你必须接受你的工作包含更广阔的责任而不仅仅是技术能力的应用。如果你想作为一个专业工程师得到尊重，那么你的行为必须合乎职业道德并且有责任感。
- 法律加以规范
  - 保密　工作能力　知识产权　计算机滥用
- 道德和职业行为准则
  - 公众感　客户和雇主　产品　判断力
  - 管理　职业感　同事　自己

1.3案例研究

- 4种不同类型的系统案例
- 说明软件工程的实践取决于所要开发的系统类型。
  - 案例1.嵌入式系统。这类系统中软件控制硬件设备并嵌入该设备中。嵌入式系统的典型问题包括物理尺寸、响应性、电量管理等。面向糖尿病患者的胰岛素泵控制软件系统。
  - 案例2.信息系统。这类系统的主要目的是管理和提供对信息数据库的访问服务。主要问题包括信息安全、可用性、隐私以及保持数据的完整性。医疗记录系统。
  - 案例3. 野外气象站。主要目的是从多个传感器收集数据，并以适当方式处理数据。关键需求是可靠性（甚至是在极端环境条件下）以及可维护性。一个野外气象站。
  - 案例4.支持环境。系统集成一系列软件工具来支持某种活动。 Eelipse这样编程环境可能是读者最熟悉的一种环境。

1.3.1 胰岛素泵控制系统

- 胰岛素泵是模拟胰腺运转的医疗系统。软件控制部分是嵌入式，从传感器收集数据，然后控制泵输送指定剂量的胰岛素给患者。
- 糖尿病患者使用这个系统。糖尿病是一种常见病症，是由于人体无法产生足够数量的胰岛素而引起的。胰岛素在血液中起到促进葡萄糖代谢的作用。糖尿病的传统治疗方法是长期规律地注射人工胰岛素。糖尿病患者使用一种外部仪器定期测量自己的血糖值，然后计算所需要注射的胰岛素剂量。
- 这方法问题在于，血液中的胰岛素浓度不仅与血液中的葡萄糖浓度相关，还与最后一次注射胰岛素时间有关。这种不规律的检查有可能导致血糖浓度太低（胰岛素太多）或血糖浓度太高（胰岛素太少）。短时间内的低血糖是一种比较严重的情况，会导致暂时的脑功能障碍，最后失去知觉甚至死亡。长期处于高血糖则会导致眼睛损伤、肾损伤和心脏问题。

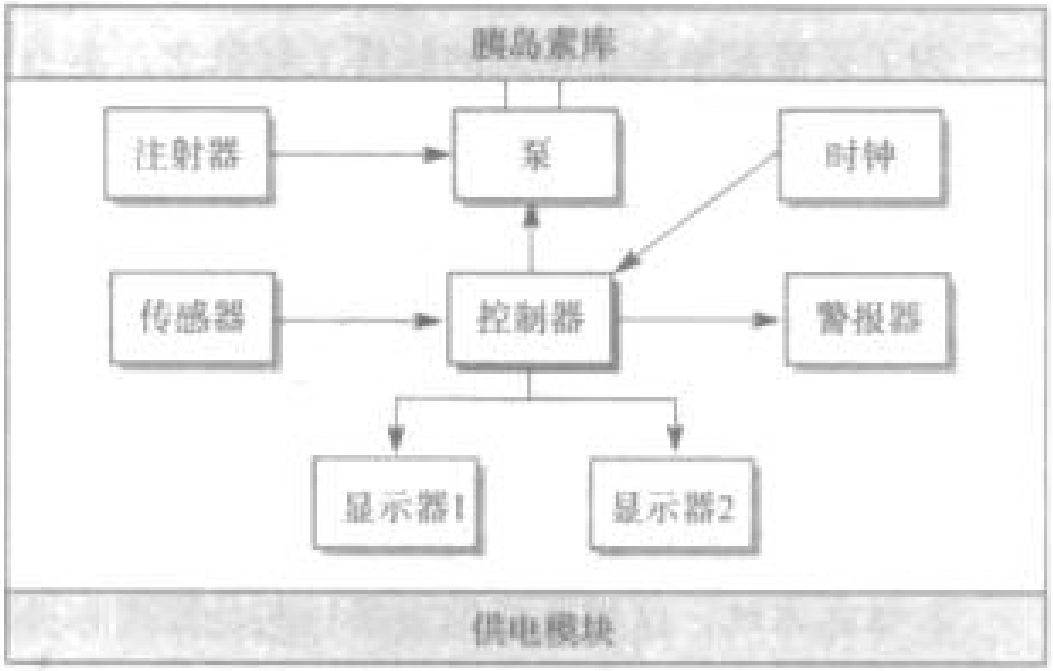
1.3.1 胰岛素泵控制系统

- 当前在微型传感器发展方面取得的进步使得自动胰岛素输送系统开发成为可能。这个系统监控血糖浓度，根据需要输送适当的胰岛素。类似这样的胰岛素输送系统现在已经有了，感觉很难控制自己的胰岛素水平的病人已经在使用这类系统了。将来还有可能将这样的系统永久地植入糖尿病患者体内。

- 该系统使用一个植入人体内的微传感器来测量一些血液参数，这些参数与血糖浓度成正比。这些参数被送到泵控制器。控制器计算血糖浓度，得出胰岛素需要量，然后向一个小型化的泵发送信号使之通过持久连接的针头输送胰岛素。

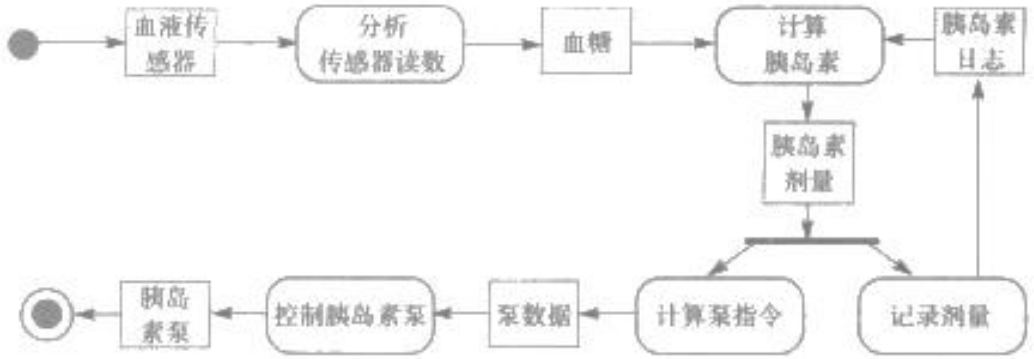
1.3.1 胰岛素泵控制系统

- 下图给出了胰岛素泵的硬件构件和组成结构。血液传感器测量血液在不同情况下的电传导率，而这些值是和血糖浓度相关的。控制器发送一个脉冲信号，胰岛素就会输送一个单位的胰岛素。因此输送10个单位的胰岛素，控制器就会向泵发送10个脉冲信号。



1.3.1 胰岛素泵控制系统

- 图是一个UML活动模型，描述了如何根据输入的血糖浓度值转换为驱动胰岛素泵的命令序列。



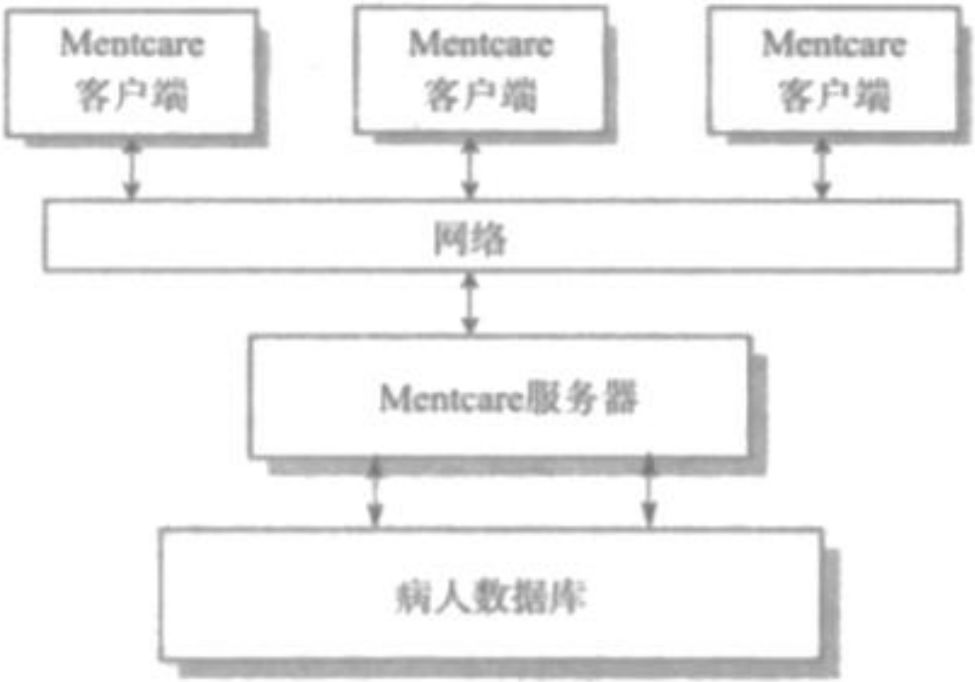
- 这是一个安全攸关的系统。如泵无法运行或者运行出问题，将会危及病人的健康，甚至使得病人因血糖浓度过低或者过高而引发昏迷或损伤脏器。系统必须满足两个关键性的高层需求：
  - 当需要时这个系统应当能够输送胰岛素；
  - 可靠运行，并根据当前血糖浓度输入正确剂量的胰岛素。
- 因此，系统的设计和实现必须确保能满足这些需求。后面更多具体需求，并讨论如何确保系统的安全性。

1.3.2 心理健康治疗病人信息系统

- 支持心理健康治疗的病人信息系统（Mentcare系统）是一个医疗信息系统，用于维护心理健康问题的病人信息以及治疗信息。多数有心理问题的人不需要住院治疗，但是要定期去那些了解他们详细病情的专科诊所看医生。为方便病人看病，这些诊所有的在医院里，也有在当地私人诊所或社区中心。
- Mentcare系统（下图）是一个在诊所使用的病人信息系统。使用集中式的病人信息数据库，也可以在笔记本上使用，这使得该系统可以在没有安全的网络连接的地方访问和使用。在有安全的网络连接时，他们使用数据库中的病人信息，但是可以下载病人记录的本地拷贝并在没有网络连接时使用。这个系统不是一个完备的医疗记录系统，因此没有记录其他的医疗信息。然而，它可以与其他诊所信息系统交互并交换数据。

1.3.2 心理健康治疗病人信息系统

- 该系统有以下两个目的:
  - 生成管理信息，使健康服务部门能够据此评估本地和政府目标的实现情况;
  - 及时为医护人员提供相关信息以支持对病人的治疗。



1.3.2 心理健康治疗病人信息系统

- 病人有时候会失去理性并有些紊乱，会错过预约时间、故意或意外遗失处方和药品、不听医生嘱咐、对医护人员提出无理要求、不期而至等。少数还可能对自己和他人造成危害。可能常换住址，有时会长期或短期离家出走。如病人具有危险性，可能需要“隔离”，限定在安全医院接受治疗和观察。
- 用户包括诊所工作人员，如医生、护士和健康随访员（上门护士）。非医务用户包括负责预约的接待员、负责维护系统医疗数据的工作人员，以及负责生成报告的行政人员。
- 系统负责记录病人信息（姓名、地址、年龄等）、诊疗情况（日期、医生、主观印象等）、病人状况和治疗方案。系统定期产生报告供医务人员和卫生部门使用。提供给医护人员的报告关注的是单个病人的信息，而提供给管理者的报告则会进行匿名化处理，主要关注的是总体状况、治疗费用等。

1.3.2 心理健康治疗病人信息系统

- 系统的主要特征是:
  - 病倒管理。医生可为病人创建记录、编辑信息、查看病人历史等。系统支持数据汇总，新医生也可以快速了解病人的主要问题和当前的治疗情况。
  - 病人监测。系统定期监测那些正在接受治疗的病人的记录，若发现可疑问题就会发出提醒。因此，若某个病人长时间没看医生，系统就会发出通知。监测系统重要特点是，能够对强制隔离的病人保持跟踪，以确保在正确的时间能够对其进行合乎法律要求的例行检查。
  - 管理报告。系统产生月度管理报告，显示每个诊所接治的病人数目、进入和离开护理系统的病人数目、采取强制隔离的病人数目、处方药物的使用情况及其价格等。

1.3.2 心理健康治疗病人信息系统

- 两项法律条款影响系统。一个是个人信息保密性的数据保护法；一个是有关强制监禁被认为对病人自己和他人会造成伤害的病人的心理健康法。心理健康在这方面是很独特，因为它是唯一可以违背病人的意愿、建议对其进行看管的医学专业。受到非常严格立法保护。Mentcare系统目的之一是确保工作人员总是能够按照法律规定工作，且对病人所有处置都能被记录下来并在必要时候用于司法审查。
- 与所有医疗系统一样，隐私是关键性系统需求。病人信息是严格保密，不能暴露给除相关医护人员和病人以外任何人。
- Mentcare系统是一个安全攸关的系统。一些精神疾病可能导致病人自杀或者对其他人造成人身伤害。系统应尽可能向医护人员警示潜在的有自杀倾向或者有危害倾向的病人。

1.3.2 心理健康治疗病人信息系统

- 系统的总体设计必须同时考虑隐私和安全两个方面的需求。
- 系统必须在需要时是可用的；否则安全性就会大打折扣，使得医生无法及时为病人拿出正确的治疗方案。

- 存在一个潜在的冲突。
  - 当系统数据只有单个拷贝时，隐私保护是最容易做到的。
  - 然而，为确保在服务器失效或没有连接网络时的系统可用性，需要维护多份数据拷贝。
- 后续章节将会讨论这些需求之间的权衡问题。

1.3.3 野外气象站

- 为了监控偏远地区的气候变化、提高气象预报的准确度，那些幅员辽阔的国家的政府会选择在偏远地区部署几百个气象站。这些气象站通过一组装置来采集气象数据，比如温度、气压、光照、降雨、风速和风向。
- 野外气象站只是一个更大的系统的一部分（见图1-7），该系统是一个从气象站采集数据并将其提供给其他系统处理的气象信息系统。

1.3.3 野外气象站

- 野外气象站系统包括:
  - 气象站系统。收集气象数据，初始处理，传输给DB系统。
  - 数据管理与存档系统。从所有的野外气象站收集数据，进行数据处理与分析，将数据存储为容易被其他系统（如天气预报系统）检索的格式。
  - 气象站维护系统。可以通过卫星与所有野外气象站通信，监控它们运行状态，并报告出现的问题。还可以更新这些气象站上的嵌入式软件。当某个野外气象站系统出问题时，该系统还可以用来远程控制气象站系统。
- 上图中使用UML包的符号表示每个系统都是一个构件的集合，并且采用UML构造型《system》表示所有的独立系统。包之间的关联表示包与包之间存在信息交换。

1.3.3 野外气象站

- 气象站有许多采集各种天气数据的仪器，如风速、风向、气温、气压、24小时降雨量等。这些设备都是在软件系统的控制下周期性地读入并管理所采集到的数据。
- 气象站系统进行气象数据观察的频率很高，如对温度的测量需每分钟一次。然而，由于卫星带宽相对较窄，气象站系统需本地对数据进行一些处理和存储。当数据收集系统请求数据时，气象站系统提交本地数据。如连接不成功，气象站系统在本地继续保留数据，直到通信恢复。
- 气象站是独立电池供电，且完全自我管理。通信是通过速度相对较慢的卫星连接。气象站必须包含自我充电机制，如太阳能或风能。由于它们是部署在野外的，直接暴露在各种恶劣环境条件下，还有可能被动物毁坏。

1.3.3 野外气象站

- 气象站软件不能仅仅进行数据采集，还必须做到以下几点：
  - 监控仪器、电源、通信硬件，并向管理系统报告故障。
  - 管理系统电源，确保电池在环境条件允许的情况下能够充电，也确保在恶劣天气情况（例如大风天气）下及时关闭发电机以免受到破坏。
  - 允许动态配置，在部分软件版本更新时或者当系统失效而切换备份装置时。
- 气象站必须是自我管理和无人看管。这就意味着尽管数据采集功能本身相对简单，但所安装的软件是复杂的。

1.3.4 学校数字化学习环境

- 使用交互式软件系统支持教育既可提高学习者兴趣又可以使得学生更深刻地理解知识。然而，对于什么是“最好的”计算机支持的学习策略并没有广泛共识。在实践中，教师通常会使用多种不同交互式、基于web的工具来支持学习。所用的工具取决于学习者的年龄、他们的文化背景、他们的计算机使用经验、可用的设备以及相关教师的偏好等。
- 数字化学习环境是一个框架，包含一些通用工具及针对特殊目的设计的工具，外加一组满足用户需求的应用。框架提供身份认证、同步和异步通信以及存储等通用服务。
- 该环境每个版本包含的工具由教师和学习者根据需要选择。工具可以是电子表格等通用的应用，也可以是管理作业提交和评分、游戏和模拟的虚拟学习环境等学习管理应用。还可以包括一些特定的内容。

1.3.4 学校数字化学习环境

- 右图是面向3~18岁学生在校内使用的数字化学习环境（iLearn）的高层体系结构模型。采用分布式系统设计，该学习环境所有的构件都是可以从互联网上任何地方访问的服务。不要求所有的学习工具汇集在一个地方。

1.3.4 学校数字化学习环境

- 这是一个面向服务的系统，其中所有的系统构件都被认为是一个可替换的服务。该系统中有以下3种类型的服务。
  - 公共服务。提供与应用无关的基础功能，可以被系统中的其他服务所使用。公共服务通常是特别针对该系统开发或修改的。
  - 应用服务。提供特定的应用（例如邮件、会议、图片共享等）以及针对特定教育内容（例如科学电影或历史资源）的访问手段。应用服务是为该系统特别购买的或从互联网上免费获取的外部服务。
  - 配置服务。用于使用特定的应用服务集合对学习环境进行调整和设置，并定义如何在学生、教师以及学生父母之间共享服务。

1.3.4 学校数字化学习环境

- 服务可以使用新服务替换，提供系统的不同版本以适应不同年龄用户需要。系统必须支持以下两个层次上的服务集成。
  - 集成服务，提供应用编程接口（API），其他服务可以通过API进行访问。这样服务到服务的直接通信就成为可能。例如身份认证服务，其他服务可以调用身份认证服务来对用户进行认证，而不是使用自己的认证机制。如果用户已经通过认证，身份认证服务可以通过API将认证信息直接发送给另一个服务，用户不需要重新对自己进行认证。
  - 独立服务，其运行与其他服务无关，可以直接通过浏览器界面进行访问。可通过显式的用户动作（如复制粘贴）与其他服务共享信息。
- 如果一个独立服务得到了广泛使用，开发团队可以集成该服务，从而使之成为一个集成的和支持性的服务。

本章要点

- 软件工程是一门覆盖软件生产的各个方面的工程学科。
- 软件不仅是程序，还包括系统用户、质量保证人员以及开发者所需要的所有电子文档。软件产品的基本属性是可维护性、可依赖性、信息安全性、效率以及可接受性。
- 软件过程包括软件开发过程中所涉及的所有活动。软件规格说明、开发、确认和维护这些活动是所有软件过程的一部分。
- 世界上存在着很多不同类型的系统。每一种类型的系统的开发都需要一种与之相适应的软件工程工具和技术。几乎不存在适用于所有类型系统的软件设计和实现技术。
- 软件工程基本思想适用于所有的软件系统。包括受管理的软件过程、软件可依赖性和信息安全性、需求工程和软件复用。
- 软件工程师对软件工程行业和整个社会负有责任，不应该只关心技术问题，而且应该对影响他们工作的道德问题有所知晓。

第2章 软件过程

- 目标：介绍软件过程（软件生产的一组相互连贯的活动）的思想。
  - 理解软件过程和软件过程模型的概念；
  - 了解3个通用的软件过程模型以及它们的适用情形；
  - 了解需求工程、开发、测试和演化等基本软件过程活动；
  - 理解软件过程有效地组织以应对软件需求和设计上的变化；
  - 理解软件过程改进的思想以及影响软件过程质量的因素。

- 软件过程是完成软件产品生产的一组相互关联的活动。
- 有许多不同类型的软件系统，没有放之四海而皆准、适用于所有类型系统的软件工程方法。也没有放之四海而皆准的软件过程。
- 不同企业中所使用的过程取决于所开发的软件的类型、软件客户的需求以及开发软件的人的技能。

第2章 软件过程

- 不同软件过程必须以某种形式包含在4个基本软件工程活动中：
  - 软件规格说明。软件的功能以及对于软件运行的约束必须在这里进行定义。
  - 软件开发。必须开发出符合规格说明的软件。
  - 软件确认。软件必须通过确认来确保软件所做的是客户所想要的。
  - 软件维护。软件必须通过维护来满足不断变化的客户需要。
- 这些活动本身也是复杂的活动，还会包括一些子活动，例如需求确认、体系结构设计、单元测试等。软件过程还包括其他一些活动，例如软件配置管理、项目计划等支持软件生产活动的活动。

第2章 软件过程

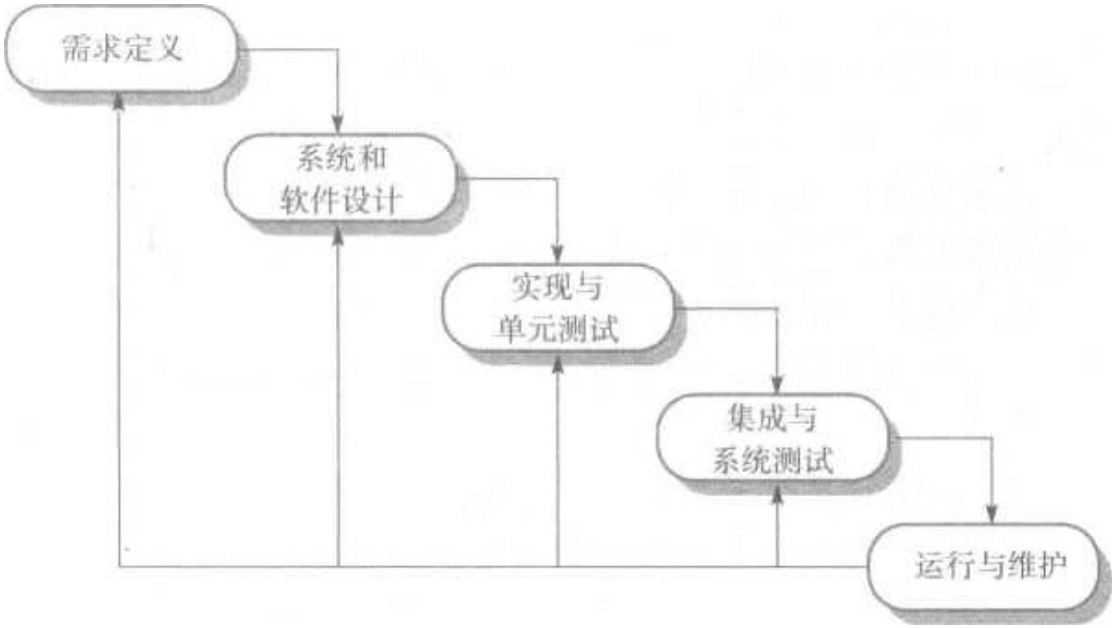
- 描述过程时，重要的是描述涉及哪些人、产生了什么以及影响活动序列的条件，具体如下。
  - 产品交付物是软件过程活动的产出物。例如，体系结构设计活动的产出物是软件体系结构模型。
  - 角色反映了参与过程的人在其中的职责。角色的例子包括项目经理、配置经理、程序员等。
  - 前置和后置条件是在一个过程活动执行之前和之后或者产品生产之前或之后须满足的条件。
- 如在体系结构设计开始前，一个前置条件可能是客户已认可所有需求；活动结束后，一个后置条件可能是描述体系结构的UML模型已进行评审。

2.1软件过程模型

- 软件过程模型（软件开发生命周期或SDLC模型）是软件过程的简化表示。表现软件过程的一个侧面，只提供过程的部分信息。
- 通用过程模型是软件过程的高层和抽象描述，用于解释不同的软件开发方法，可以看作是一种过程框架，可以通过扩展和调整来创建更加特定的软件工程过程。
- 本章讨论几个通用的过程模型：
  - 瀑布模型。包含基本的过程活动(规格说明、开发、确认、维护)，并将它们表示为独立的过程阶段，如需求规格说明、软件设计、实现、测试。
  - 增量式开发。使规格说明、开发和确认活动交错进行。系统开发体现为一系列的版本（增量），每版增加一些功能。
  - 集成和配置。依赖于可复用的构件或系统。系统开发过程关注在新的使用环境中配置构件并将它们集成为一个系统。

2.1.1瀑布模型

- 称为瀑布模型或软件生命周期模型。将软件开发过程表示为一些阶段。名称由来是该模型描述从一个阶段流动到另一个阶段。
- 瀑布模型是计划驱动的软件过程。原则上，至少应该在软件开发开始之前对所有的过程活动进行计划和进度安排。



2.1.1瀑布模型

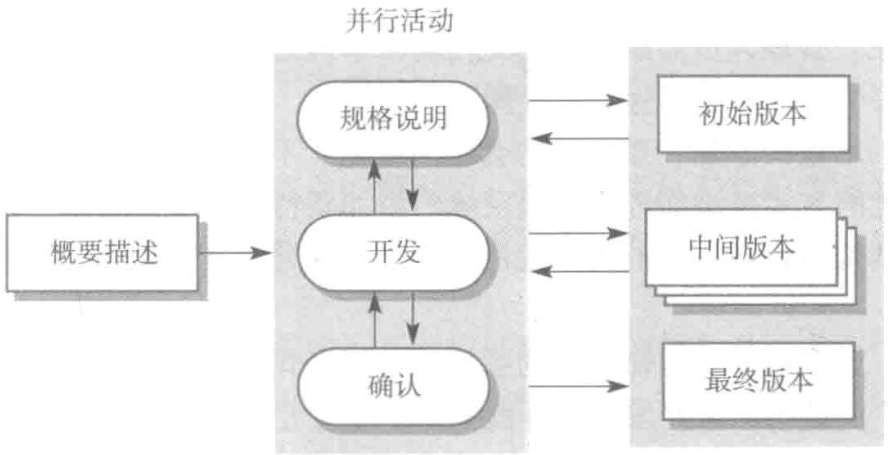
- 瀑布模型中的阶段直接反映以下这些基本的软件开发活动。
  - 需求分析和定义。通过咨询用户建立系统的服务、约束和目标，详细定义这些信息并作为系统的规格说明。
  - 系统和软件设计。系统设计过程将需求分配到硬件或软件系统上，建立一个总体的系统体系结构。软件设计涉及识别并描述基本软件系统抽象及它们之间的关系。
  - 实现和单元测试。将软件设计实现为一组程序或程序单元。单元测试验证每个单元是否满足其规格说明。
  - 集成和系统测试。各程序单元或程序集成为完整的系统，并进行测试以保证满足软件需求。测试之后，软件系统交付给客户。
  - 运行和维护。这是时间最长的生命周期阶段。系统投入实践使用。维护包括修复错误、改进系统单元的实现、随着新需求的发现而对系统的服务进行提升。

2.1.1瀑布模型

- 瀑布模型中每个阶段结果是一个或多个审批通过的文档。
- 后续的阶段在前一阶段结束前不应该开始。
- 实践中软件过程不是简单线性模型，而是包含从一个阶段到另一个阶段的反馈。
- 在现实中，软件必须在开发过程中保持灵活并容纳变更。瀑布模型要求早期的承诺并且在实施变更时要进行系统返工。
- 适用于：
  - 嵌入式系统，其中软件必须与硬件系统连接和交互。
  - 关键性系统，要求对软件规格说明和设计的安全性和信息安全进行全面的分析。
  - 大型软件系统，属于更广阔的由多家合作企业共同开发的工程化系统的一部分。

2.1.2增量式开发

- 增量式开发思想是先开发出一个初始的实现，然后从用户那里获取反馈并经过多个版本的演化直至得到所需要的系统。
- 规格说明、开发和确认等活动不是分离的而是交织在一起的，活动之间存在快速的反馈。
- 增量开发已经成为最常用的应用系统和软件产品开发方法。
- 该方法可以是计划驱动的、敏捷的，或者更为常见的是这些方法的混合。
- 在计划驱动的增量式开发中，增量是提前确定的；如果采用敏捷方法，那么早期增量是确定好的，但后面的增量开发则取决于进度和客户优先级。



2.1.2增量式开发

- 增量式开发与瀑布模型相比有以下3个主要的优势。
  - 降低了实现需求变更的成本。较之瀑布模型，重新分析和修改文档的工作量要少很多。
  - 在开发过程中更容易得到客户对于已完成的开发工作的反馈意见。客户可以对软件的演示版本进行评价。并可以看到需求已经实现了多少。如果不使用增量式开发的话，客户通常会感觉从软件设计文档中判断项目进度是很困难的。
  - 即使并未将所有的功能包含其中，也使得在早期向客户交付和部署有用的软件成为可能。与瀑布模型相比，客户可以更早地使用软件并从中获得价值。

2.1.2增量式开发

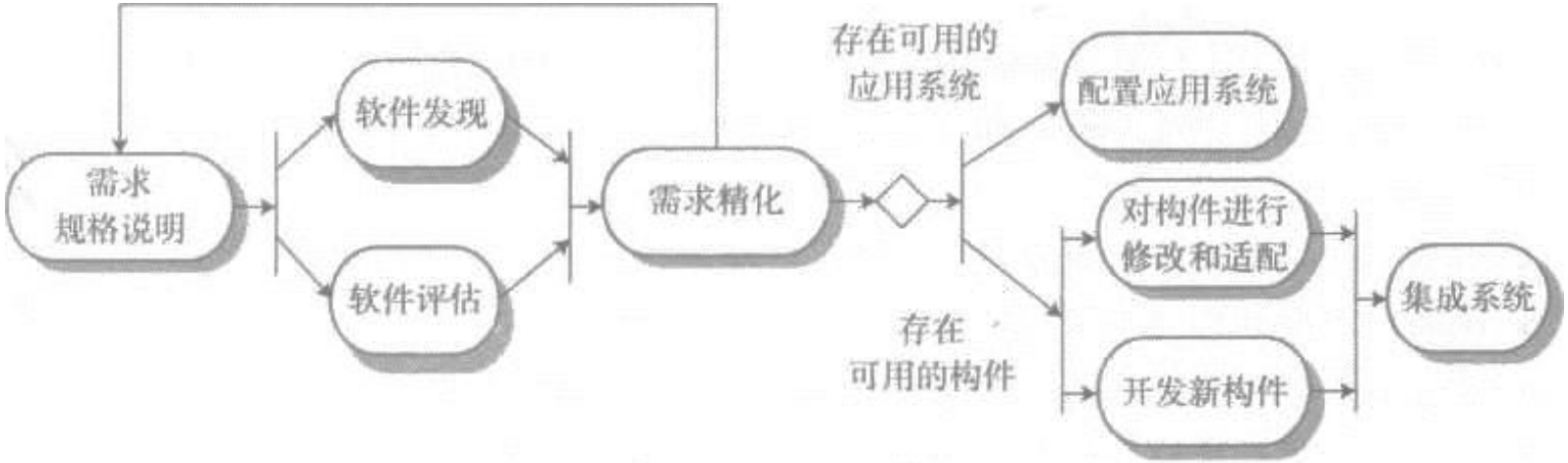
- 从管理的角度看，增量式开发方法存在两个问题：
  - 过程不可见。管理人员需要常规的交付物来掌握进度。如果系统是快速开发的，那么要产生反映系统每个版本的文档就很不合算。
  - 伴随着新的增量的添加，系统结构会逐渐退化。不断的修改导致凌乱的代码，因为新需求会以任何可能的方式被添加进来。向系统中添加新特性将变得越来越困难，成本也越来越高。为了缓解结构退化和一般的代码混乱，敏捷方法建议定期对软件进行重构（改进和结构调整）。

2.1.3集成与配置

- 大多数软件项目中都存在一定程度的软件复用。
- 2000年开始，**复用已有软件**的软件开发过程已得到广泛使用。
- 面向复用的方法依赖于一个**可复用的软件构件库**以及一个**用于构件组装的集成框架**。
- 3类软件构件经常被复用。
  - 经配置后可在特定环境中使用的独立应用系统。是拥有很多特性的通用系统，但必须在特定应用中进行调整和适配。
  - 作为一个构件或一个包开发的并且将与一个构件框架（例如JavaSpring框架）相集成的一组对象。
  - 按照服务标准并且可通过互联网进行远程调用的Web服务。

2.1.3集成与配置

- 一个通用的基于复用以及集成和配置的开发过程模型。



- 该过程中包括以下这些阶段（5个）
  - 需求规格说明。系统的初始需求被提出。
  - 软件发现和评估。搜索提供所需要的功能的构件和系统。然后对候选的构件和系统进行评估，确定它们是否满足相应的基本需求以及是否适合用于当前系统。

2.1.3集成与配置

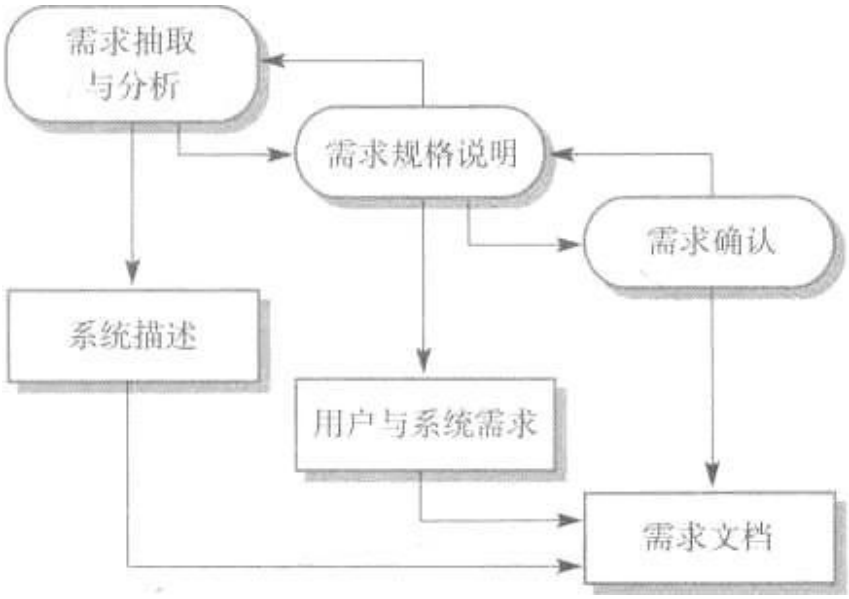
  - 需求精化。利用所发现的可复用构件和应用的信息对需求进行精化。
  - 应用系统配置。如果一个满足需求的成品应用系统是可用的，那么可以对该系统进行配置，然后使用以创建新系统。
  - 构件适配和集成。如果没有成品系统，那么可以对各个可复用构件进行修改并开发新构件。接着，这些构件被集成到一起来创建系统。
- 基于配置和集成的面向复用的软件工程在降低软件开发量以及降低成本和风险方面有着明显的优势。
- 该方法通常还可以实现更快的软件交付。
- 需求权衡不可避免，可能导致系统不完全满足用户真实需求。
- 采用面向复用的开发方法还会失去一些对系统维护的控制，因为可复用构件的新版本并不在使用该构件的组织的控制下。

2.2过程活动

- 软件过程中技术活动、协作活动以及管理活动交织在一起，其总体目标是完成一个软件系统的规格说明、设计、实现和测试。
- 软件开发者可以使用很多软件工具来帮助自己工作
  - 如需求管理系统、设计建模编辑器、程序编辑器、自动测试工具、调试器等。
- 4个基本的过程活动——规格说明、开发、确认、维护。
  - 不同的开发过程中的组织方式也各不相同，
  - 在瀑布模型中，这些活动被组织为一个序列；
  - 在增量式开发中，这些活动交织在一起。
- 这些活动如何开展取决于所开发的软件的类型、开发者的经验和能力，以及开发此软件的组织的类型。

2.2.1软件规格说明

- 软件规格说明或需求工程过程的目的是理解和定义系统需要提供哪些服务，以及识别对于系统开发和运行的约束。
- 需求工程一个特别关键的阶段，这个阶段所犯的错误将不可避免地在后续的系统设计和实现阶段造成问题。
- 需求工程过程开始前，企业可能会进行可行性或市场研究。
- 需求通常在两个细节层面上进行陈述。
  - 最终用户和客户需要一个需求的高层陈述；
  - 系统开发者需要一个更详细的系统规格说明。



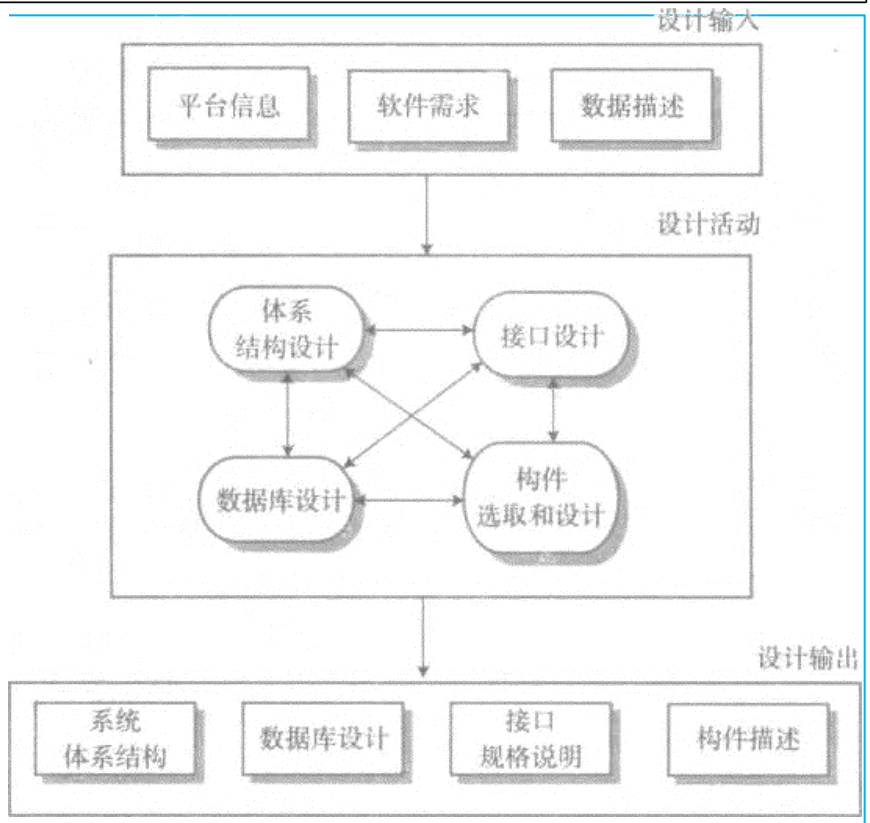
2.2.1软件规格说明

- 需求工程活动中有以下3个主要活动。
  - 需求抽取与分析。该过程通过观察已有的系统、与潜在的用户和采购方进行讨论、任务分析等手段，得出系统需求。
  - 需求规格说明。需求规格说明活动将需求分析中所收集的信息转化为定义一组需求的文档。包括用户需求、系统需求。
  - 需求确认。该活动检查需求的现实性、一致性和完整性。

- 需求分析在定义和规格说明过程中持续进行，并且在整个过程中都有可能出现新需求。因此，分析、定义和规格说明活动是相互交织的。

2.2.2软件设计和实现

- 软件开发的实现阶段是开发一个可执行的系统以交付给客户的过程。
- 软件设计是对要实现的软件结构、数据模型和结构、系统构件间的接口描述，还会包括所用的算法。
- 分阶段完成。在设计过程中要不断添加所需的细节，同时不断修改先前的设计方案。



- 右图是设计过程的抽象模型，显示了设计过程的输入、过程活动以及过程的输出。设计的修改是不可避免的。

2.2.2软件设计和实现

- 多数软件都会与其他软件系统交互。包括操作系统、数据库、中间件和其他应用系统。这些构成“软件平台”，即软件将会在其中运行的环境。
- 该平台的信息对于设计过程是一种重要的输入，因为设计者必须决定如何以最好的方式将系统与其环境集成在一起。
- 不同开发项目中设计过程活动的相同与否，取决于所开发的系统类型。如，实时系统不包含数据库，没有数据库设计阶段。
- 信息系统设计过程中可能包含的4个活动。
  - 体系结构设计。
  - 数据库设计
  - 接口设计
  - 构件选取和设计
- 设计过程输出是详细设计文档，设定精确和准确的系统描述。

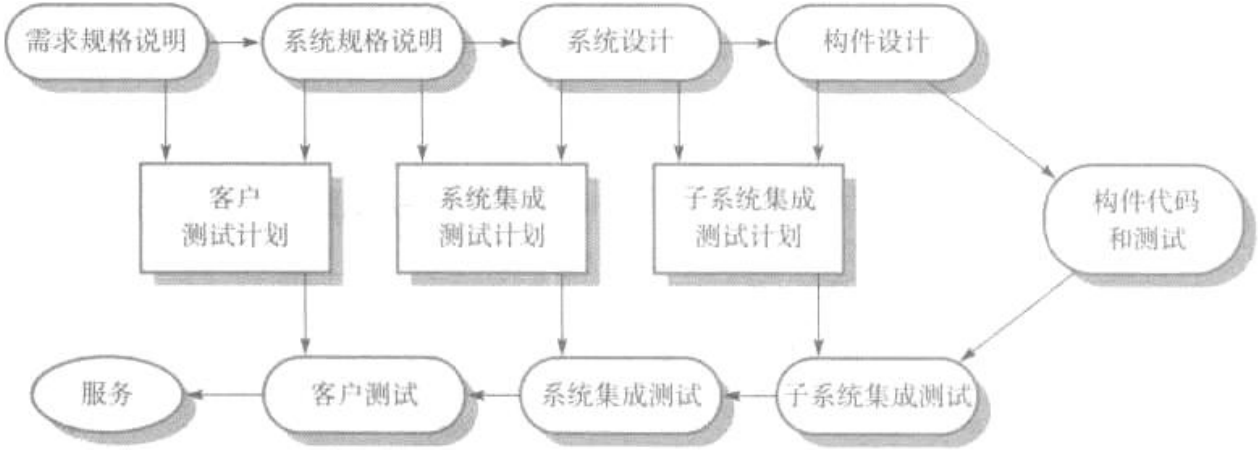
2.2.3软件确认

- 软件确认，或更一般性地，验证和确认，目的是确定系统是否符合它的规格说明，同时是否符合系统客户的期望。
- 程序测试，即用模拟测试数据运行系统，是最基本的确认技术。
- 确认还可以在从用户需求定义到程序开发的每一个软件过程阶段中包含检查性的过程（例如审查和评审）。
- 大部分验证和确认的时间和工作都是花在程序测试上。

- 测试过程包括以下这些阶段。
  - 构件测试：由系统的开发人员对组成系统的构件进行测试。
  - 系统测试：系统构件被集成到一起创建一个完整的系统。
  - 客户测试：这是系统被接受并投入运行之前的测试过程中的最后阶段。

2.2.3软件确认

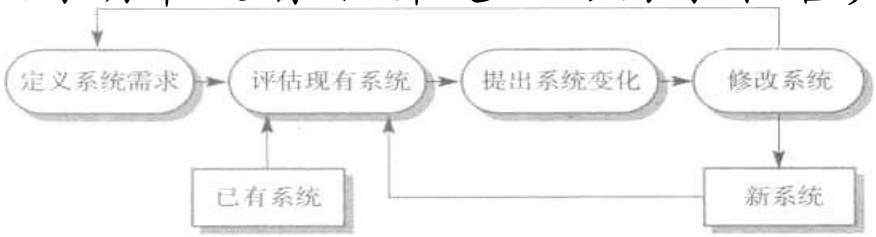
- 当使用计划驱动的软件过程时，测试是由一组测试计划驱动的。
- 独立的测试团队在基于系统规格说明和设计开发的测试计划基础上开展工作。
- 下图描述了测试计划如何将测试和开发活动链接到一起（V模型）



- 当一个系统要作为一个软件产品在市场上销售时，称为β测试。
- β测试向一些同意使用目标系统的潜在客户交付该系统。
- 他们向系统开发者报告问题。

2.2.4软件维护

- 软件的灵活性是大型、复杂系统中包含越来越多的软件的主要原因之一。
- 软件可以在系统开发之中或之后的任何时间进行修改。即使非常大范围的修改也比对于系统硬件的相应变更便宜很多。
- 软件开发过程和软件维护过程之间总是分离的。
- 开发和维护之间的区分越来越不合时宜。
- 现在很少有软件系统是全新的系统，将开发和维护视为一个连续的过程无疑是更合理的。
- 一种更加现实的认知是将软件工程视为一个演化式的过程，其中软件在其生命周期中随着不断变化的需求和客户需要而持续发生变化。



2.3应对变化

- 大型软件项目变化都无法避免。系统需求随业务应对外部压力、竞争和管理优先级的需要等变化而发生变化。
- 变化增加软件开发成本，变化意味着已完的工作须重做--返工。
- 两个相关的方法可降低返工成本。

  1.变化预测。软件过程含返工前预见或预测可能变化的活动。

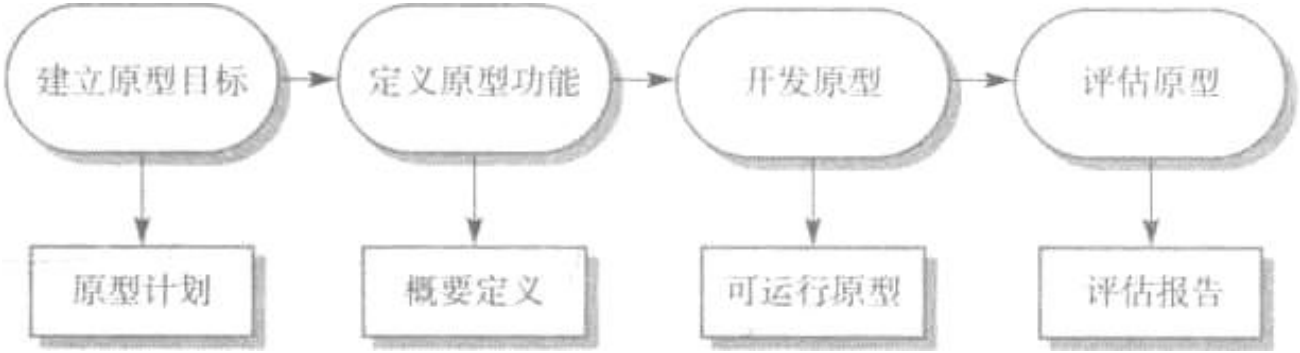
  2.变化容忍。通过过程和软件设计使得修改很容易进行。
- 两种应对变化以及修改系统需求的方法。

  1.系统原型。系统或系统的一部分的一个版本被快速开发以检验客户需求及设计决策的可行性。是一种变化预测方法.

  2.增量交付。系统增量被交付给客户进行评论和试验。这方法既支持变化预测（变化避免）又支持变化容忍。
- 重构的概念，即改进程序的结构和组织，是一种容忍的机制。

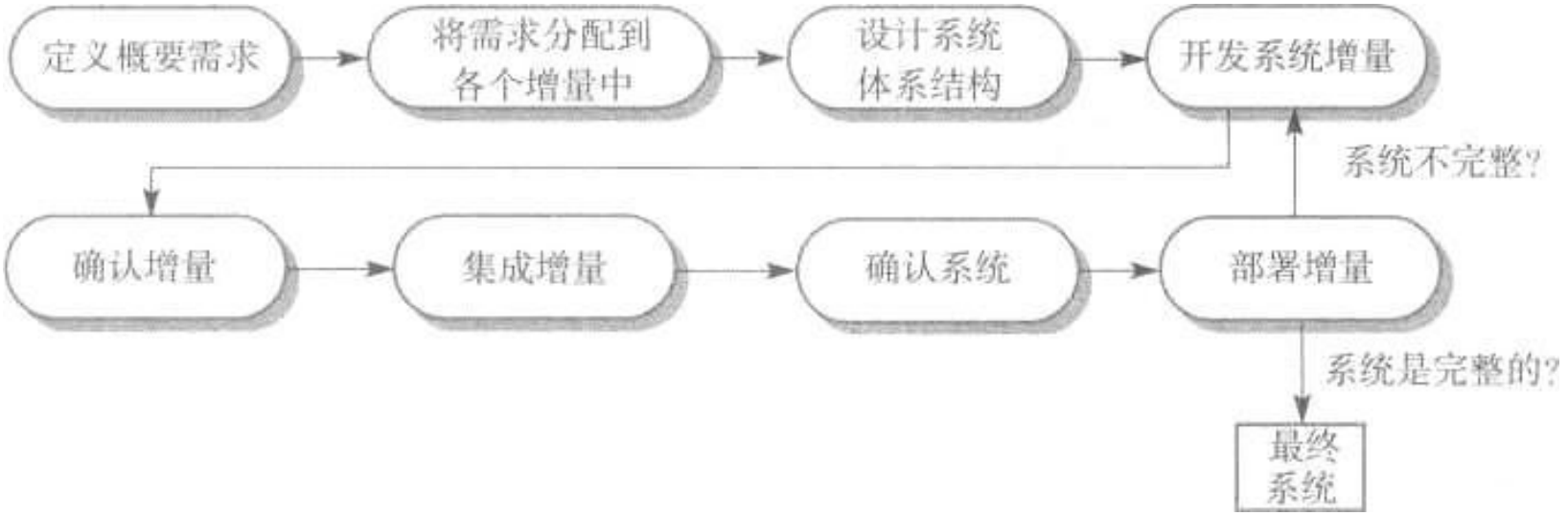
2.3.1原型

- 原型是一种软件系统的早期版本，用于演示概念、尝试候选设计方案、更好地理解问题以及可能的解决方案。
- 快速、迭代化的原型开发十分重要，可以控制成本。而系统的利益相关者也可以在软件过程的早期试用系统原型。
- 软件原型可帮助对可能需要的变化进行预测：
  1.需求工程过程中，原型可帮助对系统需求进行抽取和确认。
  2.设计过程中，原型可探索软件解决方案、用于用户界面开发
- 原型可在系统设计中进行试验，以便检查所提设计的可行性。
- 原型开发的过程模型如图所示。



2.3.2增量式交付

- 增量式交付：一部分被开发的增量会交付给客户并在他们的工作环境中进行部署和使用。在增量式交付过程中，客户定义哪些服务对他们最重要，哪些最不重要。在此基础上定义一系列交付增量，每个增量提供系统功能的一个子集。如何将服务分配到各个增量中取决于服务的优先级，其中优先级最高的服务首先被实现和交付。



2.3.2增量式交付

- 增量式交付有下面这些优势。

  1.客户可将早期增量作为原型使用，获得后续增量的需求经验。增量是真实系统一部分，系统完成时用户不需重新学习。

  2.客户不用等到整个系统交付就能从系统中获得价值。第一个增量可以满足他们最关键的需求，可以马上使用。

  3.变更可以相对容易地加入到系统中。

  4.优先权最高的服务最先交付，测试最充分。不大可能失效。

- 增量式交付也存在一些问题。

  1.当新系统替换已于系统时，迭代化交付会有问题。用户需要旧系统所有功能，不愿用一不完整的新系统试验。新旧同时使用不现实。

  2.增量实现前，需求没有详细定义，确定公共基础设施很难。

  3.迭代化过程的本质是规格说明与软件一起开发，最后才能完成规格说明。这与采购方式相冲突，因为系统规格说明是合约的一部分。

2.4过程改进

- 两种方法实现过程改进和变更

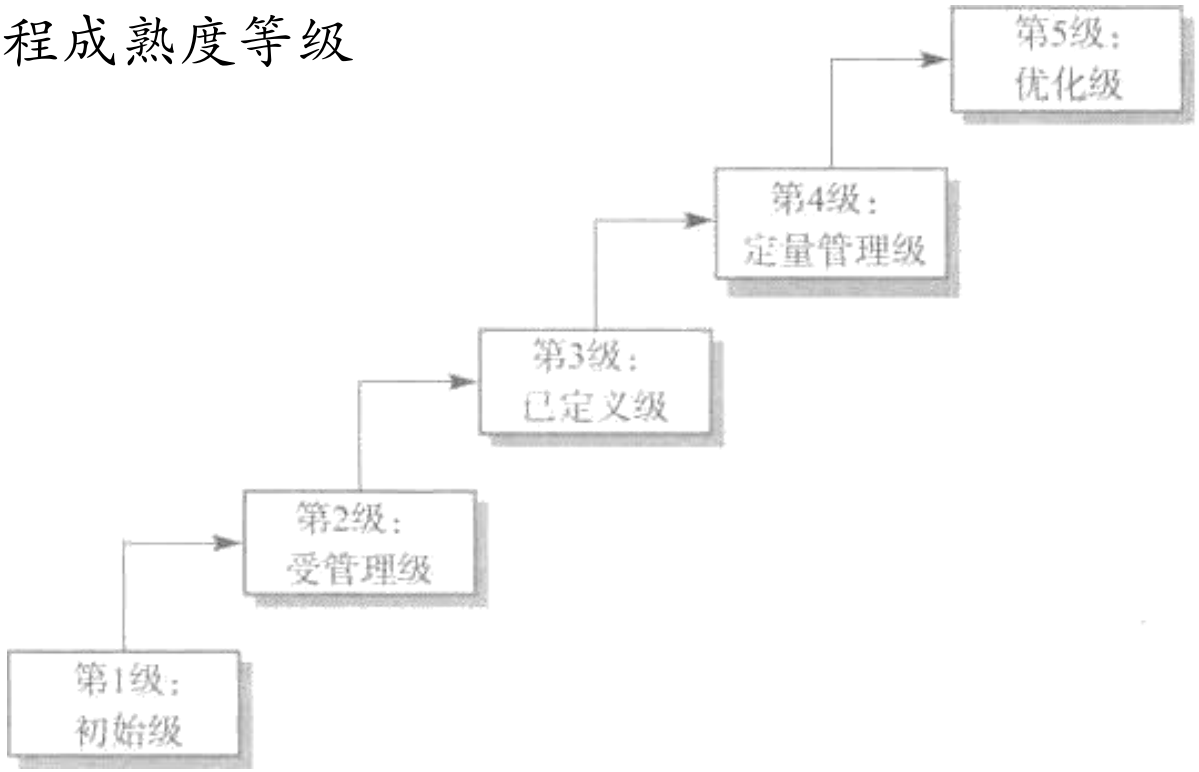
  1.过程成熟度方法，关注改进过程和项目管理，并将好的软件工程实践引入到组织中。

  2.敏捷方法，关注迭代化的开发以及降低软件过程中的额外开销，主要特点是快速交付功能以及对客户需求变更快速响应。

- 过程成熟度方法，基于过程改进的阶段如下
- 1.过程度量。对软件过程或产品的一个或多个属性进行度量。这些度量构成了一个基线，可以帮助确定过程改进是否有效。
- 2.过程分析，对当前过程进行评价，识别过程中的弱点和瓶颈。描述过程的过程模型（称为过程地图）可以在此阶段开发。
- 3.过程改变。对当前过程已识别出的过程弱点提出改变方法。引入变更后，改进循环继续收集与变化有效性相关的数据。

2.4过程改进

- 过程成熟度的思想是在20世纪80年代后期被提出的。当时，软件工程研究所（Software Engineering Institute，SEI）提出了他们的过程能力成熟度模型。
- 5个过程成熟度等级



2.4过程改进

- 5个过程成熟度等级

1.初始级。与过程域相关的目标令人满意。对于所有过程，将要进行的工作范围得到了明确定义并与团队成员进行了沟通。

2.受管理级。与过程域相关的目标得到满足，组织政策明确定义每个过程应当在何时使用。必须有文档化项目计划定义项目目标。资源管理和过程监控规程必须在整个机构中到位。

3.已定义级。关注组织的标准化以及过程的部署。每个项目都有一个受管理的过程，该过程是在一组定义好的组织过程基础上按照项目需求进行适应性调整得到的。必须收集过程资产和过程度量，并用于未来的过程改进。

4.量化管理级。存在相应的组织职责，使用统计或其他定量方法来控制子过程。即，所收集的过程和产品度量必须用于过程管理。

5.优化级。组织必须使用过程和产品度量来驱动过程改进，必须对趋势进行分析，并根据不断变化的业务需要对过程进行调整。

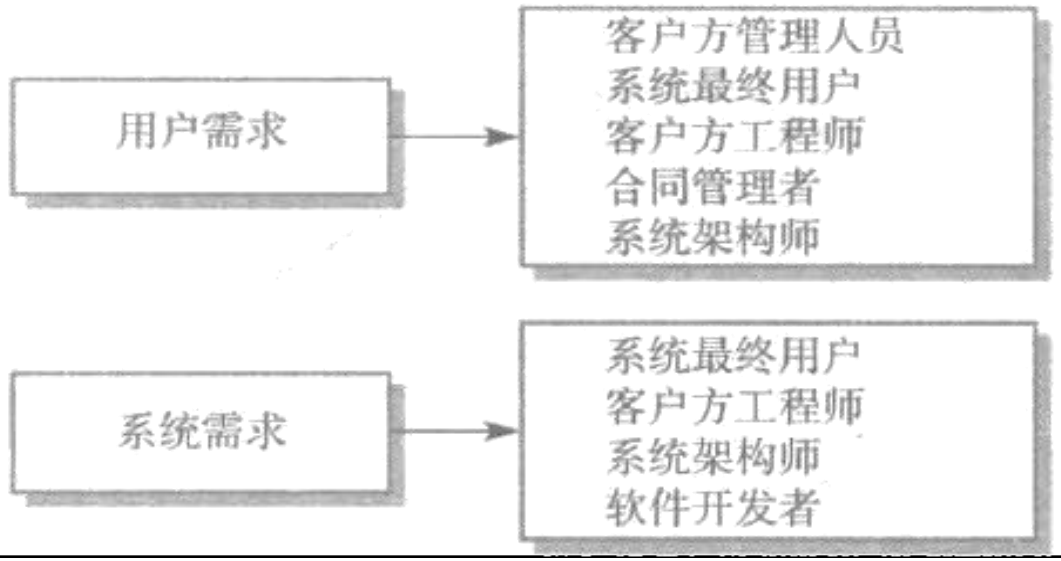
第4章 需求工程

- 本章目标：介绍软件需求，解释在发现并文档化这些需求的过程中所涉及的过程。
  - 理解用户和系统需求的概念，以及为什么这些需求应当以不同的方式进行描述；
  - 理解功能和非功能性软件需求的区别；
  - 理解主要的需求工程活动，包括抽取、分析、确认，以及这些活动之间的关系；
  - 理解需求管理的必要性，以及需求管理如何支持其他需求工程活动。
- 对一个系统的需求是关于该系统应当提供的服务以及对其运行的约束的描述。
- 需求工程：找出、分析、文档化并检查这服务和约束的过程。

第4章 需求工程

- 用户需求：使用自然语言和图形，陈述系统被期望向系统用户提供什么服务以及系统运行必须满足的约束。用户需求可以是对系统特征的大概陈述，也可以是关于系统功能的洋细和精确的描述。
- 系统需求：对软件系统功能、服务和运行约束的更详细的描述；系统需求文档(功能规格说明)精确定义要实现哪些东西。是合同的一部分。

- 向不同类型的读者传达关于系统的信息需要不同类型的需求。
- 不同类型的读者会以不同的方式使用这些需求。



第4章 需求工程

用户需求定义

1. Mentcare系统应当生成月度管理报告，显示每个诊所在当月所开的药品的成本。

系统需求规格说明

1.1 在每个月最后一个工作日，应当生成关于所开药品的汇总、它们的成本以及开药的诊所。
1.2 系统应当在当月最后一个工作日的17:30之后生成用于打印的报告。
1.3 应当为每个诊所创建一个报告，其中要列出各个药品的名字、所开的总数量、所开的剂量数字，以及所开药品的总成本。
1.4 如果药品存在不同的剂量单位（例如，10毫克、20毫克等），那么应当为每一种剂量单位创建独立的报告。
1.5 访问药品成本报告的权限应当限制，只允许管理访问控制列表中的授权用户进行访问。

图 4-1　用户需求与系统需求

第4章 需求工程

- 不同类型的文档读者都是系统利益相关者的例子。
- 从系统的最终用户、管理人员直到对系统的可接受性进行认证的外部监管者等外部利益相关者都可能是利益相关者。
- Mentcare系统包括以下利益相关者。
  - 病人以及家属；医生；护士；接待人员；IT人员；医疗伦理管理人员；医疗管理人员；医疗记录人员。

- 需求工程通常被认为是软件工程过程的第一个阶段。
- 在一个系统的采购或开发决策做出之前就开发出来。
- 建立关于系统要做什么以及系统可以提供的好处的高层视图。
- 这些需求接下来可以在可行性研究中进行考虑。
- 可行性研究结果帮助管理层决定是否继续该系统采购或开发。

4.1 功能性需求和非功能性需求

- 功能性需求。是对系统应该提供的服务、系统应该如何响应特定的输入、系统在特定的情形中应该如何表现等的陈述。某些情况，功能性需求还可明确地陈述系统不应该做什么。
- 非功能性需求。这些需求是对系统提供的服务或功能的约束，包括时间性约束、对于开发过程的约束、标准规范中所施加的约束等。非功能性需求经常适用于系统整体而不是单个的系统特征或服务。

- 不同类型的需求之间的区别并不像这里的简单定义所表达的那样清楚。我们来看一个关注信息安全的用户需求，例如，一个关于授权用户访问权限，可能看起来像是一个非功能性需求。然而，当开发得更加详细之后，这个需求可能会产生其他一些明显是功能性需求的需求，例如，要求在系统中包含用户认证的功能。

4.1.1 功能性需求

- 描述系统应该做什么。
- 这些需求取决于所开发的软件的类型、软件所期望的用户，以及该组织在书写需求时通常所采取的方法。
- 当被表达为用户需求时，功能性需求应当用自然语言描述，以使得系统用户和管理人员能够理解它们。

- 功能性系统需求将用户需求展开，是面向系统开发者描述的，应该详细描述系统功能，系统的输入、输出和异常。
- 功能性系统需求可以是关于系统应该做什么的大体上的需求，也可以是反映局部的工作方式或者组织已有系统的非常特定的需求。

4.1.1 功能性需求

- 例如，下面是Mentcare系统功能性需求的例子，用于维护那些接受心理健康问题治疗的病人的信息。

1.用户应当能够搜索到所有诊所的预约列表；

2.系统应当每天为每个诊所生成一个希望预约当天看诊的病人列表；

3.应当通过8位数字的雇员编号对使用该系统的每个工作人员进行唯一标识。

- 这些用户需求定义了系统中应当包含的一些特定的功能。
- 这些需求表明功能性需求可以在不同的抽象层次上进行描述（对比需求1和3）。
- 需求规格说明中的不精确可能导致客户和开发者之间的争执。
- 开发者通常按简化实现的方式去解读模糊的需求。然而，这不是客户所要的。为此需要建立新需求并对系统进行修改，显然会拖延系统的交付并增加成本。

4.1.1 功能性需求

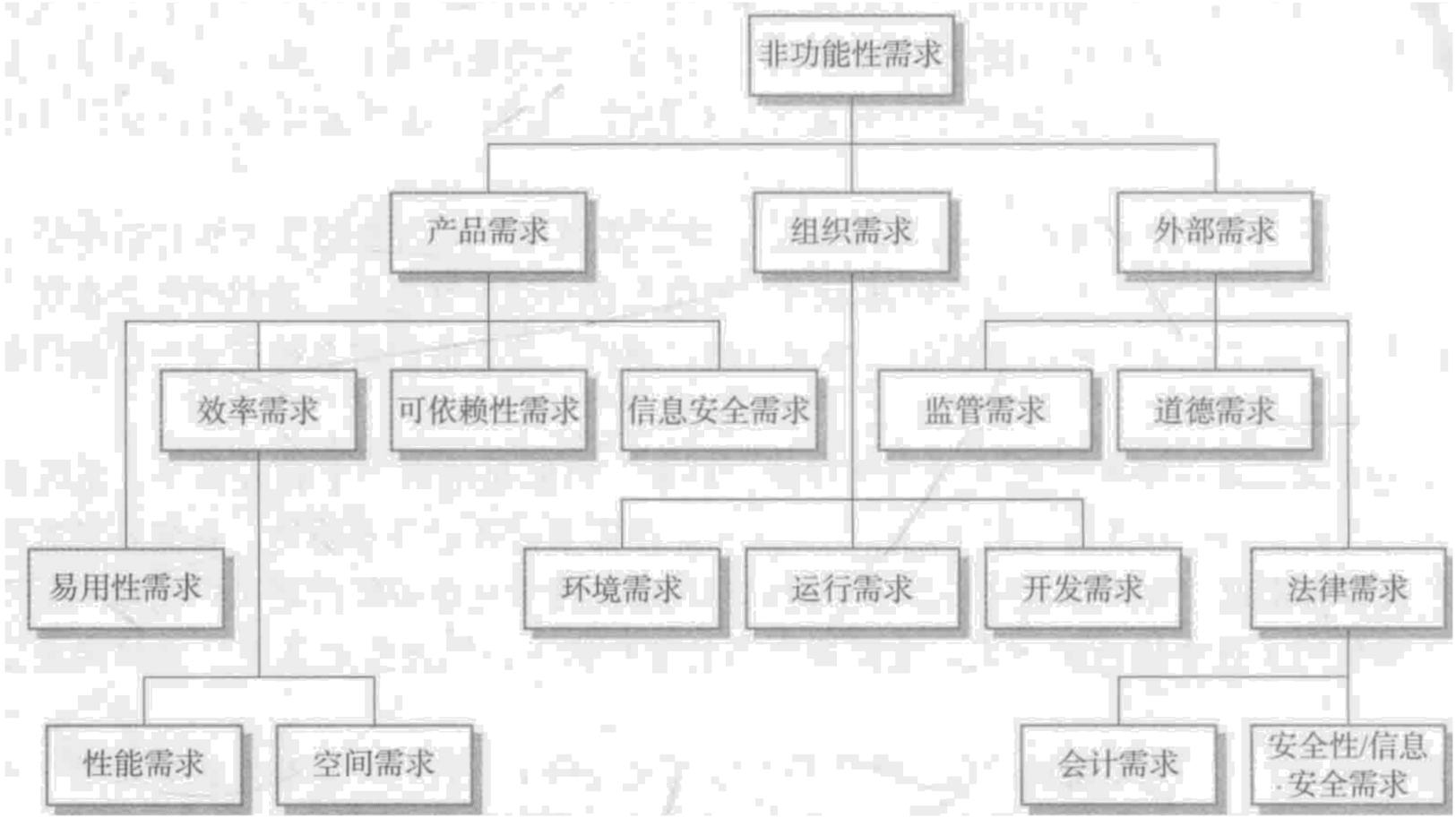
- 例如，用户应当能够搜索所有诊所的预约列表。
  - 这需求背后的考虑是，有心理健康问题的病人有时候思维混乱，他们可能在某诊所预约但实际上去了另外一家。如果他们有预约，那么将会被记录为已预约而不管是哪个诊所。
  - 提出搜索需求的医护工作人员所期望的“搜索”可能是给定一个病人名字，系统在所有诊所的所有预约中查找这个名字。然而，这需求描述并不明确。开发者会按照最容易实现的方式理解。他们所理解的搜索功能可能需要用户选择一个诊所，然后搜索预约了这个诊所的病人。这需要更多的用户输入，因此需要花更长的时间来完成搜索。
- 理想情况下，一个系统的功能性需求规约应当是完整、一致的。完整性意味着用户所需要的所有的服务和信息都应该被定义；一致性意味着需求不应当自相矛盾。

4.1.2非功能性需求

- 是指与系统向其用户提供的特定服务不直接相关的需求。
- 通常会刻画或约束系统的整体特性。
- 可能会与系统的涌现特性(如可靠性、响应时间、存储)相关。
- 或者，它们也可以定义对系统实现的约束。如，I / O能力。
- 非功能性需求经常比单个的功能性需求更关键。
- 确定哪些构件实现了非功能性需求很困难。这些非功能性需求的实现经常跨越整个系统，因为以下两点原因：
  - 非功能性需求可能影响系统整个体系结构而非单个构件。
  - 一个非功能性需求，例如信息安全需求，可能会产生多个相互联系的功能性需求，这些功能性需求定义了实现该非功能性需求所需要的新的系统服务。此外，非功能性需求还有可能产生对已有需求构成约束的新需求，例如，限制对系统中信息的访问。

4.1.2非功能性需求

- 非功能性需求的类型



4.1.2非功能性需求

- Mentcare系统非功能性需求的例子

产品需求

Mentcare 系统应当在常规工作时间（周一至周五，08:30-17:30）中对所有诊所都是可用的。任何一天在常规工作时间之内的宕机时间不应该超过 5 秒。

组织需求

Mentcare 系统用户应当使用他们的健康管理机构身份卡进行身份认证。

外部需求

系统应当按照 HStan-03-2006-priv 中的要求实现病人隐私条款。

- 产品需求定义系统必须开放时间及宕机时间可用性需求。没提任何系统功能，而清楚地明确系统设计者必须考虑的约束。
- 组织需求刻画了用户如何验证身份。都执行标准身份认证规程，替代原有用户名登录方法。刷身份卡进行身份认证。
- 外部需求源自于系统符合隐私法律的要求。要求系统开发应当符合国家的隐私标准。

4.1.2非功能性需求

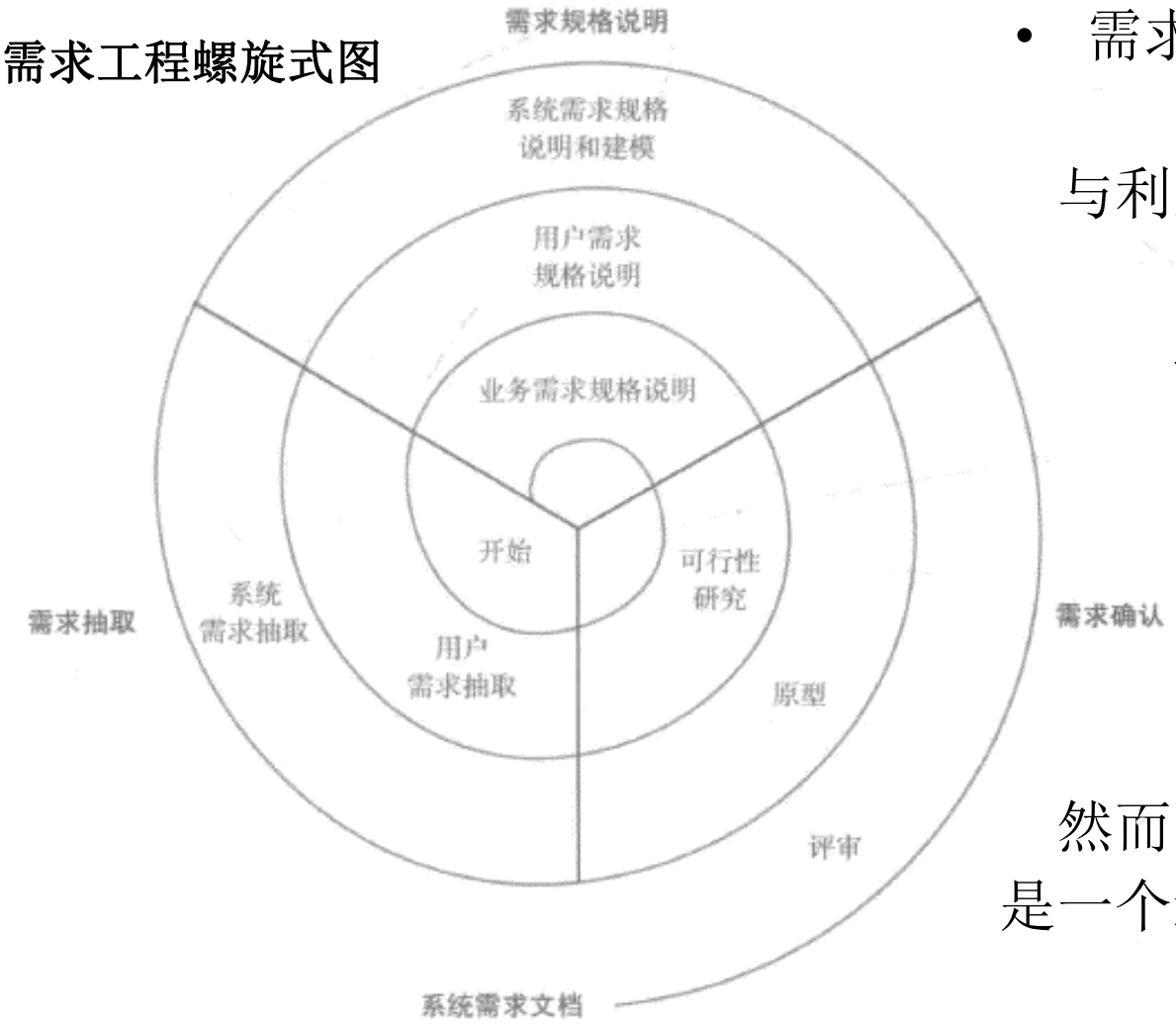
- 下图描述了可以用来刻画非功能性系统属性的度量指标，可在系统测试时对这些特性进行度量以检查系统是否满足非功能性需求。

属　　性	度量指标
速度	每秒处理的事务 用户 / 事件响应时间 屏幕刷新时间
规模	兆字节 / 只读存储器芯片数量
易于使用	培训时间 帮助画面的数量
可靠性	平均失效时间 不可用的概率 失效发生率 可用性
鲁棒性	失效后重启时间 导致失效的事件百分比 失效时数据损坏的概率
可移植性	依赖于目标的陈述百分比 目标系统的数量

- 实践中，系统的客户经常感觉很难将自己的目标表达为可度量的需求
- 非功能性需求常与其他功能性或非功能性需求冲突，如读卡器不能连PAD
- 在需求文档中很难将功能性和非功能性需求分开

4.2 需求工程过程

需求工程螺旋式图



- 需求工程包括3个关键活动
  - 抽取和分析：
与利益相关者交互发现需求
  - 规格说明：
将需求转换为标准格式
  - 确认：
检查需求是否实际上
定义了客户所
要的系统

然而，在实践中，需求工程是一个迭代化的过程，如图。

4.2 需求工程过程

- 过程早期，多数工作量花在理解高级业务和非功能性需求及用户需求上。此后，在螺旋靠外的环上更多的工作量是抽取和理解非功能性需求及更详细的系统需求。

- 在这一螺旋模型所支持的各种开发方法中，需求可以被开发到不同的抽象层次。
- 事实上所有的系统中的需求都会变化。
  - 必须对变更进行管理以理解它们对其他需求的影响，以及实施变更的成本和对系统的影响。

4.3需求抽取

- 需求抽取过程的目的是，理解利益相关者所做的事情以及他们会如何使用新系统来支持他们的工作。
- 软件工程师与利益相关者一起工作开搞清楚应用领域、工作活动、想要的服务和系统特征、系统要达到的性能、硬件约束等。
- 从系统利益相关者那里抽取和理解需求较困难，原因如下：
- 1．利益相关者不知从系统得到什么，说法泛泛；很难表达想让系统做的事情；会提出不切实际的要求，不知道哪些可行哪些不可行。
- 2．利益相关者用自己的话来表达需求，隐含他们自己工作的知识。需求工程师对业务领域没经验，无法理解需求。
- 3．不同利益相关者有各种不同的需求，不同的需求表达方式。需求工程师必须发现所有潜在的需求来源并发现共性和冲突。
- 4．政治性因素可能影响系统需求。管理人员可能会出于增加自己在组织中影响力的原因而要求某些特定系统需求。
- 5．进行需求分析时所处的经济和业务环境是动态的，不可避免地会在分析过程中发生变化。特定需求的重要性可能变化。新的需求可能会从此前没有咨询过的新利益相关者那里涌现出来。

4.3需求抽取

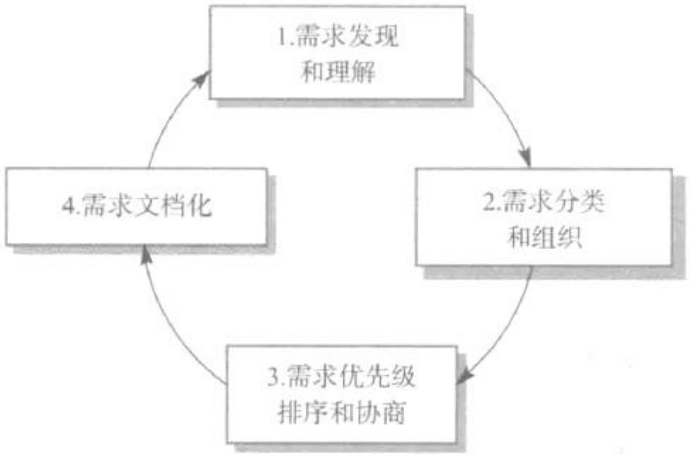
- 一个抽取和分析过程的过程模型如图所示

1．需求发现和理解：与系统利益相关者进行交互以发现需求。来自利益相关者和文档的领域需求也在此活动中发现。

2．需求分类和组织。处理所收集的未整理的需求，将相关的需求进行分组并将它们组织为内聚的聚类。

3．需求优先级排序和协商。该活动对需求进行优先级排序，找出并通过协商解决需求冲突。利益相关者必须一起协商以解决分歧并做出妥协、达成一致。

4．需求文档化。对需求进行文档化并提供给下一轮螺旋。



4.3.1 需求抽取技术

- 需求抽取包含与不同类型的利益相关者交谈以发现关于待开发系统的信息。可以补充关于现有系统及其使用的知识，以及来自不同种类文档的信息。需要花时间理解人们如何工作、他们生产什么、他们如何使用其他系统，以及他们要如何变化以适应新系统。

- 需求抽取有两个基本的方法。
  1．访谈，开发者和其他人谈论他们做的事情。
  2．观察或人种学调查，观察人们做自己的工作来了解他们使用哪些制品、他们如何使用这些制品等。
- 人种学调查是一种观察技术，可以用来理解运行过程，并且帮助得出支持这些过程的软件需求。

4.3.2故事和场景

- 人们发现，与抽象描述相比，与现实生活中例子联系会容易。
- 他们不善于告诉你系统需求。他们也许可以描述他们如何处理特定的情形，或者想象他们可能以一种新的工作方式做事情。
- 故事和场景是捕捉此类信息的手段。
- 故事和场景是一样的，它们描述系统可以如何用于一些特定的任务。故事和场景描述人们做什么，他们使用和产生什么信息，以及在此过程中他们可以使用的系统。
- 故事和场景区别在于描述的组织方式以及所呈现的抽象层次。
  - 故事被描述为叙述性的文本，并且呈现一种关于系统使用的高层描述；
  - 场景通常按照所收集的特定信息（例如，输入和输出）进行组织。
- 故事对于设定系统的“概貌”很有效；故事的一些部分可以接下来被细化并表示为场景。

4.4需求规格说明

- 需求规格说明是在需求文档中撰写用户需求和系统需求的过程：理想情况下，用户需求和系统需求应当是清晰、无二义、易于理解、完整和一致的。
- 书写系统需求的几种可能的表示法。

表示法	描述
自然语言句子	使用数字编号的自然语言句子来书写需求。每个句子应当表达一条需求
结构化自然语言	基于一个标准的表格或摸板用自然语言书写需求，每个字段提供关于需求的一个方面的信息
图形化表示法	定义系统的功能性需求，辅以文本注释的图形化模型。统一建模语言(UML)用例和顺序图被广泛使用
数学规格说明	这些表示法基于有限状态机或集合等数学概念。虽然这些无二义的规格说明可以减少需求文档中的二义性，但是大多数客户不理解形式化的规格说明，他们无法检查其是否表达了他们的想法，因此不愿意将其作为系统合约接受。

4.4需求规格说明

- 用户需求应当描述功能性需求和非功能性需求，以使不具有详细的技术知识的系统用户也可以理解。
  - 应当只刻画系统的外部行为。需求文档不应该包含系统体系结构或设计的细节。
  - 不应该使用软件术语、结构化表示法或者形式化表示法。
  - 应该用自然语言书写用户需求，带有简单的表格、表单和直观的图形。
- 系统需求是用户需求的详述版本，软件工程师将其用作系统设计的起始点。
  - 系统需求增加了细节并解释了系统应当如何提供用户需求。
  - 它们可以作为系统实现的合约的一部分
  - 系统需求应当是对整个系统的完整以及详细的规格说明。

4.4.1 自然语言规格说明

- 自然语言表达能力强、直观、具有普适性。
- 也具有潜在的模糊性和二义性，解读取决于读者的背景。
- 书写自然语言需求时尽量减少误解，遵循指南：

  1.使用标准格式并确保所有需求定义遵循该格式。使遗漏不太会发生，同时需求更易检查。尽量用一两句话自然语言书写需求。

  2.一致方式使用语言，区分必须满足和期望满足的需求。必须满足是系统必须支持的需求，用“必须”(shall)表达。期望满足不是必不可少的，用“应该”(should)来表达。

  3.强调性文本（粗体、斜体或颜色）突出需求中的关键部分。

  4.不要假设读者理解技术性语言。“体系结构”和“模块” 这样的词语容易被误解。尽量避免专业术语、缩写和首字母缩略词。

  5.只要有可能，都应当尽量将每个用户需求与其原理关联起来。

  - 原理应解释为什么需求被包含及谁提出的需求（来源），在需求要变化时咨询谁。需求原理在需求发生变化时尤其有用，因为它可以帮助判断哪些变化是不适宜的。

4.4.1 自然语言规格说明

- 下面描述这些指南该如何使用
  - 3.2系统必须每10分钟测量一次血糖，如需要就供应一次胰岛素。(血糖变化相对比较慢，因此不需要更频繁的测量；测量间隔时间过长的话会导致不必要的高血糖水平。)
  - 3.6系统必须每分钟运行一次例行的自检，测试表1定义的条件，并执行表1中所定义的相关动作。(例行的自检可以发现硬件和软件问题并警告用户常规操作可能有问题。)

4.4.2 结构化规格说明

- 结构化自然语言是一种书写系统需求的方式，即使用标准的方式而非自由文本方式书写需求
- 保持自然语言的表达能力和可理解性，同时保证一定的统一性
- 结构化语言表示法使用模板来刻画系统需求。
- 可使用编程语言元素来表示可选项和迭代，可**强调关键元素**。
- 推荐先在卡片上书写用户需求，每张卡片一条需求。建议每张卡片上包含一些字段，如需求原理、与其他需求的依赖关系、需求来源、支持性的材料等。
- 需要定义一个或多个标准的需求模板，并将这些模板表示为结构化的表单。
- 规格说明可以围绕系统操纵的对象、系统执行的功能或者系统处理的事件进行组织。

4.4.2 结构化规格说明

- 下面描述了一个基于表单的规格说明的例子，其中定义了当血糖处于安全区间时如何计算要供应的胰岛素剂量。

胰岛素泵，控制软件/SRS/3.3.2
功能：计算胰岛素剂量：安全的血糖水平。
描述：当前测量的血糖水平在安全区间3～7个单位时，计算要供给的胰岛素的剂量。
输入：当前血糖读数(r2)，前两个读数（r0和r1）。
来源：当前读数来自传感器。其他读数来自存储。
输出：CompDose-----要供给的胰岛素剂量。
目的地：主控制环。
动作：如果血糖水平稳定或在下降，或者虽然在升高但是上升率在下降，那么CompDose为0。如果血糖水平在升高并且上升率也在增长，那么CompDose通过将当前血糖水平与前一血糖水平之差除以4并四舍五入计算。如果结果为0，那么将CompDose设为可以供应的最小剂量。
要求：有前两个读数，这样可以计算血糖水平的变化率。
前置条件：胰岛素存储存有至少一个所允许的最大单剂量胰岛素。
后置条件：r0被r1替代，而r1被r2替代。
副作用：无

4.4.2 结构化规格说明

- 当使用一种标准格式来刻画功能性需求时，应包含以下信息。
  1.对所刻画的功能或实体的描述；
  2.关于其输入以及输入来源的描述；
  3.关于其输出以及输出目的地的描述；
  4.关于计算所需的信息或者所需要的系统中其他实体的信息（“要求”部分）；
  5.关于所要采取的行动的描述；
  6.如果使用一个功能性的方法，那么用一个前置条件明确该功能调用前必须满足的条件，用一个后置条件刻画该功能调用后必须满足的条件；
  7.关于该操作的副作用（如果有的话）的描述。

4.4.2 结构化规格说明

- 结构化可以去除自然语言规格说明中的一些问题。
  - 规格说明中的可变性减少了，需求可以更有效地组织。
- 但是清晰、无二义的需求很难，特别是要刻画的计算很复杂。
- 我们可以向自然语言需求中增加额外的信息，如，使用表格或系统的图形化模型。
- 这些可以显示计算是如何进行的，系统状态如何变化，用户如何与系统交互，以及动作序列如何执行。
- 当存在多种可能情况且要描述每种要采取的动作时，表格尤其有用。
- 胰岛素泵中所需要的胰岛素用量的计算是基于血糖水平的变化率。变化率是使用当前和前面的读数来计算的。

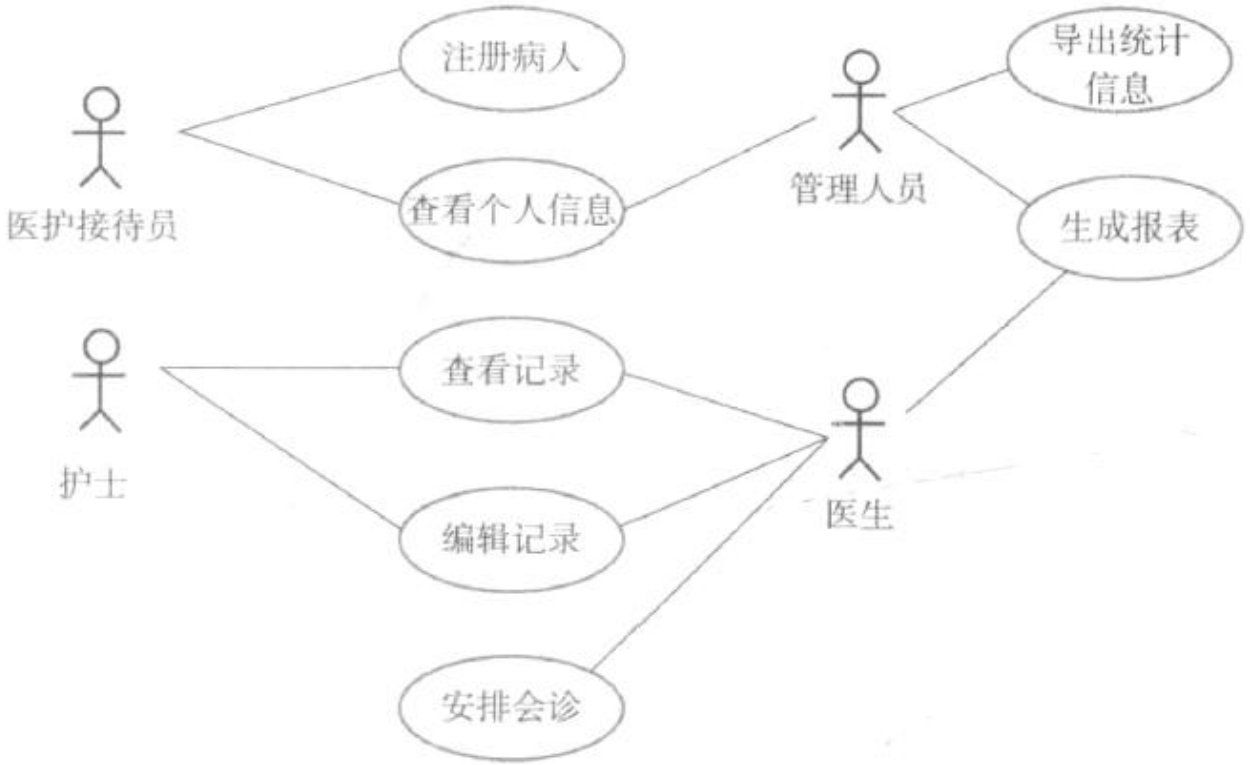
条 件	动 作
血糖水平在下降(r2< rl)	CompDose=0
血糖水平稳定(r2= rl)	CompDose=0
血糖水平在升高并且上升率在下降（（r2 - rl）< (rl - r0)）	CompDose=0
血糖水平在升高并且上升率稳定或在增加r2> rl＆（(r2 - rl)≥(rl - r0)）	CompDose= round((r2-rl)/4)如果四舍五入后为0，那么CompDose= MinimumDose

4.4.3用例

- 用例是使用图形化模型和结构化文本描述用户与系统间交互的方式
- 按照最简单的用例形式，一个用例识别参与一个交互的参与者，并且对交互的类型进行命名。
- 在此基础上，可以添加一些描述与系统交互的附加信息。
- 这些附加信息可以是一段文本描述，或者是一个或多个图形化模型，例如UML顺序图或状态图。
- 用例的简要描述可以如下。
- 安排会诊允许两个或更多不同科室的医生来同时查看同一个病人的记录。一个医生通过从当前在线的医生的下拉菜单中选择参与人来发起会诊。病人记录显示在他们的屏幕上，但只有发起的医生可以编辑记录。此外，系统会创建一个文本聊天窗口来帮助进行相关行动的协调．这里的一个假设是，可以另外安排一个电话语音交流。

4.4.3用例

- Mentcare系统的用例



4.4.4软件需求文档

- 也称为软件需求规格说明(Software Requirement Specification，SRS)，是关于系统开发者应当实现的所有东西的正式陈述。
- 可以同时包括一个用户需求及对于系统需求的详细规格说明。
- 有时用户和系统需求被集成到同一个描述中。而有时，用户需求在系统需求规格说明的一个引言章节中描述。
- 对于外包开发，即不同团队开发系统的不同部分，以及当详细的需求分析是必需的时候，需求文档十分关键。
- 其他情况，如开发软件产品或业务系统，需求文档不一定是必需的。
- 敏捷方法不使用正式文档，而是经常增量地收集用户需求并且将它们作为简短的用户故事写在卡片或白板上。在下一个增量中按照优先级排序。
- 需求文档有多种用户，从高层管理人员到软件工程师。
- 需求文档中应当包含的详细程度取决于系统类型以及所使用的开发过程。

4.4.4软件需求文档-参考结构

章	描 述
前言	定义本文档所期望的读者人群，并且描述文档的版本历史，包括创建一个新版本的原因以及对于每个版本中所作修改的总结
引言	描述系统的需要。其中应当简要描述系统的功能，并解释系统将如何与其他系统一起工作。还应当描述系统如何适应委托开发软件的组织的总体业务或战略目标
术语表	定义了文档中所用的技术术语。不应该对读者的经验或专业知识进行假设
用户需求定义	描述为用户提供的服务。非功能性系统需求也应当在这部分描述。这些描述使用客户可以理解的自然语言、图形或其他表示法。必须遵循的产品和过程标准也应该在这里描述
系统体系结构	描述所预计的系统体系结构的高层概览，显示各个系统模块上的功能分布。复用的体系结构构件应当进行强调
系统需求规格说明	更详细地描述功能性需求和非功能性需求。如果有必要，可以向非功能性需求中增加进一步的细节，还可以定义与其他系统的接口
系统模型	包括图形化的系统模型，显示系统构件之间以及系统及其环境之间的关系。可能的模型包括对象模型、数据流模型或语义数据模型
系统维护	描述系统所基于的基本假设，及所预计的由于硬件演化、用户需求变更等导致的变化，这部分对于系统设计者很有用，因为可帮助他们避免做出会限制系统未来可能的变化的设计决策
附录	这部分提供与所开发的应用相关的详细、特定的信息，例如硬件和数据库描述。硬件需求定义了系统的最小配置和优化配置。数据库需求定义了系统所使用的数据的逻辑组织以及数据之间的关系
索引	可以包含几个文档索引．除了常规的字母序索引，还可以有图索引、功能索引等

4.5需求确认

- 需求确认是检查需求是否定义了客户真正想要的系统的过程。
- 需求确认非常重要，因为如果需求文档中的错误在开发过程中或在系统投入服务后被发现，则会导致广泛的返工开销。
- 通过进行系统变更修正一个需求问题的开销通常比修复设计或编码错误要高得多。一个需求变更通常意味着系统设计和实现也必须修改。而且，系统接下来还必须重新测试。

4.5需求确认-检查

- 需求确认过程中，应该对需求文档中的需求进行不同类型的检查。
  1.正确性检查。检查需求是否反映了系统用户的真实需要。
  – 由于环境不断变化，用户需求可能在最初被抽取后已发生变化。
  2.一致性检查。文档中的需求不应该冲突。不应该有相互矛盾的约束或者对同一个系统功能的不同描述。
  3.完整性检查。需求应当定义系统用户想要的所有功能及约束。
  4.现实性检查。使用现有技术的知识，对需求进行检查以确保可以在预算范围内实现。考虑系统开发预算和进度。
  5.可验证性检查。为了减少客户和承包商之间可能的争议，所描述的系统需求应当总是可验证的。
  – 这意味着应当能够针对每一条所刻画的需求编写一组测试，以便显示所交付的系统满足该需求。

4.5需求确认-技术

- 需求确认技术

  1.需求评审。由一个评审团队系统性地分析需求，检查错误和不一致性。

  2.原型化。开发一个可执行的系统模型，并与最终用户和客户一起使用该模型，来确认是否满足他们的需要和期望。利益相关者对系统进行试验，并向开发团队反馈需求变更。

  3.测试用例生成。需求应该是可测试的。如果作为确认过程的一部分，设计针对一个需求的测试，那么经常可以揭示需求中的问题。

  - 写代码之前，根据用户需求开发测试是测试驱动开发的一个重要组成部分。
- 不应当低估需求确认中所包含的问题。

4.6 需求变更

- 大型软件系统需求总是在变化中。这样的频繁变化的一个原因是，这些系统经常被开发用于应对“非常规”的问题——无法完备定义的问题。
- 因为问题无法被充分定义，软件需求不可避免地会不完整：在软件开发过程中，利益相关者对于问题的理解在不断变化，于是，系统需求必须演化以反映对这一变化的问题的理解。

- 一旦一个系统已经安装并被正常使用，新的需求不可避免地会出现。
- 部分原因是原始需求中存在需要纠正的错误和遗漏。然而，大部分对系统需求的变更是由于系统的业务环境发生变化而产生的。

4.6 需求变更

- 1.系统业务和技术环境总是会在系统安装后发生变化。可能引入新的硬件，或者更新已有的硬件；系统可能要与其他系统建立接口；业务优先级可能发生变化（所需要的系统支持会因此发生变化）；新的法律和监管要求可能会出现并要求系统满足。
- 2.为系统付钱的人和系统的用户经常是不同的人。系统客户由于组织和预算约束而提出需求，这些可能与最终用户的需求相互冲突，并且在交付后可能必须为用户增加新特征。
- 3.大型系统通常有各种各样的利益相关者群体，他们优先级可能相互冲突或矛盾。最终的系统需求不可避免地要进行折中，而有些利益相关者必须给予较高的优先级。根据经验，经常会发现对于给予不同利益相关者的支持的平衡必须变化，而需求优先级要进行调整。

“十二五”普通高等教育本科国家级规划教材

北京高等教育精品教材

21世纪软件工程专业规划教材

软件工程导论（第6版）

张海藩，牟永敏编著

清华大学出版社

2.3 系统流程图

系统流程图是概括地描绘物理系统的传统工具。

基本思想：

用图形符号以黑盒子形式描绘组成系统的每个部件(程序、文档、数据库、人工过程等)。

系统流程图表达的是数据在系统各部件之间流动的情况，而不是对数据进行加工处理的控制过程，因此尽管系统流程图的某些符号和程序流程图的符号形式相同，但是它却是物理数据流图而不是程序流程图。

2.3 系统流程图

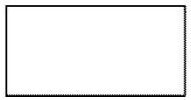
2.3.1符号

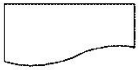
利用符号可以把一个广义的输入输出操作，具体化为读写存储在特殊设备上的文件（或数据库），把抽象处理具体化为特定的程序或手工操作等。

2.3 系统流程图

2.3.1符号

以概括方式抽象地描绘一个实际系统，下图中的基本符号就足够

符　号	名　称	说　明
	处理	能改变数据值或数据位置的加工或部件，例如程序、处理机、人工加工等都是处理
	输入输出	表示输入或输出（或既输入又输出），是一个广义的不指明具体设备的符号
	连接	指出转到图的另一部分或从图的另一部分转来，通常在同一页上
	换页连接	指出转到另一页图上或由另一页图转来
	数据流	用来连接其他符号，指明数据流动方向

符　号	名　称	说　明
	穿孔卡片	表示用穿孔卡片输入或输出，也可表示一个穿孔卡片文件
	文档	通常表示打印输出，也可表示用打印终端输入数据
	磁带	磁带输入输出，或表示一个磁带文件
	联机存储	表示任何种类的联机存储，包括磁盘、磁鼓、软盘和海量存储器件等
	磁盘	磁盘输入输出，也可表示存储在磁盘上的文件或数据库
	磁鼓	磁鼓输入输出，也可表示存储在磁鼓上的文件或数据库
	显示	CRT 终端或类似的显示部件，可用于输入或输出，也可既输入又输出
	人工输入	人工输入数据的脱机处理，例如填写表格
	人工操作	人工完成的处理，例如会计在工资支票上签名
	辅助操作	使用设备进行的脱机操作
	通信链路	通过远程通信线路或链路传送数据

需要更具体地描绘一个物理系统时还需要使用右图中列出的系统符号

2.3 系统流程图

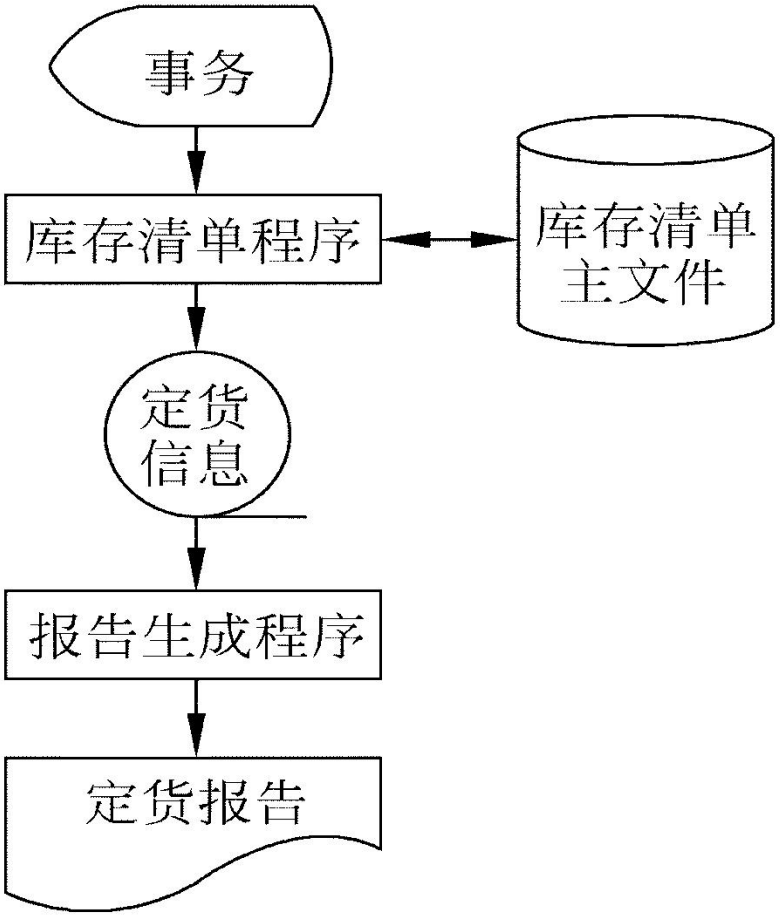
2.3.2 例子

某装配厂有一零件仓库，仓库中现有的各种零件的数量以及库存量临界值等数据记录在库存清单主文件中。当仓库中零件数量有变化时，应及时修改库存清单主文件，如零件库存量少于它的库存量临界值，则应该报告给采购部门以便订货，规定每天向采购部门送一次订货报告。

使用一台小型机处理更新库存清单主文件和产生订货报告任务。零件库存量的每一次变化称为一个事务，由仓库CRT终端输入到计算机中；库存清单程序对事务进行处理，更新磁盘上库存清单主文件，并且把必要的订货信息写在磁带上。最后，每天由报告生成程序读一次磁带，并且打印出订货报告。如下图所示。

2.3 系统流程图

2.3.3 分层



面对复杂的系统时，好的方法是分层次地描绘。首先用一张高层次的系统流程图描绘系统总体概貌，表明系统关键功能。然后分别把每个关键功能扩展到适当详细程度，画在单独一页纸上。

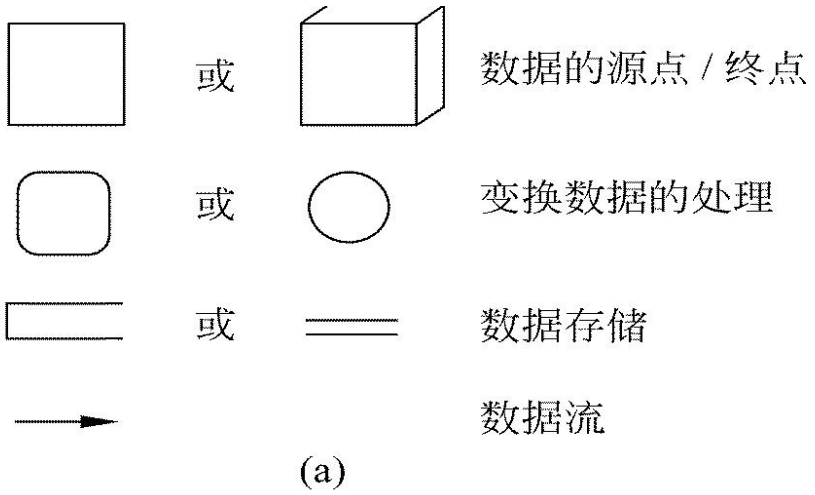
分层次的描绘方法便于阅读者按从抽象到具体的过程逐步深入地了解一个复杂的系统。

2.4 数据流图

2.4.1 符号

数据流图(DFD)是一种图形化技术，它描绘信息流和数据从输入移动到输出的过程中所经受的变换。

基本符号



(a)

正方形表示数据的源点或终点

圆角矩形代表变换数据的处理

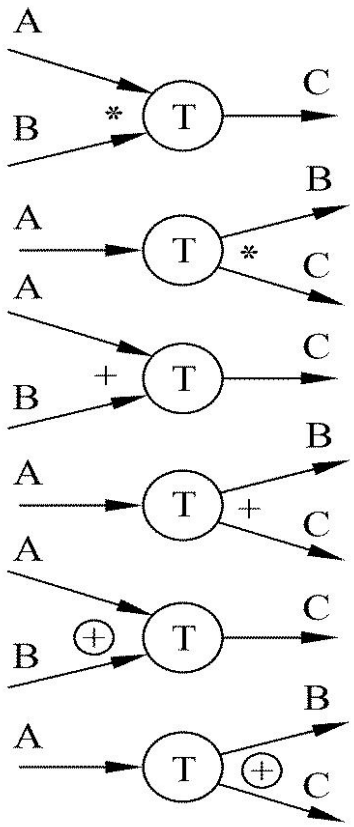
开口矩形代表数据存储

箭头表示数据流，即特定数据的流动方向

2.4 数据流图

2.4.1 符号

附加符号



数据 A 和 B 同时输入才能变换成数据 C

数据 A 变换成 B 和 C

数据 A 或 B，或 A 和 B 同时输入变换成 C

数据 A 变换成 B 或 C，或 B 和 C

只有数据 A 或只有数据 B(但不能 A、B 同时) 输入时变换成 C

数据 A 变换成 B 或 C，但不能变换成 B 和 C

(b)

2.4 数据流图

2.4.2 例子

以简单例子说明怎样画数据流图

假设一家工厂的采购部每天需要一张订货报表，报表按零件编号排序，表中列出所有需要再次订货的零件。对于每个需要再次订货的零件应该列出下述数据：零件编号，零件名称，订货数量，目前价格，主要供应者，次要供应者。零件入库或出库称为事务，通过放在仓库中的CRT终端把事务报告给订货系统。当某种零件的库存数量少于库存量临界值时就应该再次订货。

2.4 数据流图

2.4.2 例子

■ 第一步可以从问题描述中提取数据流图的4种成分：

首先考虑数据的源点和终点，从上面对系统的描述可以知道“采购部每天需要一张订货报表”，“通过放在仓库中的CRT终端把事务报告给订货系统”，所以采购员是数据终点，而仓库管理员是数据源点。

■ 第二步：再一次阅读问题描述，“采购部需要报表”

因此必须有一个用于产生报表的处理。事务的后果是改变零件库存量，然而任何改变数据的操作都是处理，因此对事务进行的加工是另一个处理。注意，在问题描述中并没有明显地提到需要对事务进行处理，但是通过分析可以看出这种需要。

2.4 数据流图

2.4.2 例子

■ 第三步：考虑数据流和数据存储

系统把订货报表送给采购部，因此订货报表是一个数据流；事务需要从仓库送到系统中，显然事务是另一个数据流。产生报表和处理事务这两个处理在时间上明显不匹配——每当有一个事务发生时立即处理它，然而每天只产生一次订货报表。因此，用来产生订货报表的数据必须存放一段时间，也就是应该有一个数据存储。

2.4 数据流图

步骤一分析结果：

源点/终点	处理
采购员 仓库管理员	产生报表 处理事务
数据流	数据存储
订货报表 零件编号 零件名称 订货数量 目前价格 主要供应者 次要供应者 事务 零件编号* 事务类型 数量*	订货信息 （见订货报表） 库存清单* 零件编号* 库存量 库存量临界值

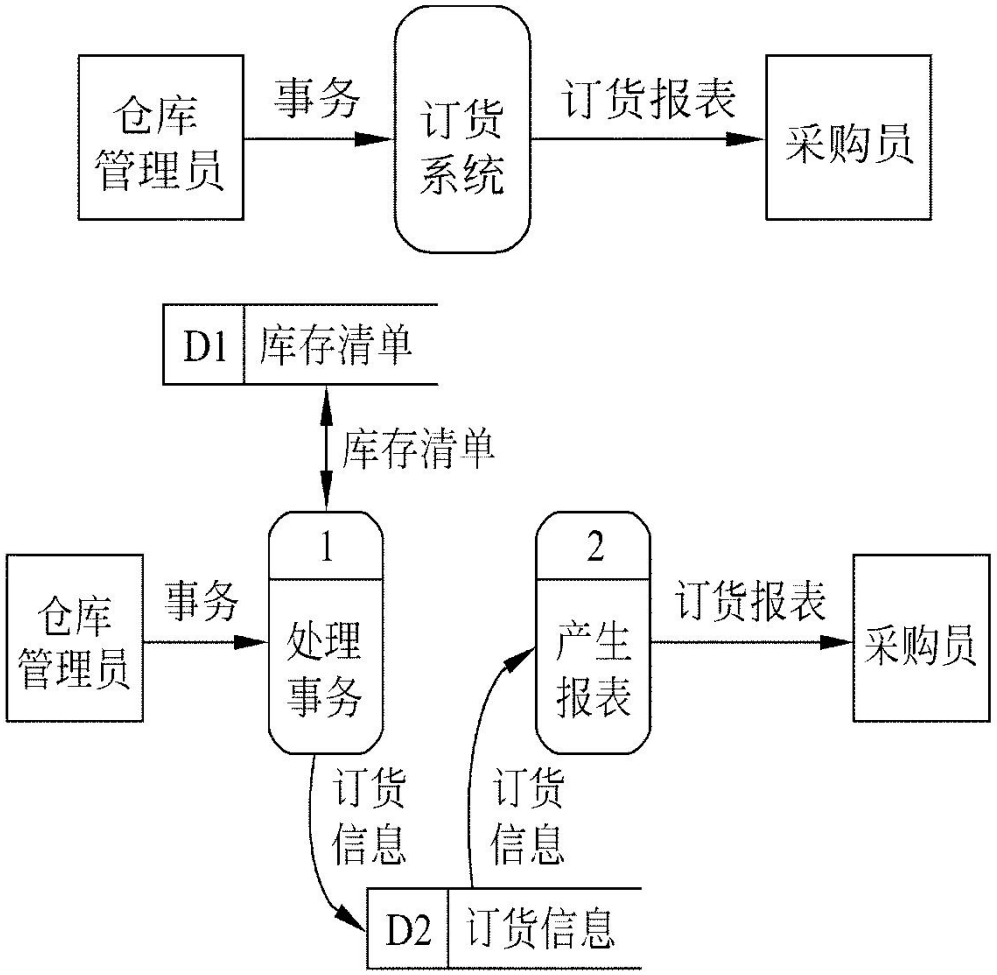
2.4 数据流图

步骤二：

把数据流图的4种成分都分离出来以后（上图所示），就可以着手画数据流图了

步骤三：

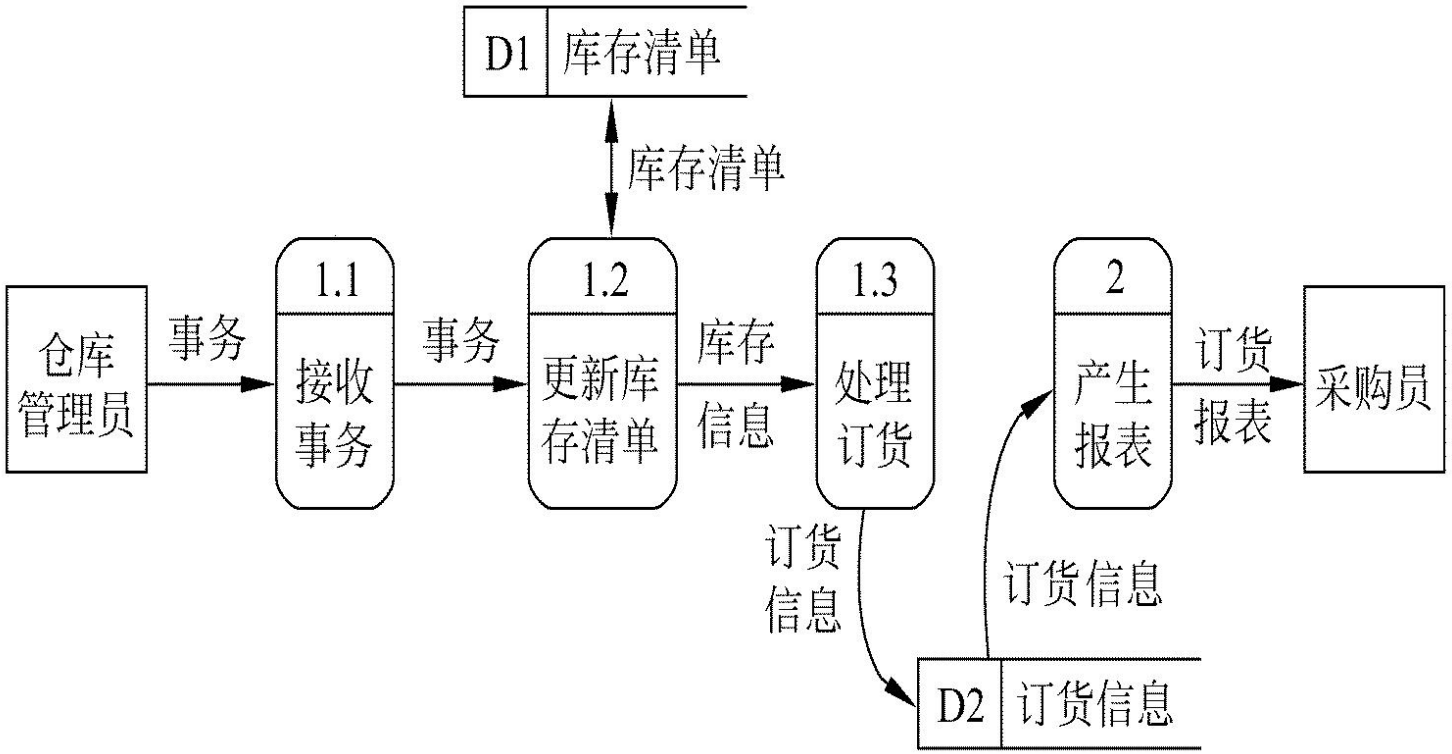
把基本系统模型细化，描绘系统的主要功能



2.4 数据流图

步骤四：

对功能级数据流图中描绘的系统主要功能进一步细化



2.4 数据流图

2.4.3 命名

数据流图中每个成分的命名是否恰当，直接影响数据流图的可理解性。因此，给这些成分起名字时应该仔细推敲。

- 数据流命名时应注意的问题
  - 名字应代表整个数据流的内容，而不仅仅反映某些成分
  - 不要使用空洞的、缺乏具体含义的名字
  - 某个数据流(或数据存储)起名字时遇到困难，则很可能是因为对数据流图分解不恰当造成的，应该试试重新分解

2.4 数据流图

2.4.3 命名

- 为处理命名时应注意的问题
  - 通常先为数据流命名，然后再为与之相关联的处理命名。
  - 名字应该反映整个处理的功能，而不是它的一部分功能。
  - 名字最好由具体的及物动词加上一个具体的宾语组成。
  - 通常名字中仅包括一个动词，如果必须用两个动词才能描述整个处理的功能，则把这个处理再分解成两个处理可能更恰当些。
  - 如果在为某个处理命名时遇到困难，则很可能是发现了分解不当的迹象，应考虑重新分解。

2.4 数据流图

2.4.4 用途

- 画数据流图的基本目的是利用它作为交流信息的工具。
- 数据流图的另一个主要用途是作为分析和设计的工具。
- 数据流图辅助物理系统的设计时，以图中不同处理的定时要求为指南，能够在数据流图上画出许多组自动化边界，每组自动化边界可能意味着一个不同的物理系统

2.5 数据字典

数据字典是关于数据的信息的集合，也就是对数据流图中包含的所有元素的定义的集合。

2.5.1 内容

- 数据字典应该由对下列4类元素的定义组成。
  - 数据流
  - 数据存储
  - 处理
  - 数据流分量
- 记录数据元素下列信息： 一般信息(名字，别名，描述等)，定义(数据类型，长度，结构等)，使用特点(值的范围，使用频率，使用方式——输入、输出、本地，条件值等)，控制信息(来源，用户，使用它的程序，改变权，使用权等)和分组信息(父结构，从属结构，物理位置——记录、文件和数据库等)。

2.5 数据字典

2.5.1 内容

- 数据元素的别名就是该元素的其他等价的名字，出现别名主要有下述3个原因：
  - 对于同样的数据，不同的用户使用了不同的名字。
  - 一个分析员在不同时期对同一个数据使用了不同的名字。
  - 两个分析员分别分析同一个数据流时，使用了不同的名字。

2.5 数据字典

2.5.2 定义数据的方法

- 由数据元素组成数据的方式只有下述3种基本类型：
  - 顺序：即以确定次序连接两个或多个分量。
  - 选择：即从两个或多个可能的元素中选取一个。
  - 重复：即把指定的分量重复零次或多次。
- 第4种关系算符
  - =意思是等价于(或定义为)；
  - +意思是和(即连接两个分量)；
  - ［］意思是或(即从方括弧内列出的若干个分量中选择一个)，通常用“|”号隔开供选择的分量；
  - {}意思是重复(即重复花括弧内的分量)；
  - ()意思是可选(即圆括弧里的分量可有可无)。

2.5 数据字典

2.5.3 数据字典的用途

- 数据字典最重要的用途是作为分析阶段的工具
- 数据字典中包含的每个数据元素的控制信息是很有价值的
- 数据字典是开发数据库的第一步，而且是很有价值的一步

2.5 数据字典

2.5.4 数据字典的实现

目前，数据字典几乎总是作为CASE“结构化分析与设计工具”的一部分实现的。在开发大型软件系统的过程中，数据字典的规模和复杂程度迅速增加，人工维护数据字典几乎是不可能的。

开发小型软件系统时暂时没有数据字典处理程序，建议用卡片形式书写数据字典，每张卡片上保存描述一个数据信息。

例子：几个数据元素的数据字典卡片，以具体说明数据字典卡片中上述几项内容的含义。

2.5 数据字典

2.5.4 数据字典的实现

名字：订货报表
别名：订货信息
描述：每天一次送给采购员的需要订货的零件表
定义：订货报表＝零件编号＋零件名称＋订货数量＋目前价格＋主要供应者＋次要供应者
位置：输出到打印机

名字：零件编号
别名：
描述：唯一地标识库存清单中一个特定零件的关键域
定义：零件编号＝8{字符}8
位置：订货报表
订货信息
库存清单
事务

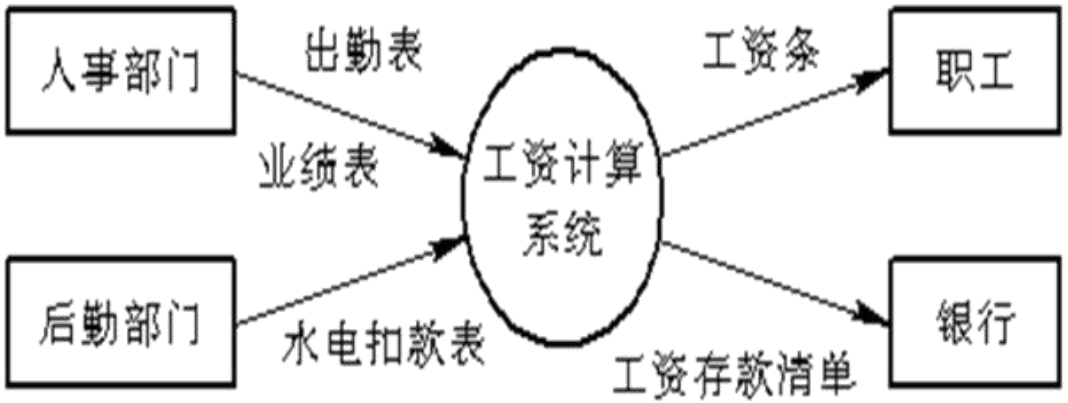
名字：订货数量
别名：
描述：某个零件一次订货的数量
定义：订货数量＝1{数字}5
位置：订货报表
订货信息

案例分析

- 工资计算系统
- 计算工资：根据人事部门给出的出勤表和业绩表计算奖金和缺勤扣款，通过生成的奖金发放表及工资基本信息库的信息计算应发工资，根据应发工资表计算所得税，根据后勤部门给出的水电扣款及缺勤扣款表和所得税款计算出实发工资，生成实发工资表和工资清单。
- 打印工资清单：根据工资清单完成工资条打印，给职工工资转存：根据实发工资表生成职工工资存款清单并将其发送到银行

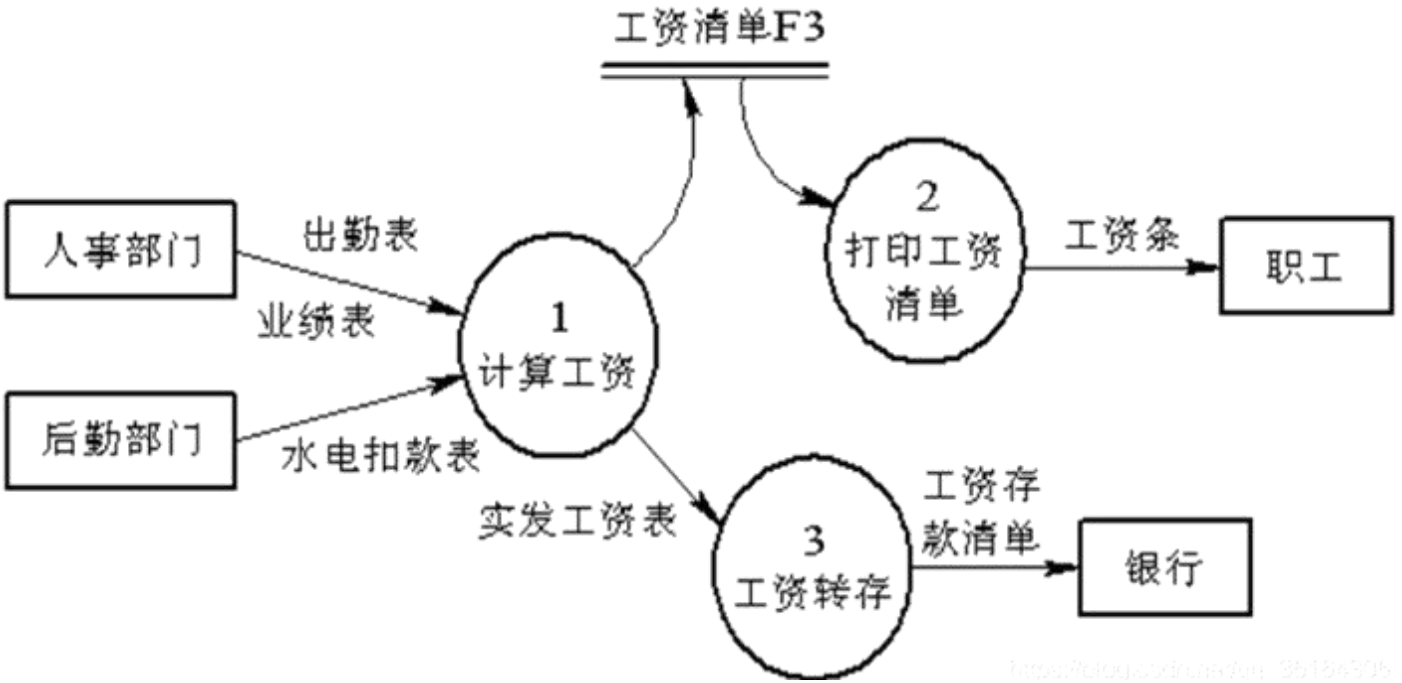
分析数据流图基本元素

- 数据的源点和终点：人事部门、后勤部门；职工、银行
- 处理：人事部门给出的出勤表和业绩表计算奖金和缺勤扣款，工资清单完成工资条的打印，实发工资表生成职工工资存款清单并将其发送到银行
- 数据流：出勤表、业绩表；水电扣款表；工资条、工资存款清单
- 数据存储：工资基本信息库、工资清单
- 画顶层数据流图



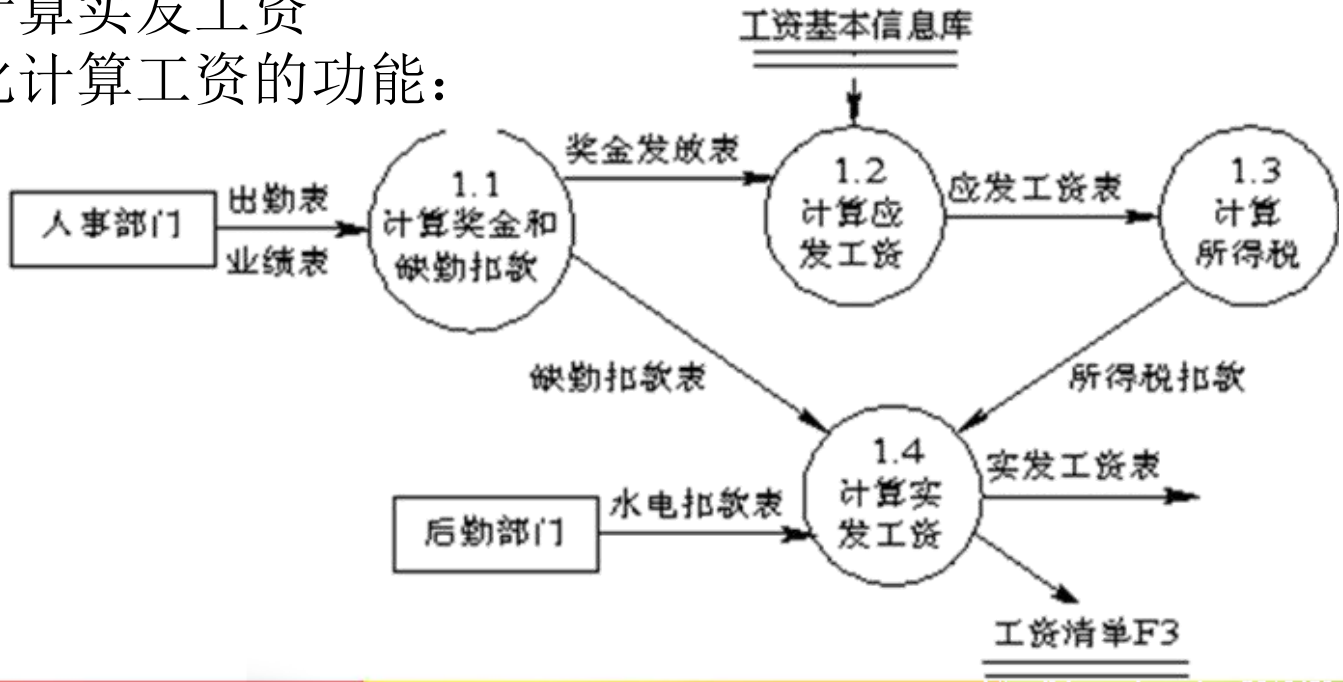
细分功能

- 工资计算系统包括处理：计算工资、打印工资清单、工资转存
- 细化数据流图



细化功能

- 细化功能：打印工资清单、工资转存需要的数据已经完善了，不需做其他处理，因此这两个处理不用细分。
- 计算工资，计算奖金和缺勤扣款，计算应发工资，计算所得税，计算实发工资
- 细化计算工资的功能：



第5章 系统建模

- 系统建模就是建立系统抽象模型的过程，其中每一个模型表示系统的一个不同的视角或观点。
- 在需术工程过程中使用模型，是为了帮助得到详细的系统需求；在设计过程中使用模型，是为了向实现系统的工程师描述系统；在实现系统之后还要使用模型，是为了描述系统的结构和运行。
- 可以同时针对现有系统和待开发系统开发系统模型。
  - 现有系统模型在需求工程过程使用。帮助阐明现系统做什么，且可用于让利益相关者间讨论聚焦于现系统优势和弱点。
  - 新系统模型在需求工程过程中使用。帮助解释对其他系统利益相关者所提 出的需求。
  - 工程师使用这些模型来讨论设计方案并描述系统以用于实现

第5章 系统建模

- 理解系统模型并不是系统的一个完备表示，这一点很重要。
  - 系统模型有意去掉一些细节以使模型更容易理解。
  - 模型是所研究系统的一种抽象，而不是系统的另一种表示。
  - 系统的一个表示应当包含关于所表示的实体的所有信息。
  - 一个抽象则有意简化一个系统设计并选取最显著的特性。
- 可以开发不同的模型来从不同的视角表示系统。
  - 外部视角，对系统的上下文或环境进行建模；
  - 交互视角，对系统及其环境或者系统构件之间交互进行建模；
  - 结构化视角，对系统组织或系统所处理数据的结构进行建模；
  - 行为视角，对系统的动态行为及系统如何响应事件进行建模。
- 开发系统模型时，开发者可灵活决定图形化表示法使用方式。
- 开发者并不总是需要严格坚持一些细节的。一个模型的细节和严格性取决于开发者打算如何使用它。

第5章 系统建模

- 图形化模型的3种常见的使用方式。
- 1、作为推动关于现有或新系统的讨论以及使讨论聚焦的方式。
  - 模型目的是推动以及聚焦参与系统开发的软件工程师之间讨论。模型可以不完整(但是要覆盖讨论要点)，且可能会以一种非正式的方式使用建模表示法。这是敏捷建模中常见的模型使用方式。
- 2、作为一种文档化现有系统的方式。
  - 当模型被用于文档化时，它们不需要完整，因为你可能只需要使用模型去描述系统的一些部分。必须是正确的—它们应当正确地使用相应建模表示法，且对系统准确描述。
- 3、作为一种可以用于生成系统实现的详细系统描述。
  - 当模型作为基于模型的开发过程的一部分被使用时，系统模型必须是完整正确的。模型可以作为生成系统源代码基础，因此不要混淆相似但含义各不相同符号(如，线形箭头和块状箭头)。

5.1上下文模型

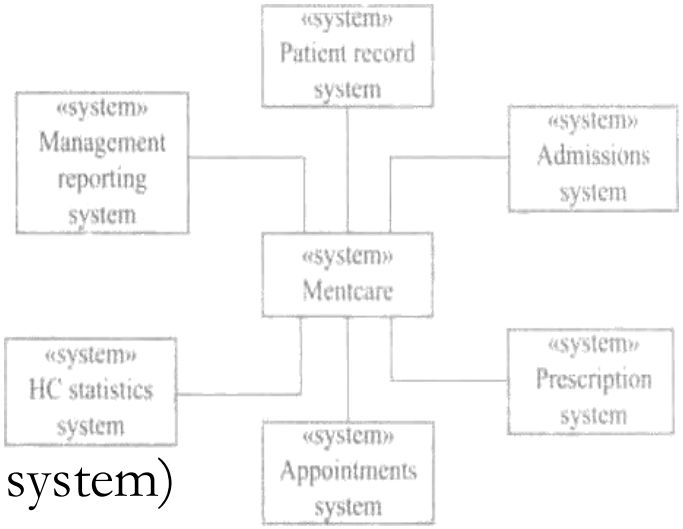
- 在系统规格说明的早期阶段，你应当确定系统的边界，也就是说确定哪些属于、哪些不属于所开发的系统。
- 与系统利益相关者一起决定哪些功能应当包含在系统中，以及哪些处理和操作应当在系统的运行环境中执行。
- 开发者决定哪些业务自动化，哪些由手工或由其他系统支持。
- 应考虑新系统的功能与原系统可能存在的重叠部分，并决定新功能应当在哪里实现。
- 这些决定应早做，以控制理解需求和设计所需的成本和时间。
- 有时，系统及其环境的边界相对清楚。如，自动化系统准备取代已有手工或计算机化系统时，新旧系统的环境通常一样。
- 而在其他情况下则存在更多的灵活性，需要在需求工程过程中确定系统及其环境的边界由哪些因素构成。

5.1上下文模型

- 如，假设正在为Mentcare开发规格说明。
- 系统将会管理参加心理健康诊断以及安排治疗的病人的信息。
- 必须确定系统是否应当只关注收集诊疗的信息，还是系统也应该收集病人信息。依赖其他系统获取病人信息好处是避免数据重复。缺点是使用其他系统可能信息访问慢，另外如果这些系统不可用那么Mentcare系统也会无法使用。
- 有些情况，系统的用户基础非常分散，有很多不同需求，那么我们可能会不定义系统边界，而去开发一个可以通过调整来适应不同用户需要的可配置系统。iLearn系统的开发中就采用了这种方法。用户范围很广，从很小的还无法阅读的小孩子到年轻人、他们的老师以及学校管理人员。用户群体需要不同的系统边界，设计可以允许系统在部署时再确定边界的可配置系统。

5.1上下文模型

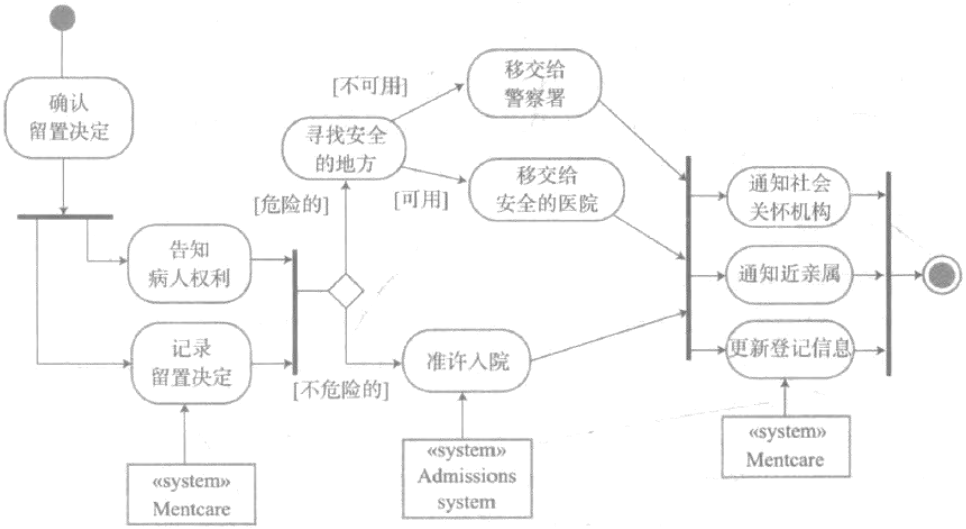
- 边界确定，定义系统上下文以及系统对于其环境的依赖。
- Mentcare系统：
  预约系统(Appointments system)
  病人记录系统(Patient record system)
  医院许可系统(Admissions system)
  统计系统(HC statistics system)
  处方系统(Prescription system)
  管理报表系统(Management reporting system)



- 上下文模型没有显示环境中的系统之间关系的类型
- 外部系统可能提供或消费系统数据、共享数据，可能直接网络连接或没有连接，可能位于同一位置或位于不同建筑中。
- 所有这些关系都可能会影响所定义的系统的需求和设计，必须加以考虑。可以和其他模型，如业务过程模型一起使用。

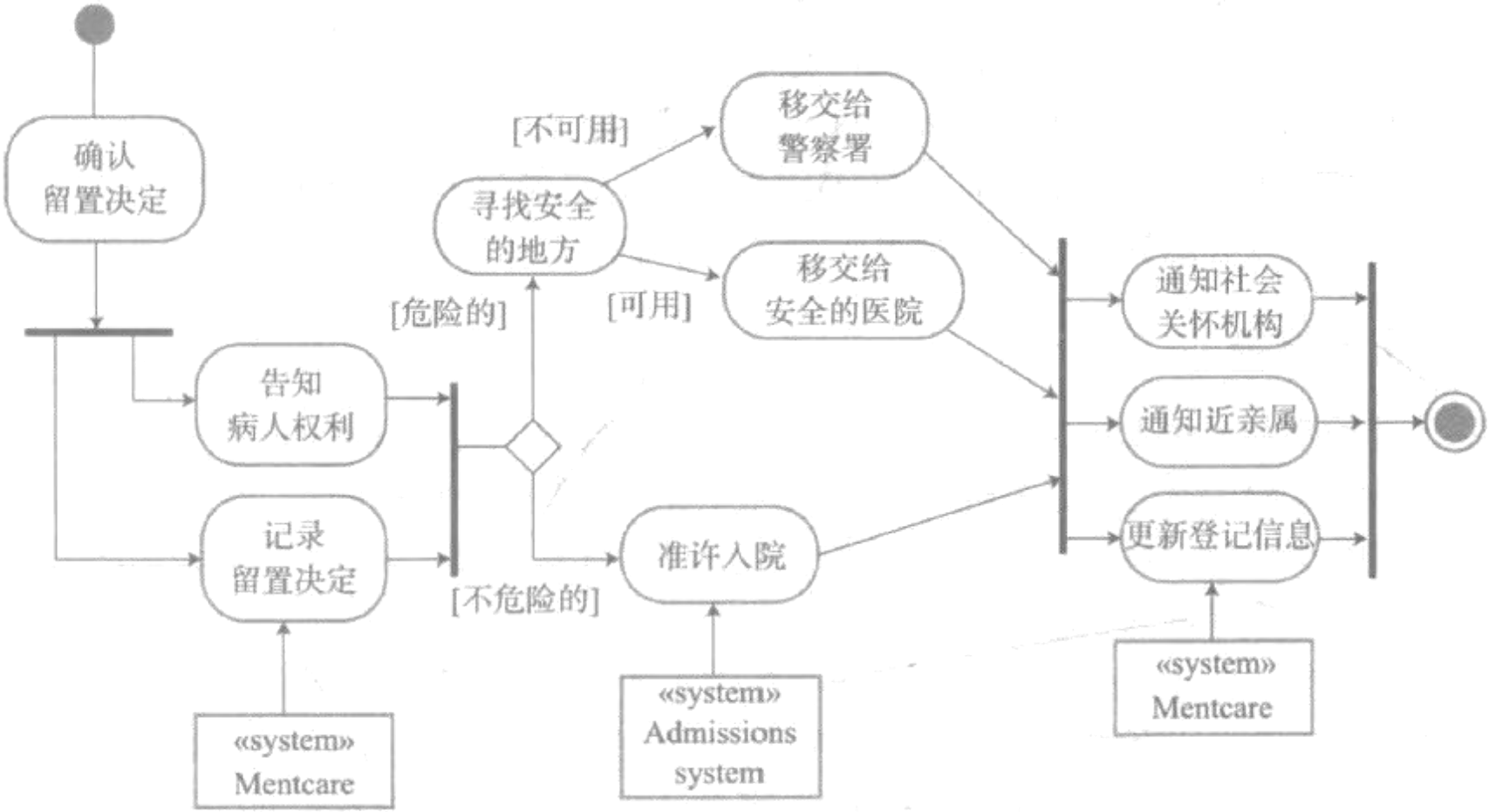
5.1上下文模型

- UML活动图可以显示系统使用所处的业务过程。
- 描述 Mentcare系统心理健康问题治疗过程—强制留置。
- 活动图强调从活动到活动的控制流，是内部处理驱动的流程。
- UML是用于对系统动态为建模的一种常用工具，它的活动图描述活动的顺序，展现从一个活动到另一个活动的控制流。



- 活动图是一种表述过程基理、业务过程以及工作流的技术。它可以用来对业务过程、工作流建模，也可以对用例实现甚至是程序实现来建模

5.1上下文模型



5.2交互模型

- 系统都包含某种交互：可以是与用户的交互(输入和输出)；也可能是开发的软件和其他系统间的交互；或者软件内部构件间交互。
- 用户交互建模很重要，因为它可以帮助识别用户需求。
  - 建模系统间的交互可以突出可能出现的通信问题。
  - 建模构件交互可以帮助我们理解所提出的系统结构是否能实现所要求的系统性能和可依赖性。
- 用例建模，建模系统与外部主体(人类用户或其他系统)间交互；
- 顺序图，建模系统构件之间的交互，也可以包含外部主体。
- 用例模型和顺序图表示不同抽象层次的交互，可以一起使用。
- 例如，一个高层用例中所包含的交互的细节可以在一个顺序图中描述。

5.2.1 用例建模

- 下图展示Mentcare系统的一个用例，描述了一个从Mentcare系统向更通用的病人记录系统上传数据的任务，这个更通用的系统保存了关于病人的总结数据，而不是每次诊疗的数据。

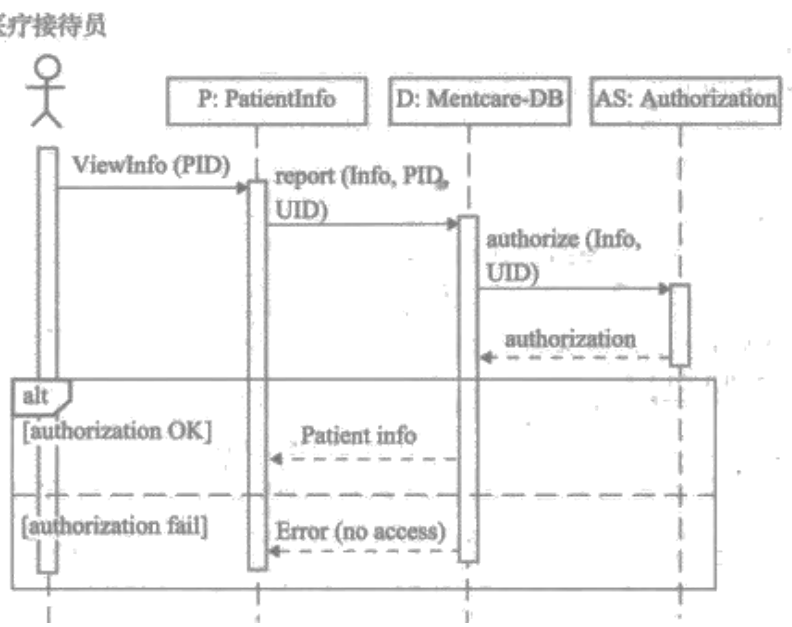
- 使用表格的结构化描述

Mentcare系统：传输数据	
参与者	医疗接待员、病人记录系统（PRS）
描述	医疗接待员可以从Mentcare系统向健康管理机构维护的通用的病人记录数据库传输数据，所传输的信息可以是更新的个人信息（地址、电话号码等）或者病人的诊断和治疗情况总结
数据	病人个人信息、治疗总结
触发激励	医疗接待员发出的用户命令
响应	PRS已经更新的确认信息
注解	医疗接待员必须具有访问病人信息以及PRS的适当的信息安全许可

5.2.2顺序图

- UML顺序图用于建模参与者与系统中的对象之间的交互以及这些对象自身相互间的交互。
- 顺序图显示在一个特定的用例或用例实例执行过程中发生的交互序列。下图是描述顺序图的基本表示法的例子。

- 顺序图顶部列出参与交互的对象和参与者，下面有垂直方向的虚线。矩形表示对象生命线。
- 箭头表示对象间交互。标签表示对对象的调用、调用参数及返回值。
- 按照从上到下顺序阅读交互。
- 可选项的表示法：带有alt名称的方框、方括号中的条件、用虚线分隔的可选的交互选项。



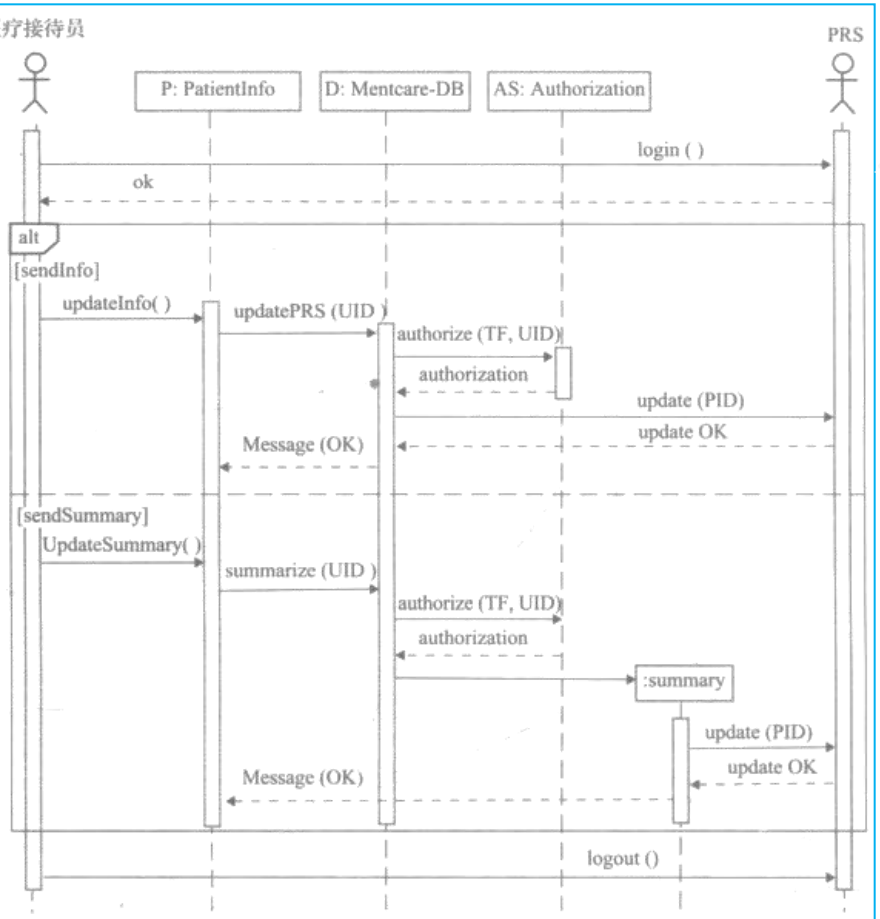
5.2交互模型

- 医疗接待员触发PatientInfo(病人信息)对象类的一个实例P中的ViewInfo(查看信息) 方法，提供病人的标识符P1D来确定所需要的信息。P是个用户界面对象，是一个显示病人信息的表单。
- 实例P调用数据库返回所需的信息，提供接待员的标识符UID用于信息安全检查。
- 数据库通过权限系统检查接待员是否具有执行此动作的权限。
- 如果权限检查通过，那么病人信息被返回并呈现在用户屏幕上的一个表单上。
- 如果权限检查不通过，那么返回一个错误消息。
- 左上角那个带有alt标签的方框是一个选择框，表示所包含的交互中某一个会被执行。选择每个选项的条件显示在方括号中。

5.2交互模型

例：增加两个新的特性

- 1、系统参与者之间直接通信
  - ✓接待员登录到PRS上。
- 2、作为操作序列一部分的对象创建
  - ✓例，Summary (总结)类型的一个对象被创建，用于封装总结数据，这些数据将被上传到一个国家 PRS (病人记录系统)中



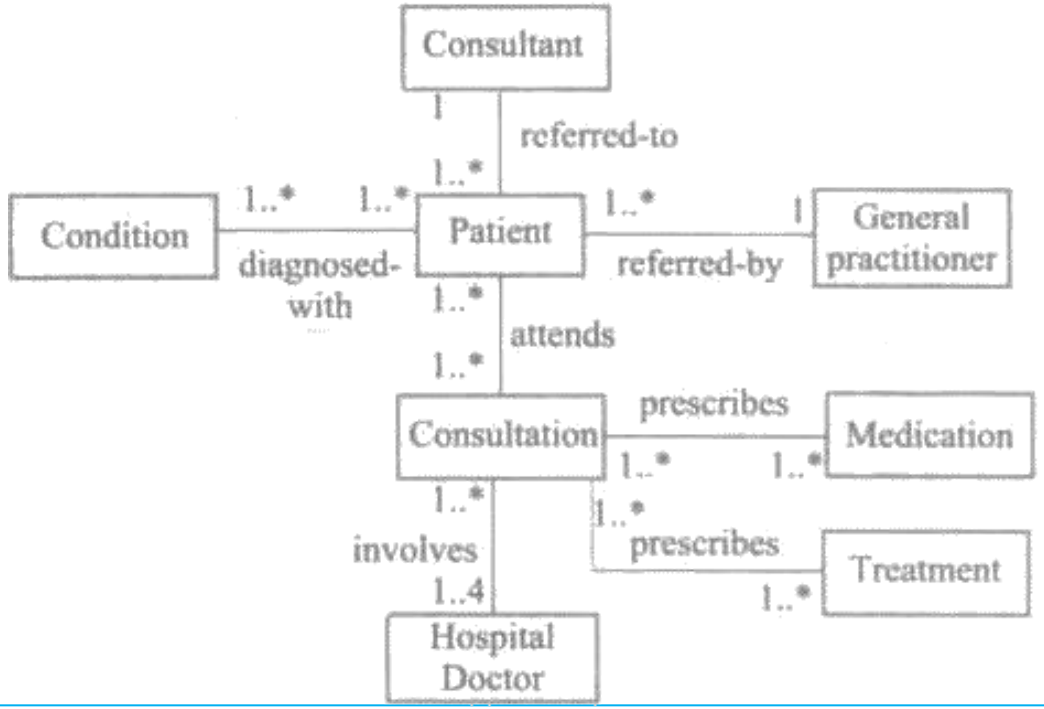
5.3结构模型

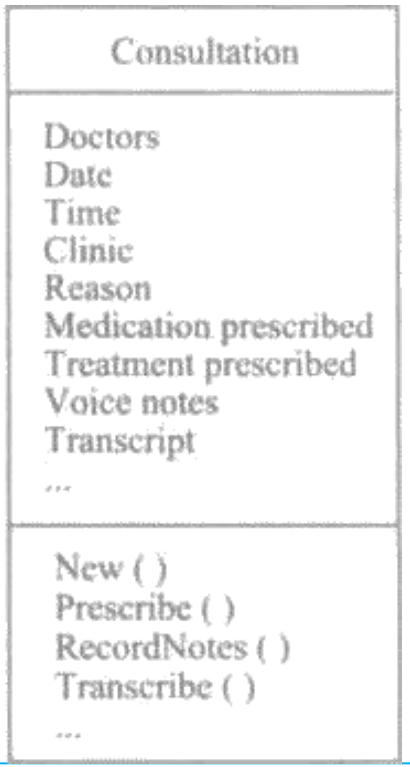
- 软件的结构模型按照构成系统的构件以及它们之间的关系显示系统的组织。
- 结构模型可以是描述系统设计组织的静态模型，也可以是描述系统执行时的组织的动态模型。
- 系统的动态组织是一组相互交互的线程。

- 可以在讨论和设计系统体系结构时创建系统的结构模型。
- 模型可以是总体的系统体系结构模型或者更加详细的系统中的对象及其关系的模型。
- 类图是对软件系统中的对象类的静态结构进行建模的。
- UML的构件图、包图、部署图都可以用于描述体系结构模型。

5.3.1 类图

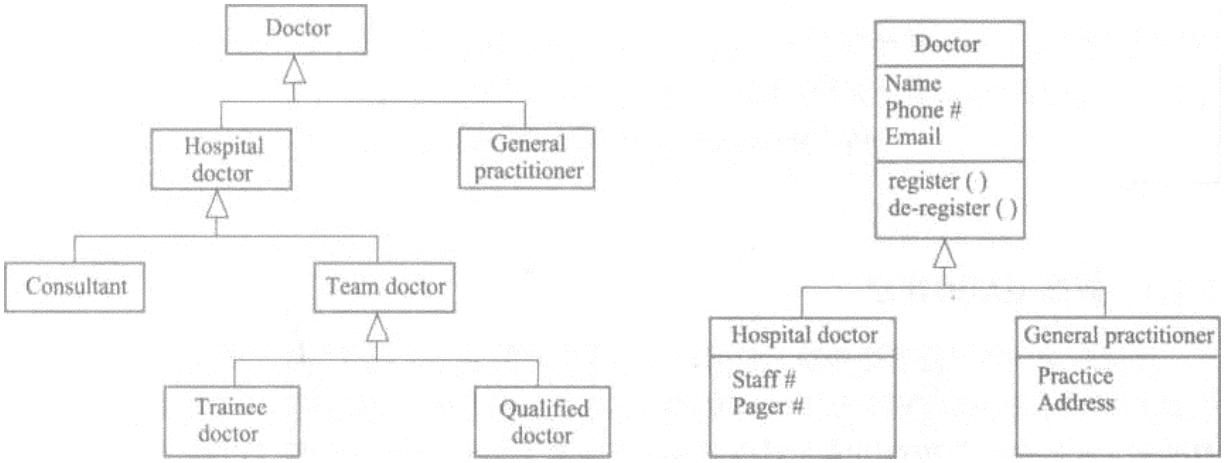
- 当开发一个面向对象系统模型来显示系统中的类以及类之间的关联时，可以使用类图。下面左图描述了Patient类的对象参与了其它类之间的关系（包括关于重数的表达）；右图描述了某个类（Consulation-咨询）的具体的属性和操作。





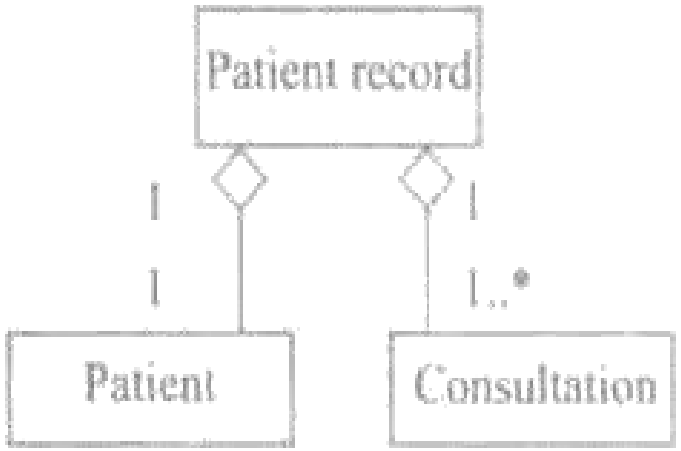
5.3.2泛化

- 我们在日常生活中并不是从经历的所有事情的详细特性中进行学习的，而是学习通用的类(动物、汽车等)及这些类的特性。
- 在此基础上通过对事物进行分类来复用这些知识，并关注它们与它们所属的类之间的差异。
- 在建模系统时，检查系统中是否存在泛化和创建类的空间。这意味通用的信息只需在一个地方保存，变更时只需在最高泛化层上变更。
- 在Java等面向对象语言中，使用类继承的机制来实现。



5.3.3聚集

- 现实世界中的对象经常由不同的部分组成。例如，一个课程的学习包可能包括一本书、 PowerPoint.课堂测验、推荐阅读等。有时候我们需要描述这种组成关系。
- UML的聚集，意思是一个对象(整体)由其他对象(部分)组成。
- 下图表明一个病人记录是一个病人及不确定数量的诊疗的聚集。即，病人记录保存着病人个人信息及每一次医生对其的诊疗记录。



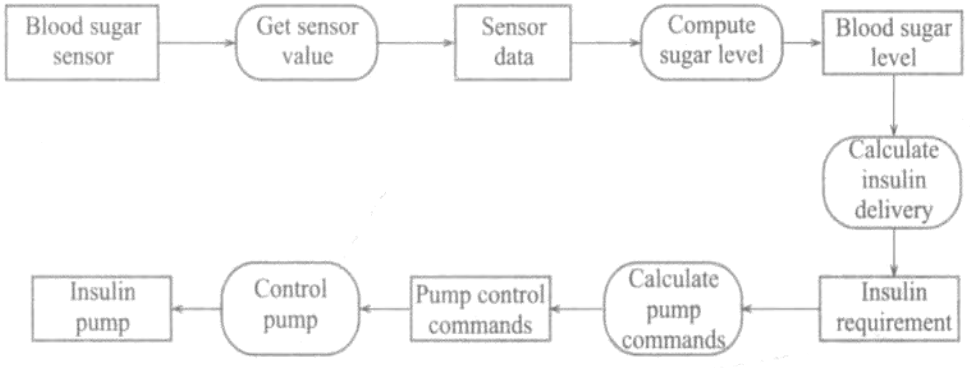
5.4行为模型

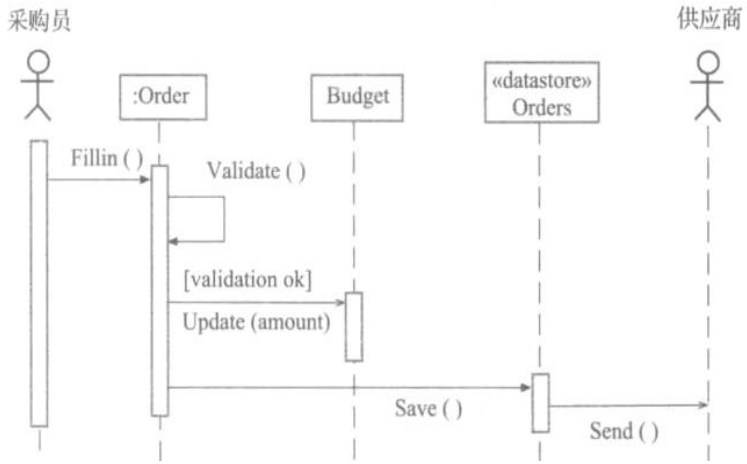
- 行为模型是关于系统在运行时的动态行为的模型，描述当系统对来自环境的激励进行响应时发生什么或者应该发生什么。
- 激励可以是数据或事件，如：
  - 系统必须处理的数据产生了。数据的产生触发相应的处理过程
  - 一个事件发生了，触发了系统处理。事件可以有相关联的数据，虽然并非总是如此。如实时系统通常是事件驱动的。

- 数据流图是展现功能性视角的系统模型，每个转换表示单个功能或过程，用于描述数据流如何从一系列处理步骤中流过。
- UML中的活动图可以用于表达数据流图。

5.4.1数据驱动的建模

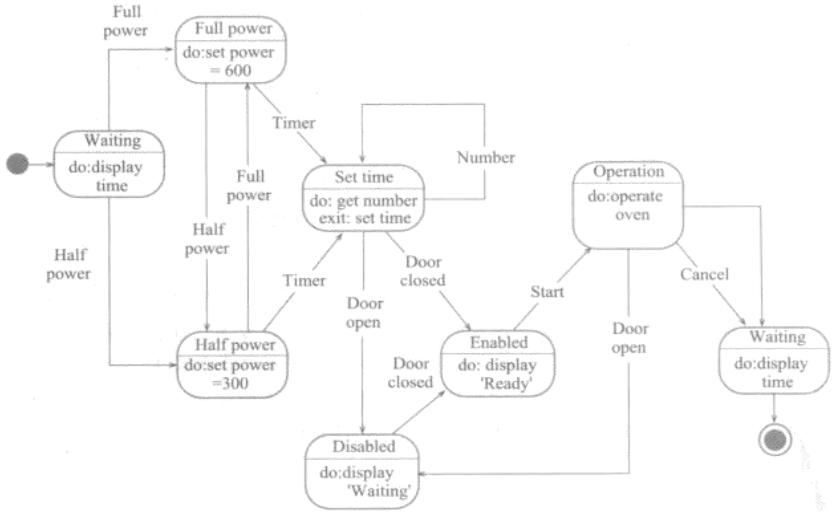
- 数据驱动的模型描述处理输入数据以及生成相关的输出过程中所涉及的动作序列。描述了系统中端到端的处理。
- 需求分析过程中使用
- 通过UML的活动图描述
- 另一种描述系统中的处理序列的方法是使用UML的顺序图



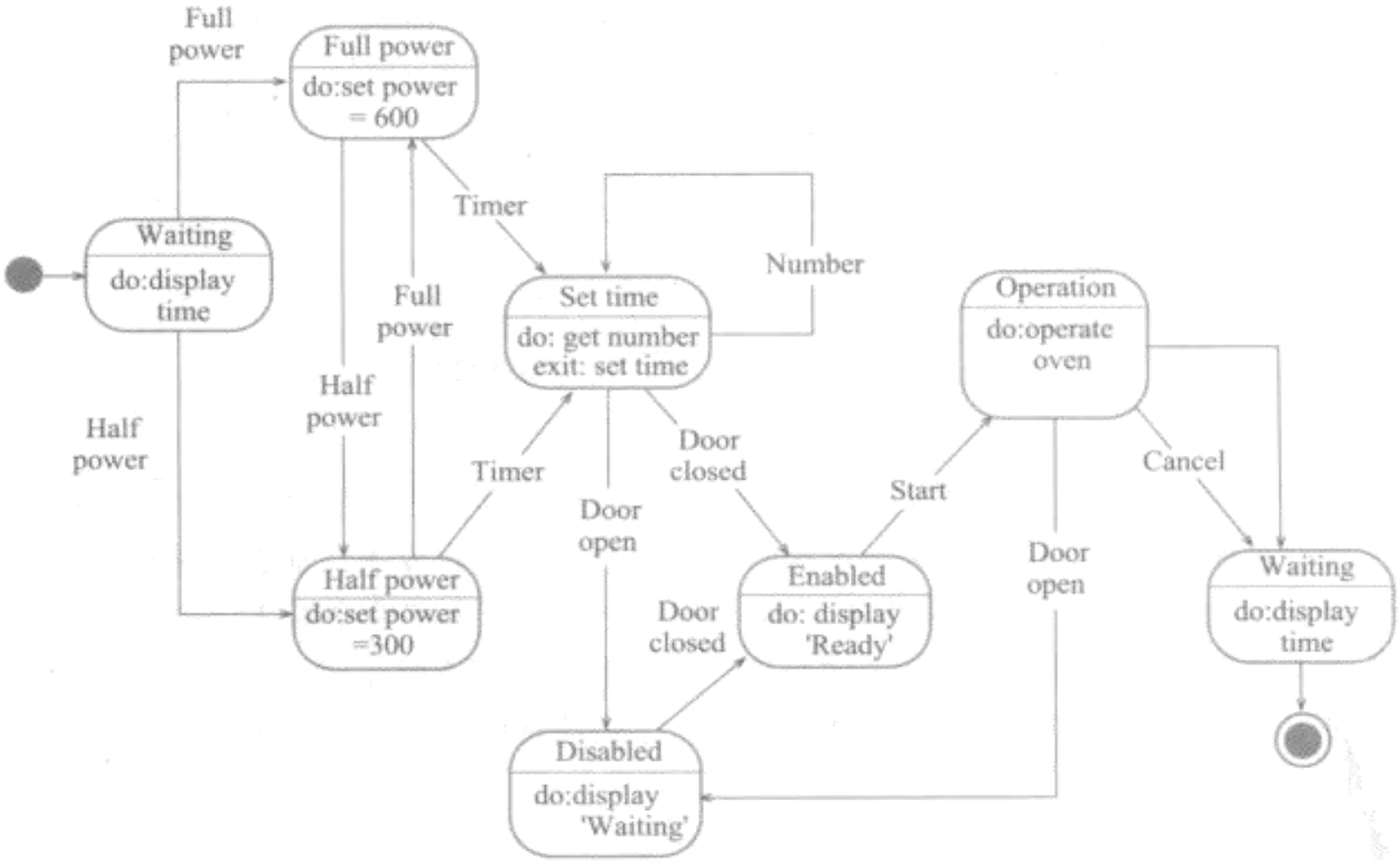


5.4.2事件驱动的建模

- 事件驱动的建模描述一个系统如何对外部和内部事件做出响应。这种模型建立在系统具有有限数量的状态以及事件（激励）可以导致状态间的转换的假设基础上。
- UML通过状态图支持基于事件的建模，
- 状态图描述了系统状态以及导致状态间转换的事件。
- 状态图不会描述系统中的数据流，但是可以包含关于每个状态中所执行的计算的额外信息。

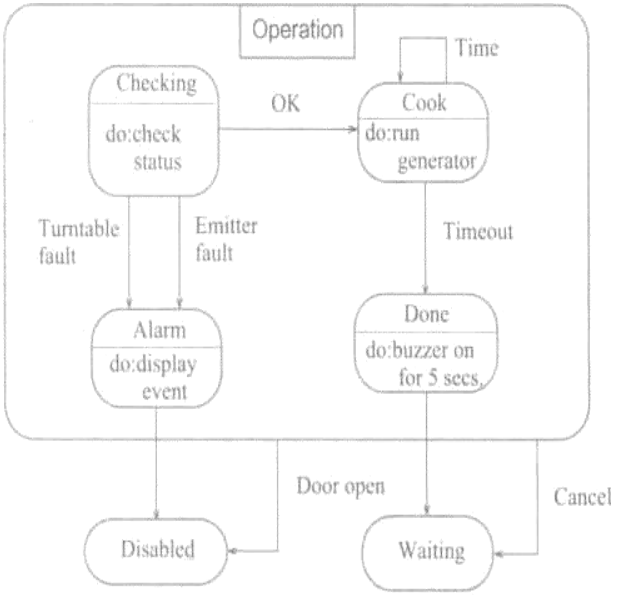


5.4.2事件驱动的建模



5.4.2事件驱动的建模

- 基于状态的建模的问题是，可能的状态的数量会快速增长。因此，对于大的系统模型，需要在模型中隐藏细节。一种隐藏细节的方式是使用“父状态"（superstate）的概念，在其中封装了一些独立的状态。这种父状态看起来像是高层模型中的单个状态，但是可以在另一个图中展开以显示更多的细节。
- 右图是Operation状态的状态模型
- 系统状态模型提供了事件处理概览，但通常还要在此基础上扩展更详细的关于激励以及系统状态的描述。可以使用一个表格来列出状态以及激励状态转换的事件，同时包括对每个状态和事件的描述。



5.4.2事件驱动的建模

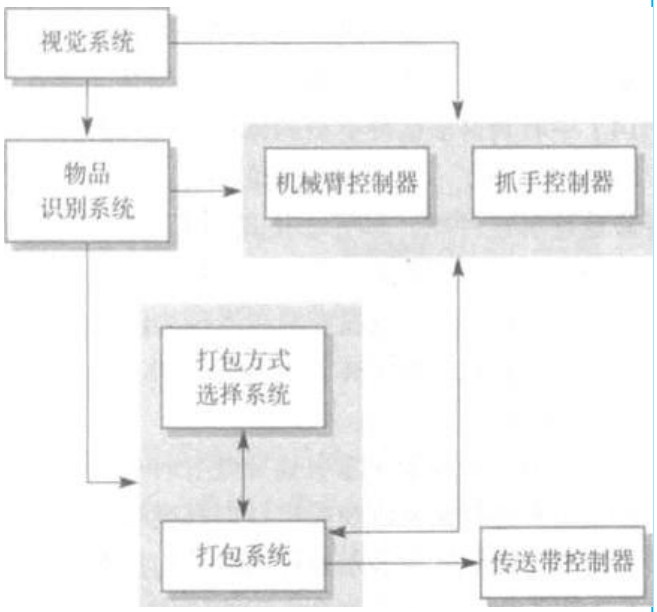
状　态	描　述
等待（Waiting）	微波炉在等待输入。显示器显示当前时间
半功率（Half power）	微波炉功率被设置为 300 瓦特。显示器显示“半功率”
全功率（Full power）	微波炉功率被设置为 600 瓦特。显示器显示“全功率”
设置时间（Set time）	按照用户的输入值设置加热时间。显示器显示所选择的加热时间并在设置时间时进行更新
未激活（Disabled）	微波炉运转出于安全原因未激活。微波炉内部的灯亮起。显示器显示“未就绪”
激活（Enabled）	微波炉运转被激活。微波炉内部的灯关闭。显示器显示“准备加热”
运转（Operation）	微波炉正在运转。微波炉内部的灯亮起。显示器显示计时器倒数计时。加热完成后，蜂鸣器响起 5 秒钟。微波炉灯亮起。显示器在蜂鸣器响的过程中显示“加热完成”
激　励	描　述
半功率（Half power）	用户按下了半功率按钮
全功率（Full power）	用户按下了全功率按钮
计时器（Timer）	用户按下了其中一个计时器按钮
数字（Number）	用户按下了一个数字键
门打开（Door open）	微波炉门开关没有关上
门关上（Door closed）	微波炉门开关关上了
启动（Start）	用户按下了启动键
取消（Cancel）	用户按下了取消键

图 5-18　微波炉的状态和激励

第6章 体系结构设计

- 体系结构设计关注理解一个软件系统应当如何组织，以及设计该系统的整体结构。
- 体系结构设计是设计和需求工程之间关键性衔接环节，因为它会确定组成一个系统的主要结构构件以及它们之间的关系。
- 体系结构设计过程的输出是一个体系结构模型，该模型描述系统如何被组织为一组相互通信的构件。

– 机器人系统可以为不同类型的物品打包。
– 视觉构件来挑选传送带上的物品，识别类型，并选择正确的打包方式
– 将物品从传送带上取下进行打包
– 将打好包的物品放在另一个传送带上
– 这个体系结构模型显示了这些构件以及它们之间的连接关系。



第6章 体系结构设计

- **作为需求工程过程的一部分**，要提出一个抽象的系统体系结构，其中将一组组的系统功能或特征与大规模的构件或子系统关联起来。接下来，要使用这一分解结构来与利益相关者讨论需求以及更加详细的系统特征。

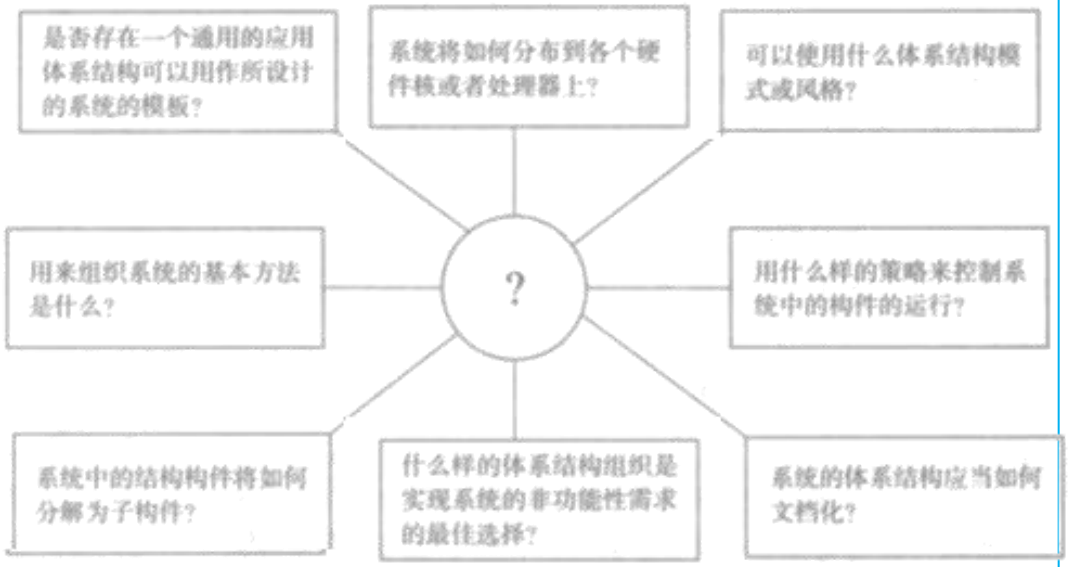
- 可在两个抽象层次上设计软件体系结构
  - 小体系结构关注单个程序的体系结构。关注单个的程序是如何被分解为构件的。
  - 大体系结构关注包括其他系统、程序和程序构件的复杂企业系统体系结构，这些企业系统可以分布在不同的计算机上，这些计算机可以由不同的公司拥有和管理。

第6章 体系结构设计

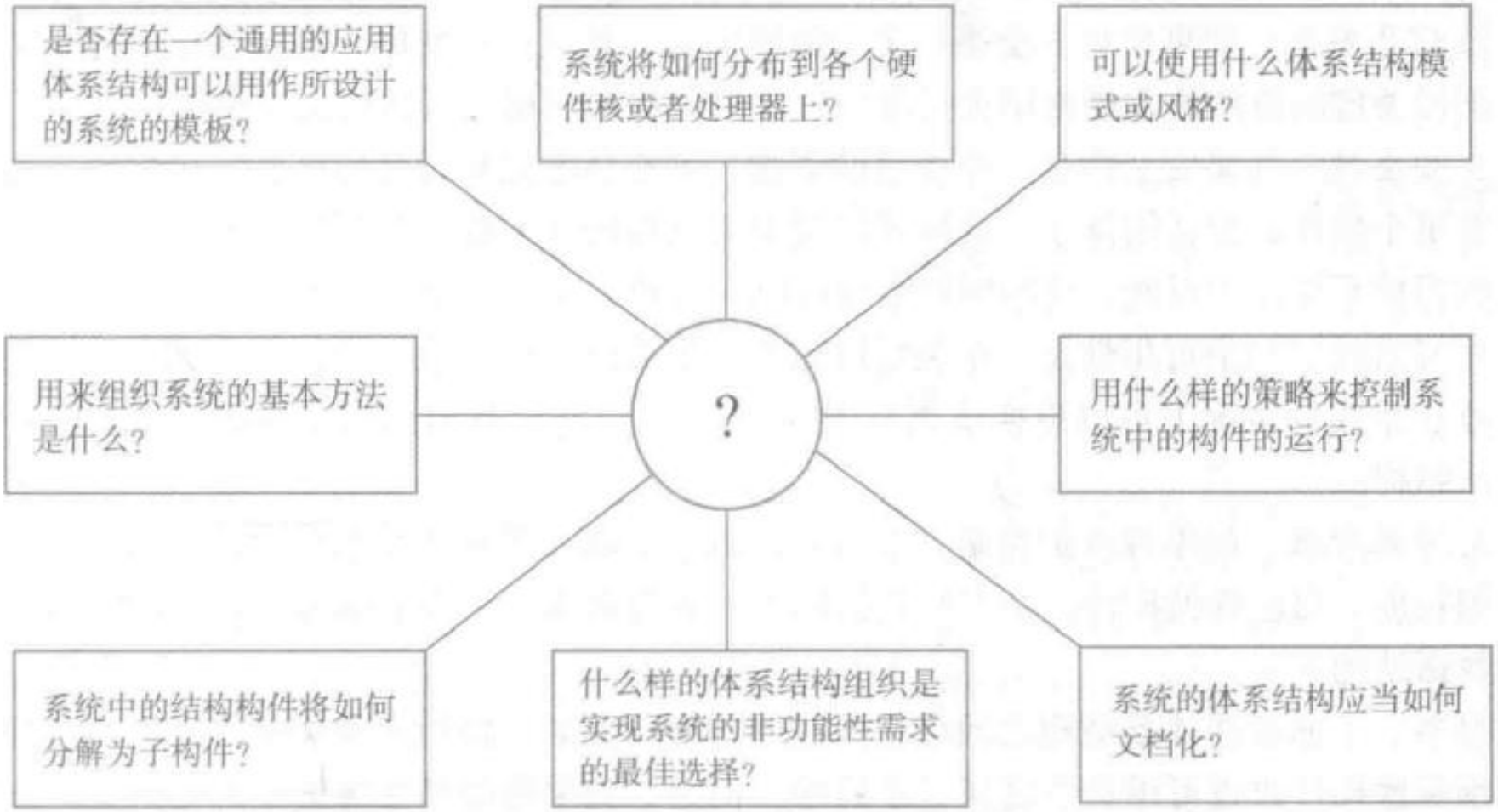
- 软件体系结构很重要，它会影响系统的性能、健壮性、分布性和可维护性。单个构件实现了功能性系统需求，非功能性系统特性的主导影响因素是系统的体系结构。非功能性需求对于系统体系结构的影响最大。设计软件体系结构的3个好处。
  - 1、利益相关者交流。体系结构是一种可以作为许多不同利益相关者讨论的焦点的系统的高层表示
  - 2、系统分析。在系统开发的早期阶段明确系统的体系结构要求进行一些分析。体系结构设计决策对于系统是否能够满足性能、可靠性、可维护性等关键需求具有深远的影响
  - 3、大范围复用。体系结构模型是关于系统如何组织以及构件如何互操作的一个紧凑、可管理的描述。具有相似需求的系统经常具有相同的系统体系结构，因此可支持大范围软件复用

6.1体系结构设计决策

- 体系结构设计是一个创造性的过程，在这个过程中你会设计一个满足系统功能性和非功能性需求的系统组织结构。
- 并不存在公式化的体系结构设计过程。
- 具体的体系结构设计过程 取决于所开发的系统的类型、系统架构师的背景和经验，以及系统的特定需求。
- 最好是将体系结构设计作为一系列的决策而不是活动序列。
- 在体系结构设计中，系统架构师必须做出一系列深刻影响系统及其开发过程的结构决策，基于他们的知识和经验，必须考虑图中所显示基本问题



6.1体系结构设计决策



6.1体系结构设计决策

- 每个软件系统都是独特的，但同一个应用领域中的系统经常具有相似的、反映领域的基本概念的体系结构。
- 当设计系统体系结构时，必须确定系统和更广阔的应用类型有什么共性，确定来自这些体系结构中的知识有多少可以被复用。
  - 针对嵌入式系统及PC和移动设备设计的应用，不需要设计分布式体系结构。大型系统通常是分布式系统。分布式体系结构的选择是一个影响系统性能和可靠性的关键决策。
- 软件体系结构可以基于特定的体系结构模式或风格。是一种系统组织方式的描述，例如，C/S组织或者层次化体系结构。
- 体系结构模式捕捉了一个在不同的软件系统中使用的体系结构的本质特性。在进行系统体系结构决策时应当了解通用的模式、这些模式的适用场合、它们的优势和弱点。6.3节介绍了几个经常使用的模式。

6.1体系结构设计决策

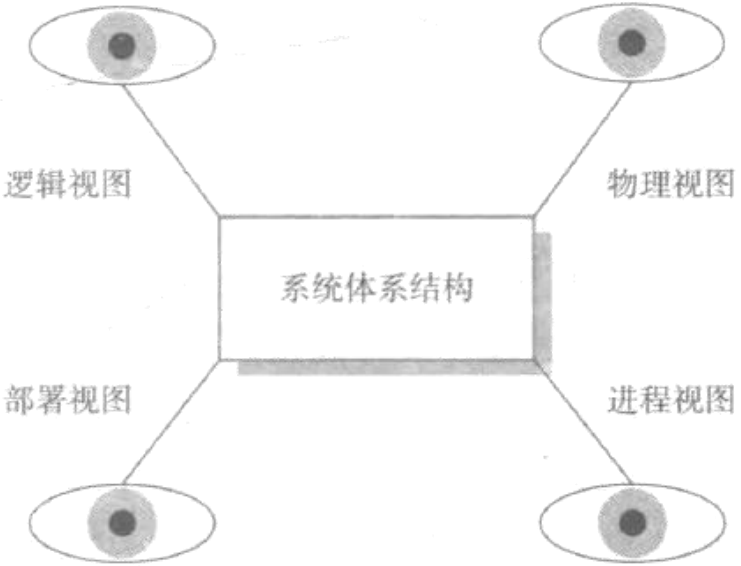
- 体系结构风格和结构选择应当根据系统非功能性需求来决定。
  - 性能。如果是关键性需求，应将关键性操作局部化到少量的构件上，尽量使用一些相对较大颗粒的构件部署在同一台计算机上而不要分布在网络上-。这样可以减少构件间的通信量。还可考虑运行时系统中允许系统多副本部署在不同处理器上执行。
  - 信息安全。如是关键性需求，应该使用层次化体系结构，把最关键的资产放在最内层进行保护，对这些层应用高级的信息安全确认。
  - 安全性。如是关键性需求，应将安全性相关操作集中在单个或少量构件上。减少安全性确认成本和问题，并提供相关的保护系统，保护系统可以在失效发生时安全地关闭系统。
  - 可用性。如是关键性需求，那么体系结构设计应该包含冗余的构件以便在不停止系统的情况下可以更换或更新构件。
  - 可维护性。如是关键性需求，那么系统体系结构设计应当使用容易改变的细粒度、自包含的构件。应当将数据的生产者与数据的消费者相分离，同时应当避免共享的数据结构。

6.2 体系结构视图

- 体系结构模型可以与软件需求或设计相关
- 体系结构模型也可以用于设计的文档化，体系结构可以作为更加详细的系统设计以及实现的基础。
- 我们需要理解下面两个问题
  - 在设计和描述一个体系结构时，哪些视图或视角是有用的?
  - 应当使用哪些表示法来描述体系结构模型?
- 单个图表示出一个体系结构相关的所有信息是不可能的，因为一个图形化模型只能显示一个系统的视图或视角。
- 单个图可以显示系统如何分解为模块、运行时进程如何交互，或者系统构件在一个网络上的不同分布方式。
- 通常都需要从多个不同的视图来呈现软件体系结构。

6.2 体系结构视图

- 著名的“4+1”软件体系结构视图模型提出应当有4个基本的体系结构视图，这些视图可通过共同的用例或场景连接在一起。
  - 逻辑视图，将系统中的关键抽象显示为对象或对象类。应该可以将系统需求与这个逻辑视图中的实体联系起来。
  - 进程视图，显示系统运行时如何通过相互交互的进程来构成。对于做出关于非功能性特性（性能、可用性）的判断很有用。
  - 开发视图。显示软件如何面向开发任务进行分解；即，显示了软件如何分解构件。对于软件开发管理者和程序员都很有用。
  - 物理视图。显示系统硬件及软件构件如何分布在系统中的处理器上。对于规划系统部署方案的系统工程师很有用。



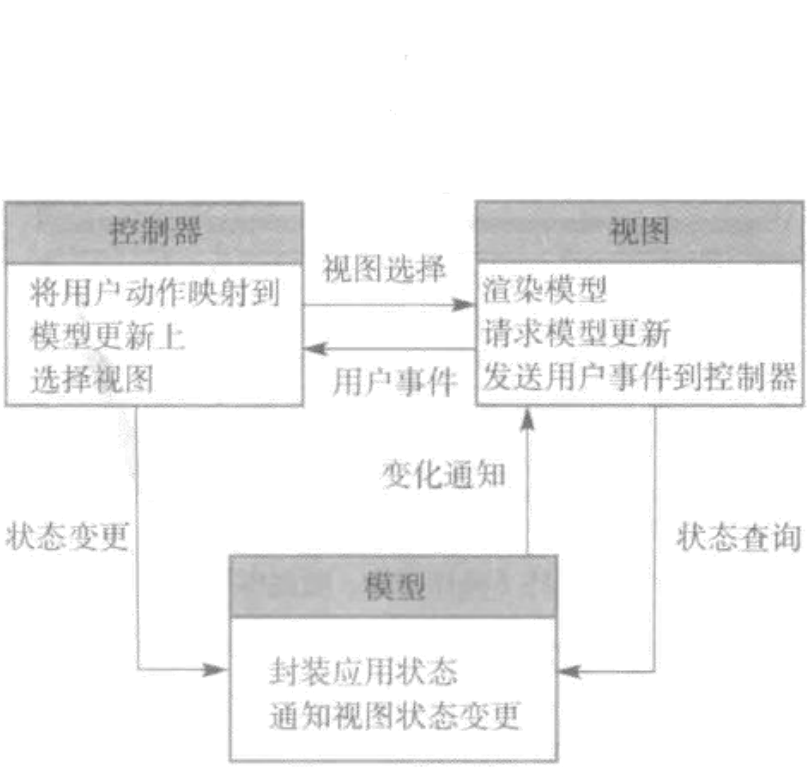
6.3 体系结构模式

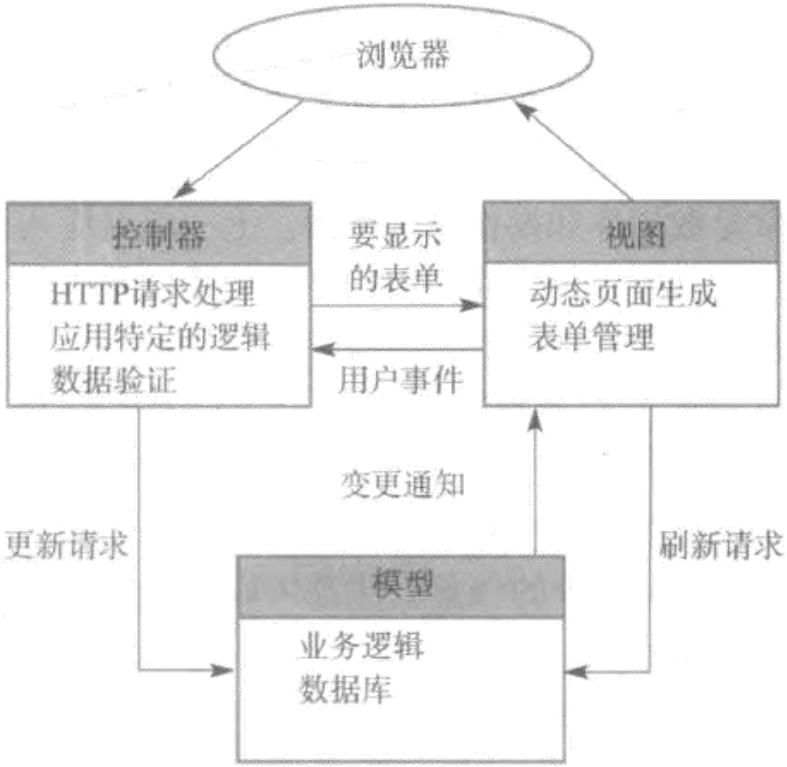
- 将模式作为一种表示、分享、复用软件系统相关知识的思想已在软件工程中的一系列领域中得到采用。在90年代提出体系结构模式。
- 体系结构模式理解为是对好的实践的一种风格化、抽象化的描述。
- 一个体系结构模式应当描述一种已经在此前的系统中得到成功应用的系统的组织。它应当包含何时适合以及何时不适合使用该模式，还应包含关于该模式的优缺点的详细描述等信息。

名 称	MVC （模型-视图-控制器）
描述	将呈现和交互从系统数据中分离出来，系统被组织为3个相互交互的逻辑构件. 模 型（Model）构件管理系统数据以及相关的对这些数据的操作，视图（View 构件定义并管理数据呈现给用户的方式，控制器（Controller）构件管理用户界面（例如按键、 鼠标点击等）并将这些交互传递给视图和模型。
例子	下页图显示了一个使用MVC模式进行组织的基于Web的应用系统的体系结构
何时使用	当存在多种查看数据以及与数据交互的方式时使用，也可以在未来关于数据的交互 和呈现的需求未知时使用
优点	允许数据独立于它的呈现方式进行变更，反之亦然。支持以不同的方式呈现同样的数据。在某一个呈现方式中进行的修改可以在所有呈现方式中显示
缺点	当数据模型和交互比较简单时，可能包含额外的代码以及代码复杂性

6.3体系结构模式

- 与MVC模式相关的体系结构的图形化模型显示





- 模型-视图-控制器的组织　　使用MVC模式的Web应用体系结构

6.3.1分层体系结构

- 分离和独立性的思想是体系结构设计的基础，因为这可以使变更被局部化。MVC模式将系统的元素分离，允许它们独立变更。例如，增加一个新的视图或者改变一个现有的视图，可以在不改变所依赖的模型中的数据的情况下实现。
- 分层体系结构模式是另一种实现分离和独立性的方式，系统功能被组织为多个分离的层次，每个层次只依赖于紧邻的下一层所提供的设施和服务。这一分层的方法支持系统的增量开发。一个层次开发好之后，该层次所提供的一些服务可以提供给用户。
- 该体系结构也是可变化和可前移的。如果接口不变，那么一个功能进行扩展的新层可在不影响系统其他部分的情况下取代已有层。
- 当一个层次的接口变化或者增加新设施时，只有相邻的层次受影响。
- 分层系统将机器的依赖局部化了，这使提供一个应用系统的多平台实现变得更容易了。只需将与机器有依赖关系的层重新实现，便可以适应不同的操作系统或数据库的设施。

6.3.1分层体系结构

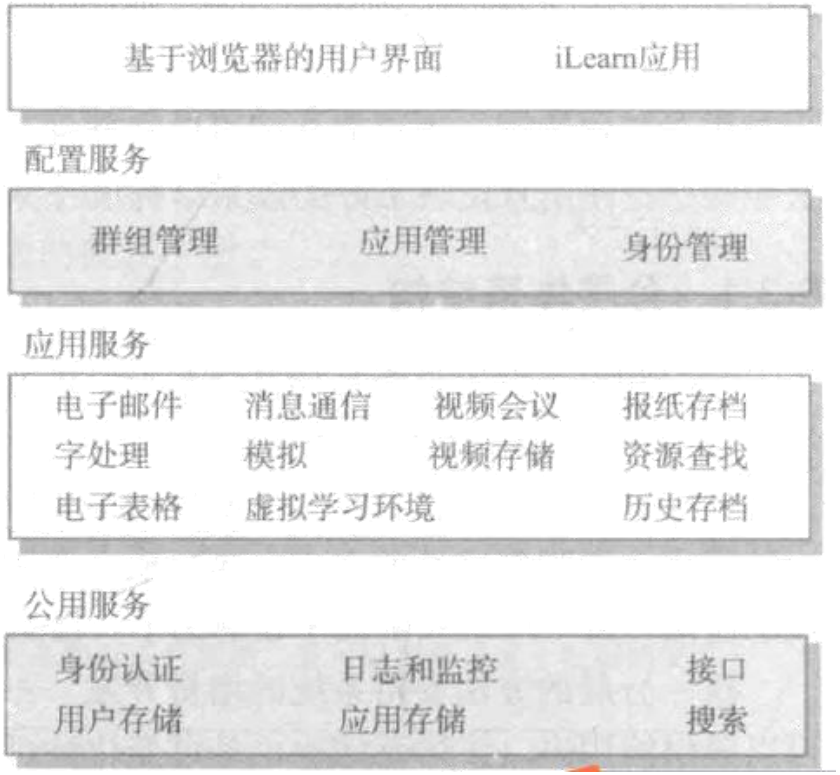
名 称	分层体系结构
描述	将系统组织为多个层次，每个层次与一些相关的功能相联系：每个层次向其上的一层提供服务，因此那些最底的层次表示很有可能在整个系统中使用的核心服务
例子	一个数字化学习系统的分层模型支持学校中所有科目的学习
何时使用	在已有系统之上构建新的设施时使用；开发涉及多个团队，每个团队负责一个层次上的功能时使用；存在多个层次上的信息安全需求时使用
优点	只要接口保持不变，允许对整个层进行替换，可以在每个层次上提供冗余设施（如，身份认证）以增加系统的可依赖性
缺点	在实践中，将各层之间干净地分离经常很难做到，较高层次上的层可能不得不与较低层次上的层直接交互，而不是通过紧邻着的下一层。性能可能是一个问题，因为当一个服务请求在各个层次上进行处理时需要经过多层的解析

6.3.1分层体系结构

- 例如，一个4层的分层体系结构。最底层包括系统支持软件，通常是数据库和操作系统支持。上一层是应用层，其中包括关注应用功能的构件以及由其他应用构件使用的公用构件。
- 右图是一个通用的分层体系结构
  - 从下向上数第三层关注用户界面管理以及提供用户身份认证和授权，最顶层提供用户界面设施。层的数量是任意的。任意一层都可以进一步划分为两层或更多层。
  - iLearn 数字化学习系统就有一个遵循这一模式的4 层体系结构。

6.3.2知识库体系结构

- 作为体系结构模式的例子，分层体系结构和MVC模式所呈现的视图都是一个系统的概念化组织。
- 下一个体系结构模式的例子是知识库模式，它描述了一组相互交互的构件如何共享数据。
  - 大多数使用大量数据的系统都是围绕一个共享数据库或知识库组织的。
  - 这个模型适合于数据由一个构件生成同时由另一个构件使用的应用。
  - 这类系统例子包括指挥控制系统、管理信息系统、计算机辅助设计系统、交互式软件开发环境等。

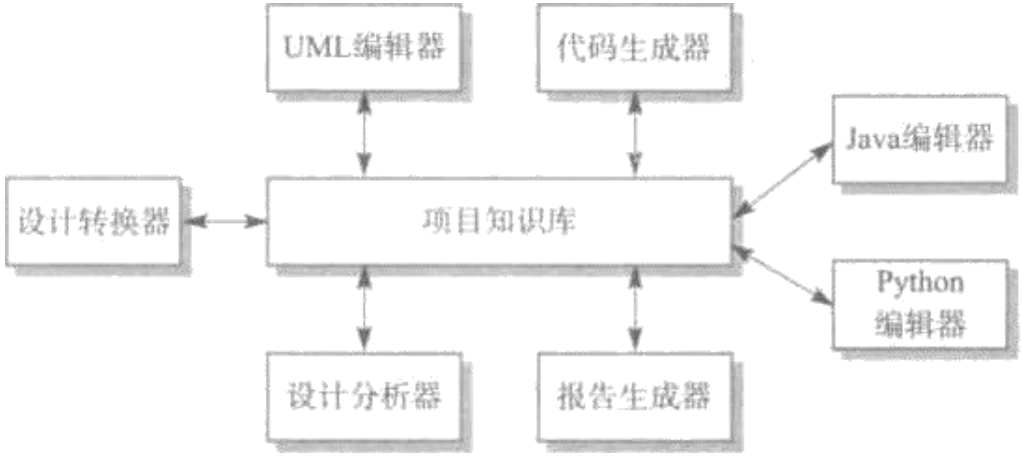


6.3.2知识库体系结构

名称	知识库
描述	一个系统中的所有数据在所有系统构件都能访问的中心知识库中进行管理。构件相互之间并不直接交互，仅通过知识库进行交互
例子	一个集成开发环境（IDE）的例子，其中构件使用系统设计信息的知识库。每个软件工具生成的信息都会提供给其他工具使用
何时使用	当系统生成大量需要长时间保存的信息时应当使用这个模式。还可以在数据驱动的系统中使用，这类系统中的知识库中增加新数据时会触发一个动作或工具
优点	构件可以保持独立，它们不需要知道其他构件的存在；一个构件进行的修改可以被传播到所有构件；所有数据都可以一致地进行管理（例如同时进行备份），因为这些数据都位于一个地方
缺点	知识库可能存在单点失效问题，因此知识库中的问题会影响整个系统；通过知识库组织所有的通信可能效率不高；将知识库分布到多台计算机上可能比较困难

6.3.2知识库体系结构

- 下图面向集成开发环境（IDE），其中包含不同的支持模型驱动开发的工具。知识库可以是一个追踪对软件的修改并且允许回滚到早期版本的版本控制环境。
- 围绕知识库组织工具是共享大量数据的有效方法。不需显式地在构件间传输数据。构件须围绕达成共识的知识库数据模型运转。
- 要对每个工具的特定需要进行权衡，而且如果一个新构件与数据模型不相符，那么就可能很难甚至无法集成了



6.3.2知识库体系结构

- 在实践中，可能很难将知识库分布到多台不同的机器上。虽然有可能实现一个逻辑上的集中知识库的分布式部署，这其中会涉及维护数据的多份拷贝。保证这些拷贝的一致性以及及时更新，会给系统增加更多的额外负担。

- 知识库是被动的，使用知识库的构件负责进行控制。另一种从人工智能（AI）系统中派生出来的方法是，使用一个“黑板”模型在特定的数据可用时触发构件运行。当知识库中的数据是非结构化数据时，这种方法是合适的。关于应当激活哪个工具的决策只能在对数据进行分析之后做出。

6.3.3客户-服务器体系结构

- 知识库模式关注系统的静态结构，没有显示系统的运行时组织。
- 客户-服务器模式描述了一种常用的分布式系统运行时的组织方式。一个遵循客户-服务器模式的系统被组织为一组服务和相关联服务器，以及访问并使用服务的客户端。主要构件包括：

– 一组向其他构件提供服务的服务器。服务器的例子包括：提供打印服务的打印服务器，提供文件管理服务的文件服务器，提供编程语言编译服务的编译服务器。服务器是软件构件，多个服务器可能运行在同一台计算机上。
– 一组调用服务器提供的服务的客户端，通常会有一个客户端程序的多个实例并行运行在多台计算机上。
– 一个允许客户端访问这些服务的网络。客户-服务器系统通常实现为分布式系统，使用互联网协议连接。

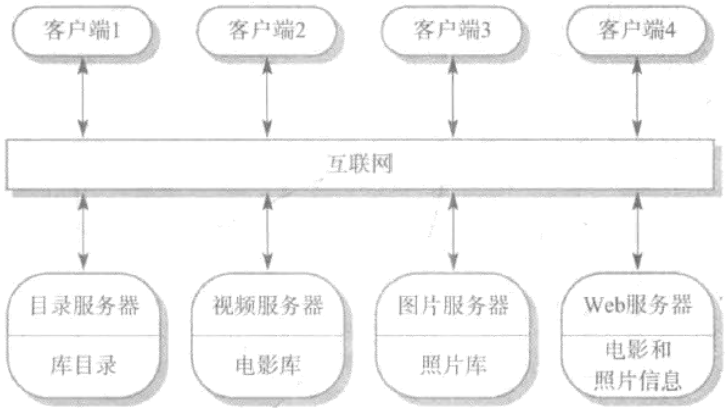
6.3.3客户-服务器体系结构

名 称	客户-服务器
描述	系统被呈现为一组服务，其中每个服务都由一个独立的服务器提供。客户端是这些服务的用户。通过访问服务器来利用这些服务。
例子	一个组织成客户-服务器系统的电影和视频/DVD库的例子
何时使用	当一个共享数据库中的数据必须从一系列不同的位置进行访问时使用，因为服务器可以复制。因此也可以在一个系统的负载多变的情况下使用
优点	主要优点是服务器可以分布在网络上。通用的功能（例如一个打印服务）可以向所有客户端开放使用，不需要由所有服务实现。
缺点	每个服务都存在单点失效的问题，因此容易受到拒绝服务攻击或服务器失效的影响；性能可能不可预测，因为这取决于网络以及系统；如果服务器属于不同的组织，那么还可能会出现管理问题。

- B-S体系结构被认为是分布式系统体系结构。重要优势仍然是分离和独立性。服务和服务器可以在不影响系统的其他部分的情况下被修改。
- 客户端可能必须知道可用的服务器以及它们所提供的服务的名称。
- 但是服务器不需要知道客户端的身份或者有多少客户端正在访问它们的服务。客户端使用请求-应答协议（http）的远程过程调用访问一个服务器提供的服务

6.3.3客户-服务器体系结构

- 这是基于B-S的例子，提供电影和照片库的多用户、基于Web的系统。
- 多个服务器管理并显示不同类型的媒体，视频帧需要以同步的方式快速传输，分辨率相对较低。压缩存储库中，视频服务器以不同的格式处理视频压缩和解压缩。图片则必须以一种高分辨率保存。因此将它们保存在一个单独的服务器上更为适合。
- 目录必须处理多种不同的查询，并提供到Web信息系统的链接，包括电影和视频片段的数据，及支持照片、电影、视频片段销售的电子商务系统。客户端使用Web浏览器访问这些服务。
- B-S模型优势在于它是一个分布式体系结构，带有很多分布式处理器的网络化系统可以获得有效的使用。很容易增加一个新的服务器并将其与系统的剩余部分相集成，或者在不影响系统其他部分的情况下透明地升级服务器。



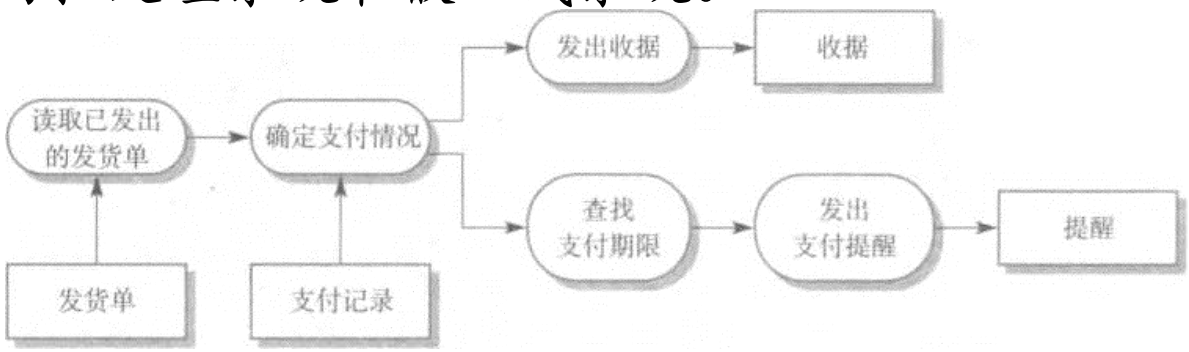
6.3.4管道和过滤器体系结构

- 这是个系统的运行时组织结构的模型，
- 它通过功能性的变换来处理输入并产生输出，数据从一个构件流动到另一个构件，在流经某序列时进行变换，每个处理步骤被实现为一个变换。
- 输入数据经过这些变换直到被转换为输出。变换可以串行或并行执行。每个变换可以一项项地处理数据，也可以在单个批次中一次性处理。

名 称	管道和过滤器
描述	系统中的数据处理通过组织，每个处理构件（过滤器）可以分离开来并执行一种数据转换。数据从一个构件流动到另一个构件进行处理
例子	一个用于处理发货单的管道和过滤器系统的例子
何时使用	在数据处理应用（无论批处理还是事务）中得到了广泛应用，这类应用中输入在多个分离的阶段中进行处理，并最终生成相关的输出
优点	容易理解并且支持变换的复用；这种工作流风格与许多业务过程的结构相匹配；通过增加变换来进行演化是非常直观的；可以被实现为一个顺序系统或一个并发系统
缺点	针对数据转换的格式必须在相互通信的多个变换之间达成一致；每个变换必须解析它的输入，并将输出转换成所约定的形式，增加了系统的负担，无法复用使用不兼容的数据结构的体系结构构件

6.3.4管道和过滤器体系结构

- 管道和过滤器这个名字来自于Unix操作系统。
- 从计算机最初被用于自动化地数据处理时，就已经使用该模式的变体了
- 当变换是顺序化的而所处理的是批量数据时，这个管道和过滤器体系结构模型就成为批量顺序模型，一种通用的数据处理系统（如账单系统）的体系结构。
- 例子：一个组织向客户发出发货单。每个星期他们都会将已经收到的付款与这些发货单进行对账。对于已经付款的发货单，他们会发出一个收据。对于那些在所允许的支付时间内还没有付款的发货单，他们会发出一个付款提醒。
- 最适合于用户交互很少的批处理系统和嵌入式系统。
- 交互式系统很难使用管道和过滤器模型，因为管道和过滤器模型需要处理数据流



6.4 应用体系结构

- 应用系统的目的是满足企业的需要。企业有很多共性：都要雇人、开发货单、保存账户等；同行业的企业使用通用的行业特定的应用
- 因此，与通用企业的功能一样，所有的电话公司需要系统连接呼叫和计费、管理网络、向客户发出账单。它们的应用系统有很多共性。
- 共性促使人们去开发描述特定类型软件系统结构和组织软件体系结构
- 应用体系结构封装了一类系统的主要特性。
- 例如实时系统可能包含不同的通用体系结构模型，如数据收集系统或监控系统。虽然实例细节存在差异，但是共性体系结构可以被复用。
- 应用体系结构可以在开发新系统时重新实现。
- 然而，对于许多企业系统，当通过配置通用的应用系统来创建一个新应用时，应用体系结构的复用是隐式的。
- 在ERP及可配置的成品应用系统(如财务和仓库管理)中可以看到，这些系统有标准的体系结构及相关的构件。这些构件可以通过配置和适应性调整来创建一个特定的企业应用。
- 例如，一个供应链管理系统可通过适应性调整来支持不同类型的供应商、商品、合同关系。

6.4应用体系结构

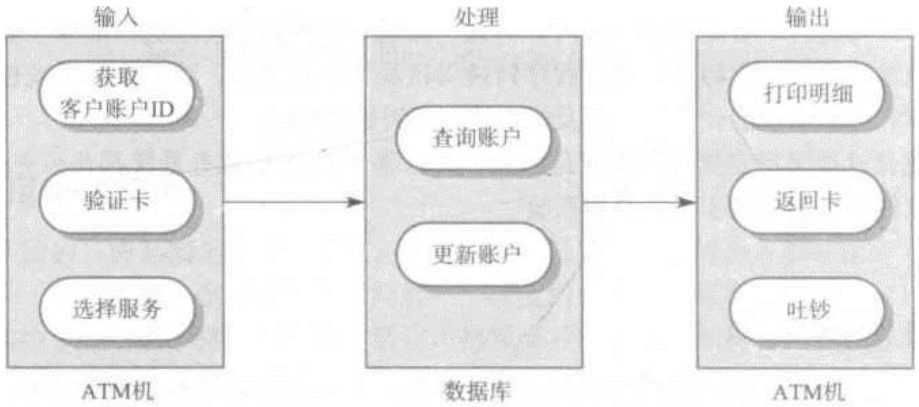
- 软件设计者可通过以下不同的方式来使用应用体系结构模型。
  - 1、体系结构设计过程的起点。如果不熟悉开发应用类型，可以将初始设计建立在一个通用的应用体系结构基础上。在此基础上可针对所开发的特定系统对体系结构进行特化。
  - 2、作为一个设计检查表。如果已经为应用系统设计了体系结构，可将它与通用体系结构比较，检查设计是否与通用的体系结构一致。
  - 3、作为组织开发团队工作的一种方式。应用体系结构识别出稳定的结构特征，这些结构单元很多时候可以并行开发。可以按照这一结构向团队成员分配工作，实现体系结构中不同构件。
  - 4、作为评价构件复用的一种方式。如果有可复用的构件，将它们与通用结构相比较，确定应用体系结构中是否有与之相当的构件。
  - 5、作为谈论应用时的一种词汇表。如果正在讨论一个特定的应用或者试图比较不同的应用。那么可以使用通用体系结构中所识别的概念来讨论这些应用。

6.4应用体系结构

- 许多类型的应用系统，看上去很不一样。然而，表面上不一样的应用可能有很多共性，因此可以共享一个抽象应用体系结构。
  - 1、事务处理应用。事务处理应用是以数据库为中心的应用，处理用户的信息请求并且更新数据库中的信息。这些系统是最常见的交互式业务系统类型。这些系统的组织结构使用户动作不会相互干涉，而数据库的完整性得以保持。这类系统包括交互式银行系统、电子商务系统、信息系统、预订系统。
  - 2、语言处理系统。语言处理系统中用户的意图用形式化语言（例如编程语言）来表达，语言处理系统将这种语言处理成一种内部格式，然后对这种内部表示进行解释。最著名的语言处理系统是编译器，它将高层语言程序转换为机器代码。然而，语言处理系统也可以用于解释数据库和信息系统的命令语言以及标记语言（例如XML）。
- 介绍上面这两类特定类型的系统是因为，大量基于Web的业务系统都是事务处理系统，而所有的软件开发都依赖于语言处理系统。

6.4.1 事务处理系统

- 事务处理系统用于处理用户获取数据库中信息的请求或者更新数据库的请求。
- 从技术上看，一个数据库事务包含一个操作序列并且该序列作为整体处理（一个原子单元）。
- 一个事务中的所有操作必须在数据库修改被持久化之前完成。这保证了一个事务中失败的操作不会导致数据库中的不一致。
- 从用户的角度看，一个事务是任何满足一个目标的内聚的操作序列，例如“查找从伦敦到巴黎的航班时间”。如果用户事务不需要对数据库进行修改，那么不一定要将其作为一个技术上的数据库事务。
- 数据库事务例子，一个客户使用一台ATM机请求从一个银行账户中取一些钱。在所有步骤完成之前，这个事务就不会提交。



6.4.2信息系统

- 所有包含与共享数据库交互的系统都可认为是基于事务的信息系统。
- 信息系统允许对一个很大的信息库（例如，图书馆目录、航班时间表或者医院里的病人记录）的受控访问。信息系统几乎都是基于Web的系统，其中用户界面在一个Web浏览器中实现。
- 通用的信息系统模型

图 6-18　分层的信息系统体系结构

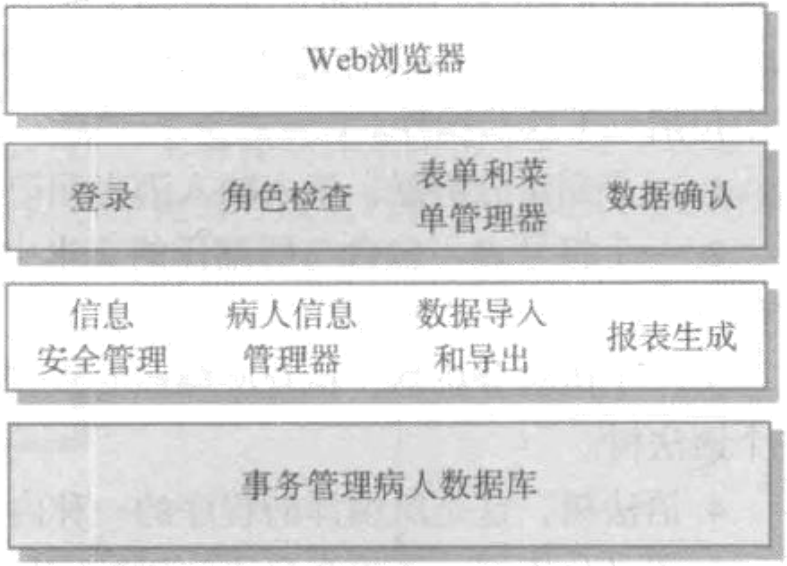


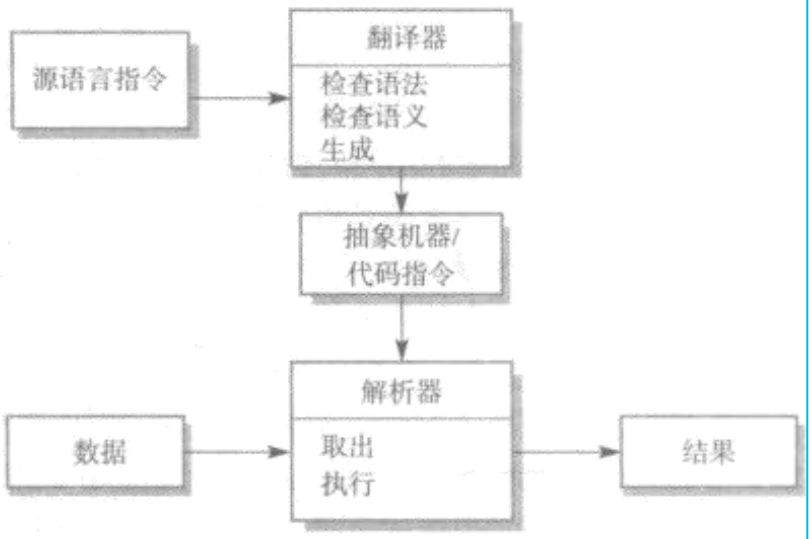
图 6-19　Mentcare 系统的体系结构

6.4.2信息系统

- 信息和资源管理系统有时候也是事务处理系统。
- 例如：电子商务系统是基于互联网的资源管理系统，接受商品或服务的电子订单，然后安排这些商品或服务向客户交付。系统应用特定层包括附加功能以支持购物车，用户可以通过多个分开的事务放入一些购物项，然后在一个事务中一起支付所有购物项。
- 系统中的服务器组织通常反映了4层通用模型。这些系统经常实现为带有多层客户端/服务器体系结构的分布式系统。
  - 1、Web服务器负责所有的用户通信，并带有使用Web浏览器实现的用户界面。
  - 2、应用服务器负责实现特定应用逻辑以及信息存储和检索请求。
  - 3、数据库服务器将信息移动到数据库，从数据库中获取数据，同时处理事务管理。
- 使用多个服务器可以实现高吞吐量，使每分钟处理成千上万的事务成为可能。随着请求量的增长，每个层次都可以增加服务器以应对所需要的额外处理能力。

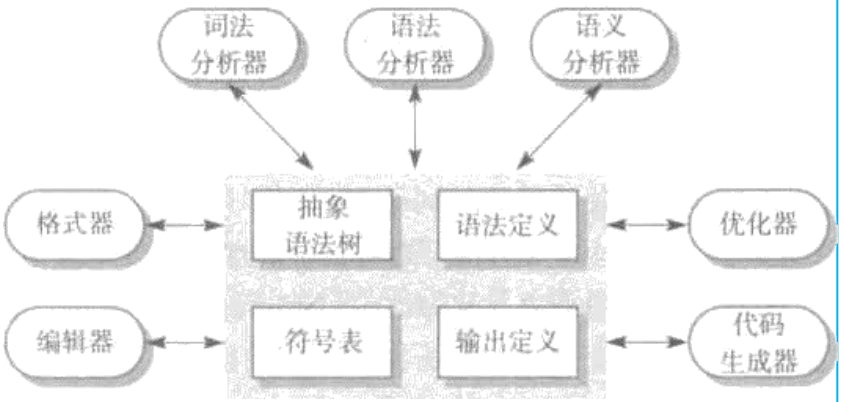
6.4.3语言处理系统

- 语言处理系统将一种语言翻译为另一种表示方式，对于编程语言还可以执行所产生的代码。
- 如：编译器将一种编程语言翻译为机器代码。
- 其他语言处理系统可以将XML数据 描述翻译为数据库查询命令或者另一种XML表示形式。
- 自然语言处理系统可将一种自然语言翻译为另一种语言。
- 面向编程语言的语言处理系统体系结构如图所示。
- 源语言指令定义了要执行的程序，一个翻译器将这些转换为面向抽象机器的指令。指令被另一个构件解析，该构件取出指令并且使用来自环境的数据执行这些指令。



6.4.3语言处理系统

- 作为编程环境的一部分，编程语言编译器具有一个通用的体系结构：
  - 1、词法分析器，读入输入语言标记符号，并转换为一种内部形式。
  - 2、符号表，包含与所翻译的文本中使用的实体（变量、类名、对象名等）名称相关的信息。
  - 3、语法分析器，检查所翻译的语言语法。它使用该语言所定义的语法，并且构造一个语法树。
  - 4、语法树，这是所编译程序的一种内部表示结构。
  - 5、语义分析器，使用来自语法树的信息以及符号表来检查输入语言文本的语义正确性。
  - 6、代码生成器，遍历语法树并且生成抽象的机器代码
- 还可包含其他对语法树进行分析和转换以提高效率并从所生成的机器代码中消除冗余的构件。



第7章 设计和实现

- 本章的目标是介绍使用UML的面向对象软件设计，并强调一些重要的实现关注点。
  - 理解一个通用的面向对象设计过程中最重要的活动；
  - 理解一些不同的面向对象设计描述模型；
  - 了解设计模式的思想，以及如何利用设计模式实现设计知识和经验的复用；
  - 了解在实现软件时必须考虑的一些关键问题，包括软件复用和开源开发。

第7章 设计和实现

- 软件设计和实现是软件工程过程中的一个阶段，在此阶段中会开发一个可执行的软件系统。
- 对于简单系统，软件工程意味软件设计和实现，合并了其它活动。
- 对于大型系统，软件设计和实现只是一系列软件工程过程中的一个。
- 软件设计和实现活动活动一般总是存在重叠。
- 软件设计是创造性活动，基于客户需求识别软件构件及其关系。
- 实现是将设计实现为一个程序的过程。
- 设计是关于如何解决一个问题，因此总是有一个设计过程。
- 设计和实现密切联系，设计时通常都应该将实现问题考虑进来。
- 例如，如果使用面向对象语言(Java或C#)编程，那么使用UML描述一个设计可能是正确的选择。如果使用Python这样的动态类型语言，那么UML就没那么有用了。如果通过配置一个成品软件包来实现系统，那么使用UML就没什么道理了。

第7章 设计和实现

- 在软件项目早期阶段必须的决策是，确定要构建还是购买应用软件。
- 许多类型的应用，能买到可以按照用户需求进行适应性调整和裁剪的成品应用系统。如，购买一个已经在使用的软件包。采用这种方法通常比用传统开发语言开发新系统会更加便宜也更快。
- 通过复用一个成品软件产品开发一个应用系统时，设计过程关注如何配置该系统产品 以满足应用需求。而不需要开发系统的设计模型，例如，关于系统对象及其交互的模型。
- 本章有以下两个目的：
  - 1、展示系统建模和体系结构设计如何在开发一个面向对象软件设计时进行实践应用；
  - 2、介绍在编程相关的书中通常不会涉及的一些重要的实现问题，包括软件复用、配置管理和开源开发。
- 由于存在大量不同的开发平台，我们不会关注特定的编程语言，所有的例子都是用UML而不是编程语言(例如Java或Python)来呈现的。

7.1使用UML的面向对象设计

- 一个面向对象系统由相互交互的对象组成，这些对象保持自己的本地状态同时基于状态 提供相应的操作。
- 状态的表示是私有的，无法从该对象的外部直接访问。
- 面向对象设计过程 包括设计对象类以及这些类之间的关系。
- 这些类定义了系统中的对象以及它们的交互。
- 当设计被实现为执行程序时，对象会在这些类定义的基础上被动态创建出来。
- 对象同时包括数据以及操纵这些数据的操作。因此，它们可以作为独立的实体进行理解和修改。
- 改变一个对象的实现或者增加服务不应该影响其他系统对象。因为对象与事物相关 联，现实世界实体(例如硬件构件)和它们在系统中的控制对象之间经常存在清晰的映射。
- 这改进了设计的可理解性，因此也有利于可维护性。

7.1使用UML的面向对象设计

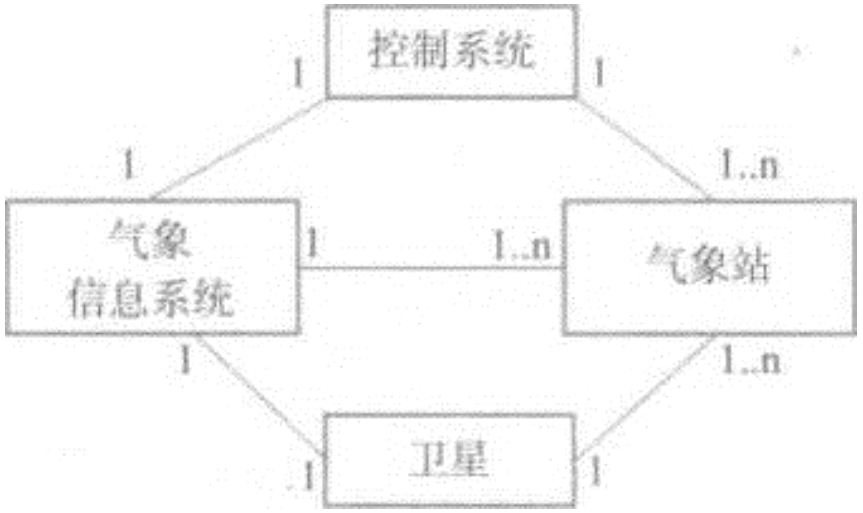
- 为了开发一个系统设计(从概念到详细的面向对象设计)，需要:
  - 1、理解并定义上下文以及与系统的外部交互;
  - 2、设计系统体系结构;
  - 3、识别系统中的主要对象;
  - 4、开发设计模型;
  - 5、刻画接口。
- 注意，设计不是一个清晰的顺序性过程。
- 在开发一个设计的过程中会获得想法，提出解决方案，随着更多信息的获得来精化这些解决方案。
- 当出现问题时，不可避免地必须回溯并重新尝试。
- 有时候要详细探索各种选项看它们是否奏效；而其他时候则会忽略细节直到过程中的后期。有时候使用UML等建模表示法来精确地阐明设计的各个方面；而其他时候，则可以以非正式的方式使用一些表示法来激发讨论。

7.1.1 系统上下文和交互

- 任何软件设计过程的第一个阶段都是理解所设计的软件与外部环境间的关系。这很重要，因为可以据此决定如何提供所需系统的功能，以及如何对系统进行组织从而与环境进行通信。
- 设定系统边界可以帮助开发人员决定哪些特征在所设计的系统中实现，以及哪些特征在其他相关联的系统中实现。
- 在野外气象站案例中，需要决定功能如何在所有气象站的控制软件以及气象站自身的嵌入式软件之间进行分布。
- 系统上下文模型和交互模型所呈现的关于系统及其环境之间关系的视图是互补的。
  - 1、系统上下文模型是一种结构化模型，其中展示了所开发的系统的环境中的其他系统。
  - 2、交互模型是一种动态模型，其中显示系统在使用时如何与环境进行交互。

7.1.1 系统上下文和交互

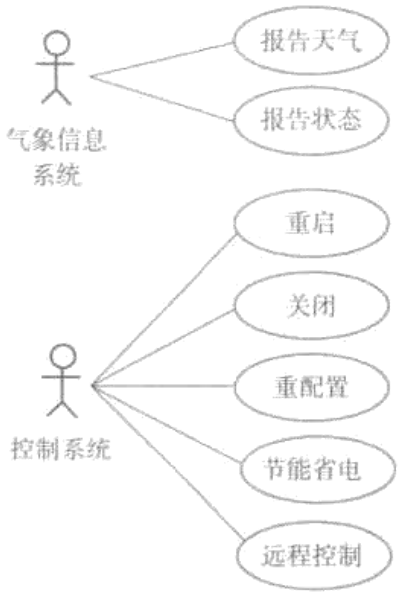
- 一个系统的上下文模型可以使用关联表示。关联只是显示关联中所包含的实体间存在某些关系。可以使用简单的框图来描述系统的环境，显示系统中的实体以及它们之间的关联关系。
- 图7-1显示了气象站的系统上下文，每个气象站环境中的系统包括一个气象信息系统、一个卫星系统以及一个控制系统。链接关系上的基数信息显 示，有一个控制系统、多个气象站、一个卫星、一个通用的气象信息系统。



7.1.1 系统上下文和交互

- 当对系统与环境间的交互进行建模时，应使用一种不包含太多细节的抽象方法。可以使用用例模型，每个用例表示与系统的一种交互。
- 气象站的用例模型如图所示。图中显示气象 站与气象信息系统进行交互以报告气象数据以及气象站硬件的状态。
- 还包括与一个控制系统的交互，通过该交互可以发出特定气象站控制命令。线条人形在UML中用于表示其他系统以及人用户。
- 气象站用例

– 报告天气——发送天气数据到气象信息系统
– 报告状态——发送状态信息到气象信息系统
– 重启——如果气象站被关闭了，那么重启系统
– 关闭——关闭气象站
– 重配置——重配置气象站软件
– 省电——将气象站设置为节电模式
– 远程控制——发送控制命令到任何气象站子系统



7.1.1 系统上下文和交互

- 这些用例中的每一个都应当使用结构化自然语言进行描述。
- 有助于设计人员识别系统中的对象并理解系统意图要做什么。
- 下面是一种标准格式描述，识别要交换什么信息、交互如何发起等。
- 如嵌入式系统经常通过描述它们如何响应内部或外部激励的方式进行建模。因此，激励以及相关的响应在描述中进行列举。

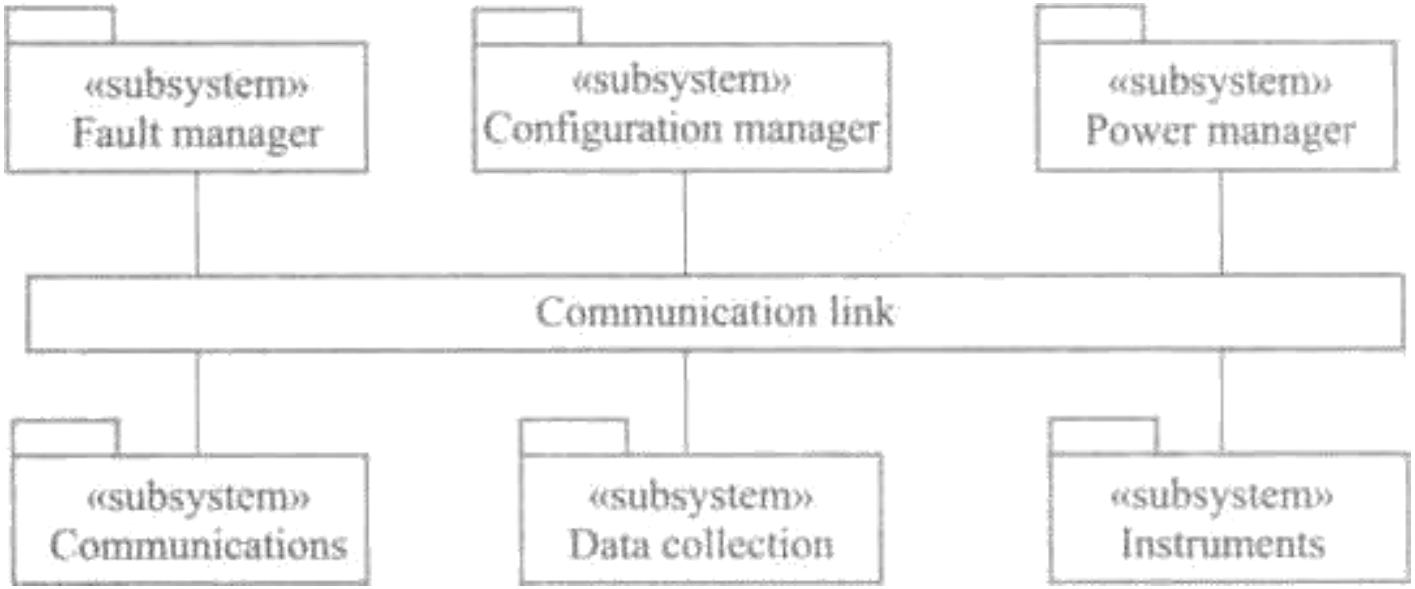
系统 气象站
用例 报告天气
参与者 气象信息系统，气象站
数据 气象站将在收集阶段从各种仪器上收集的气象数据的一个汇总发送给气象信息系统。所发送的数据包括最高、最低以及平均的地面温度和空气温度；最高、最低以及平均的气压；最高、最低以及平均的风速；总降雨量；以5分钟间隔采样的风向。
激励 气象信息系统与气象站之间建立一个微信通信链路并请求传输数据
响应 汇总后的数据被发送到气象信息系统。
备注 通常会让气象站每小时报告一次，但不同气象站的报告频率可能会有所差别并且可能在未来发生变化。

7.1.2体系结构设计

- 软件系统和系统环境之间的交互定义好之后，就可以以此为基础来设计系统体系结构。
- 设计人员需要将这一知识与自身关于体系结构设计原则的通用知识以及更加详细的领 域知识相结合。
- 需要识别构成系统的主要构件以及它们之间的交互，然后要使用一种体系结 构模式(如，分层或B-S模型)来设计系统组织结构
- 气象站软件的高层体系结构设计如图所示。
- 气象站由一些独立子系统组成：故障管理器(Fault manager)、配置管理器(Configuration manager)、电源管理器(Power manager)、通信(Communications)、数据收集(Data collection)、仪器(Instruments)子系统。

7.1.2体系结构设计

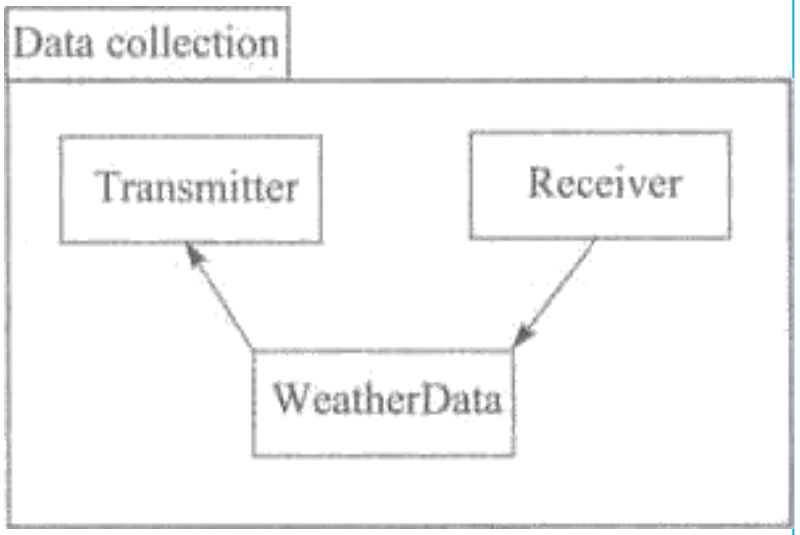
- 这些子系统通过在公共基础设施上广播消息来进行通信，如图中的通信(Communication)链路所示。每个子系统都会在该基础设施上监听消息并选取与它们相关的消息。
- 这一“监听器模型”是一种广泛使用的分布式系统体系结构风格。
- 气象站的高层体系结构



7.1.2体系结构设计

- 当通信子系统收到一个控制命令(例如关闭)时，其他每个子系统也将获得该命令，这些子系统接收者会执行命令(如，按照正确的方式关闭自身)。
- 这一体系结构的关键优点是很容易支持不同的子系统配置，因为一条消息的发送者不需要指定接收该消息的特定子系统。
- 这里展示了上图中所包含的数据收集子系统的体系结构。

– 其中的发送器(Transmitter) 和接收器(Receiver)对象关注通信管理，而气象数据(WeatherData)对象则封装从仪器那里收集并传送给气象信息系统的信息。
- 这一设计遵循了生产者-消费者模式。



7.1.3对象类识别

- 在设计过程的阶段之前，我们应该对系统中要包含的重要对象已经有了一些想法。
- 随着对设计的理解进一步加深，要对这些对象想法进行精化。
- 用例描述帮助我们识别系统中的对象和操作。
- 从“报告天 气”用例描述可以看到， 我们要实现两个对象：表示收集气象数据的各种仪器对象，及一个表示气象数据汇总的对象。通常需要一个高层的系统对象或用来封装用例中定义的系统交互的对象。脑海中有了这些对象后，就可以开始识别系统中的通用对象类。
- 不同的识别面向对象系统中对象类的方法：

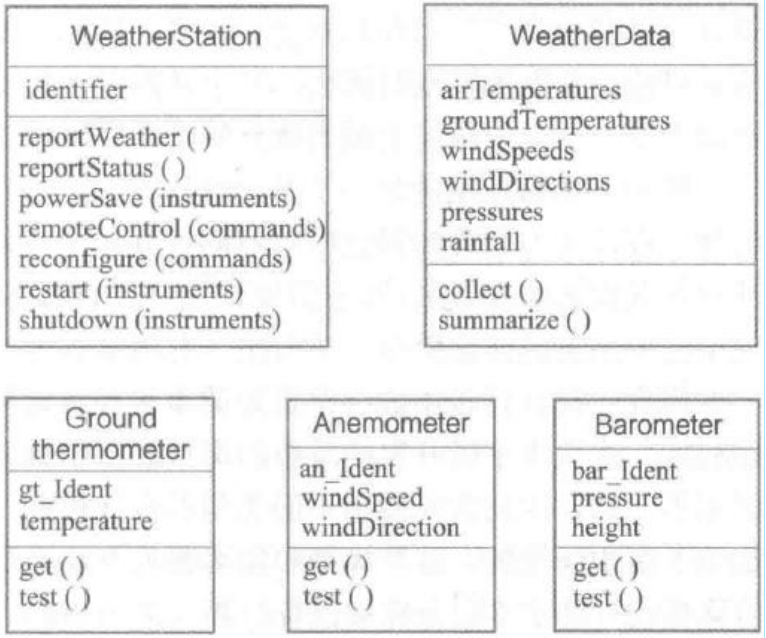
– 1、分析系统自然语言描述：对象和属性是名词，操作或服务是动词。
– 2、使用领域中有形实体或事物(如飞机)、角色(如经理)、事件(如请求)、 交互(如会议)、位置(如办公室)、组织单元(如公司)等。
– 3、使用一种基于场景的分析方法，其中依次识别并分析系统使用的各种场景。分析每个场景时，负责分析的团队必须识别所需要的对象、属性以及操作。

7.1.3对象类识别

- 实践中必须使用多种知识源来发现对象类。最初从非正式的系统描述中所识别出的对象类、属性和操作可以成为设计的起点。
- 接下来，来自应用领域知识或场景分析的信息可以用于精化和扩展初始的对象。
- 这些信息可以从需求文档、与用户的讨论或者对现有系统的分析中收集到。除了表示系统外部实体的对象，还必须设计出用于提供通用服务（例如，搜索和正确性检查）的“实现对象”。
- 在野外气象站中，对象识别是基于系统中有形的硬件。由于篇幅原因，这里无法列举所有的系统对象。图7-6中展示了5个对象类。Ground thermometer （地面温度计）、Anemometer（风速计）和Barometer（气压计）对象是应用领域对象，Weatherstation（气象站）和 WeatherData（气象数据）对象是从系统描述和场景（用例）描述中识别出来的。

7.1.3对象类识别

- Weatherstation对象类提供了气象站及其环境的基本接口。这里使用单个对象类，其中 包括所有这些交互。也可以将系统接口设计为多个不同的类，每个类对应一个交互。
- WeatherData对象类负责处理报告天气的命令。它将来自气象站仪器的汇总数据发送 到气象信息系统。
- Ground thermometer， Anemometer和Barometer对象类直接与系统中的仪器相关联。它们反映了系统中有形的硬件实体，而它们的操作关注控制这些硬件。这些对象自治地运行，按照指定的频率收集数据，并在本地保存所收集的数据。这些数据在请求时提供给 WeatherData对象。



7.1.3对象类识别

- 可以使用应用领域知识来识别其他对象、属性和服务。
  - 气象站经常位于偏远的地方，而且所部署的各种仪器有时会坏。仪器失效应当自动报告。这意味着你需要提供用于检查这些仪器是否正确工作的属性和操作。
  - 有许多远程气象站，因此每个气象站应当有自己的标识符以便在通信过程中进行唯一标识。
  - 气象站是在不同时间安装的，仪器类型会不一样。因此，每个仪器也应被唯一标识，且应当要维护一个仪器信息数据库。
- 在这个设计过程阶段中，应当关注对象自身，不需要考虑这些对象要如何实现，一旦识别出这些对象后，可以再进行精化。发现共性特征，然后为系统设计继承层次。
- 例如，识别出一个Instrument(仪器)父类，其中定义了所有仪器的共性特征(如，标识符、读取信息操作、测试操作等)。还可以向父类中增加新属性和操作，例如，一个记录数据应当多久收集一次的属性。

7.1.4 设计模型

- 设计或系统模型，展示了一个系统中的对象或者对象类，还展示了实体之间的关联和关系。
- 这些模型是系统需求和系统实现之间的桥梁。是抽象的，以使得不必要的细节不会隐藏它们和系统需求之间的关系。模型还必须包含足够的细节，以使得程序员可以做出实现决策。
- 设计模型所需要的详细程度取决于开发人员所使用的设计过程。
- 需求工程师、设计人员和程序员之间存在密切的联系，抽象模型可能就是所有所需要的东西。
- 特定设计决策会在实现时做出，问题通过非正式讨论解决。如果使用敏捷开发，在白板上画出概要设计模型就足以提供所需信息了。
- 如果使用一个基于计划的开发过程，需要更详细的模型。如果需求工程师、设计人员和程序员间仅有间接联系，则需要详细设计描述以便于沟通。此时要使用从高层的抽象模型中派生出来的详细模型，从而使所有的团队成员对于设计有一个共同的理解。

7.1.4 设计模型

- 设计过程中一个重要的步骤是决定所需要的设计模型及模型所需要的详细程度。
- 这取决于开发的系统类型。顺序性数据处理系统与嵌入式实时系统就很不一样，因此需要使用不同类型的设计模型。
- UML支持13种不同类型的模型。尽量减少所产生的模型的数量可以降低设计的成本以及完成设计过程所需的时间。
- 使用UML来开发一个设计时，应当开发两类设计模型。
  - 结构模型，使用静态类及其关系描述系统的静态结构。这个阶段需要描述的重要的关 系类型包括泛化（继承）关系、使用/被使用关系、组合关系。
  - 动态模型，描述了系统的动态结构并展示了所期望的系统对象之间的运行时交互。可以描述的交互包括对象发出的服务请求的序列以及由这些对象交互所触发的状态变化。

7.1.4 设计模型

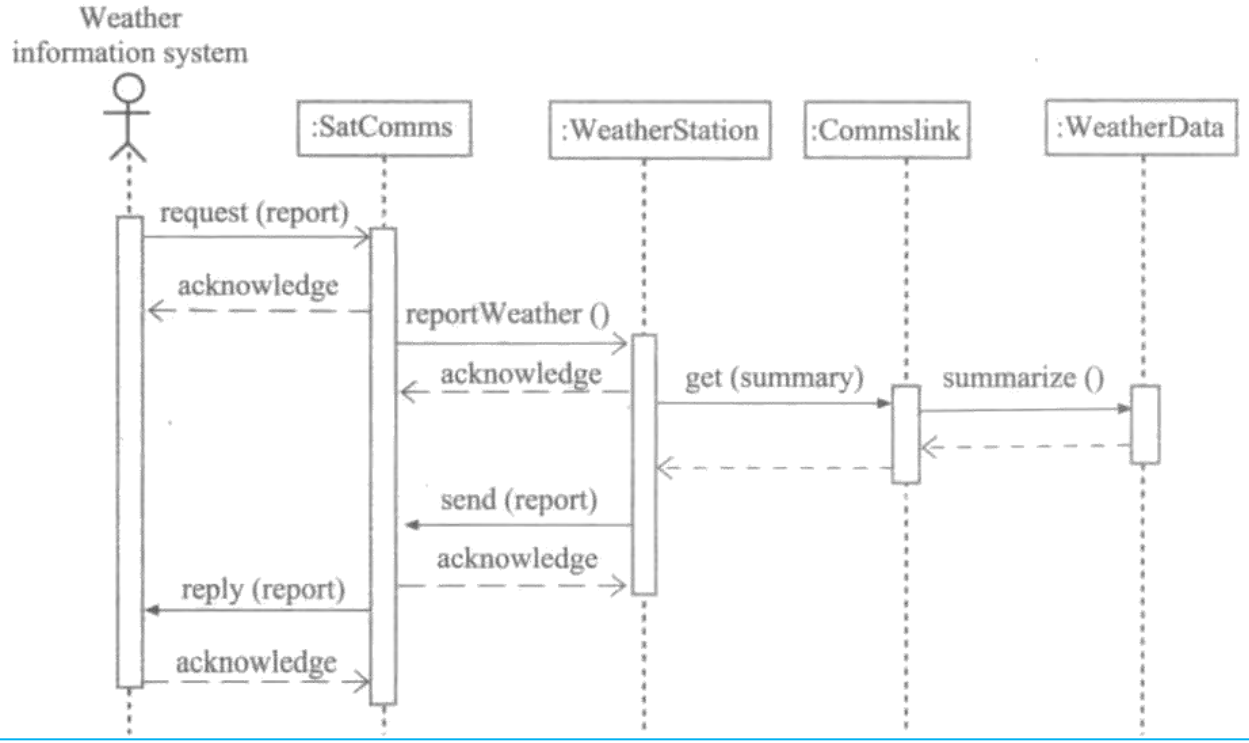
- 下面这3种UML模型对于增加用例和体系结构模型的细节很有用。
  - 1、子系统模型，展示了如何对对象进行逻辑分组以构成内聚的子系统。这些可以使用一种类图的形式来表示，其中每个子系统表示为一个包含对象的包。子系统模型是结构模型。
  - 2、顺序模型，展示了对象的交互序列。这些可以使用UML顺序图或协作图来表示。顺序模型是动态模型。
  - 3、状态机模型，展示了对象如何在事件响应中改变自己的状态。这些可以使用UML状态图来表示。状态机模型是动态模型。
- 子系统模型是有用的静态模型，展示如何将一个设计组织为逻辑上相关的对象 组。除了子系统模型，还可设计详细对象模型，展示系统中对象及它们的关联关系(继承、泛化、聚 集等)。
- 顺序模型是动态模型，描述了每个交互模式下所发生的对象交互序列。描述一个设计时，应当为每个有意义的交互产生一个序列模型。如果已经开发了一个用例模型，那么每一个所识别出的用例都应该有一个顺序模型。

7.1.4 设计模型

- UML顺序模型的例子，下图展示了当一个外部系统向气象站请求汇总数据时发生的交互序列。
  - SatComms (气象站命令)对象接收来自气象信息系统的请求，从气象站收集气象报告。确认收到请求。
  - SatComms对象通过一个卫星链路发送消息给Weatherstation(气象站)对象以创建一个所收集的气象数据汇总。
  - Weatherstation对象发消息给Coinmslink(通信链路)对象汇总气象数据。
  - Commslink对象调用WeatherData(气象数据)对象汇总方法并等待回复。
  - 气象数据汇总完毕通过Commslink对象返回Weatherstation对象。
  - Weatherstation对象接着调用SatComms对象，将汇总后的数据通过卫星通信系统传送给气象信息系统。
- SatComms和Weatherstation对象可以被实现为并发进程，它们的执行可以被挂起和恢复。SatComms对象实例监听来自外部系统的消息，对这些消息进行解码，并初始化气象站操作。

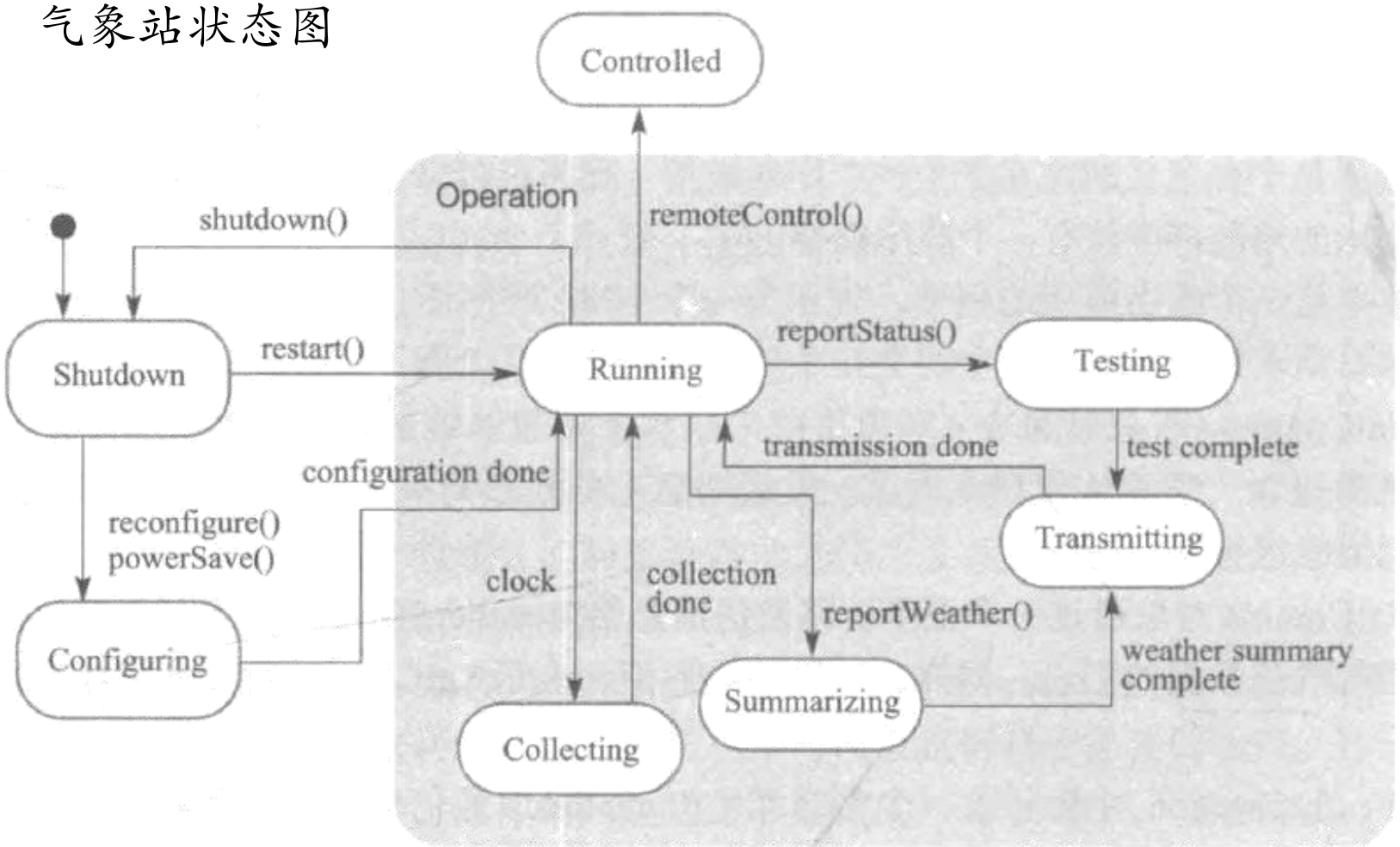
7.1.4 设计模型

- 顺序图用于对一组对象的组合行为进行建模
- 如果要按照消息或事件对一个对象或子系统的行为进行总结，则可以使用一个状态机模型来展示对象实例如何根据所收到消息改变状态。



7.1.4 设计模型

- 气象站状态图



7.1.4 设计模型

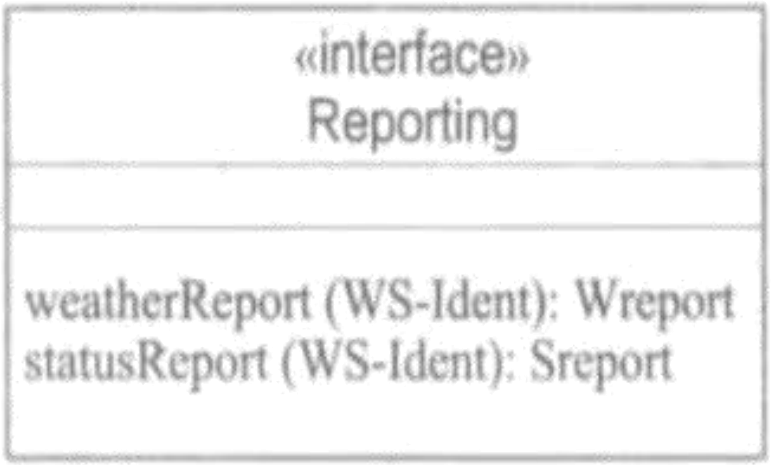
- 系统是 Shutdown (关闭)状态时，可以对 restart()、reconfigure()或 powerSave()消息进行响应。Shutdown是初始状态。restart() 消息使得状态转换为正常运行状态。powerSave()和reconfigure()消息使系统转换为对自身进行重配置状态。只有在系统关闭状态下才能重配置。
- Running(运行)状态，系统等待进一步的消息。如果收到 shutdown() 消息，则对象返回关闭状态。
- 收到reportWeather()消息，进入Summarizing(汇总)状态。汇总完成后，进入Transmitting(发送)状态，信息发送远程系统。然后返回Running状态。
- 收到时钟信号，那么系统会进入Collecting (收集)状态，系统收集仪器数据：仪器按顺序接受指令收集相关联传感器数据。
- 如果收到remoteControl()消息，则系统进入一个受控的状态，在此状态下系统对 另一组来自远程控制室的消息进行应答。

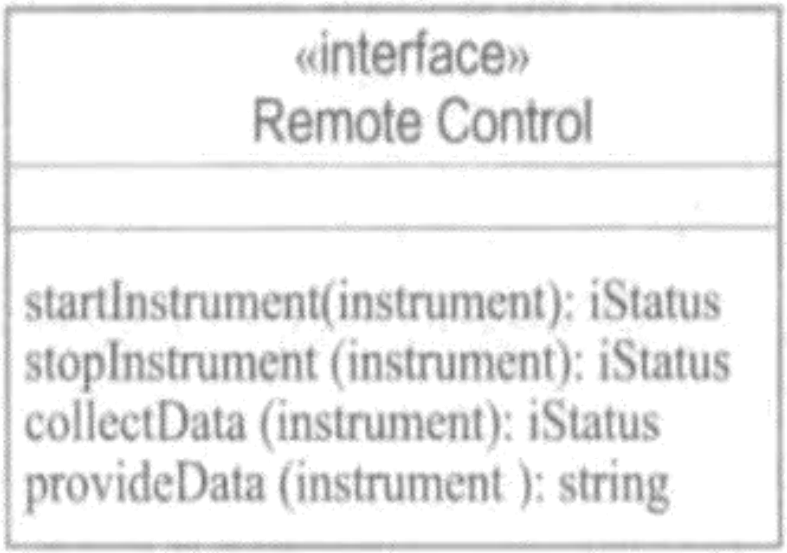
7.1.5接口规格说明

- 这是设计中的重要部分，要设计构件之间的接口规格说明。
- 接口设计关注刻画一个对象或一组对象的接口细节。要定义由一个对象或者一组对象所提供服务的**型构(signature)**和语义。
- UML中接口可以用类图一样的表示法来刻画。接口没有属性部分，而名称部分应当包含UML构造型(stereotype) «interface» 。接口的语义可以使用对象约束语言(OCL)来定义。
- 接口中不包含数据表示细节，接口规格说明中不会定义属性。
- 接口设计应当包含访问和更新数据的操作。由于数据表示被隐藏，因此可以在不影响使用该数据的对象的情况下很容易地修改数据表示。这样可以具有更好的可维护性。例如，一个堆栈的数组表示可以在不影响使用堆栈的其他对象的情况下改为列表表示。
- 对象与接口间不是一对一的关系。一对象可有多个接口，其中每个接口都是一个对于该对象所提供的方法的视角。这些咋Java中都是支持的，接口是独立于对象进行声明的，而对象会“实现”接口。同样，一组对象也可以通过单个接口一起访问到。

7.1.5接口规格说明

- 下图显示气象站定义的两个接口。左侧的接口是一个报告(Reporting)接口，定义了用于生成气象和状态报告的操作的名称。这些直接与Weatherstation对象中的操作相对应。
- 远程控制(Remote Control)接口提供了4个操作，它们映射到Weatherstation对象的同一个方法上。例子中，每个不同的操作都是在与remoteControl方法相关联的命令字符串中进行编码的。



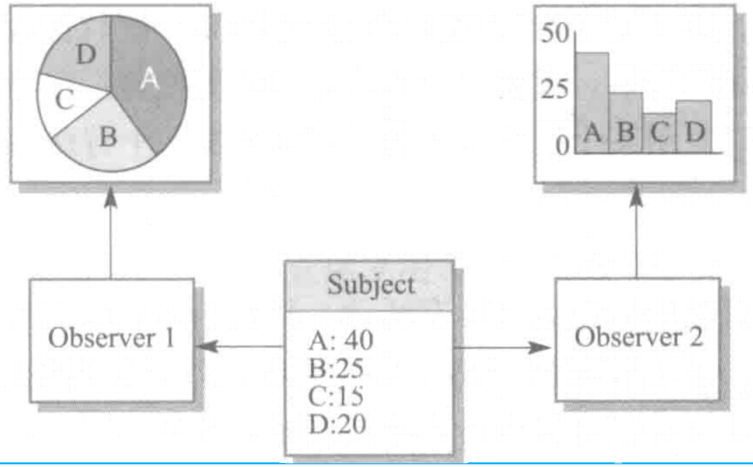


7.2设计模式

- 模式是一种对于问题及其解决方案的本质的描述，从而使得解决方案可以在不同的环境中进行复用。模式不是一种详细的规格说明，应当将模式理解为一种对于逐渐累积的智慧和经验的描述，一种对某个共性问题的经过成功尝试的解决方案。
- 模式和模式语言是描述最佳实践、好的设计的方式，其中捕捉了相关的经验，从而使其他人复用这些经验成为可能。
- 模式对于面向对象软件设计已经产生了巨大的影响。除了作为针对共性问题的经过验证的解决方案之外，模式还已经成为一种谈论设计的词汇表。我们可以通过描述所使用的模式来解释设计。
- 模式是一种复用其他设计者的知识和经验的方式。设计模式通常与面向对象设计相关联。
- 模式经常依赖于继承和多态等对象特性来提供通用性。
- 可以拥有针对实例化可复用的应用系统的配置模式。

7.2设计模式

- 设计模式的4个基本元素。
  - 1、名字，作为对模式的有意义的参照。
  - 2、问题域的描述，解释了该模式何时适用。
  - 3、对于设计解决方案的各个部分、它们之间的关系以及职责的描述。这不是一个具体的设计描述，而是一个设计解决方案的模板，可以以不同的方式进行实例化。模板经常可以通过图形化的方式进行表达，展示解决方案中的对象和对象类之间的关系。
  - 4、对效果的陈述——应用该模式的结果以及权衡。有助于设计者理解一个模式是否可以用于某个特定的情形中。
- 该模式将必须展示的对象与它的不同展示形式分离开了。这一点在图中有所体现，其中展示了同一个数据集的两种不同的图形化呈现方式。



7.2设计模式

- 观察者（Observer）模式例子

模式名称：观察者

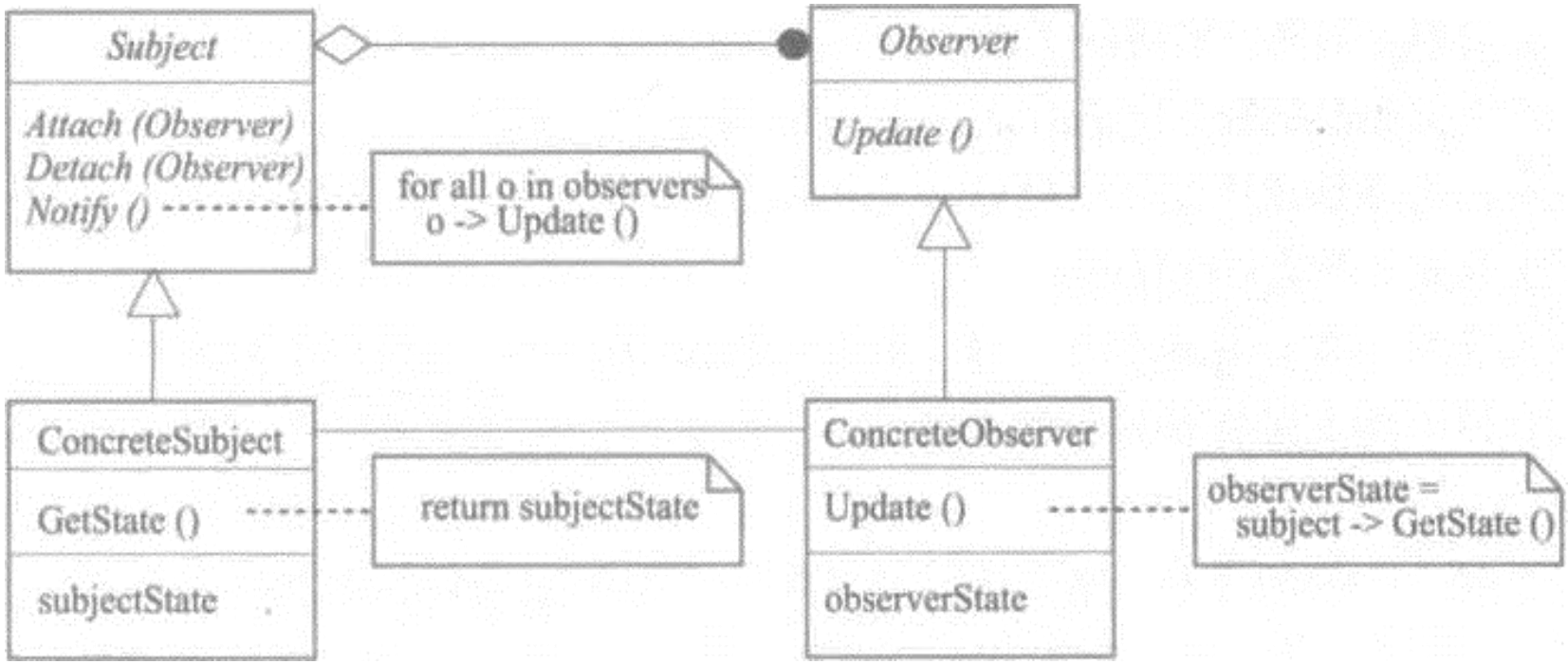
描述：将一个对象状态的呈现与对象本身分离开，允许为对象提供不同的呈现方式。当对象状态变化时，所有的呈现都会自动得到通知并进行更新以反映变化。

问题描述：在很多情况下你都必须为状态信息提供多种呈现方式，例如，一个图形化呈现方式和一个表格呈现方式。这些呈现方式在对信息进行规格说明时并不都是已知的。所有不同的呈现方式都应该支持交互，在状态发生变化时所有呈现都必须更新。这个模式的使用情形是，状态信息需要不止一种呈现方式，并且维护状态信息的对象不需要知道所使用的特定的呈现格式。

解决方案描述：解决方案包括两个抽象对象Subject(被观察者)和Observer(观察者)，以及两个继承相关抽象对象属性的具体对象ConcreteSubject(具体主题)和 ConcreteObserver(具体观察者)。抽象对象包括适用于所有情形的通用操作。要呈现的状态在ConcreteSubjeci中进行维护，该对象继承Subject的操作从而允许该对象增加和移除观察者(每个观察者对应一种呈现方式)以及在状态发生变化时发出通知。ConcreteObserver维护了ConcreteSubject状态的一份拷贝，并且实现了Observer的Update () 接口，该接口允许这些拷贝能够被同步更新。ConcreteObserver自动呈现状态并在任何时候当状态更新时自动反映变化。

7.2设计模式

效果：主题对象只知道抽象的观察者对象，而不知道具体类的细节。因此，这些对象之间的耦合被最小化了。由于缺少这些知识，提高呈现性能的优化可能无法实现：对主题对象的变化可能导致生成与之相关的针对观察者的一组更新，而其中一些可能并没有必要。



7.2设计模式

- 为了在设计中使用模式，我们要知道，任何设计问题都可能会有一个与之相关联、可以应用的模式。以下是关于这种问题的例子。
  - 1、向多个对象告知一些对象的状态发生了变化--观察者模式。
  - 2、整理面向一些相关的且经常是增量地开发出来的对象的接口--门面模式。
  - 3、为一个合集（collection）中的元素提供一种标准的访问方式，不用考虑合集是如何实现的--迭代器模式。
  - 4、允许在运行时扩展一个已有的类的功能--装饰者模式。
- 模式支持高层的概念复用。试图复用可执行的构件时，会受到构件实现中所做出的详细设计决策的约束和限制。使用模式意味着对思想进行复用，但可以对实现进行调整以适应正在开发的系统。
- 当开始设计一个系统时，很难提前知道是否会需要某个特定的模式。因此，在设计过程中使用模式经常要经过开发设计、体验问题、然后认识到可以使用某个模式的过程。
- 模式是伟大的思想，有效地使用还需要一些软件设计经验，必须能够认识到一个模式可以被应用的情形。没有经验的程序员，即使读过模式的书，也总是会发现很难决定是否复用某个模式或者是否需要开发一个特殊目的的解决方案。

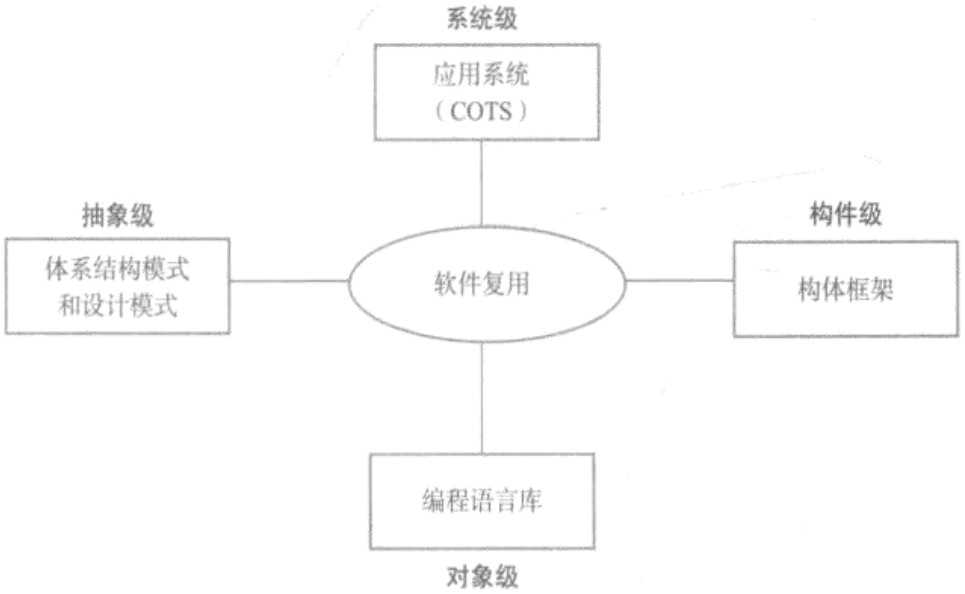
7.3实现问题

- 软件工程过程中的另一个关键阶段当然是系统实现，其中会创建软件的一个可执行版本。
- 实现可能会包括：使用高层或底层编程语言开发程序，或者对通用的成品系统进行裁剪和适 配以满足一个组织的特定需求。
- 本章的目的是提供一种与语言无关的方法，因此这里并不会关注与良好的编程实践相关的一些问 题。
- 这里我们介绍实现中的一些对软件工程尤其重要并且经常不会在编程相关的书中介绍的方面，包括如下这些。

– 1、复用。大多数现代的软件都是通过复用已有的构件或系统来构造的。在开发软件时，应该尽可能多地利用已有的代码。
– 2、配置管理。开发过程中，每个软件构件会创建不同的版本。配置管理系统保持对这些版本的追踪，避免系统中包含构件的错误版本。
– 3、宿主机-目标机开发。运行生产软件的计算机通常与软件开发环境中的计算机并不相同。软件一般在一台计算机上开发（宿主机）；另一台计算机上运行（目标机），宿主机和目标机有时候是同一种类型，但却又经常完全不同。

7.3.1 复用

- 60年代到90年代，多数软件都是从头开始开发的，一般都是通过使用某种高级编程语言编写所有代码。
- 唯一显著的软件复用是对编程语言库中的功能和对象的复用。然而，成本和进度压力使这一方法越来越不可行，特别是对于商业系统和基于互联网的系统。
- 基于复用已有软件的开发方法现在已经成为许多不同类型系统开发的基本准则，基于复用的方法现在已经在很多领域得到广泛使用，包括所有类型的基于Web的系统、科学软件，以及越来越多的嵌入式系统软件。



7.3.1 复用

- 软件复用可以在多个不同的级别上发生，如前页图所示。
  - 1、抽象级。并不是直接复用软件而是在软件的设计中使用成功的抽象知识。设计模式和体系结构模式都是面向复用的抽象知识的表示。
  - 2、对象级。直接复用来自库中的对象而不是自己写代码。实现这种类型复用，必须找到合适的库，并且确定其中的对象和方法是否提供了所需要的功能。例如，Java程序中处理电子邮件消息，那么可以使用JavaMail库中的对象和方法。
  - 3、构件级。构件是对象和对象类的合集，它们一起运行以提供相关的功能和服务。经常都要通过增加一些代码来对构件进行适配和扩展。例子是利用一个框架来构建自己的用户界面。
  - 4、系统级。复用整个应用系统。通常会包括对这些系统的某种方式的配置。这可以通过增加和修改代码来实现，或者通过使用系统自身的配置接口来实现。大多数商业化系统现在都是用这种方式构建的，其中会对通用的应用系统进行适配和复用。

7.3.1 复用

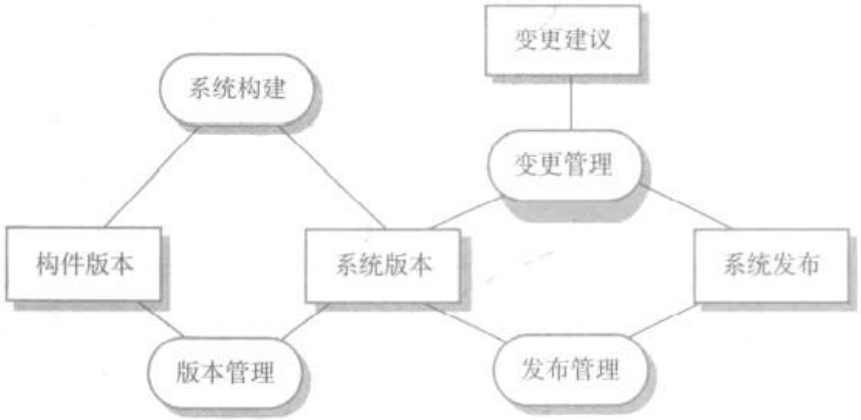
- 复用已有软件，可以更快地开发新系统，而且风险和成本更低。当然，会有一些与复用相关的成本：
  - 1、寻找可复用的软件以及评价其是否满足需要所花费的时间成本，可能还必须对软件进行测试以确保它可以在你的环境中工作。
  - 2、在适用的情况下，购买可复用软件的成本。对于大型的成品系统，这些成本可能会很高。
  - 3、适配和配置可复用软件构件或系统以反映开发系统需求的成本
  - 4、可复用软件元素相互之间集成以及与所开发的新代码相集成的成本。集成来自不同提供者的可复用软件可能会很难并且代价较高，因为这些提供者针对他们各自的软件将会如何被复用会做出一些相互冲突的假设。
- 如何复用已有的知识和软件应当是在开始一个软件开发项目时要考虑的第一件事情，应当在详细设计软件之前考虑复用的可能性。因为可能要对设计进行调整以适应对已有软件资产的复用。在开发过程中，要搜索可复用元素然后修改需求和设计以充分利用这些可复用元素。

7.3.2配置管理

- 在软件开发中，变化总是在不断发生的，因此变更管理绝对是很重要的，当多个人参与软件系统开发时，必须确保团队成员不会干扰其他人的工作。
- 如果两个人都在开发同一个构件，那么他们的修改必须进行协调，否则一个程序员所做的修改可能会覆盖其他人的工作。
- 还要确保每个人都可以访问到软件构件的最新版本，否则开发人员可能要重做已经做过的工作。当一个系统的新版本出现问题时，必须能够回到该系统或构件此前的一个可以工作的版本上。
- 配置管理是管理一个不断变化的软件系统的一般过程。配置管理的目的是支持系统集成过程以使所有的开发者都可以以一种受控的方式访问项目的代码和文档，找出代码和文档做了哪些修改，以及对构件进行编译和链接来创建一个系统。

7.3.2配置管理

- 如图所示，有以下4个基本的配置管理活动。
  - 1、版本管理，为保持对软件构件的不同版本的追踪提供支持。
  - 2、系统集成，为帮助开发人员定义用于创建每一个系统版本的构件的版本提供支持。然后，这一描述被用于通过编译和链接所需要的构件来自动构建一个系统。
  - 3、问题追踪，为用户报告bug和其他问题提供支持，为所有开发人员查看谁在处理这些 问题以及问题何时被修复提供支持。
  - 4、发布管理，会向客户发布一个软件系统的新版本。发布管理关注规划新发布的功能并 为了分发而对软件进行组织。
- 软件配置工具支持以上每一个活动。通常安装在集成开发环境(如Eclipse)中。版本管理使用版本管理系统来支持，如Subversion或者Git，支持多地点、多团队开发。

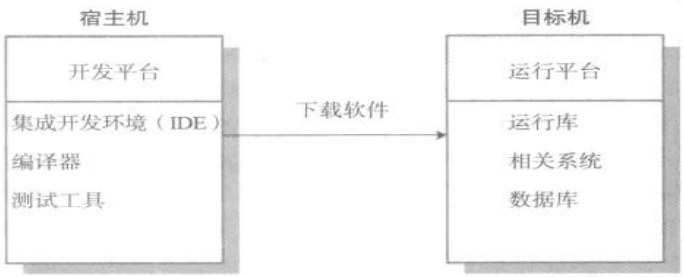


7.3.3宿主机-目标机开发

- 大多数专业化的软件开发都是基于宿主机-目标机模型的。
- 软件在一台计算机（宿主机）上开发，但是在另一台机器（目标机）上运行：更泛化一些的话，可以说一个开发平台（宿主机）和一个执行平台（目标机）。
- 一个平台不仅仅是硬件，还包括所安装的操作系统和其他支持性的软件（例如数据库管理系统）；或者对于开发平台而言，一个交互式的开发环境。有时候，开发平台和执行平台是一样的，这使软件的开发和测试可以在同一台机器上进行。
- 如果用Java开发，那么目标环境是Java虚拟机。原则上讲，这个目标环境在每台计算机上都一样，因此程序应该可以从一台机器迁移到另一台机器上。
- 对于嵌入式系统和移动系统，开发和执行平台是不一样的，需要将所开发的软件移动到执行平台上进行测试，或者在开发机上运行一个模拟器。

7.3.3宿主机-目标机开发

- 模拟器经常在开发嵌入式系统时使用，硬件设备(如传感器)以及系统将被部署的 环境中的事件进行模拟。



- 模拟器加快了嵌入式系统的开发过程，因为每个开发人员都可以有自己的执行平台，而不需要将软件下载到目标硬件上。然而，模拟器的开发很昂贵，因此可用的模拟器通常都是那些针对最流行的硬件体系结构的。
- 如果目标系统已经安装了需要使用的中间件或其他软件，那么要能够使用这些软件来测试系统。有时候在开发机上安装这些软件是不现实的，即使开发机和目标平台是一样的。
- 也可能因为许可证的限制而无法实现这一目的。那就需要将所开发的代码转移到执行平台上，从而对系统进行测试。软件开发平台应当提供一系列工具来支持软件工程过程。

7.3.3宿主机-目标机开发

- 工具可能会包括:
  - 一个允许开发人员创建、编辑和编译代码的集成编译器以及语法制导的编辑系统;
  - 一个语言调试系统;
  - 图形化编辑工具，例如编辑UML模型的工具;
  - 可以自动在程序的新版本上运行一组测试的测试工具，例如JUnit;
  - 支持重构和程序可视化的工具;
  - 管理源代码版本以及集成和构建系统的配置管理工具。

- 除了这些标准化的工具，开发系统还可以提供更加专业的工具，例如静态分析器。通常，面向团队的开发环境还会包括一个运行变更和配置管理系统的共享服务器，还可能会有一个支持需求管理的系统。

7.3.3宿主机-目标机开发

- 软件开发工具通常都会安装在一个集成开发环境中（IDE）。一个集成开发环境是一组位于一些公共框架和用户界面中的支持软件开发的不同方面的软件工具。
- 集成开发环境都支持使用特定的编程语言（例如 Java）的开发。针对特定语言的集成开发环境可以是特别开发的，也可以在通用集成开发环境基础上加入特定的语言支持工具。
- 一个通用的集成开发环境是一个为所开发的软件提供数据管理设施以及允许各种工具 一起工作的集成机制的框架，可以容纳各种软件工具。
- 例如：通用集成开发环境 Eclipse环境，它基于一个插件体系结构，从而使其可以面向 不同的语言（例如Java）和应用领域进行定制。
- 可以安装Eclipse，并且通过增加插 件来按照自己的特定目的对其进行裁剪。例如，可以增加一组插件以支持使用Java的网络 化系统开发，或者使用C语言进行嵌入式系统编程。

7.3.3宿主机-目标机开发

- 作为开发过程的一部分，需要针对所开发的软件将如何在目标平台上部署做出决策。做出决策时必须考虑的问题。
  - 1、一个构件的硬件和软件需求。如果一个构件是为一个特定的硬件体系结构而设计的，或者依赖于其他一些软件系统，那么很明显该构件要被部署到提供所需要的硬件和软件支持 的平台上。
  - 2、可用性需求。高可用性系统可能要求将构件部署到多个平台上，如果发生平台失效，那么该构件的其他实现是可用的。
  - 3、构件通信，如果有许多构件间通信，那么最好将这些构件部署在同一个平台上或者在物理上相互接近的多个平台上，降低通信延迟。
- 可以使用UML部署图来描述你对于硬件和软件部署的决策，其中可以显示软件构件如何分布在不同的硬件平台之上。
- 如果你正在开发一个嵌入式系统，你可能不得不考虑目标机的特性，例如，它的物理大小、电量、对于传感器事件的实时响应要求、效用器的物理特性以及它的实时操作系统。

7.4开源开发

- 开源开发是一种软件开发方法，其中软件系统的源代码被公开并邀请志愿者参加开发过程。
- 开源开发源于自由软件基金会（Free Software Foundation），主张源代码不应是私有的，而应该总是面向用户开放的，使他们可以按照自己的意愿检查并修改代码。其中有一个假设，代码将由一个小的核心小组而不是代码的用户进行控制和开发。
- 开源软件通过使用互联网来招募更大规模的志愿开发人员扩展了这一思想。他们中的许多人本身也是代码的用户。至少在原则上，一个开源项目的任何贡献者都可以报告和修复bug，并提出新的特征和功能建议。然而，在实践中，成功的开源系统仍然依赖于一个核心开发人员小组来控制软件的变更。
- 开源软件是互联网和软件工程的支柱。Linux操作系统、Apache web、Java、Eclipse集成开发环境、MySQL数据库管理系统、Android等都是被广泛使用的开源产品。IBM和0racle等计算机产业巨头，都支持开源运动并且将他们的软件建立在开源产品基础上。

7.4开源开发

- 获取开源软件通常很便宜甚至是免费的，通常可以免费下载开源软件。然而，如果想获得文档和相应的支持，那么可能必须要为此付费。使用开源产品的另一个关键的优势是广泛使用的开源系统非常稳定。这些系统有大量用户自愿去修复软件中的问题，而不是把这些问题报告给开发者并等待系统新的发布版本，bug的发现和修复通常要比私有软件要快。
- 对应参与软件开发的公司而言，有两个开源问题必须要考虑。
  - 1、正在开发的产品应该利用开源构件吗?
  - 2、应该使用开源的方法来开发自己的软件吗?
- 回答取决于所开发的软件的类型以及开发团队的背景和经验。
- 开发用于销售的软件，上市时间和降低成本是很重要的。
- 如果正在一个存在可用的高质量开源系统的领域中开发软件，那么可以通过使用这些系统来节省时间和费用。
- 如果针对一组特定组织的需求开发软件，使用开源构件可能是不可行的选项，可能必须要将软件和与可用开源系统不兼容的系统相集成。

7.4开源开发

- 许多软件产品公司现在都在使用开源方法来开发，特别是针对专业的系统。他们的业务模型并不依赖于销售软件产品，而是提供对于该产品的服务。他们认为将开源社区纳入进来会使软件开发更快同时更加便宜，并且将为该软件创造一个用户社区。
- 有的公司认为，采用开源方法会将机密业务知识透露给竞争对手，因此不愿意采用这一开发模型。如果在小公司工作并将软件开源，公司歇业了，那么你可以自己支持该软件。
- 发布系统源代码并不意味着来自更大范围内的社区人们将要帮助进行开发，多数成功开源产品都是平台产品而不是应用系统。
- 对专业应用系统感兴趣的开发者数量有限。让软件系统开源并不能够保证社区的参与度，Sourceforge和GitHub上有成千上万的开源项目都只有很少的下载次数。然而，如果你的软件用户对该软件未来是否可用存在疑虑，那么让软件开源就意味着用户可以获取自己的拷贝，并且放心他们不会失去对代码的访问权，

7.4.1 开源许可证

- 虽然开源软件开发的一个基本原则是，源代码应当是免费提供的，但是这并不意味着 任何人都可以随心所欲地处置这些代码。
- 从法律意义上讲，代码的开发者（一个公司或一个 人）拥有代码。他们可以通过在一个开源软件许可证中包含法律约束条件来对代码如何使用 加以限制（St. Laurent 2004 ）。
- 有一些开源开发者认为如果一个开源构件被用于开发一个新系统，那么该系统也应当是开源的。
- 其他人则愿意让别人在没有这些限制的情况下使用他们的代码。所开发的系统可以是私有的并作为闭源系统出售。

7.4.1 开源许可证

- 大多数开源许可证都是以下3个通用模型中某一个的变体。
  - 1、自由软件基金会（GNU）通用公共许可证（General Public License，GPL），这是一个所谓的互惠许可证。可以简单理解为，如果你使用GPL许可证下的开源软件，那么你必须让你软件开源。
  - 2、自由软件基金会（GNU）宽通用公共许可证（Lesser General Public License，LGPL），这 是GPL许可证的一个变体，按照该许可证你可以编写链接到开源代码的构件而不用发布这些构件的源代码。但是，如果你修改了受许可证保护的构件，那么你必须将其发布作为开源软件。
  - 3、伯克利标准分发（Berkley Standard Distribution， BSD）许可证：这是一个单向的许 可证，意味着你没有责任重新发布任何对于开源代码所做出的修改，你可以在所销售的私有系统中包含这些代码。如果你使用了开源构件，你必须承认这些代码最初的创造者。 MIT许可证是BSD许可证的一个变体，条款相似。

7.4.1 开源许可证

- 许可证问题很重要，因为如果在一个软件产品中使用开源软件，那么你可能会被强制 要求服从许可证条款来使你自己的产品开源，如果你试图销售自己的软件，你可能希望对其 保密。这意味着你可能会希望避免在开发中使用GPL许可证的开源软件。

- 如果你正在构造运行在开源平台上的软件，但是并没有复用开源构件，那么许可证不是个 问题。然而，如果你将开源软件嵌入到你的软件中，那么你需要建立相应的过程和数据库来保持对已经使用的开源代码及其许可证条件的追踪。

7.4.1 开源许可证

- 管理使用了开源代码的项目的公司应当做到以下几点，
  - 1、建立一个系统来维护关于所下载和使用的开源构件的信息。必须针对每一个构件保 存一个在使用构件时有效的许可证的副本。许可证可能会发生变化，要知道已经认可的那些许可证条款。
  - 2、了解不同类型的许可证，并理解一个构件在使用前是如何确定许可证的。你可能会决定在某个系统中使用一个构件而不在另一个系统中使用它，因为你计划以不同的方式使用这些系统。
  - 3、了解构件的演化路径。你需要知道一点关于开发构件的开源项目的事情，以理解这些构 件未来会如何变化。
  - 4、进行开源软件教育。仅仅建立规程来确保符合许可证条款是不够的，还需要对开发人员进行关于开源软件以及开源许可证的教育。
  - 5、建立审计系统。处于严格的交付期限压力下的开发人员可能会试图违反许可证条款。 如果可能的话，应当使用软件来侦测违反许可证条款的情况并进行阻止。
  - 6、参与开源社区。依赖于开源产品，应当参与相关社区并帮助支持他们开发。