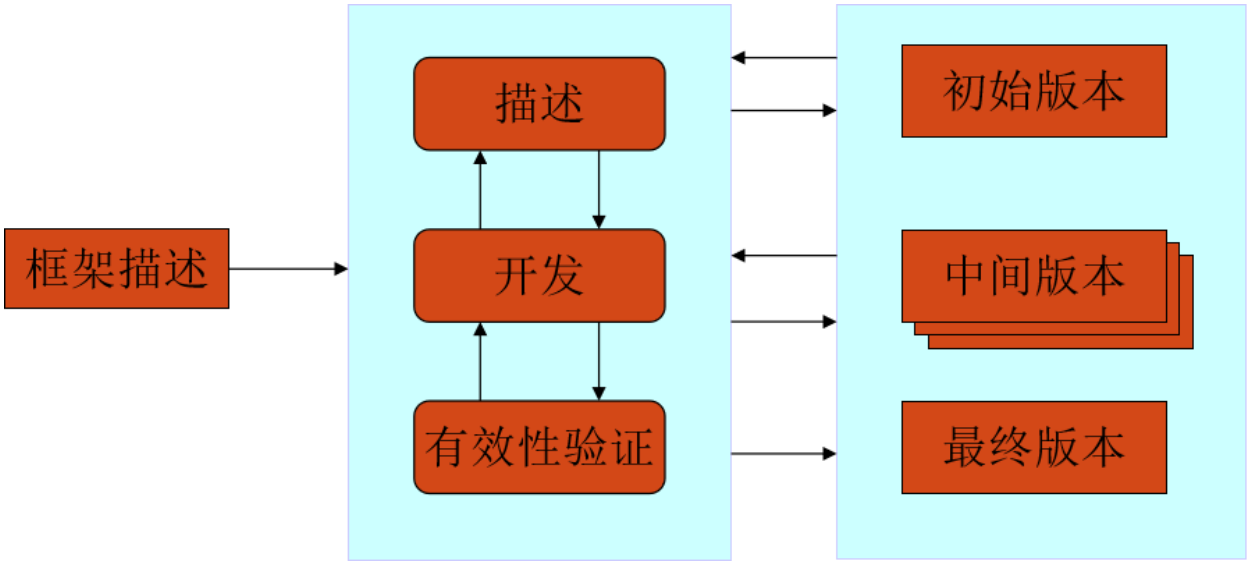
other1_软件工程

- **第一讲 概述**
  - **1.1 软件工程与软件危机**
    - 软件危机指的是软件的发展过程中出现的一系列严重的问题，如开发效率低下、成本高、可维护性差。
    - 软件工程被认为是能够解决软件开发严重问题的有效途径。
  - **1.2什么是软件**
    - 软件=计算机程序+相关文档
    - 专业化开发的软件包括：
      - 1.能够提供客户所需功能与性能的计算机程序；
      - 2.用于设置程序的配置文件、用于描述程序结构及开发过程的系统文档及解释如何使用系统的用户文档。
    - 软件产品可以为一个特定的用户设计开发，也可以为某一类通用的市场设计开发。
    - 软件开发可以分成：
      - 通用软件
      - 定制软件
    - 一个新的软件并不一定是全新开发，可以由现有软件或可复用软件成分配置形成。
  - **1.3什么是软件工程?**
    - 软件工程是涉及软件生产各个方面的一门工程学科。
    - 软件工程涉及软件生命周期的各个方面，从软件需求的确定到软件退役。
    - 软件工程还涉及软件开发中的人为因素，如团队组织；经济因素，如成本估算；法律因素，如版权保护。
    - 软件工程：
      - （1）将系统化的、规范的、可度量的方法应用于软件开发、运行和维护的过程、将工程化应用于软件；
      - （2）研究（1）中的方法。
  - **1.4什么是成功的项目**
    - 三要素：
      - 按时交付
      - 不超过预算
      - 满足用户要求
  - **1.5软件过程与软件生命周期的相关概念**
    - 软件过程是指开发或制作软件产品的一系列活动及成果。
    - 软件过程包括四个基本活动

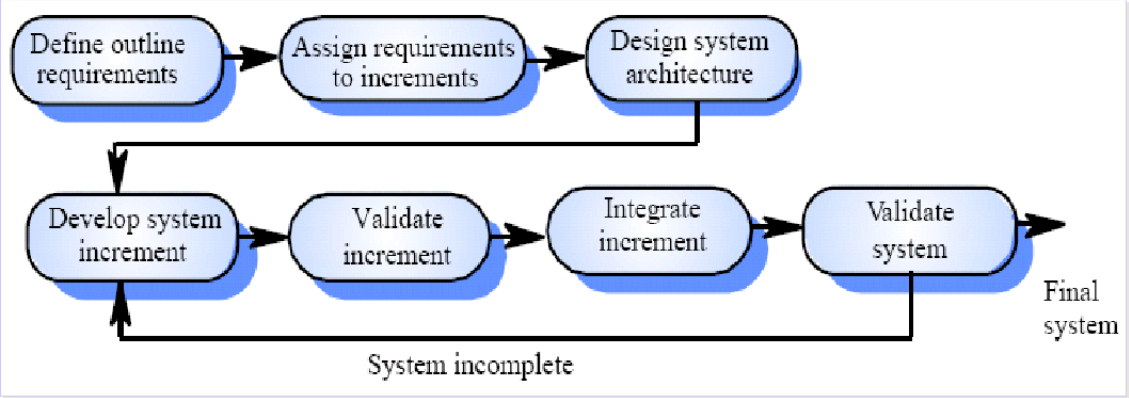
        - 1.描述
        - 2.开发
        - 3.有效性验证
        - 4.进化
    - 软件生命周期：软件过程的另一种形象描述，通常包括需求定义、分析与描述、软件设计、实现、测试、维护、退役等活动。
- **1.6优良软件的属性**
    - 可维护性
    - 可依赖性
    - 有效性
    - 可用性
- **1.7软件工程方法**
    - 结构方法
    - **面向对象方法**
- **1.8职业道德与责任**
    - 行为准则
        - 保密
        - 工作能力
        - 知识产权
        - 计算机滥用
- **第二讲 软件过程**
    - **2.1瀑布模型**
        - 各阶段：
            - 需求定义分析
            - 系统和软件设计
            - 实现和单元测试
            - 集成与系统测试
            - 运行和维护
            - 反馈表是在软件开发过程中要针对一些具体情况做一些适应性调整。
        - 缺点及适用情况：
            - 顺序严格，很难根据用户需求的变更做出及时调整。
            - 只适合需求非常清楚和需求变更被严格限制的情况
            - 实际的软件开发中，几乎没有多少业务系统是稳定的需求。
            - 瀑布模型反映了工程设计的基本思想。
    - **2.2进化式开发模型**

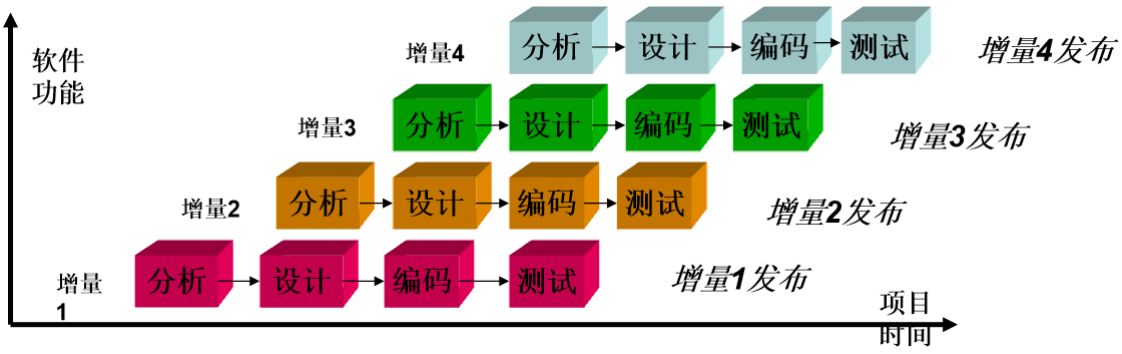
- 基本思想：通过开发系统原型和用户反复交互，已明确需求，使系统在不断调整与修改中得以进化成熟，又叫做原型式开发方法。
-



- 问题
  - 缺乏过程可见性
  - 系统结构通常会很差；
  - 需要一些特别的技术（如原型快速开发技术），通常与主流技术不兼容。
- 适用情况
  - 适合中小规模的交互系统
  - 可用于大型系统的局部开发（如系统界面），可以和瀑布模型混合使用；
  - 生命周期较短的系统。

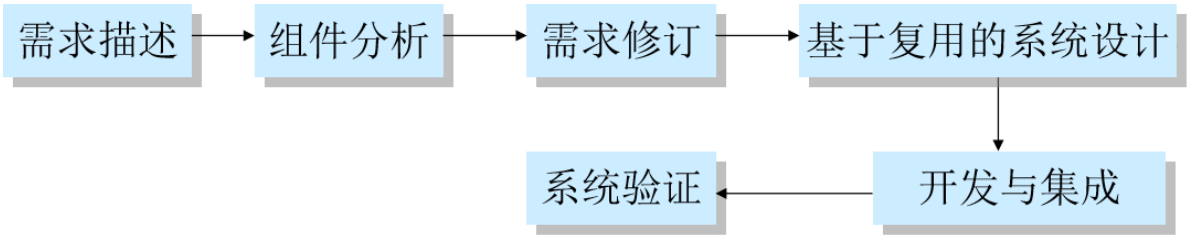
- **2.3基于过程反复的过程模型**
  - 对于大型项目而言，系统需求的变更是无法避免的，因此开发过程的反复是软件开发的必要手段；
  - 过程反复可以和任何一种一般过程模型结合使用。
  - 支持过程反复的两种模型：
    - 增量式
      - 不把系统作为一个整体交付，而是把系统分解为若干个增量，每个增量交付系统部分功能；
      - 用户需求按优先级进行排序，优先级最高的需求有最早交付的增量来完成；
      - 一旦增量进入开发阶段，其需求的变更便被“冻结”，而同时后续的增量的需求分析可以同步进行。
      -





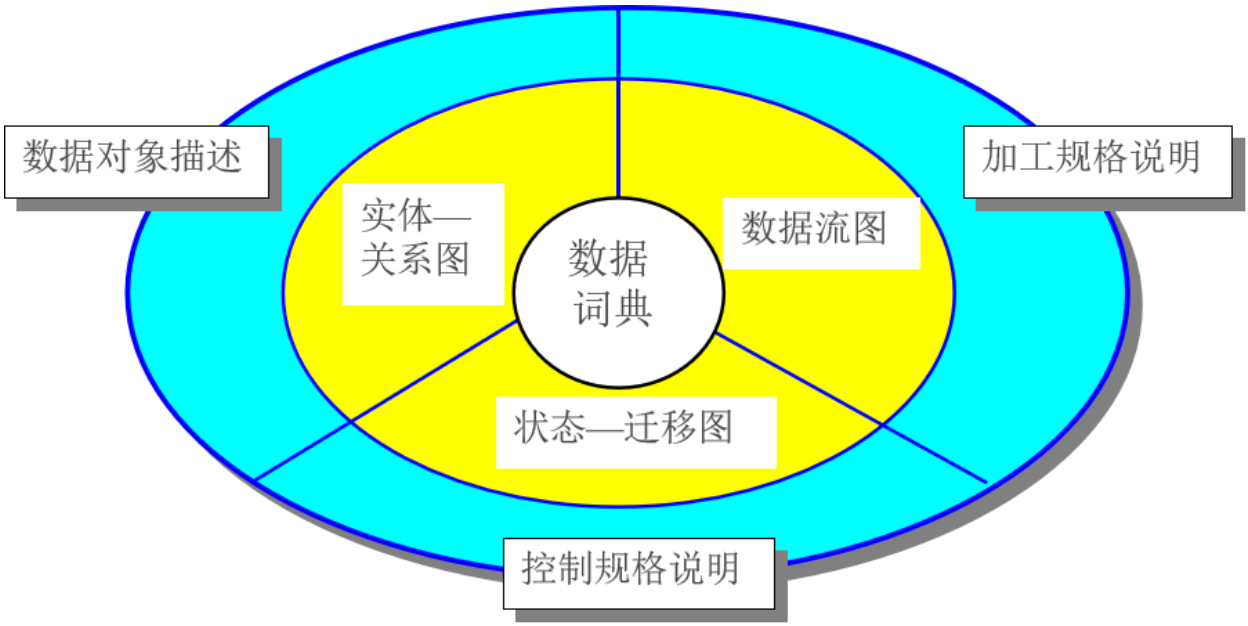
- 特点：
  - 由于每个增量可以交付部分系统功能，因此软件可以较早的交付用户使用（部分功能）；
  - 早期交付的增量可以作为后期增量的原型帮助后期需求的确定；
  - 项目总体失败率较低；
  - 优先级较高的系统功能得到最多的测试。
- 螺旋式
  - 用螺旋线

- **2.4基于构件的软件工程**
  - 软件复用是指在两次或多次不同的软件开发过程中重复是要那个相同或相似软件元素（通常称为可复用构件、组件或软件部件）的过程。
  - 软件构件是标准的、可互换的、经过装配可随时使用的软件模块。
  - 软件复用的意义：
    - 出发点是使软件系统的开发不再“一切从零开始”，能够充分利用已有的知识和经验。
    - 能够在软件开发中避免重复劳动，充分利用已有的开发成果，提高开发效率，降低开发成本。
    - 还可避免全新开发可能引入的错误，从而提高软件的开发质量。
    -



- **第三讲 需求工程**
  - **3.1需求分析基础**
    - 1）两个不同层次描述
      - 用户需求：从客户角度，采用自然语言配合以图标对目标系统应提供的服务以及系统操作要受到的约束进行的声明。
      - 系统需求：一种结构化文档，要运用一些专业的模型详细的描述系统的功能及其约束。系统需求文档有时也称为功能描述，应当是精确的，它可以成为双方之间合同的重要内容，同时作为开发工作的依据。
    - 2）功能需求和非功能需求
      - 功能需求：对系统提供的功能，系统在特定输入下做出的反应以及特定条件下的行为描述。某些情况下还要包括系统不应做什么。
      - 非功能需求：对系统提供服务或功能时受到的约束进行描述。如时间约束、开发过程约束和标准等。
      - 领域需求：来自系统的应用领域，反映领域特征。可能是功能需求也可能是非功能需求。
      - 非功能需求往往更为关键，表示的是系统的整体特征。而功能性需求描述的使局部功能。
      - 功能性需求描述应具备全面性和一致性。
      - 非功能需求的常见问题使检验困难，描述时应该设定明确目标并设定可测试的度量标准。
    - 3）使用自然语言描述需求的准则
      - 设计一个标准格式
      - 使用一致语言
      - 使用文本加亮
      - 避免使用计算机专用术语
    -
  - **3.2需求工程过程**
    - 可行性研究
      - 决定被提议的系统是否值得去做
      - 包括信息评估、信息汇总、书写报告
      - 信息评估与汇总需要分析人员与信息持有者沟通，提出问题汇总信息。
    - 需求导出与分析

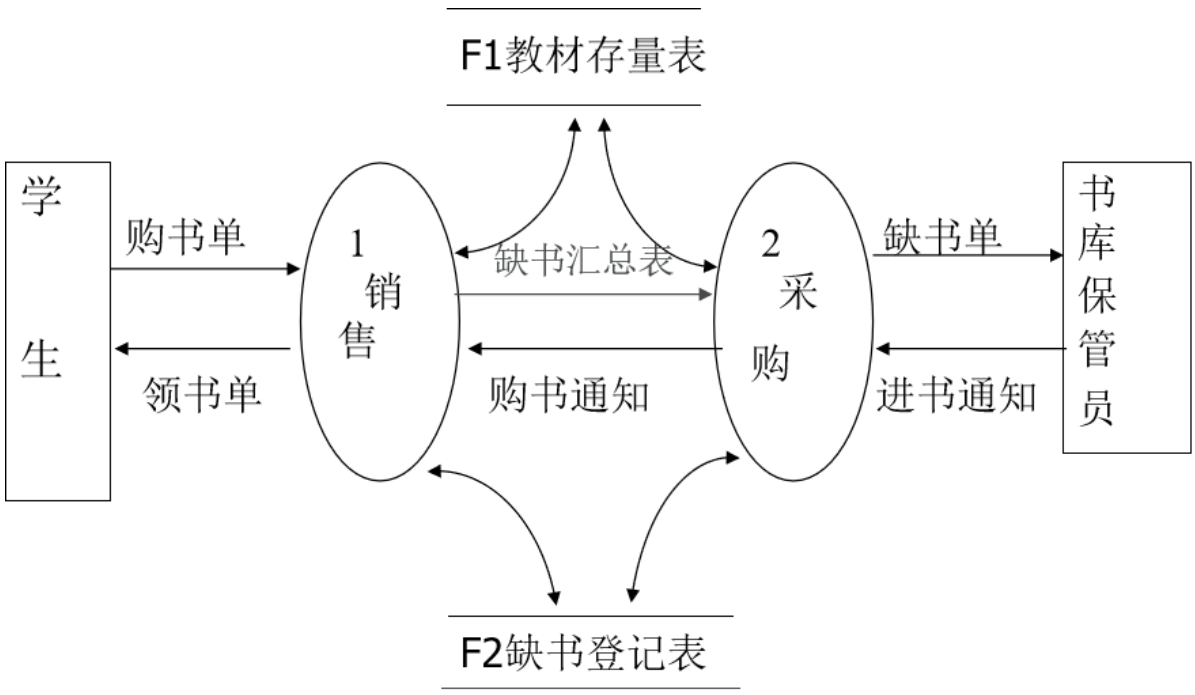
        - 在可行性研究之后进行，通常与需求描述交叉进行。
        - 包括：需求发现，需求分类与组织、优先排序和冲突解决、需求文档化。
        - 需求的发现与识别最为关键。
        - 信息来源包括已有文件，系统的信息持有者以及相近系统的规约描述。
        - 从多个视点进行分析
        - 视点用来描述不同角度的需求来源（信息持有者、其他相关系统及领域）。每一个视点代表系统需求的一个子集。
    - 需求描述（访谈）
        - 封闭式访谈：预先定义要提的问题，又叫限定式访谈。
        - 开放式访谈：不预先设定内同，给定一个范围，自由交流。
        - 访谈者应该是思维开阔，愿意聆听他人想法，并且不会对问题有先入为主的看法。
        - 善于用问题和建议去启发访问者的思维，或使用原型与被访问者讨论，而不仅仅用“what do you want”进行简单的提问。
    - 需求有效性验证
    - 需求管理
- **3.3结构化分析（SA）建模**
    - 面向数据流的系统建模技术，从数据加工的角度对系统进行规格描述。
    - 帮助分析者理解系统的功能，并采用模型与用户进行交流；
    - 不同的模型从不同的角度对系统进行描述。
    -



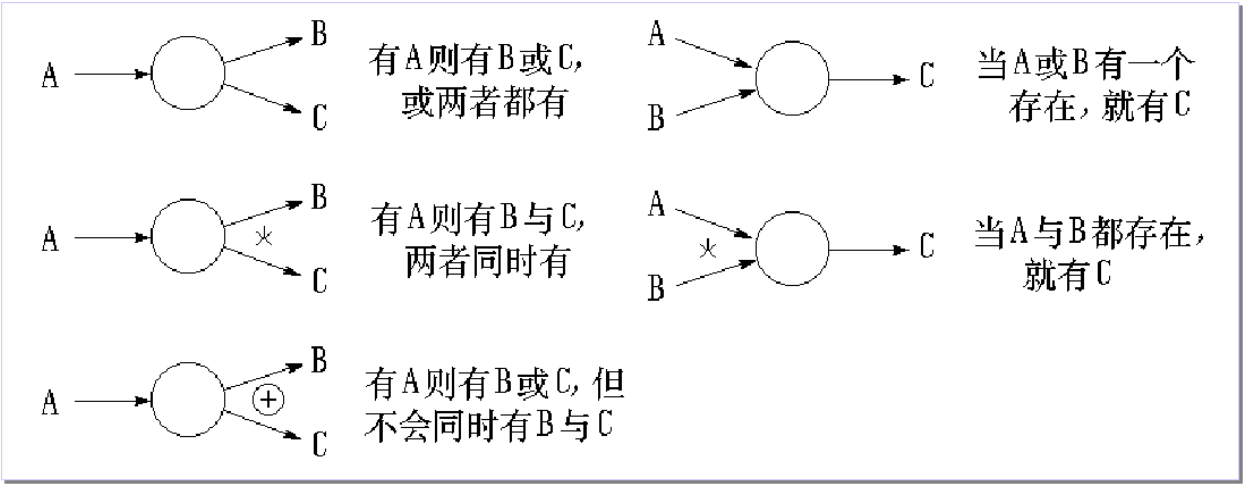
    - 核心：数据词典
        - 用于精确严格地定义每一个与系统相关的数据元素（包括数据流、数据存储和数据项），并以字典顺序将它们组织起来，使得用户和分析员有共同的理解。
        - 包含：

- 1.名称：数据对象或控制项，数据存储或外部实体名字
- 2.别名或编号
- 3.分类：数据对象，加工，数据流，数据文件，外部实体，控制项（事件/状态）
- 4.描述：描述内容或数据结构等
- 5.何处使用：使用该词条（数据或控制项）的加工
- 6.注释：数据量，峰值，限制，组织方式等

- 实体关系图（ER）：数据对象及数据对象之间的关系
- 数据流图（DFD）：数据在系统中如何被传送或变换，以及描述如何对数据流进行变换的功能（子功能）；

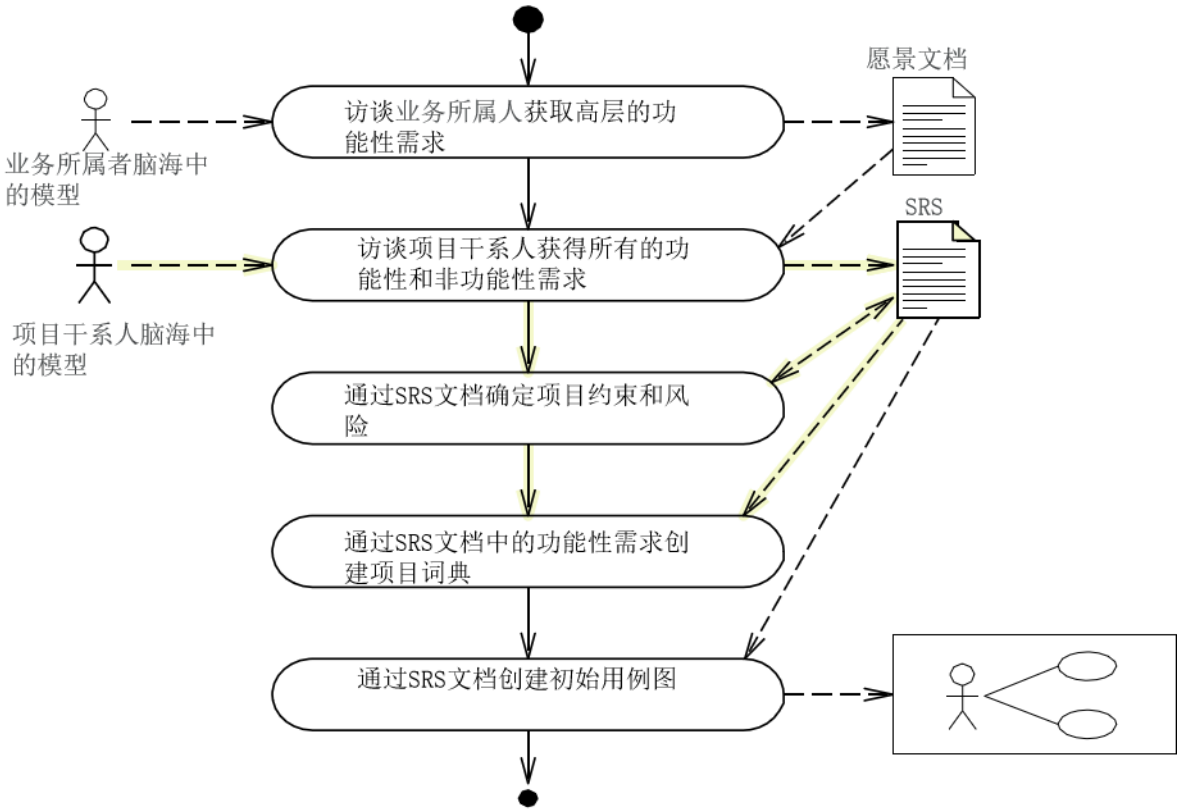


- 数据流图中的常用图元有以下四种：
-  表示外部实体，代表数据源和数据池（终点）。
-  表示加工，代表接收输入，经过变换，继而产生输出的处理过程。
- 表示数据流，代表数据的流向和路径。
-  表示数据存储，代表系统加工的数据所存储的地方。

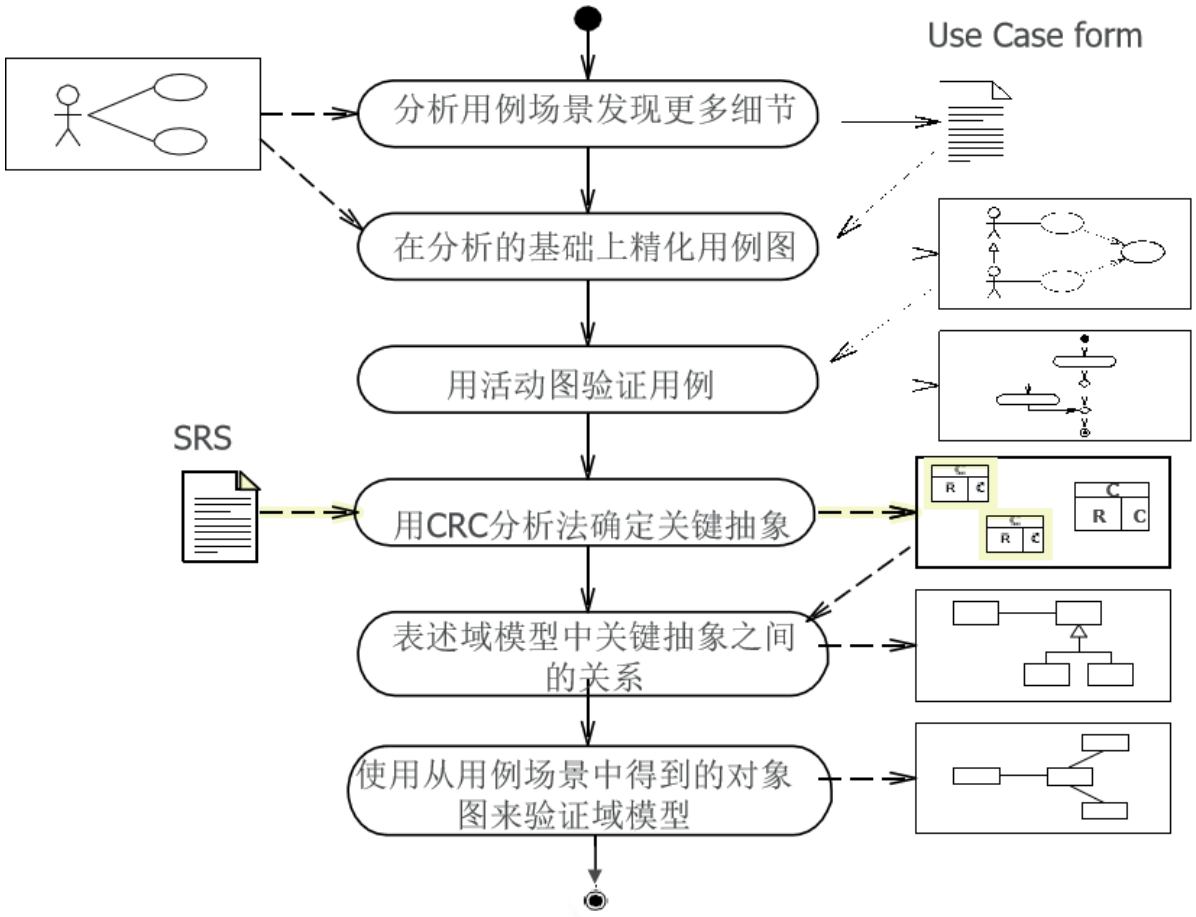


- 状态前意图（STD）：描述系统对外部事件如何响应，如何动作。
- ERD数据建模，DFD功能建模，STD行为建模

- **3.4 UML与面向对象分析方法**
  - 3.4.1理解UML
    - 一中标准的图形化建模语言，为不同领域的人提供一种统一的交流标准，这种标准使得系统构造者能够用标准等、易于理解的方式建立能表达出他们想象力的系统蓝图，并使客户、分析员、设计人员、程序员和系统其他涉及者能够互相理解和达成一致，从而能够有效地共享和交流设计结果。
  - 3.4.2需求导出工作流程
    -

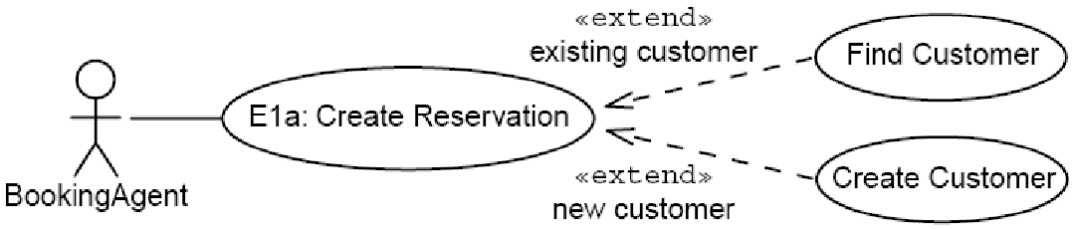


- 3.4.3需求精化工作流程
  -

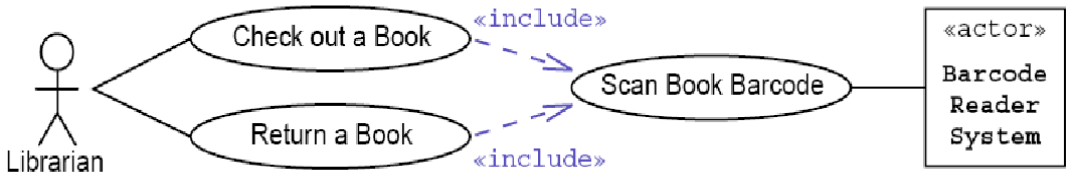


  - 用例图的画法
    - 表示了参与者与用例之间关系的图

- 一个用例场景应该：
  - 尽可能详细
  - 从不包含有条件的陈述
  - 以相同的方法开始但有不同的结果
  - 不指明过多用户界面方面的细节
  - 显示成功的也显示不成功的结果
- 用例：描述了所有高级行为和有哪些参与者参与了该行为。
- <<extend>>依赖使你找出那些不属于主事件流，而是可选场景中的系统行为。
  -

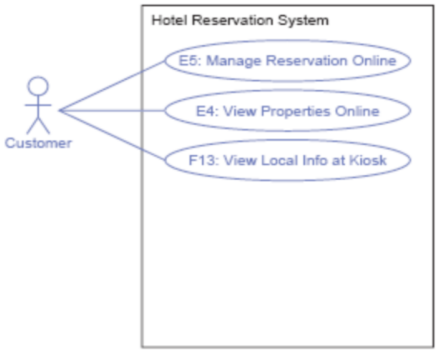


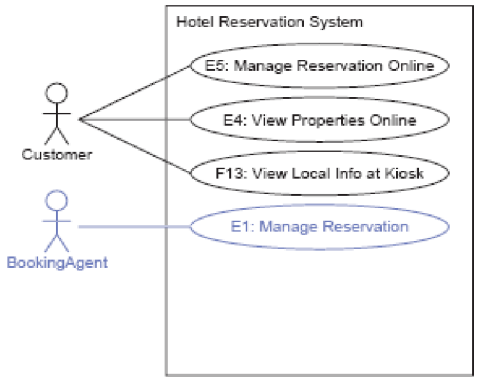
- <<include>>包含依赖可以使你能够把多个用例共同的行为识别出来。
  -

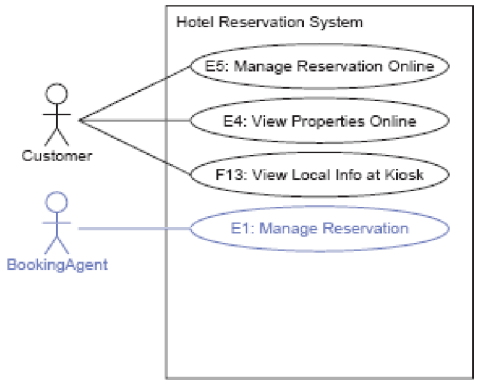


- 1.创建和命名系统的边界长方形
  -

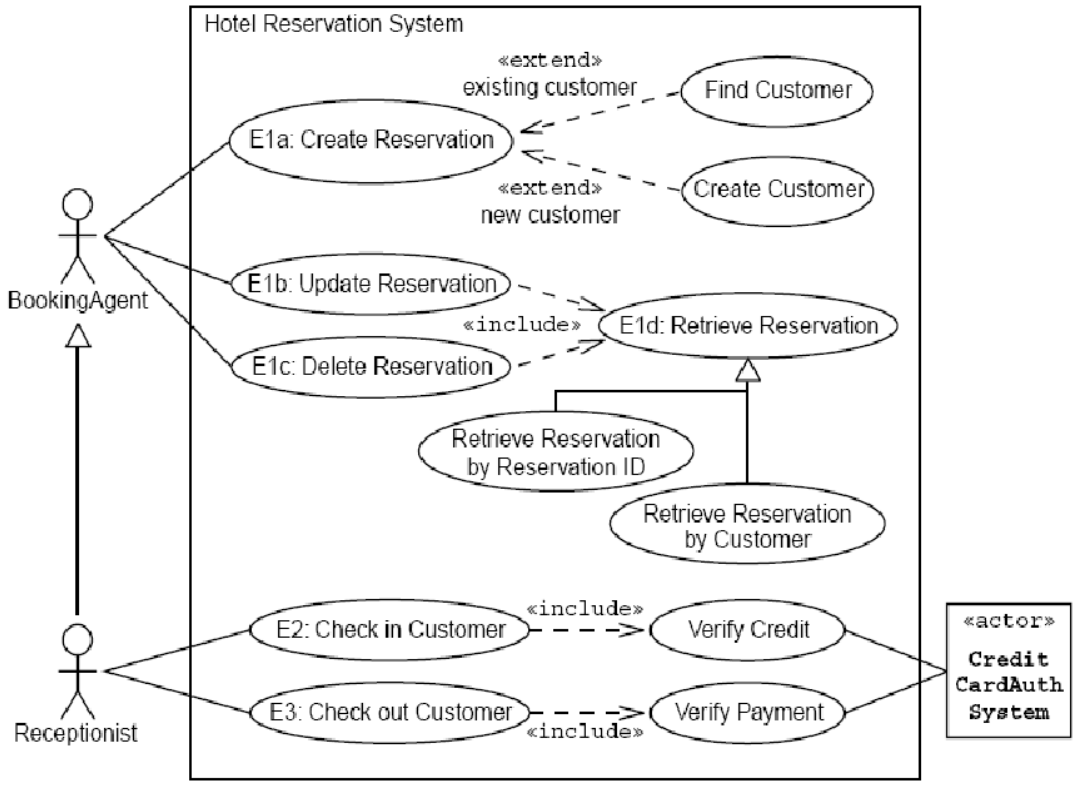
- 2.确定所有参与者
  -



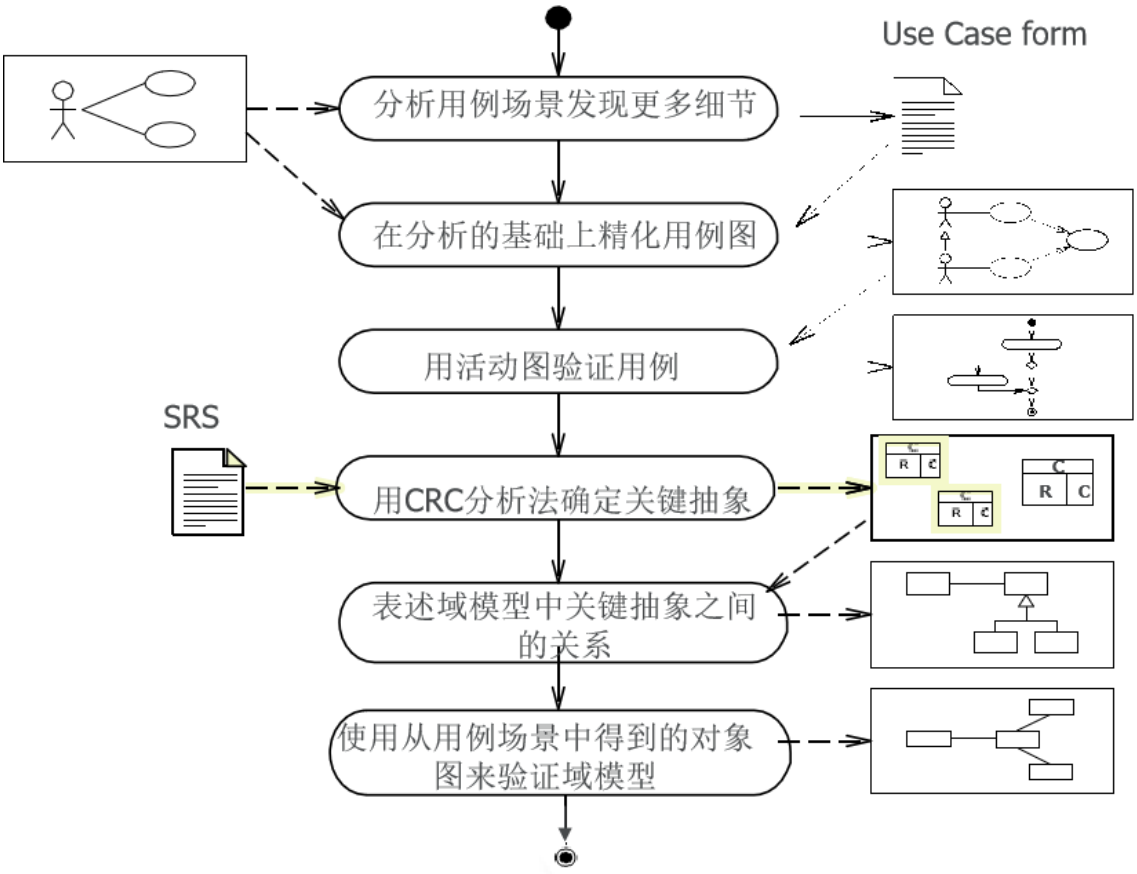
- 3.每个参与者添加图标
  - 

- 4.每个参与者参与的图中增加用例
  - 

- 5.画出参与者的用例关联
- 旅店预约组合实例
  -



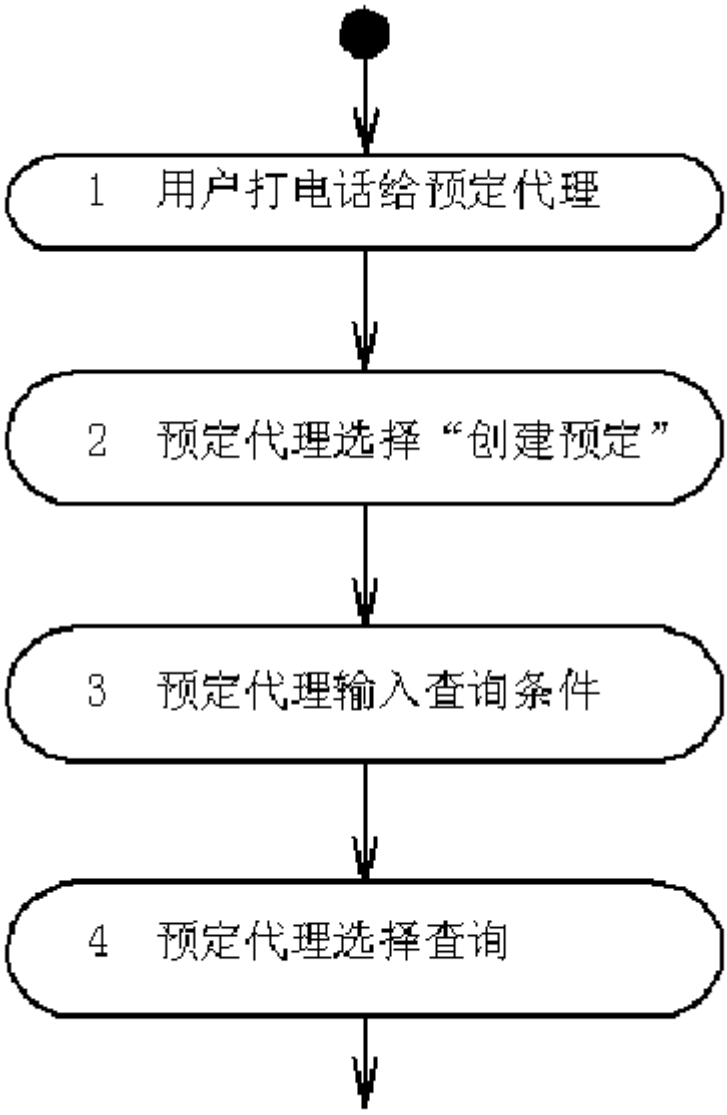
- 用例表：
  - 表格要素 描述



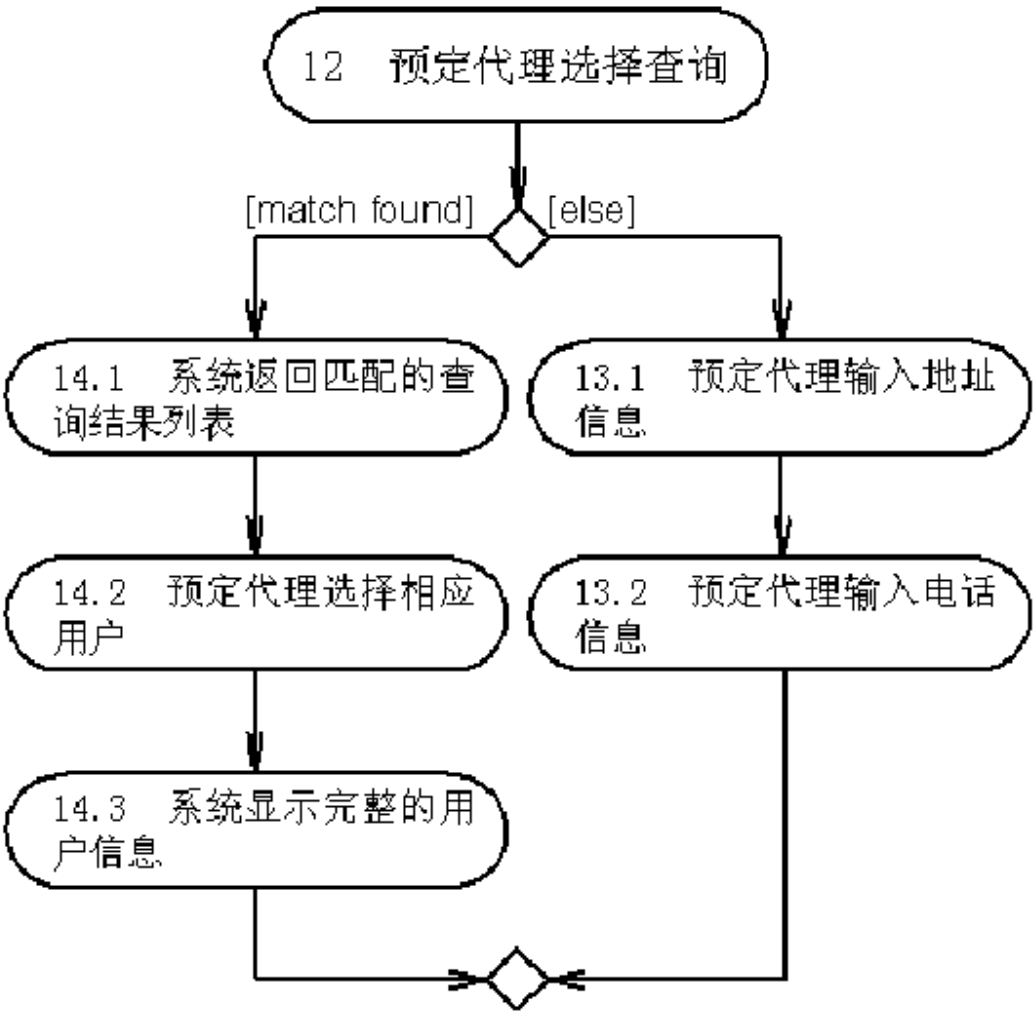
  - 用例标识和名称 需求规约中定义的用例编号和名称
  - 描述 一到两行关于用例目的的描述
  - 参与者 列出可以使用此用例的所有相关的参与者

- 优先级　需求规约中定义的用例的优先级别（Essential, High-value, or Follow-on ）
- 风险　用例的风险因素（按高、中、低分类）
- 前置条件和假设 用例被调用时的（系统）状态
- 触发条件 通知参与者用例应该被调用的条件
- 主事件流 组成此用例的用户动作的执行序列
- 可选事件流　任何可能发生的次要的操作和事件
- 后置条件 用例完成时系统应该所处的状态
- 非功能性需求　与用例相关的非功能性需求列表。可以是非功能性需求的概述，也可以是需求规约中的非功能性需求的编码列表

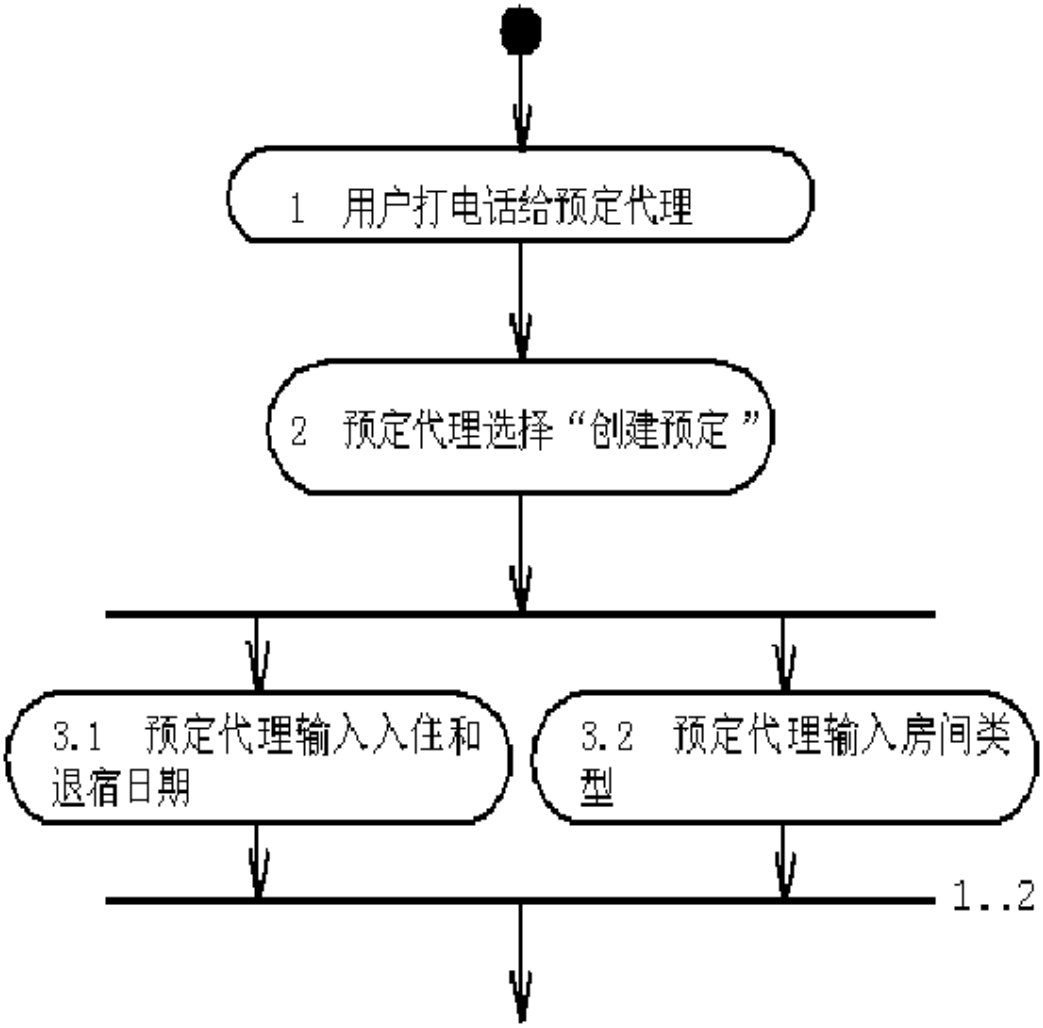
- 活动图：
  - 分析用例表中的主事件流：
    - 确定哪些活动
    - 确定分支和循环
    - 确定并发活动
    - 创建活动图
  -



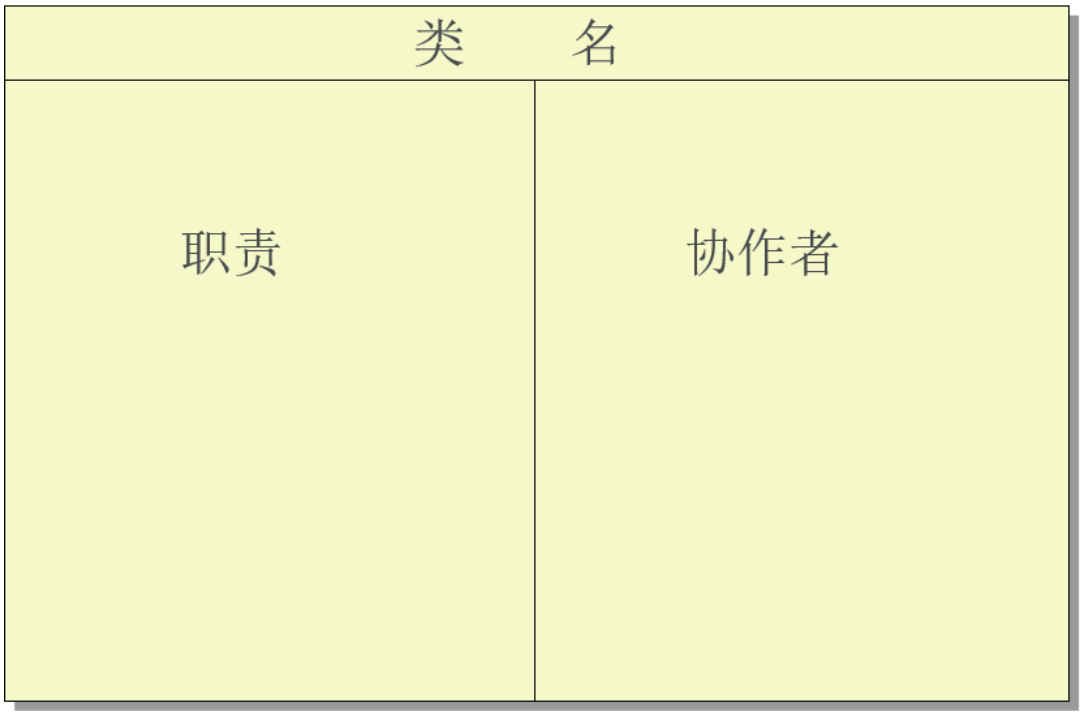
- 分支流

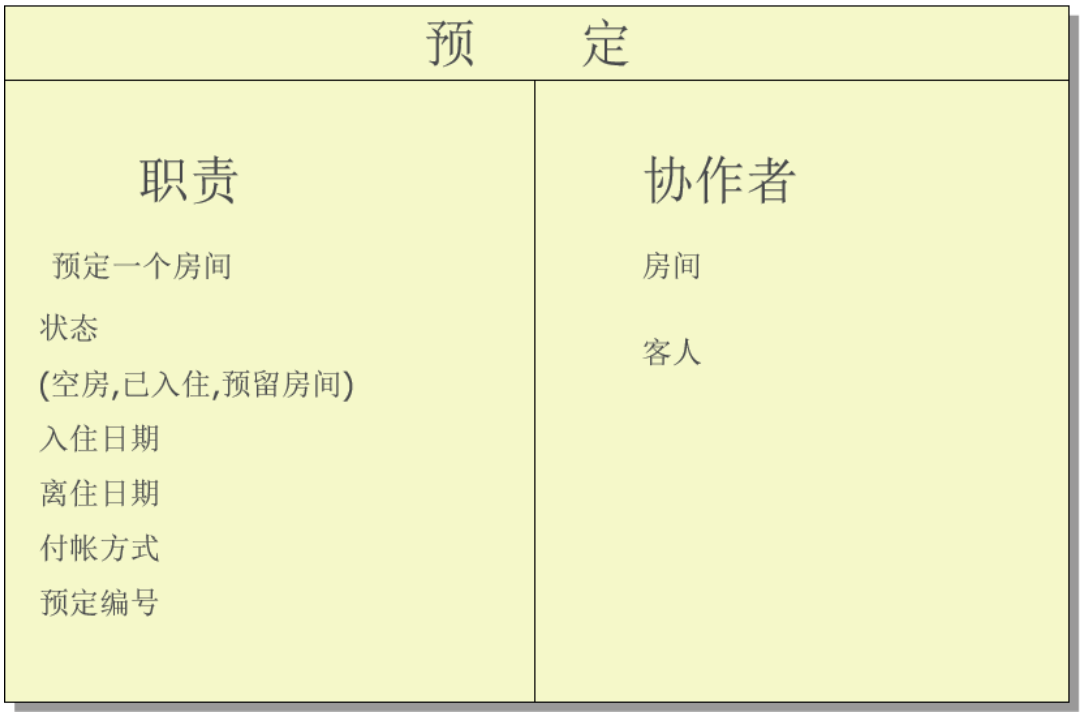

12　预定代理选择查询
[match found]
[else]
14.1　系统返回匹配的查询结果列表
13.1　预定代理输入地址信息
14.2　预定代理选择相应用户
13.2　预定代理输入电话信息
14.3　系统显示完整的用户信息

- 并发流

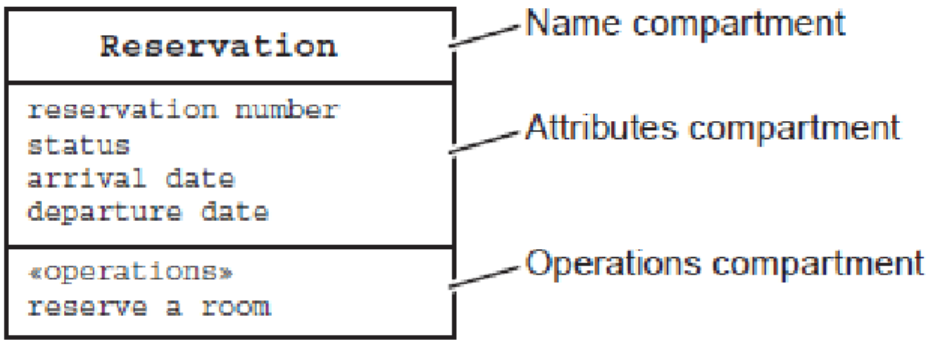


- CRC确定关键抽象的过程：
  - 关键抽象：有自己的职责同时被其他对象使用
  - 关键抽象的过程就是发现类的过程
  - 首先找出需求规格说明书的所有名词
  - CRC确定最基本的一组关键抽象
  - CRC卡记录关键抽象
    -



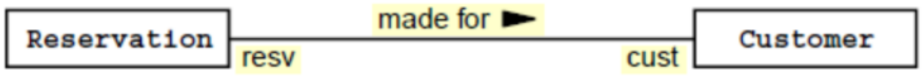


- 用类模型表示关键抽象：
  - 类节点组成：

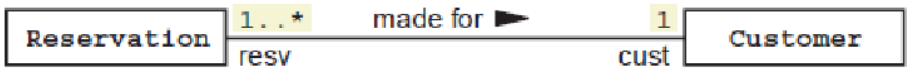


- 上图中，名字部分记录类名，类型部分记录类的属性（或实例变量），操作部分记录类的操作（或方法）。

- 关联：
  - 类之间的联系
  - 只有运行时才能够明确，但是这些模型表示对象在运行时所有可能的排列关系和角色。
    - “订单是为用户定制的”

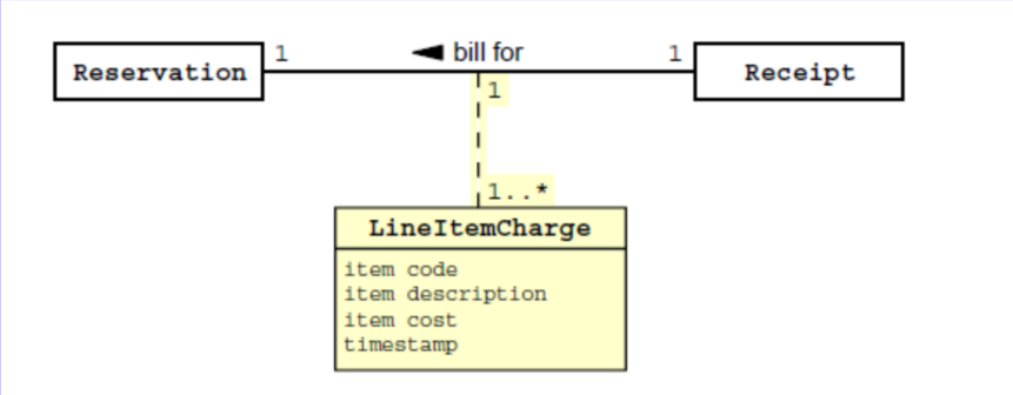


  - 多重性：决定多少个对象可能出现在关系中：
    - “订单只能为一个顾客制定，而一个顾客可以预定一个或者多个订单”



  - 方向性：关联上的箭头代表关联方向，即系统运行时，是否可以从这一方到另一方。
    - “系统可以一句订单找到有关的房间”

  - 关联类：有时会出现两个类之间的一些星系包含在两个类的关联中，这是需要把关联中的信息单独定义为一个类，该类称作关联类。
    - LineItemCharge是一个记录了订单与发票之间关联信息（消费明细）的关联类



- **3.5需求有效性验证**
  - 需求有效性验证的目的是检验需求描述是否正确地反映了客户的意愿
  - 好的需求对软件系统的开发效率及软件质量起着至关重要的作用。一个错误发现的越晚，修改它所付出的代价就越大。
  -

阶段	需求分析	软件设计	程序编码	单元测试	验收测试
相对修复代价	0.1 0.2	0.5	1	2	5

  - 需求检查：
    - 有效性
    - 一致性
    - 完备性
    - 现实性
    - 可检查性
- **第四讲 设计工程**
  - **4.1软件工程中的设计工程**
    - 设计是一个把问题转换成解决方案的创造性过程；
    - 设计解决的是“如何实现系统”；
    - 从工程管理的角度，软件设计可分为概要设计（总体设计、系统设计）与细节设计（详细设计）
  - **4.2设计概念**
    - 4.2.1模块化
      - 把软件划分为可独立命名和编址的构件，每个构件称为一个模块，每个模块完成一个子功能，当把所有的模块组装到一起成为一个整体时，便可完成指定的功能。
      - 模块组成系统或子系统。
      - 一个复杂的问题分割成若干个容易解决，容易管理的小问题后更易于求解，模块化正是以此为依据把系统划分成若干个模块，各个击破。
    - 4.2.2信息隐藏与独立性

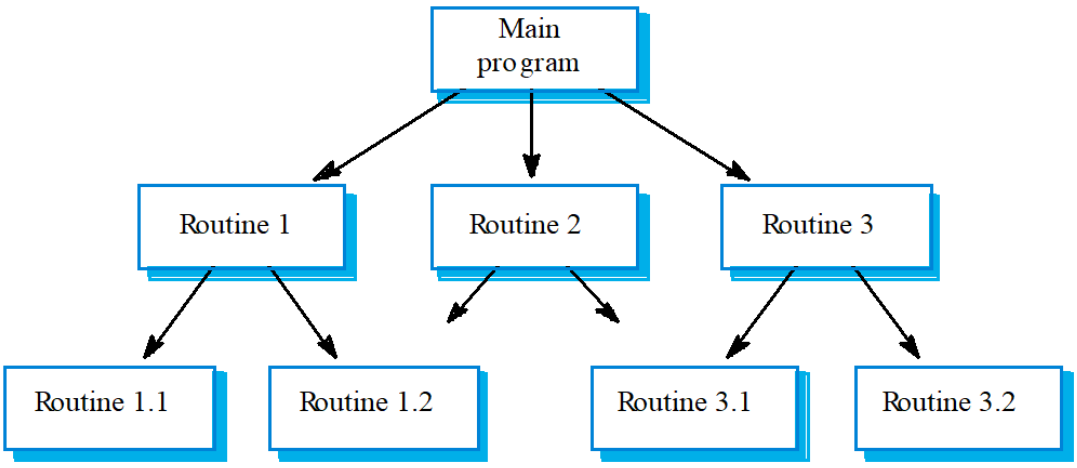
- 模块应该设计得使其所含信息（过程和数据）对于那些不需要这些信息的模块来说不可见；每个模块只完成一个相对独立的特定功能；模块之间仅交换那些为完成系统功能必须交换的信息，即模块应该功能独立的。
- 信息隐藏原理 的好处：
  - 支持模块的并发运行；
  - 减少测试和后期维护的工作量，因为测试和维护阶段不可避免的要修改设计和代码，模块对大多数数据和过程处理细节的隐藏可以减少错误向外传播。
  - 整个系统扩充功能只需要插入新模块，原有模块无需改动。
- 模块独立性：模块化、抽象和信息隐藏概念的直接产物，模块独立性是通过开发具有单一功能和其他模块没有过多交互作用的模块来达到的。
- 独立性好的模块对于其他模块依赖性小，修改时对其他模块影响小，易于修改和扩充，因此有良好的可维护性。
- 模块独立性可用两个定性准则来度量：耦合性和内聚性。
  - 耦合：模块之间相对独立性的度量
  - 内聚：模块功能相对强度的度量。
  - 模块的内聚性越强，耦合性越弱，独立性越强。

- 4.2.3抽象
  - 认识复杂现象的过程中使用的最强有力的思维工具。
  - 抽出事物本质特征而暂不考虑它们的细节。
  - 一个复杂的动态系统首先可以用一些高级的抽象概念构造和理解，这些高级概念可用一些低级的概念构造和理解，如此近进行下去，直至最底层的具体元素。
- 4.2.4逐步求精（细化）
  - 早期的自顶向下设计策略。
  - 主要思想：针对某个功能的宏观描述，用逐步求精的方法不断地分解，逐步确立过程细节，直至该功能用程序语言描述的算法实现为止。
  - 求精的每一步都是用更为详细的描述替代上一层次的抽象描述，所以在整个设计过程中产生的、具有不同详细程度的各种描述，组成了系统的层次结构。层次结构的上一层是下一层 的抽象，下一层是对上一层的求精。

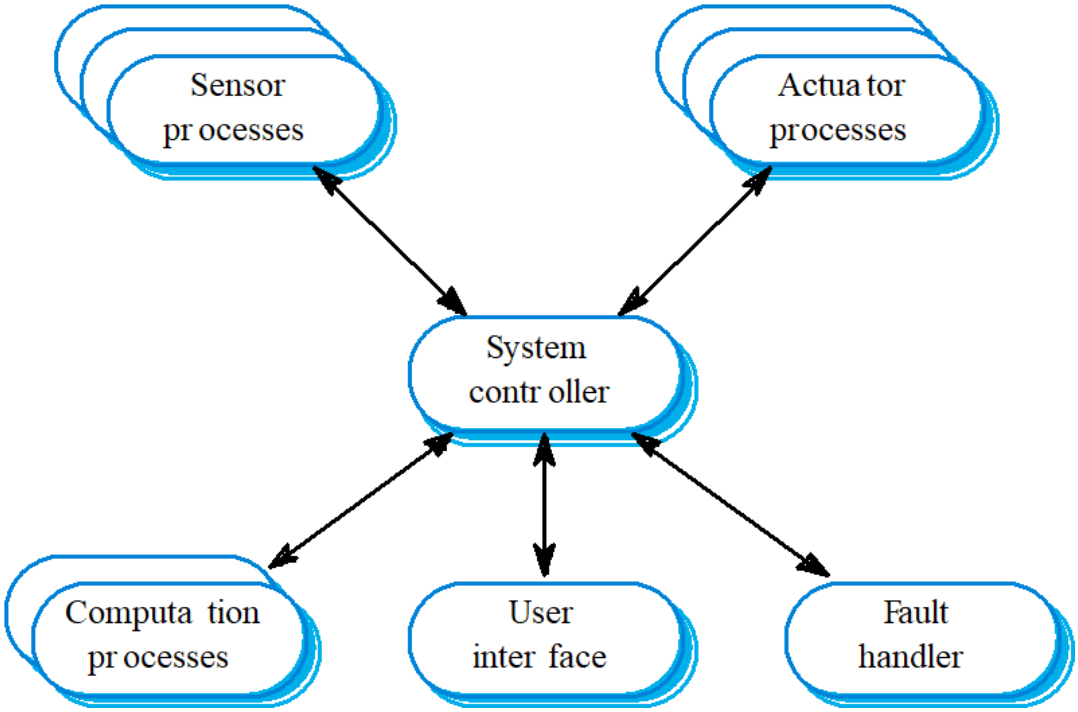
- **4.3体系结构设计**
  - 体系结构设计的任务是要识别出组成系统的子系统并建立子系统的控制和通信框架。
  - 体系结构设计是联系需求描述与其他设计活动的桥梁。
  - 系统组成：反应系统组织所采用的基本策略。
  - 三种广泛的组成类型：
    - 数据中心体系结构（容器模型）：
      - 所有数据共享到一个中心数据库，所有子系统都能从中存取数据。
      - 适合存在大量共享数据的系统。
    - 客户/服务器体系结构：

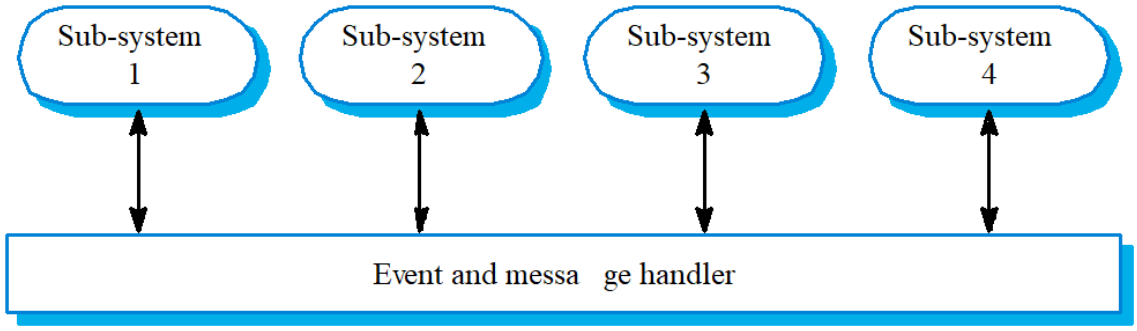
- 分布式系统模型，数据和加工过程在多个处理器之间分配器‘
- 主要组成：
  - 一组为其他子系统提供服务的单机服务器；
  - 一组向服务器请求服务的客户机；
  - 连接客户机与服务器的网络。
- 分层（抽象机）体系结构：
  - 把系统组织一系列的层次（抽象机），每一层提供一组服务；
  - 这种模型支持增量式开发，不同层次的服务可以单独交付
  - 层与层之间以接口联系，一个接口改变，毗邻的层会受影响。
  -

- **4.4体系结构视图**
  - 软件体系结构是对系统构成与部分间联系的多角度描述，每种模型只能描述系统的一个侧面（视角）；
  - 一个体系结构视图是对于从某一视角或某一点上看到的系统所作的简化描述，描述中涵盖了系统某一特定方面，而省略了于此方面无关的实体
- **4.5控制模式**
  - 集中式控制：这种控制方式中有一个子系统被指定为系统控制器来负责管理其他子系统的执行。
    - 按子系统顺序执行还是并行执行分为两类：
      - 调用-返回模型：
        -

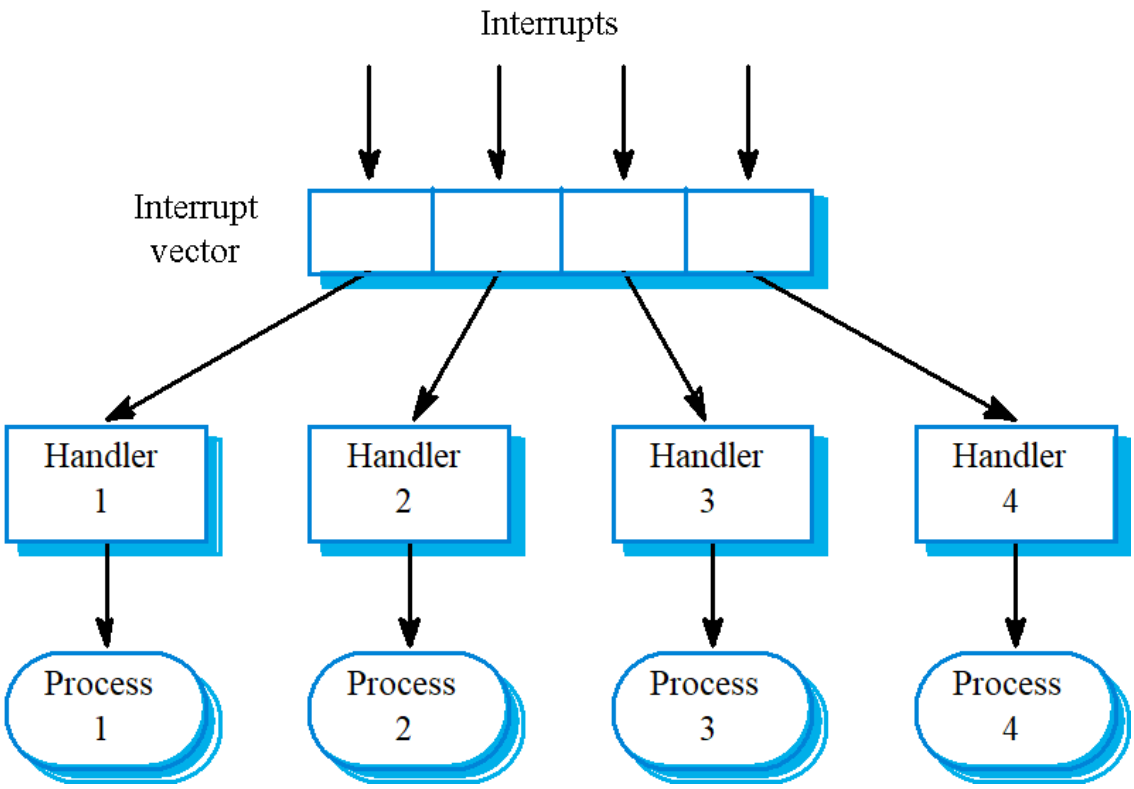


      - 管理者模型：
        -



- 基于事件的控制：事件驱动的控制模型通过外部产生的事件来驱动系统。
- 主要有两种：
  - 广播模型：
    -
    

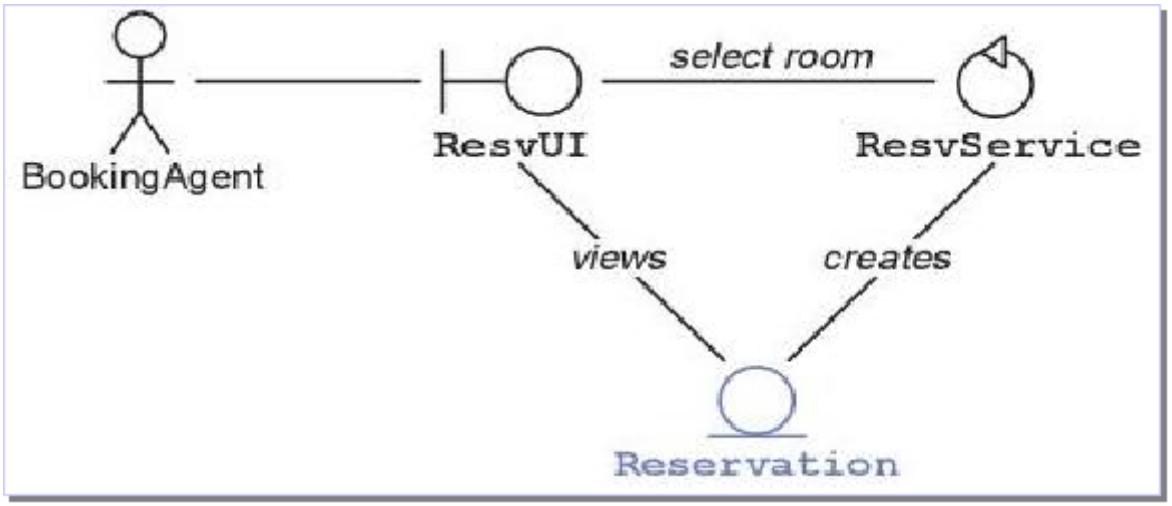
    - 对于基于网络的分布式系统很有效
    - 这种模式中子系统注册其感兴趣的特别事件，当这些事件发生时，控制就被转移到能处理这些事件的子系统。
    - 这种模式的控制策略不在事件和消息处理器的内部。由子系统决定需要哪些使劲按，消息处理器只负责将事件发给它们。
    - 易于新系统的集成，缺点是系统不能知道是否和什么时候事件将会被处理。
  - 中断驱动模型：
    -



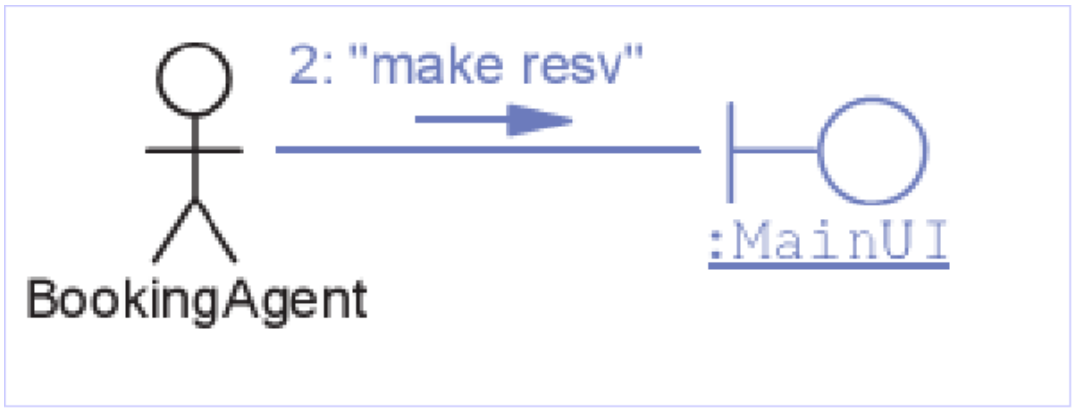
- 需要对外部事件迅速做出处理，适合与实时系统。
- 每种终端类型对应一种要处理的使劲按，分别与一个内存地址相连，该内存地址存放的时与中断类型相对应的事件处理器地址
- 有点事能够对事件做出非常迅速的反应，缺点在于它的编程较为复杂且不易验证有效性。

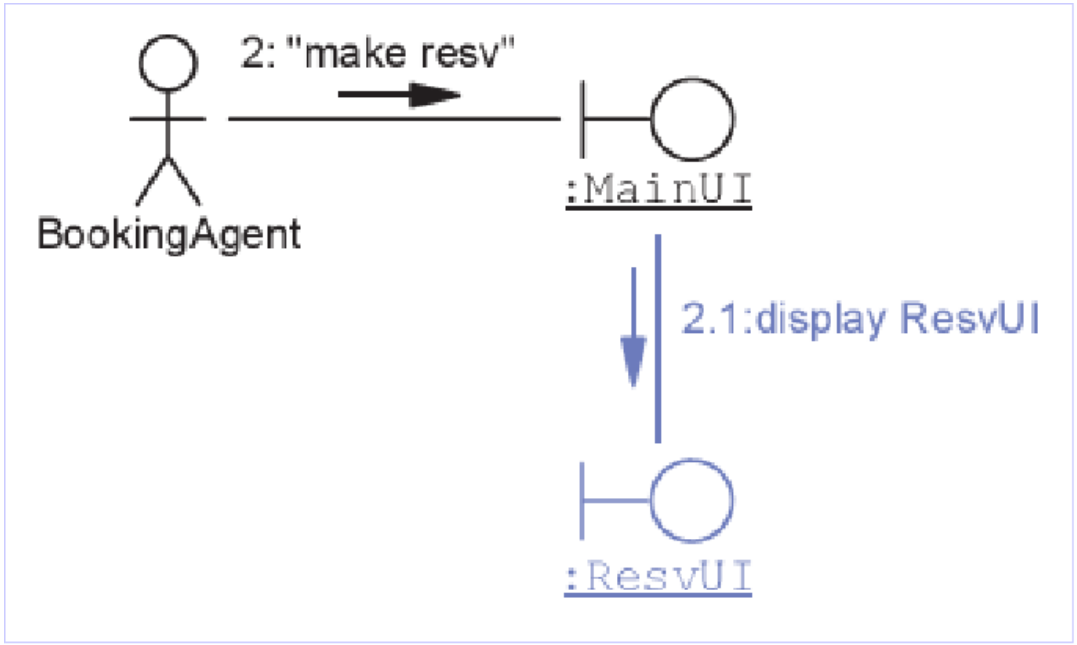
- **4.6从分析到设计的转换——鲁棒性分析：**
  - 鲁棒性分析是这样一个过程，用鲁棒图引导我们从用例转换为支持用例实现的职责模型：
  - 输入：
    - 一个用例
    - 用例场景
    - 活动图
    - 域模型
  - 输出：
    - uml序列图和一些组件设计：边界、服务、实体组件，得出用鲁棒图表示的设计模型。
  - 边界组件：
    - 用于针对系统和参与者之间的交互建模。
    - 抽取用户界面】传感器、通信接口等
    - 高层用户接口组件
    - 每一个边界组件必须至善与一个参与者关联起来
    -

- 服务组件：
  - 

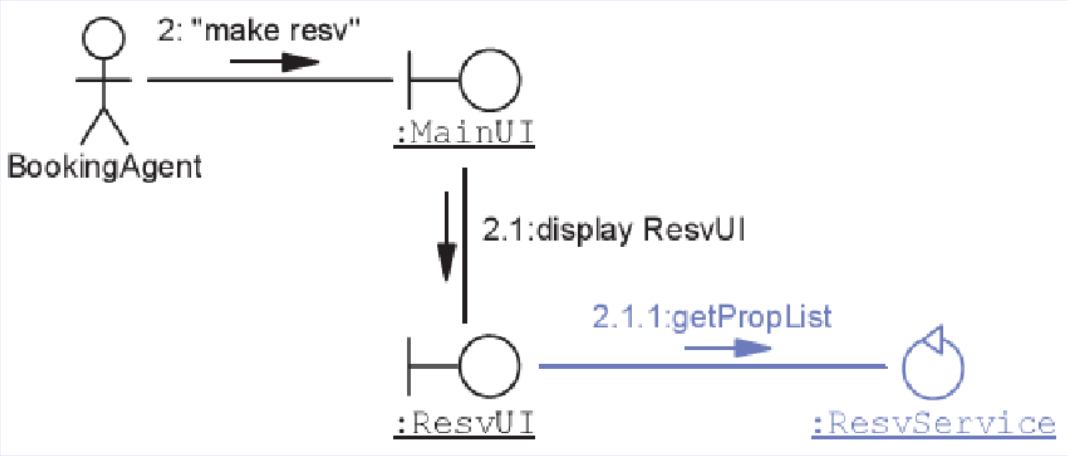
  - 协调、序列化、事务以及控制另外的对象的角色，而且他们经常被用来封装与某个特定用例的控制
  - 调整控制流
  - 在工作流种隔离所有从边界组件到实体组件的更改。
- 实体组件：
  - 

  - 模拟长期存在并具有持久性的信息。
  - 通常与域对象通信
  - 大多数实体是具有持久性的。
  - 尸体可以有非常复制的行为
- 分析过程：
  - 1.选择适当用例
  - 2.把参与者放到协作图里
    -

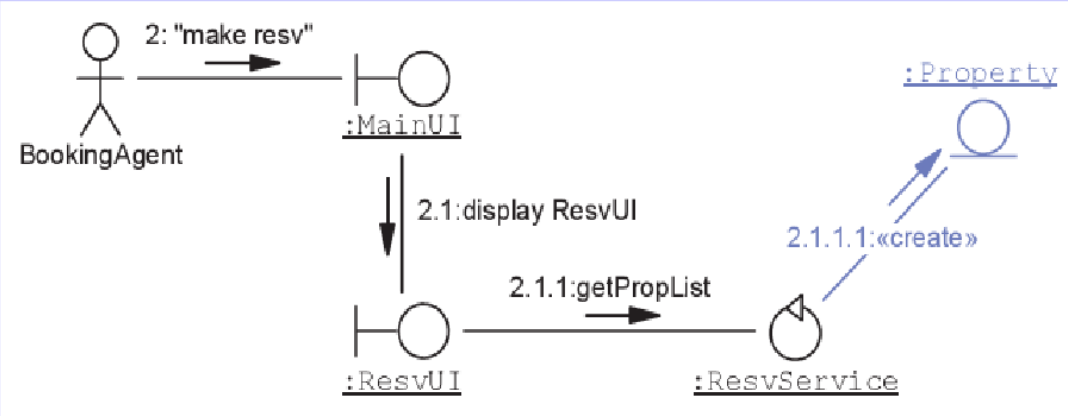
- 3.分析用例
  - 对每一个动作：
  - 确定并增加边界组件：
    -
    

    -
    

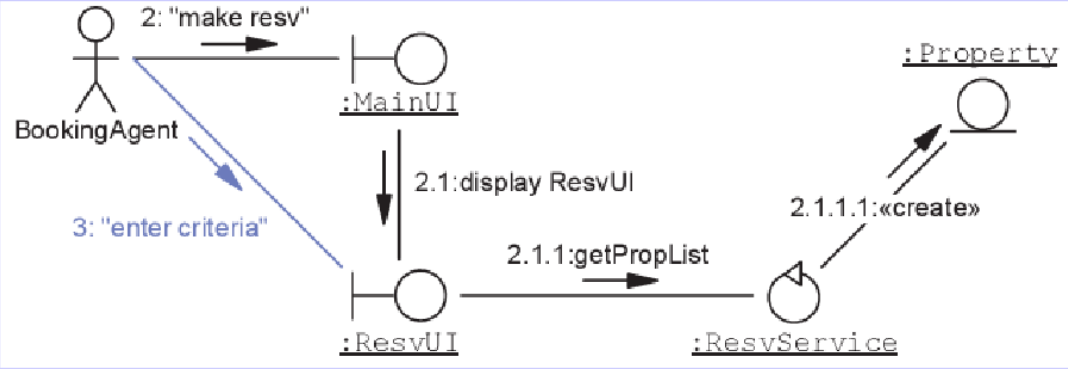
  - 确定并增加服务组件：
    -



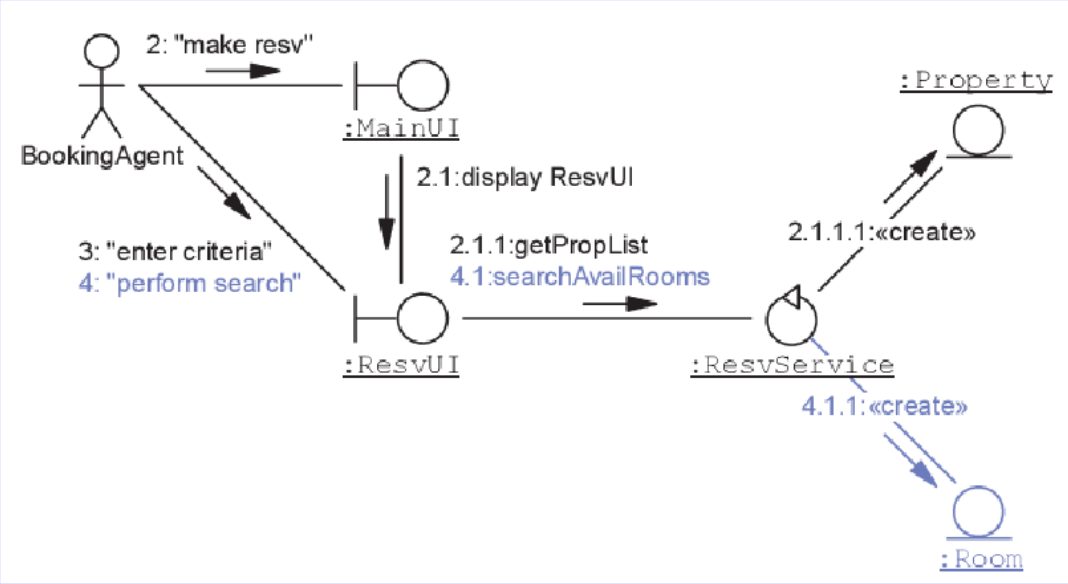
- 确定并增加实体组件：
  -



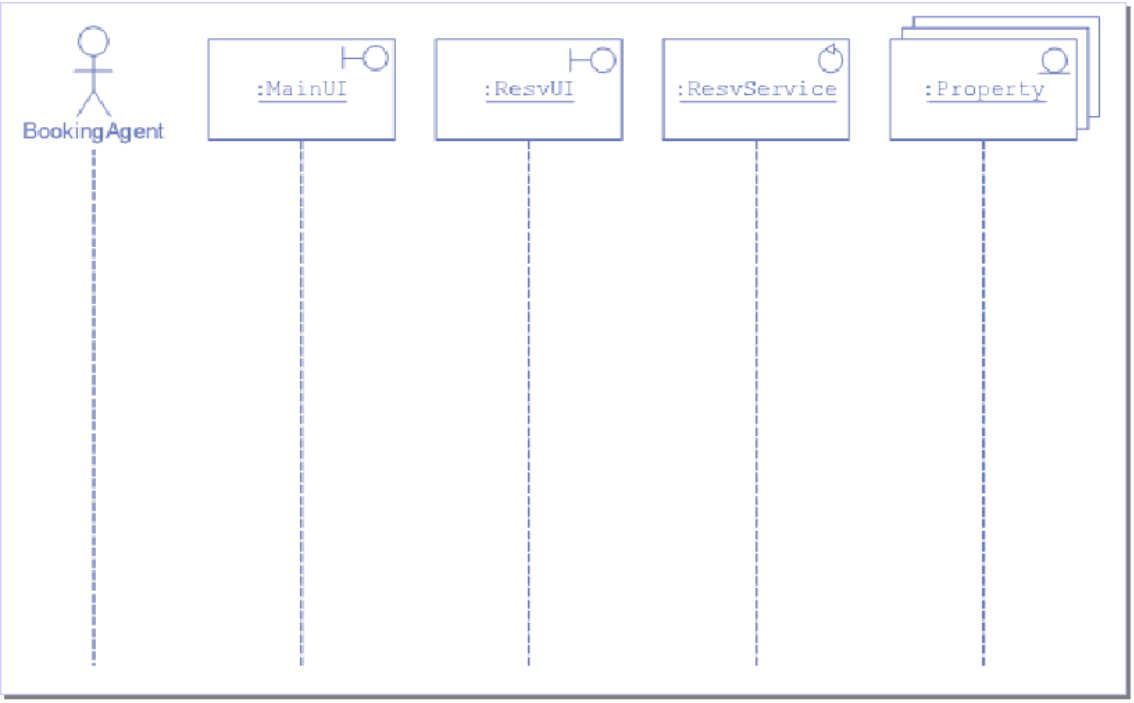
- 画出组件的关联
  -



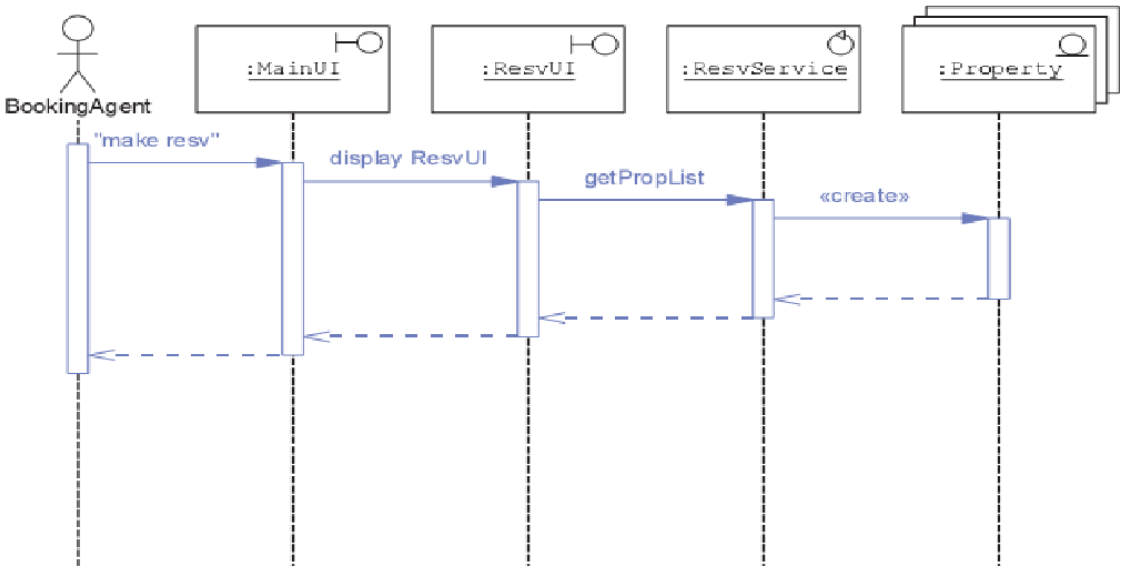
- 把每个组件都贴上用来满足用例交互的动作标签
  -



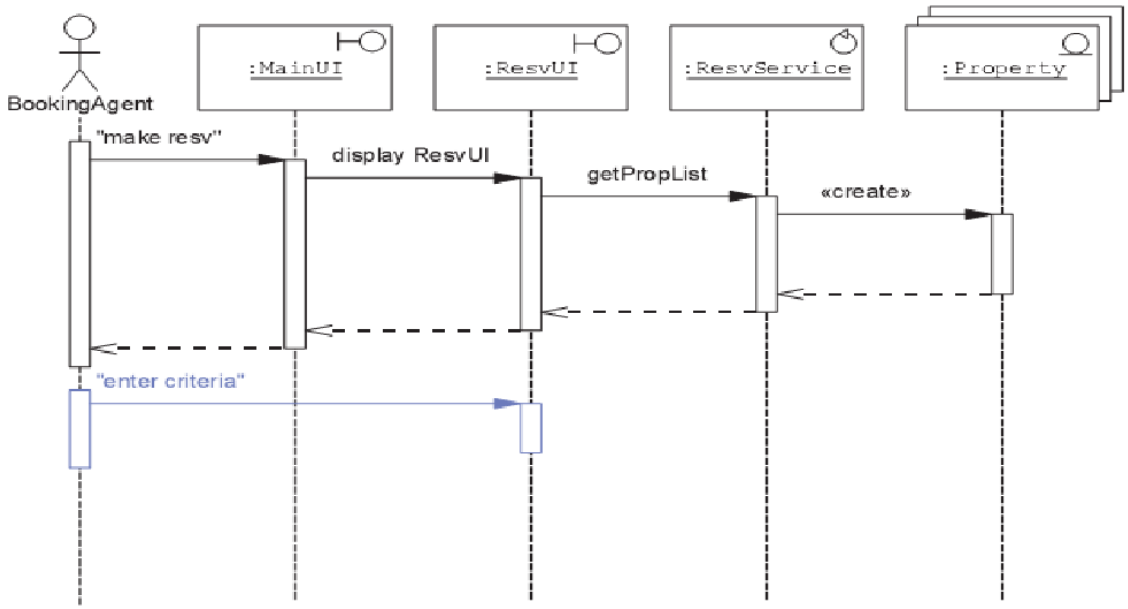
- 4. 把协作图转换成序列图（可选）
  - 1.为第一个活动安排组件:

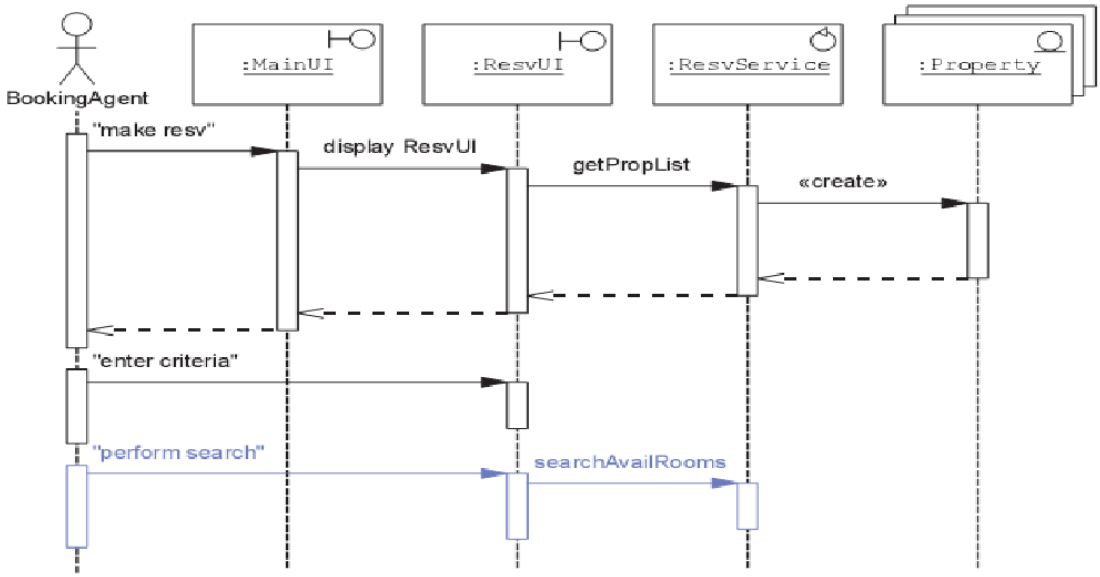


  - 2.添加消息连接和活动条：

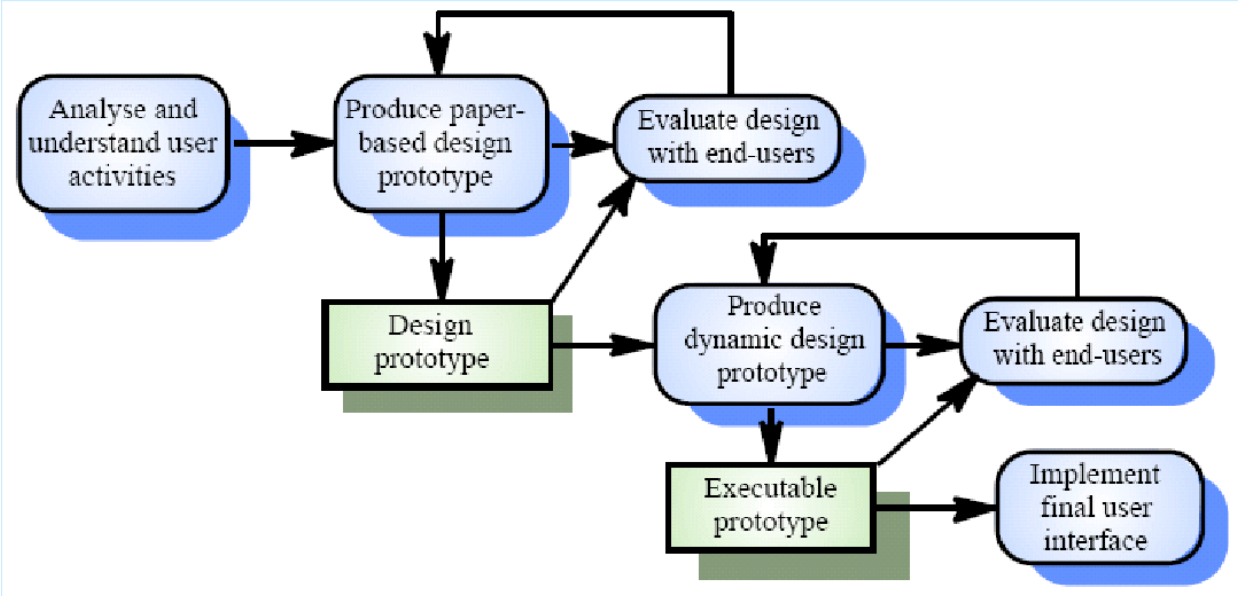


- 3.每一个活动重复2：



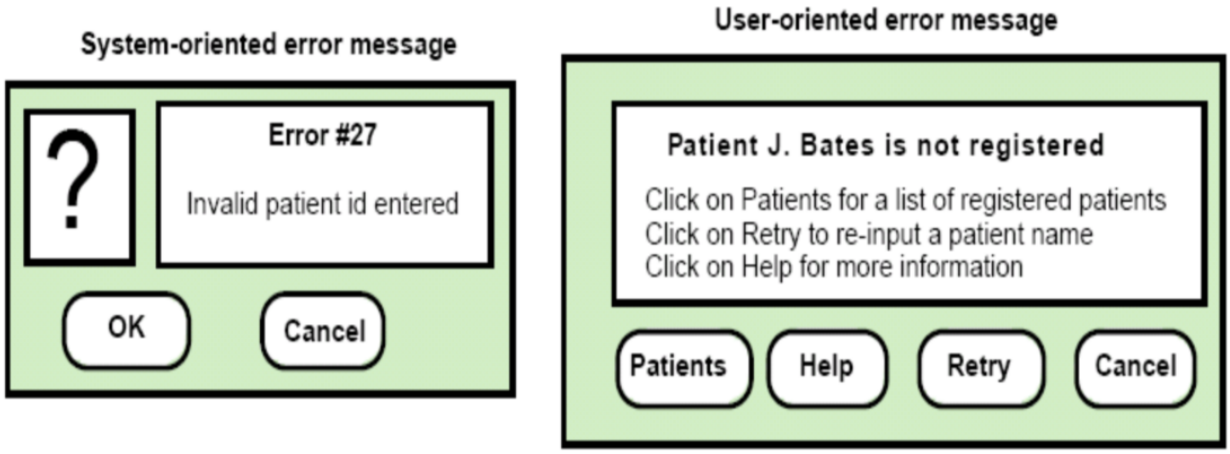


- **4.7用户界面设计**
  - 过程：



- 一般原则：
  - 用户熟悉；
  - 一致性；
  - 意外最小化；
  - 可恢复性；
  - 用户差异性

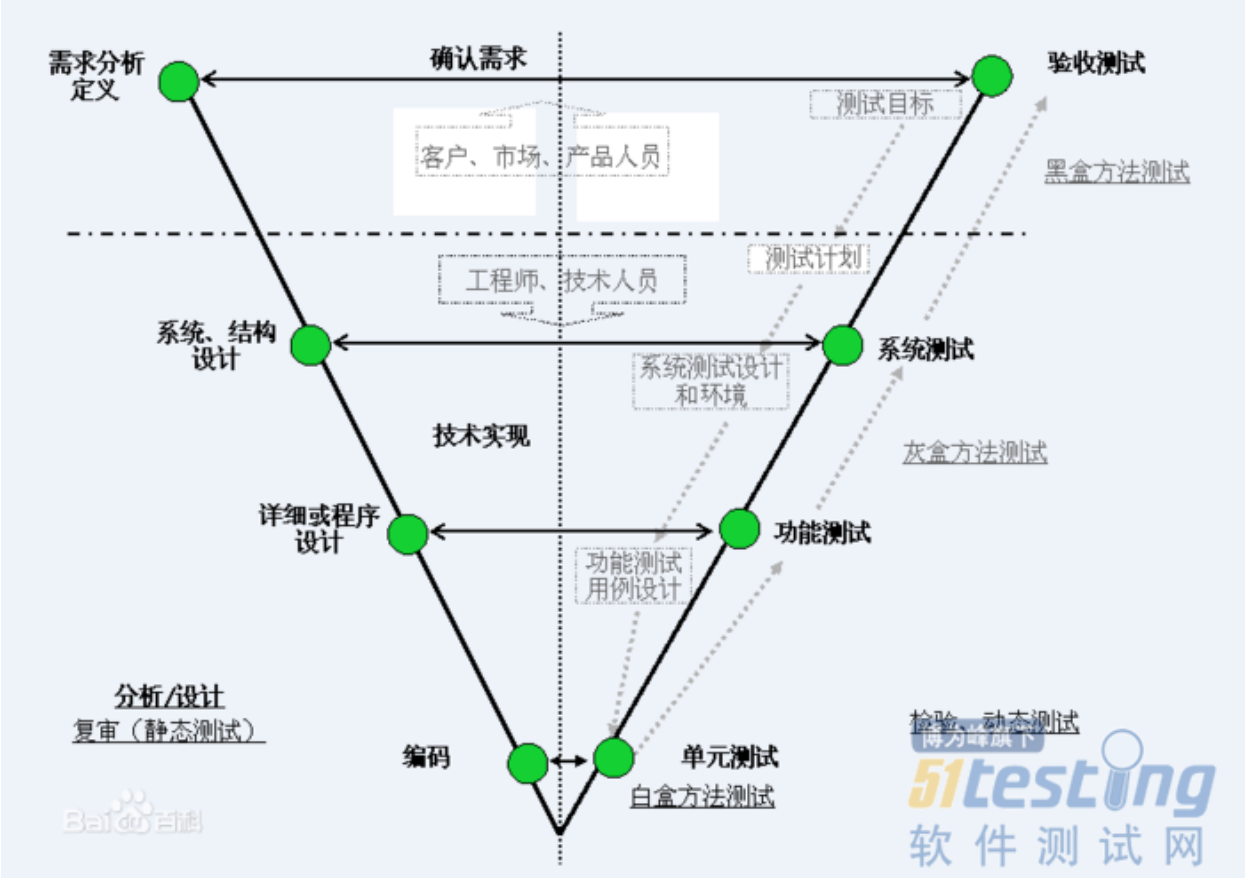
- **4.8错误消息**
  - 礼貌、简明、一致、建设性。
  -



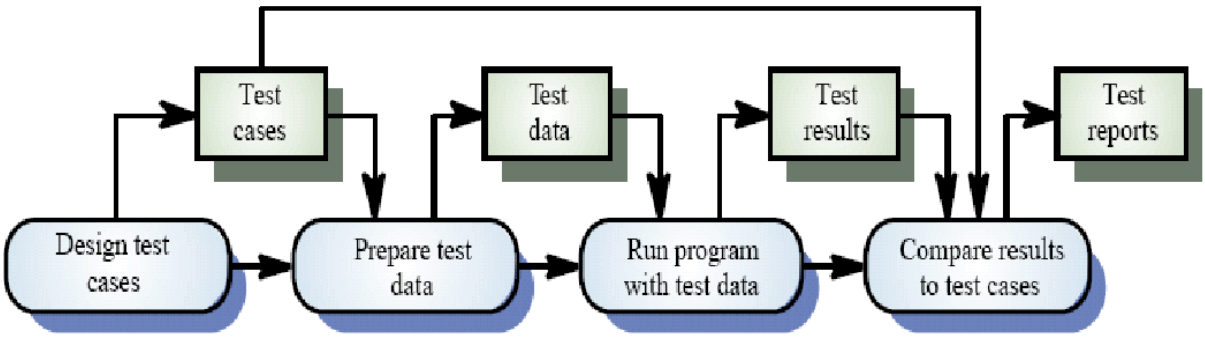
- **4.9帮助系统设计**
  - 不能是简单的用户手册复制，应该由合理的组织与结构，应该为用户提供不同的入口。

- **第五讲 软件实现与验证**
  - **5.1程序设计与调试**
    - 任务：把设计转换成程序以及在程序中去除错误，包括编程和调试。
    - 调试：程序员对自己开发的程序进行测试，这时程序中的一些明显的错误暴露出来并被根除，这叫做调试。

- **5.2验证和有效性确认**
  - 验证：检查软件是否符合它的规格描述（不符合描述的即为软件缺陷）
  - 有效性确认：检车软件是否满足客户的期待（要检查系统是否符合客户的真实愿望）
- **5.3用来实现&目标的测试类型**
  - 缺陷测试
    - 目标是找出软件中的缺陷和不足（发现程序与描述之间的不一致性）
    - 一个成功的测试就是找出了系统存在某个目前未知的缺陷。
  - 有效性确认
    - 目标是向开发者和用证实系统满足了客户需求
    - 每个需求至少要有一个测试，以证明系统是按预期运行的。（这种测试要反映真实的用户输入及其频率）
- **5.4过程的两种基本方法**
  - 软件审查
    - 通过对系统的各种静态成果，如需求文档、设计文档、源代码，进行检查和分析发现问题。
    - 静态审查无法检验软件是否可用，也不能检验非功能需求，因此程序测试是必不可少的，是起决定性作用的技术。
  - 软件测试
    - 通过使用测试数据执行系统，检查运行结果来发现问题
    - 目的使为了揭示程序中存在的错误，而不是没有错误。
    - 可分为有效性测试与缺陷测试
  - 通常程序测试和静态审查结合使用
  - 测试和调试
    - 通常交叉进行
    - 缺陷测试的目的是确定系统中存在缺陷；
    - 调试考虑的是定位和修改缺陷。
- **5.5规划**
  - 仔细的规划能够使程序检查和测试的工作得到更多的回报。
  - 过程的规划应该从开发过程的早期就开始。
  - 规划应该明确的说明静态检查与测试任务与分工。
  - 测试规划主要是制定测试过程标准，而不是描述测试本身。
- **5.6系统开发的V模型**
  -

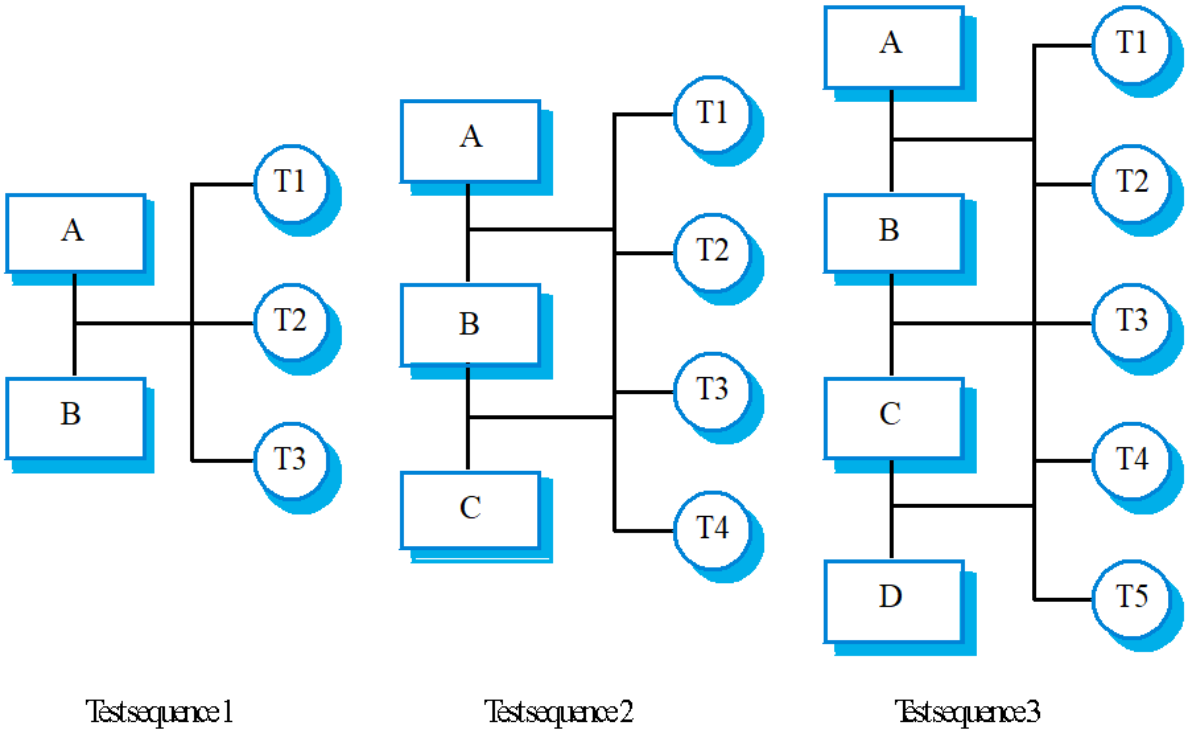


- **5.7软件测试阶段的活动**
  - 活动过程：



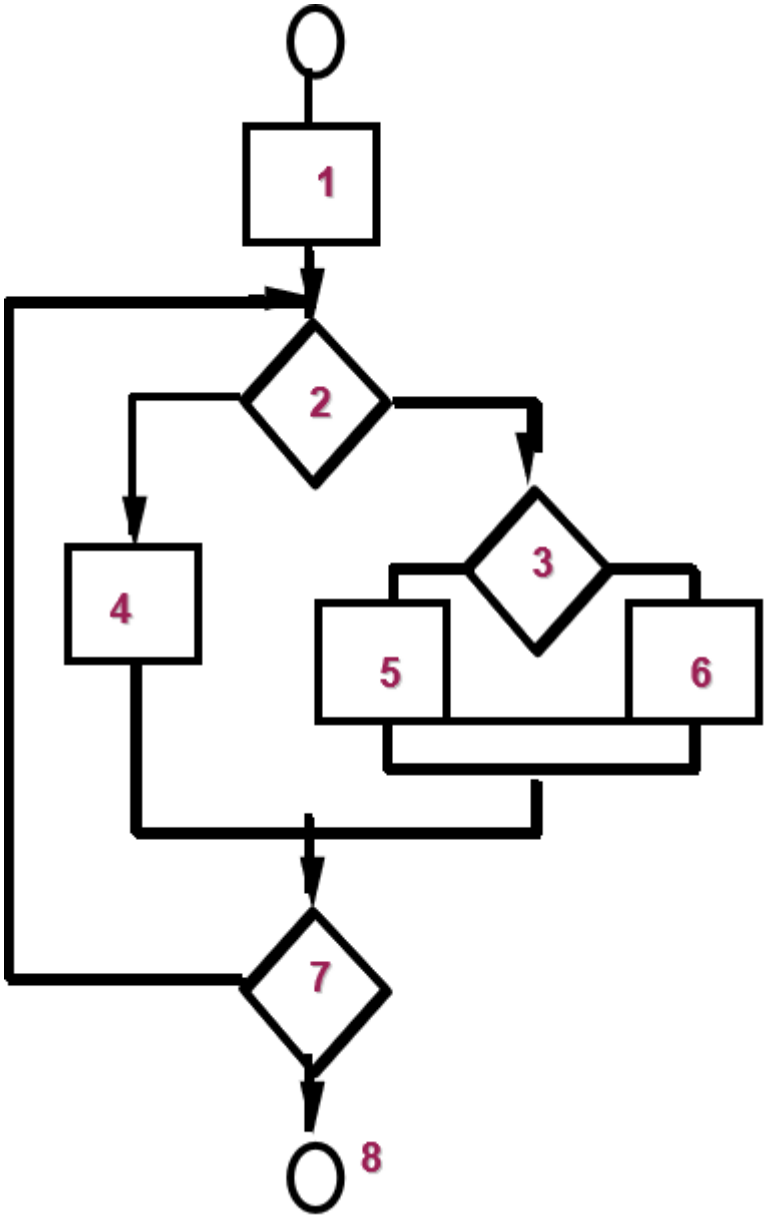
  - 阶段测试
    - 测试单个程序组件
    - 通常由程序开发者完成（除了要求特别高的系统）；
    - 这个阶段大多依靠测试者的经验。
  - 系统测试
    - 测试由组件整合的子系统和系统
    - 有专门的测试团队进行测试
    - 测试要依据需求规格说明进行
- **5.8集成测试**
  - 包括吧组件继承为系统和对合成的系统进行测试，以发现组件集成过程带来的问题，集成方式可分为：

- 自顶向下集成：
  - 从主控模块开始，沿着控制层次结构逐步向下，利用深度优先或广度优先将从属主控模块的其他模块集成到系统结构中。
- 自底向上：
  - 从原子模块开始，从底层开始把模块逐步向上集成为更大的子系统和系统。
- 增量式集成
  - 为了简化测试中错误定位的问题，可以采用增量集成的方法：
  -



- **5.9测试用例设计**
  - 用例基本构成：设计输入，期望输出，测试环境，测试对象的描述。
  - 设计测试用例是系统测试和组件测试的关键工作，主要是通过设计输入数据与预计的输出来测试系统。
  - 测试用例的设计目的是建立一组测试用例集合，用尽可能少的测试代价有效的发现系统缺陷并证明系统能够满足其需求。
  - 常用方法：
    - 划分测试与边界值分析
    - 结构化测试（白盒）
  - 黑盒测试：又叫做功能测试，测试者只关心系统的功能而不关心实现。不必了解有关系统的任何细节，只把系统看成是一个能够处理输入，产生用例的“黑盒子”，仅从功能设计的角度测试用例。
  - 白盒测试：又叫做结构测试，根据软件结构知识导出测试用例的设计方法，测试者把来测试组件看成是一个打开的白盒子，组件内部的结构对测试者是透明的，通过所用算法结构的分析设计测试用例

- **5.10等价划分测试**
  - 设计测试用例时，按特征把数据输入域划分若干等价类，等价类的每个数据应以相同方式获得，因此对揭露程序中的错误是等效的。少量有代表性的输入数据作为测试数据，以期用较小代价暴露较多的程序错误
- **5.11结构化测试**
  - 根据软件的结构知识导出测试用例。
  - 又叫做白盒测试
  - 目标：
    - 保证一个模块中所有独立路径至少被执行一次
    - 对所有逻辑值均测试真假
    - 上下边界以及可操作的范围执行所有循环
    - 检验内部结构以确保其有效性。
  - 可发现比黑盒测试更细小的缺陷。
- **5.12逻辑覆盖法**
  - 覆盖一系列测试过程的总称。这组测试会逐渐进行越来越完整的通路测试。
  - 按照不同的测试目标，逻辑覆盖可分为：
    - 语句覆盖
    - 判定覆盖
    - 条件覆盖
    - 判定-条件覆盖
    - 条件组合覆盖以及路径覆盖
    - 其中语句覆盖覆盖度最弱，路径覆盖最强！
- **5.13基本路径测试**
  - 原理：在程序控制流图基础上，分析控制结构的环路复杂度，并用这个复杂度为指南定义执行路径的基本集合，从而导出基本可执行路径集合，设计出测试用例并保证每个可执行语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值。
  -


1
2
3
4
5
6
7
8