目录

1. 软件工程概述

1.1 计算机软件

1.1.1 软件的定义

1.1.1.1 是包括程序、数据及其相关文档的完整集合

1.1.2 软件的特点

1.1.2.1 一种抽象的逻辑思维实体

1.1.2.2 开发是思维成熟的过程，没有明显的制造过程

1.1.2.3 无磨损、老化，但会退化

1.1.2.4 开发原始

1.1.2.5 高度复杂的逻辑体

1.1.3 软件的分类

1.2 软件的发展和软件危机

1.2.1 软件危机

1.2.1.1 软件工程产生的历史背景

1.2.1.1.1 软件危机的产生导致软件工程概念的诞生

1.2.1.1.2 落后的软件生产方式无法满足日益增长的计算机软件需求

1.2.1.1.3 软件危机的表现（任写三条）

1.2.1.2 软件危机的背景

1.2.1.2.1 落后的软件生产方式无法满足日益增长的计算机软件需求，导致软件开发与维护过程中产生

一系列严重问题

1.2.1.3 软件危机的表现

1.2.1.3.1 开发计划难以制定、费用和进度难以控制、难以让用户满意、一般不可维护、质量难以保证、没有适当的程序文档、成本在计算机中所占比例提高

1.2.1.3.2 原因：自身的复杂性、开发与维护过程不合理

1.2.1.4 软件危机解决途径

1.2.1.4.1 软件工程学，即采用工程化的方法从事软件系统的研究和维护。

1.3 软件工程

1.3.1 软件工程定义

1.3.1.1 应用系统化、规范化、定量化的方法来开发、运行和维护软件

1.3.2 软件工程要素

1.3.2.1 工具、方法和过程

1.3.3 软件工程目标

1.3.3.1 在给定成本、期限的前提下，开发出满足用户需求的具有可修改性、可维护性、可移植性、可重用性、可靠性、可适应性的软件产品。

1.3.4 软件工程的最终目的

1.3.4.1 摆脱手工生产软件的状况，逐步实现软件研制和维护的自动化

1.3.5 软件工程研究内容

1.3.5.1 软件开发技术、工程管理

1.3.6 软件工程的原则

1.3.6.1 选取合适的开发模型

1.3.6.2 采用适合的设计方法

1.3.6.3 高质量的工程支持力度

1.3.6.4 重视开发过程的管理

1.3.7 软件工程的基本原理

1.3.7.1 用分阶段的生命周期计划严格管理

1.3.7.2 阶段评审

1.3.7.3 产品控制

1.3.7.4 现代程序设计技术

1.3.7.5 结果能清楚的表现

1.3.7.6 开发小组小而精

1.3.7.7 承认软件工程实践的意义

1.3.8 软件工程的一般原理：

1.3.8.1 抽象、信息隐藏、模块化、局部化、确定性、一致性、完备性、可验证性

2. 软件生命周期模型

2.1 模型及软件生命周期定义

2.1.1 模型定义

2.1.1.1 模型是对实际事物的抽象表示方法；针对所需了解和解决的问题，抽取主要因素和矛盾，忽略一些不影响基本性质的因素，形成对问题域的表示方法。

2.1.2 软件生命周期

2.1.2.1 软件产品从其概念的形成开始，到该软件不再使用为止的整个周期过程。一般包括概念阶段、分析与

设计阶段、构造阶段、移交和运行阶段等不同时期。

2.1.2.2 6 个基本活动

2.1.2.2.1 制定计划

2.1.2.2.2 需求分析

2.1.2.2.3 软件设计

2.1.2.2.4 程序编写

2.1.2.2.5 软件测试

2.1.2.2.6 运行维护

2.2 传统软件生命周期模型

2.2.1.1 了解模型特点、区别

2.2.1.2 比较瀑布模型、演化模型、螺旋模型的优缺点

2.2.1.2.1 瀑布模型利用阶段评审和文档控制保证软件项目的进度和质量，但缺乏适应变化需求的灵活性；

2.2.1.2.2 演化模型能够适应变化性的需求，但是系统结构混乱；

2.2.1.2.3 增量模型结合了前两者的优点；

2.2.1.2.4 喷泉模型提高了开发项目的效率，节省开发时间；但是开发人员的管理和阶段生成的工件管理存在困难；

2.2.1.2.5 螺旋模型包含其他模型，适合于大型软件的开发；但是模型的应用有难度。

2.2.2 瀑布模型

2.2.2.1 定义：规定一些基本工程活动，并且规定他们自下而上、相互衔接的次序，如同瀑布流水，逐级下落。

2.2.2.2 主要优点

2.2.2.2.1 生命周期的阶段划分降低了软件开发的难度、提高开发过程的透明度；

2.2.2.2.2 “推迟软件实现”：强调软件实现之前分析设计工作；

2.2.2.2.3 阶段评审和文档控制能对整个开发过程进行指导，保证阶段之间的衔接性

2.2.2.3 缺点：

2.2.2.3.1 缺乏灵活性；文档控制引发的一系列问题；风险控制能力弱

2.2.3 演化模型

2.2.3.1 可以处理需求不明确的软件项目；用户参与较多，最终项目既能够让用户满意，又能保证质量

2.2.3.2 软件系统结构较差；文档控制的丧失，使得开发过程对管理人员不透明；可能需要特殊工具

2.2.4 增量模型

2.2.4.1 瀑布模型利用阶段评审和文档控制保证软件项目的进度和质量，但缺乏适应变化需求的灵活性；演化模型能够适应变化性的需求，但是系统结构混乱； 增量模型结合了前两者的优点；

2.2.5 喷泉模型

2.2.5.1 喷泉模型提高了开发项目的效率，节省开发时间；但是开发人员的管理和阶段生成的工件管理存在困难

2.2.6 螺旋模型

2.2.6.1 螺旋模型包含其他模型，适合于大型软件的开发；但是模型的应用有难度

2.2.7 原型方法

2.2.7.1 原型：模拟某种最终产品的原始模型

2.2.7.2 原型方法的应用过程

2.2.7.2.1 快速分析

2.2.7.2.2 快速构造

2.2.7.2.3 用户使用

2.2.7.2.4 评价反馈

2.2.7.2.5 修改

2.2.7.3 原型方法支持的软件生命周期

2.2.7.3.1 辅助或代替分析阶段

2.2.7.3.2 辅助设计阶段

2.2.7.3.3 代替分析与设计阶段

2.2.7.3.4 代替分析、设计和实现阶段

2.2.7.3.5 代替全部开发阶段

2.2.7.4 原型方法的作用（优点）:

2.2.7.4.1 逐渐明确用户需求，适应需求变化；用户的介入能够及早发现问题，降低风险。

2.3 新型软件生命周期模型

2.3.1 RUP

2.3.1.1 特点

2.3.1.1.1 RUP 模型定义与增量模型最为贴近

2.3.1.1.2 RUP 模型融合了增量模型与喷泉模型的特点

2.3.1.1.3 RUP 是一个面向对象的程序开发方法论

2.3.1.2 RUP 模型通过迭代增量建模思想提高风险控制能力;相比之下，瀑布模型的风险控制能力较弱

2.3.1.2.1 迭代计划安排是风险驱动的，高风险因素集中在前两个阶段解决

2.3.1.2.2 每一次迭代都包括需求、设计、实施、部署和测试活动，每个中间件产品都进行集成测试

2.3.1.2.3 每一个阶段结束时有严格的质量评审

2.3.1.2.4 根据市场竞争情况适时推出中间版本，降低市场风险

2.3.2 敏捷思想与 XP 方法

2.3.2.1 相比于瀑布模型，XP 适合于需求经常改变的领域

2.3.2.2 敏捷思想的目的：

2.3.2.2.1 在实际开发环境中提高开发质量和效率，同时能够避免简化和不切实际的期望

2.3.2.3 敏思思想本身不是一个完整的方法论

3. 软件需求分析

3.1 软件需求分析的目标及任务

3.1.1 需求分析

3.1.1.1 需求分析是软件设计和软件实现活动的前提，也是保证软件质量的重要因素。

3.1.2 对象、目标和任务

3.1.2.1 需求分析阶段研究的对象是用户要求

3.1.2.2 任务是借助业务系统的逻辑模型导出目标系统的逻辑模型，解决目标系统“做什么”的问题

3.2 软件需求分析的原则和方法

3.2.1 三条原则

3.2.1.1 三条原则（软件需求分析必须展现的三个方面）：

3.2.1.1.1 问题的信息域必须表示和理解

3.2.1.1.2 软件将完成的功能必须定义

3.2.1.1.3 软件的行为必须表示

3.2.2 三种建模

3.2.2.1 数据建模

3.2.2.2 功能建模

3.2.2.3 行为建模

3.3 软件需求工程

3.3.1 需求开发

3.3.1.1 需求获取

3.3.1.2 需求分析

3.3.1.3 需求定义

3.3.2 需求管理

3.3.2.1 需求确认

3.3.2.2 需求跟踪

3.3.2.3 需求变更控制

3.4 软件需求分析过程

3.4.1.1 需求沟通、需求获取、需求分析与综合、需求建模、制定需求分析规格说明、需求确认、需求评审

3.4.2 需求获取的对象

3.4.3 需求获取

3.4.3.1 主要步骤

3.4.3.1.1 准备调查

3.4.3.1.2 调查与记录

3.4.3.1.3 分析需求信息

3.4.3.1.4 撰写用户需求说明书

3.4.3.1.5 需求确认

3.4.3.2 需求获取与记录（注意事项）

3.4.3.2.1 按时到达，注意礼节，获得好感

3.4.3.2.2 实现了解用户身份、背景，随机应变

3.4.3.2.3 灵活调查方式，不轻易打断用户

3.4.3.2.4 尽量避免给用户添麻烦，但也不能因此降低调查力度

3.4.3.2.5 避免片面听取一部分用户需求

3.4.3.3 《用户需求说明书》以及《软件需求分析规格说明书》的区别

3.4.3.3.1 前者主要采用自然语言来表达用户需求，内容相对于后者而言比较粗糙，不够详细

3.4.3.3.2 后者是前者的细化，更多的采用计算机语言和图形化符号来描述需求，产品需求是软件系统设计的直接依据。

3.4.3.3.3 两者之间可能不存在一一对应的关系，产品需求可能在不断变化中

3.4.4 需求类别

3.4.4.1 功能需求、性能需求、环境需求、可靠性需求、安权保密要求、用户界面需求、资源使用需求、软件成本消耗与开发进度需求

3.4.4.2 非功能性需求

3.4.5 需求分析与综合

3.4.5.1 细化并分析用户需求

3.4.5.2 撰写软件需求规格说明书

3.4.5.3 进行需求确认

3.4.6 需求建模

3.4.7 编制需求分析文档

3.4.8 需求确认

4. 面向对象需求分析方法

4.1 统一建模语言

4.1.1 UML 概述

4.1.1.1 图形化建模语言

4.1.1.2 用例图、类图、对象图、顺序图、协作图、状态图、活动图、组件图、部署图

4.2 面向对象的需求分析建模

4.2.1.1 目标：构建目标系统的逻辑模型

4.2.1.2 面向对象分析建模的活动

4.2.1.2.1 领域建模

4.2.1.2.2 用例建模

4.2.1.2.2.1.1 用例图

4.2.1.2.2.1.2 用例描述

4.2.1.2.2.1.3 系统顺序图

4.2.1.2.2.1.4 操作契约

4.3 领域建模

4.3.1 领域模型的表示

4.3.1.1 领域模型是针对某一特定领域内概念类或者对象的抽象可视化表示

4.3.1.2 **类图+活动图（一般画类图就足够）**

4.3.1.2.1 类图表达概念及其关系。

4.3.1.2.2 活动图表示业务流程。活动图在附录三

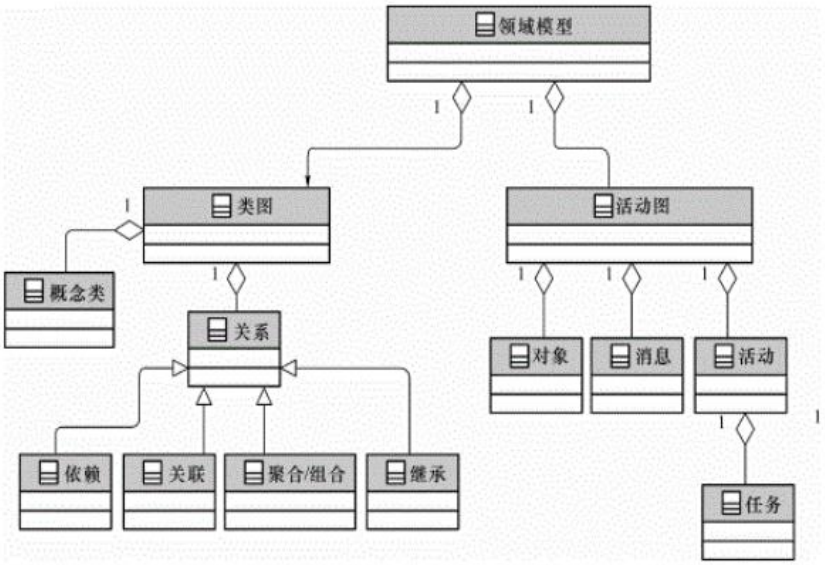
4.3.1.3 概念类及其关系的创建步骤：

4.3.1.3.1 识别或抽象出领域的概念类或对象

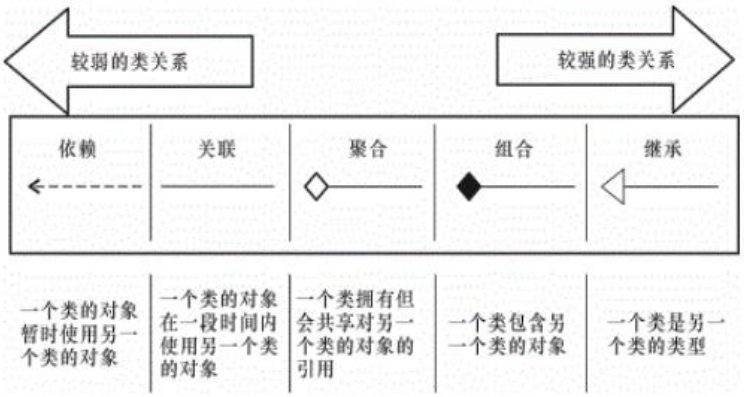
4.3.1.3.2 建立概念类之间的关系

4.3.1.3.3 设置概念类的关键属性

4.3.1.4 领域建模画图时不用描述出属性



4.3.2 UML 概念类及关联表示



4.3.2.1 继承

4.3.2.1.1 子类可以用于父类所有的属性和操作，且可以定义自身的一些属性和操作

4.3.2.2 组合

4.3.2.2.1 一个类 A(整体类)完全拥有另一个类 B(部分类)，且其他类都不能拥有类 B

4.3.2.2.2 整体消失时，部分类也不存在

4.3.2.3 聚合

4.3.2.3.1 一个类 A(整体类)拥有另一个类 B(部分类)，同时其他的类 C 也能拥有部分类 B；

4.3.2.3.2 部分类的对象不因为整体类的消失而不存在，这是聚合与组合最大的区别。

4.3.2.4 关联关系

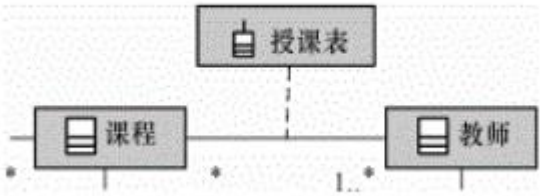
4.3.2.4.1 一个类可以将另一个类的属性或者类对象的整体作为其属性

4.3.2.5 依赖

4.3.2.5.1 对象的级别上存在关系，而不是对象的内部属性和操作

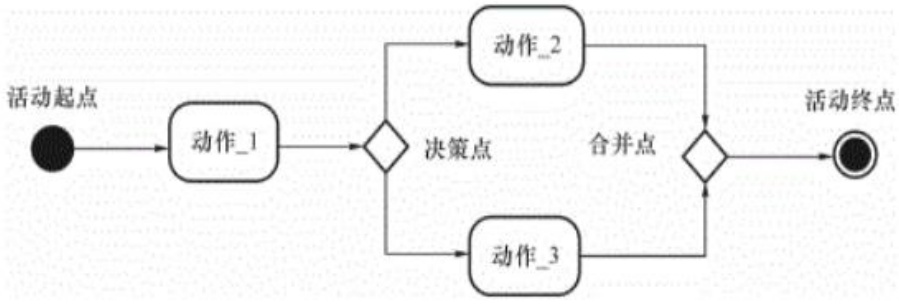
4.3.2.6 关联类

4.3.2.6.1 主类与关联类之间用一条相交的点线来连接



4.3.3 活动图

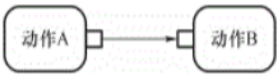
4.3.3.1 基本元素



4.3.3.2 启动

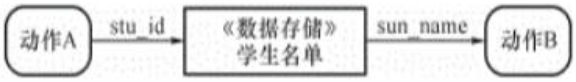
4.3.3.3 动作

包含有输入输出引脚的动作



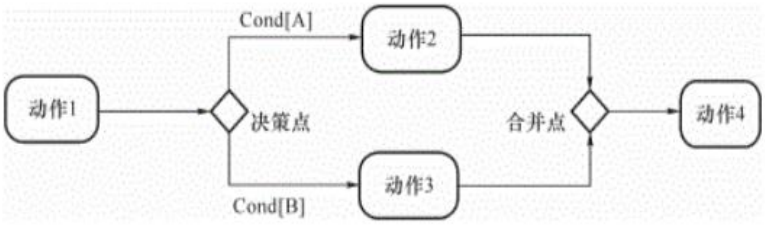
4.3.3.4 活动：活动包含起点到终点的全部内容，可以包含多个动作。

4.3.3.5 数据对象：动作执行中的中间产物。以矩形框表示



4.3.3.6 控制流：单箭头表示动作之间的连接

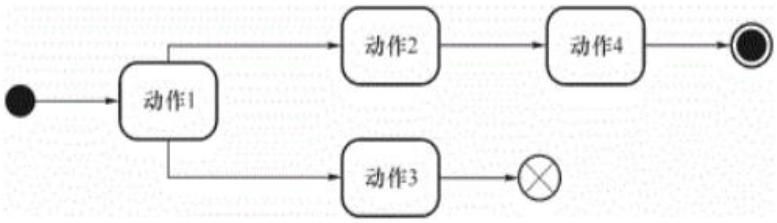
4.3.3.7 决策点与合并点



4.3.3.8 终止

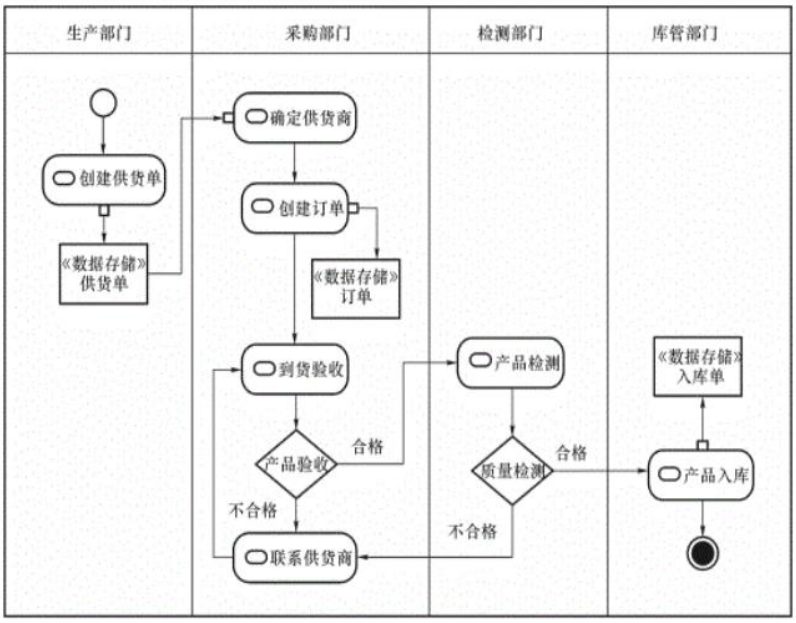
4.3.3.8.1 终点：停止所有活动图的动作

4.3.3.8.2 流结束点：结束当前路径



4.3.4 活动图的扩展结构

4.3.4.1 多泳道活动图



4.4 用例建模

4.4.1 用例及用例模型

4.4.1.1 **用例**：从使用系统的角色出发描述使用者与系统或系统某个功能之间的交互场景

4.4.1.2 **用例模型**：以用例为核心，从使用者的角度描述和解释对于待建软件系统的功能需求，并通过 UML 的用例图规范化表示，使用系统顺序图和操作契约为软件对象确定和软件系统结构设计提供充分的依据。

4.4.1.3 为什么用例方法比传统需求分析中的功能特性列表更有效

4.4.1.3.1 用例将系统的特性和功能放到面向用户目标的语境中去考虑，从而能使识别出来的功能是真正为用户提供价值的功能

4.4.1.3.2 用例描述功能的方式是：让用户写出多种使用系统的场景，和用户日常工作序列一致，使得交流更简单有效

4.4.2 用例图

4.4.2.1 问题域边界

4.4.2.1.1 说明当前讨论用例图应用的范围，用一个矩形表示，边界名称要标出

4.4.2.2 角色

4.4.2.2.1 人、其他系统或设备、时钟

4.4.2.3 用例

4.4.2.3.1 一个具有功能名称的椭圆形图标

4.4.2.4 关联

4.4.2.4.1 角色与用例之间建立关联

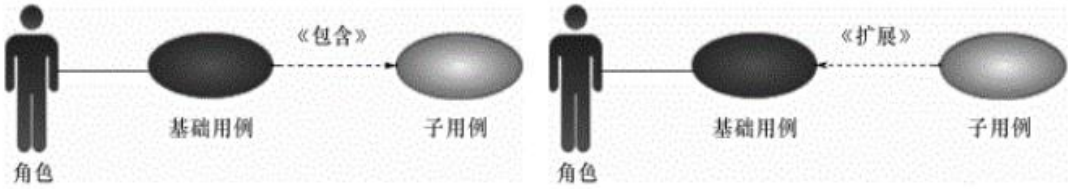
4.4.3 用例之间的关系

4.4.3.1 包含关系

在用例说明中，发现一段或几段相似的操作步骤，而且这些都与角色有交互，同时又可表示为一个系统的独立功能，可以把这部分提取出来作为一个新的用例

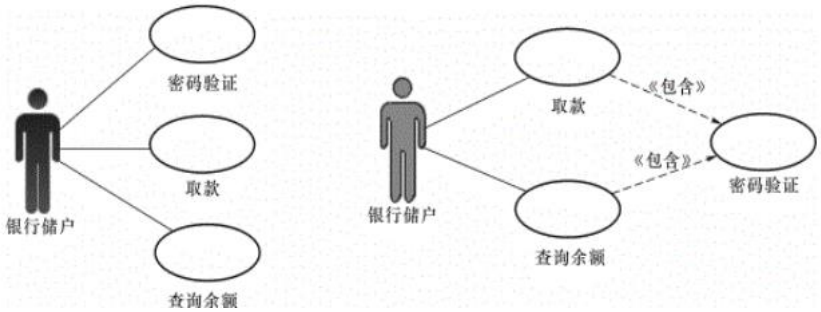
4.4.3.2 扩展关系

存在判断分支的情况时，可将分支部分的操作步骤独立为一个子用例，但是该子用例与角色之间没有直接的关联关系，必须在基础用例执行过程中某个条件成立时被调用，为此将这部分用例视为基础用例具有扩展关系的子用例

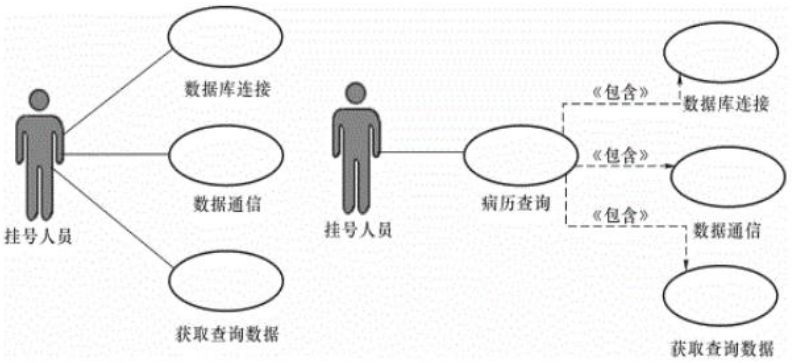


4.4.3.3 常见误区

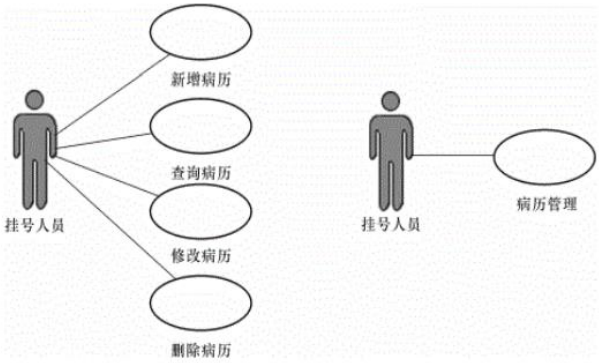
4.4.3.3.1 非目的性用例（如登陆系统）



4.4.3.3.2 系统操作型用例



4.4.3.3.3 用例表示的粒度



4.4.4 用例描述（用例说明）

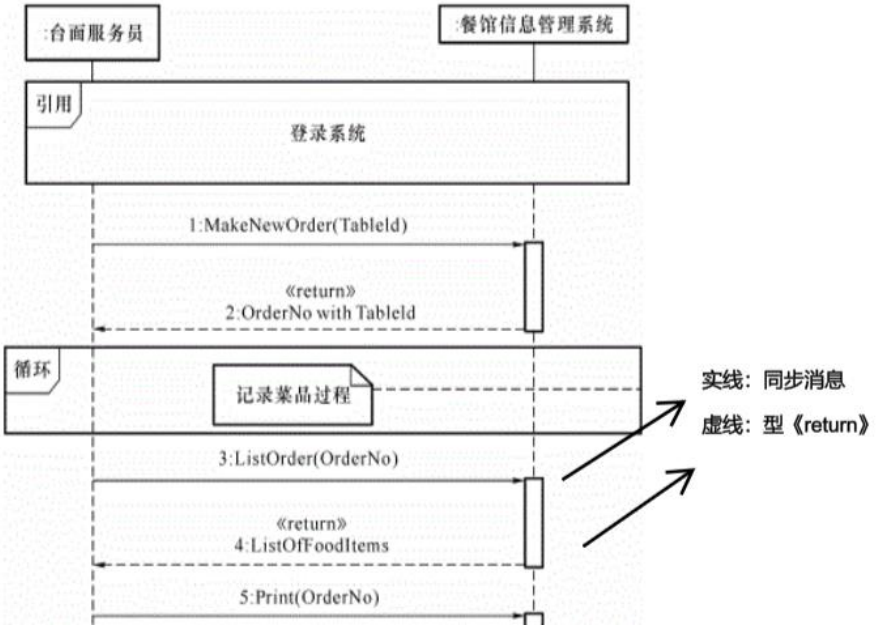
用例编号	
用例名称	动+宾
范围	作用范围
级别	企业/用户/子系统目标级别
参与者	调用系统服务的主要参与者
项目相关人员及兴趣	相关人员兴趣的内容
前置条件	开始之前必须为‘真’的条件

后置条件	成功结束之后必须为‘真’的条件	
成功场景	步骤	活动
	1	
	2	
特殊需求	非功能性需求	
发生频率		

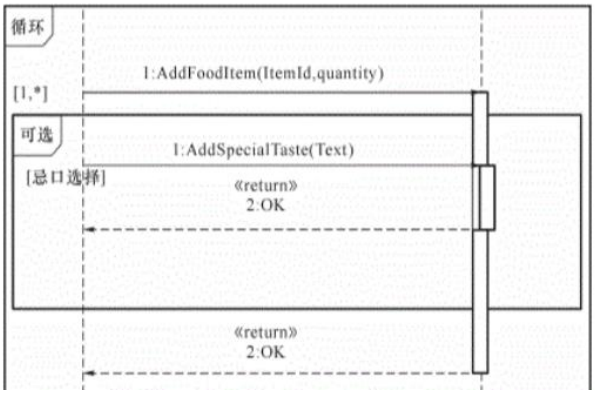
4.4.5 系统顺序图 （SSD）

4.4.5.1 通过交互图元素，描述一个用例中角色与系统之间某一个场景的消息交互形式

4.4.5.2 两类对象：用例中的角色对象、待构建系统的对象（仅有一个）



4.4.5.3 循环与可选的表示方法



4.4.6 操作契约

4.4.6.1 系统顺序图上的代表构建的软件对象接受角色发送的系统事件请求后，系统对象根据需求的具体内容返回一个明确的结果，称之为操作契约，即根据明确的角色的要求（系统事件）系统必须返回的契约结果。

4.4.6.2

（系统事件）操作	操作名称及参数
交叉引用	产生该操作的用例
前置条件	执行该操作之前的系统或领域模型的状态
后置条件	执行之后系统或领域模型的状态 ● 对象创建或删除 ● 对象之间的‘关联’创建或删除 ● 对象的属性值修改

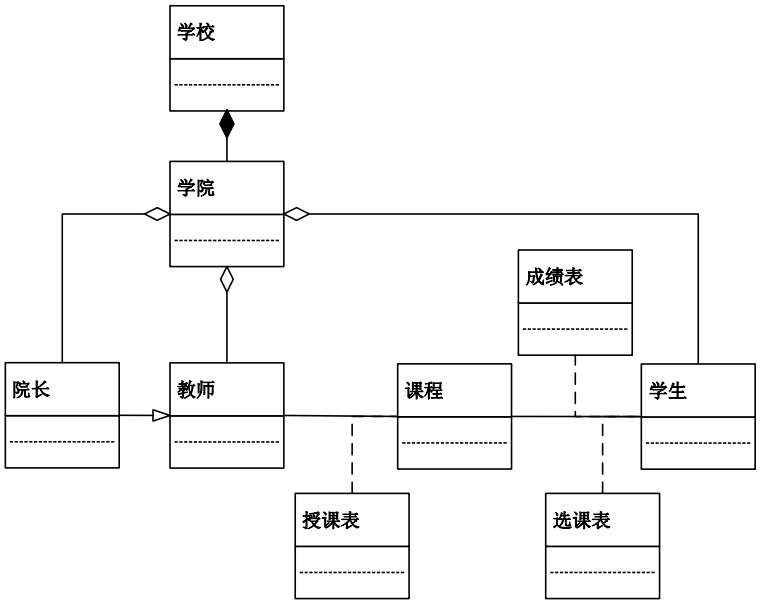
4.4.7 用例描述与操作契约的区别

用例描述	
前置条件	开始之前必须为‘真’的条件
后置条件	成功结束之后必须为‘真’的条件

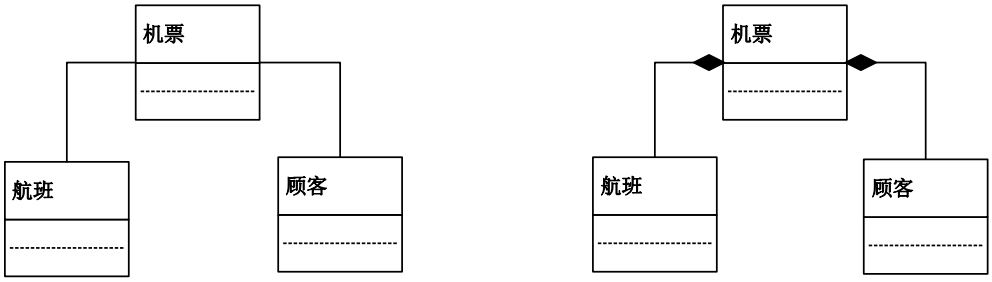
操作契约	
前置条件	执行该操作之前的系统或领域模型的状态
后置条件	执行之后系统或领域模型的状态 ● 对象创建或删除 ● 对象之间的‘关联’创建或删除 ● 对象的属性值修改

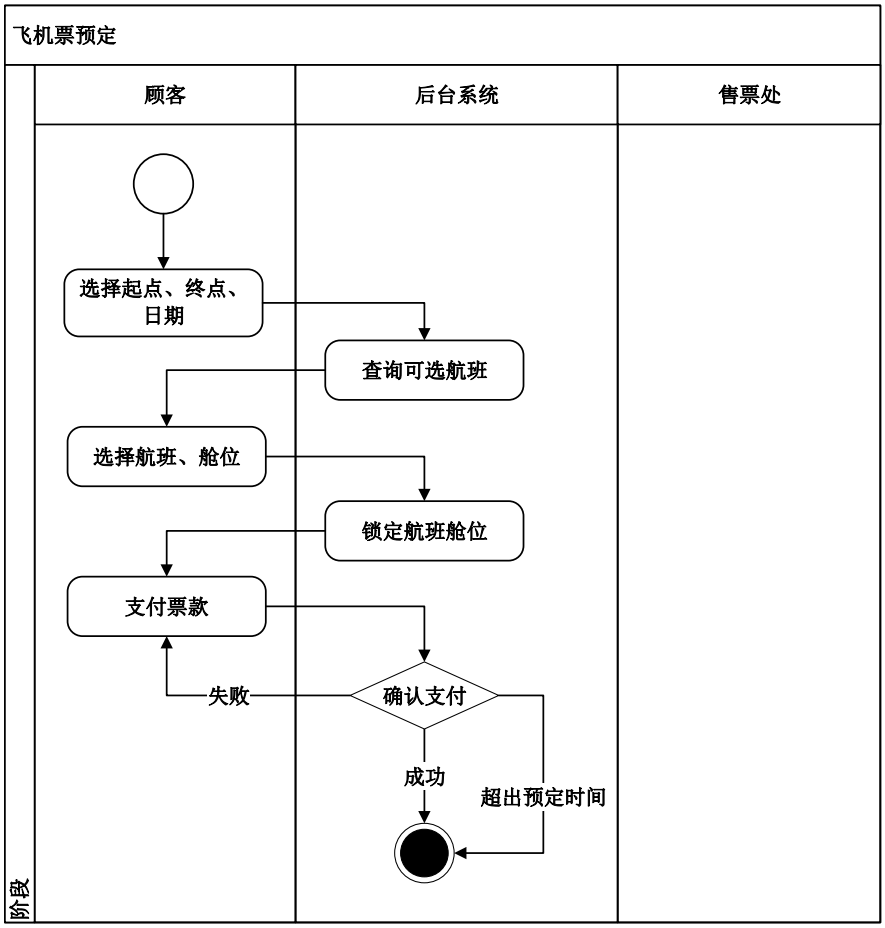
4.5 习题

4.5.1.1 大学知识领域



4.5.1.2 飞机票预定系统





4.5.1.3 试给出图书管理系统中“借书”用例的系统事件。针对其中的每一个系统事件，给出操作契约

系统事件	VerifyCard(CardId)
交叉引用	借书
前置条件	操作员（可以是自动借书机）验证通过
后置条件	借书卡被验证

系统事件	MakeNewOrder(CardId)
交叉引用	借书
前置条件	CardId 通过验证，开始订单处理
后置条件	（1） 一个新的订单创建

	（2） 订单与借书卡建立关联 （3） 订单与操作员建立关联

系统事件	AddBookItem(BookId)
交叉引用	借书
前置条件	操作员正在处理订单
后置条件	（1） 一个借书项创建 （2） 订单与书项建立关联

系统事件	ListOrder(OrderNo)
交叉引用	借书
前置条件	操作员完成订单处理
后置条件	订单属性被修改：isCompleted

系统事件	Print(OrderNo)
交叉引用	借书
前置条件	操作员完成订单处理
后置条件	订单属性被修改：isPrinted

系统事件	EndOrder()
交叉引用	借书
前置条件	操作员完成订单处理
后置条件	退出当前订单

5. 结构化需求方法

5.1 面向数据化流图的结构化分析模型

5.1.1 数据建模（ER 图）

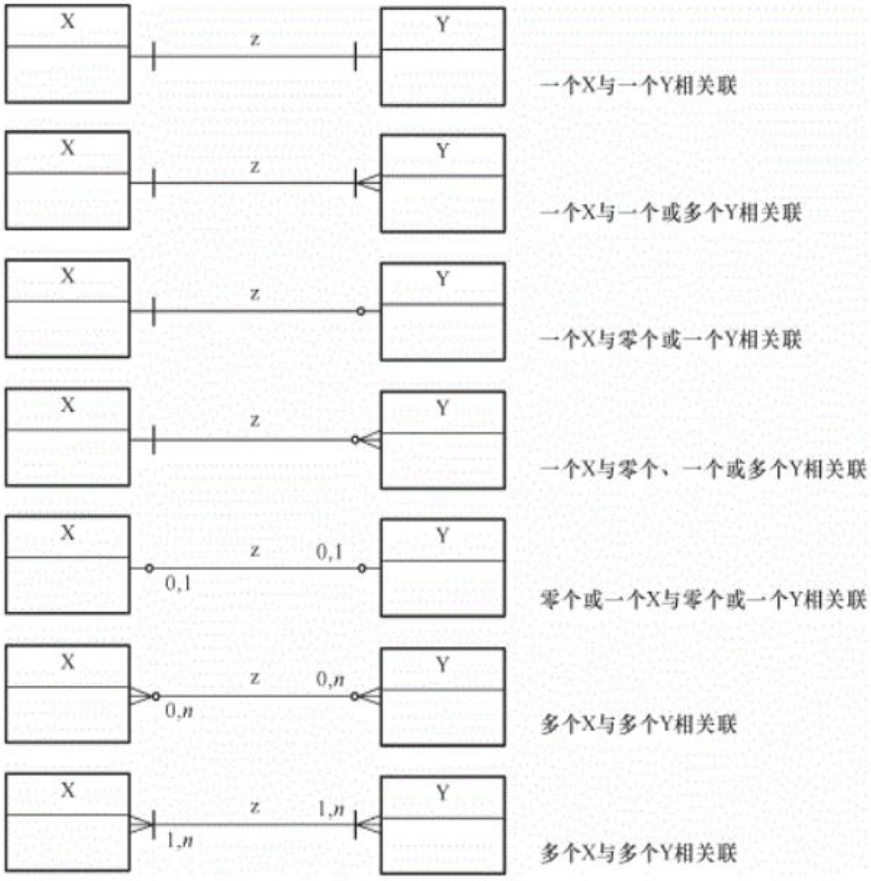
5.1.1.1 数据对象、属性和关系

5.1.1.1.1 属性

5.1.1.1.1.1 描述属性、命名属性、引用属性

5.1.1.1.2 关系

5.1.1.1.3 基数



5.1.1.2 数据结构规范化

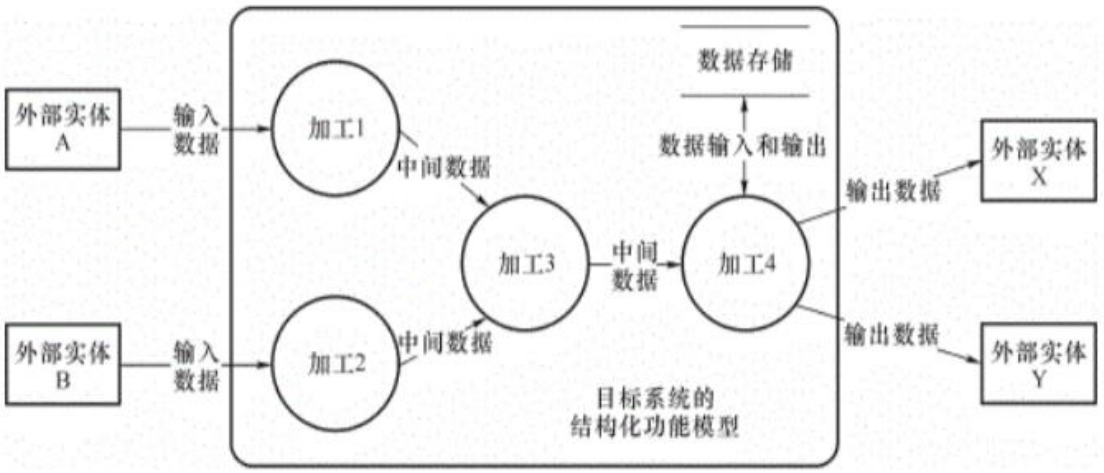
5.1.1.2.1 目的：消除数据模型中不必要的冗余

5.1.1.2.2 关系规范化的程度通常按属性间的依赖来划分，以范式（NF）来表达。

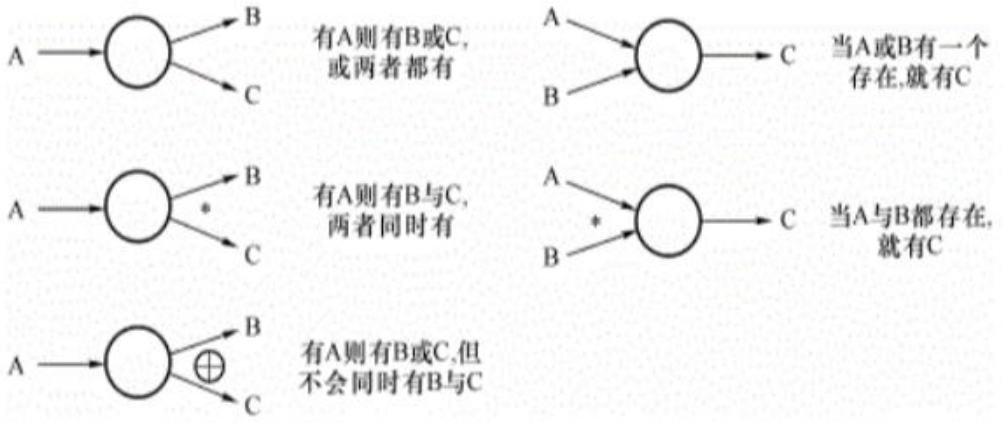
5.1.1.2.3 范式是符合某一种级别关系模式的集合；范式等级越高，冗余越少，数据库表越多

5.1.2 数据流图

5.1.2.1 结构

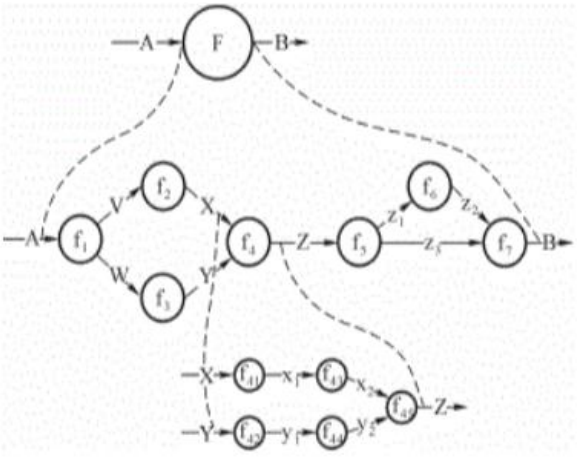


5.1.2.2　数据流与加工之间的关系



5.1.2.3 分层的数据流图

5.1.2.3.1　顶层流图只有一个‘加工’，代表被开发系统



5.1.2.3.1.1　确定外部实体

5.1.2.3.1.2　确定加工

5.1.2.3.1.3　分析实体与加工间的数据流

5.1.2.3.1.4　确定有哪些数据存储

5.1.2.3.2　中间层数据流图可以有多层，对其父层细化

5.1.2.3.2.1　中间层的第一层：增加一些子系统以及子加工之间的、子系统与外部实体之间

的数据流

5.1.2.3.2.2 中间层的第二层：进一步解释每一个输入数据进入加工后，系统内是否还存在一些加工接收数据流、分解数据流、转换数据流知道存储必要的数据信息进入数据文件存储。

5.1.2.3.3 底层流图的加工无需再细化

5.1.2.3.4 **高、低层输入输出要保持一致**

5.1.2.4 数据流图的绘制步骤

5.1.2.4.1 找出外部实体,它们是系统的数据源点与汇点,界定系统的分析范围。

5.1.2.4.2 在图的边上画出系统的外部实体。

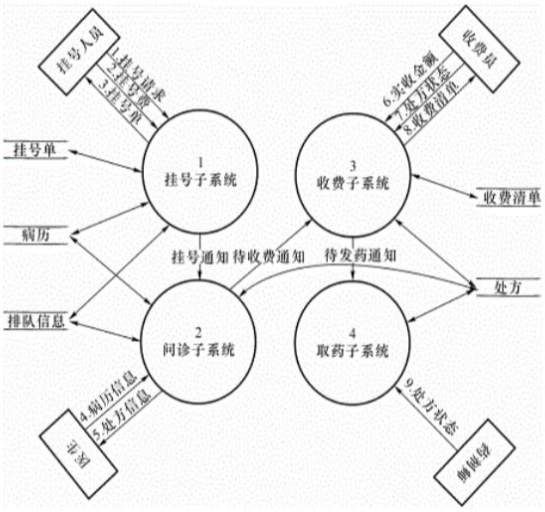
5.1.2.4.3 找出外部实体的输入数据流与输出数据流。

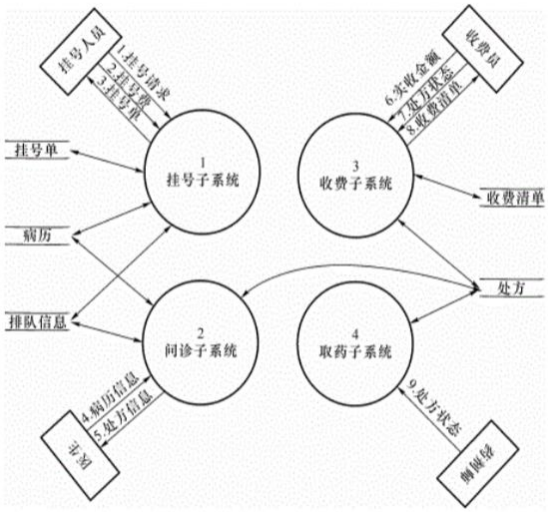
5.1.2.4.4 从外部实体的输出数据流(即系统的源点)出发,按照系统的逻辑需要,逐步画出系列逻辑“加工”,直到找到外部实体所需的输入数据流(即系统的汇点),形成封闭的数据流。(特别提示,此时的加工完全是个人分析的结果,没有绝对意义上的对错)

5.1.2.4.5 按照既定的原则进行检查和修改。

5.1.2.4.6 按照上述步骤,再从各“加工”出发,画出所需的子图。

5.1.2.5 实例





5.1.2.5.1 区别在于子加工之间是否加入数据流

5.1.3 系统行为建模

5.1.3.1 状态迁移图（了解）

5.1.4 数据词典

5.1.4.1 数据词典的构成

5.1.4.1.1 数据流词条描述

数据流是数据结构在系统内传播的路径

数据流名称	唯一标注数据流的名称
简要描述	简要介绍该数据流的作用
数据流来源	来源
数据流去向	去向
数据流组成	内部数据元素的组成成分
备注	数据流量、流通量

5.1.4.1.2 数据元素词条描述

数据元素名称	唯一标识名称或编号

简要描述	数据元素的作用，位于那个数据结构
类型	数字、字符型
长度	
取值范围	

5.1.4.1.3 数据文件词条描述

数据文件名称	唯一标识数据文件名
简要描述	存放什么数据
输入数据	写入文件的数据内容或数据结构
输出数据	从文件读出的数据内容或数据结构
数据文件组成	数据结构组成
存储方式	数据文件的操作方式

5.1.4.1.4 加工逻辑词条描述

加工名称	唯一标识加工的名称
简要描述	逻辑、规则、功能
加工编号	加工的层次
输入数据流	流入的数据流
输出数据流	流出的数据流
加工逻辑	加工的逻辑、规则

5.1.4.1.5 外部实体词条描述

外部实体名称	唯一标识外部实体的名称
简要描述	实体的性质、与系统的关系
有关数据流	与之交互的数据流
备注	