
东南大学计算机科学与工程学院

《软件工程》复习总结

（根据张敏灵老师的英文 PPT）

第一章

软件系统的特点：

1. 复杂的创造

很多功能

实现许多不同的（往往又是矛盾的）目标

包含许多组成部分

不同的参与者

开发流程和软件生命周期经常持续很多年

2. 容易发生变化

客户或终端用户需求变化

发现错误

开发者有了很好的理解

新技术出现，员工变迁

软件工程的定义：

1. 建模

软件工程师通过建模解决复杂性问题

模型：系统的抽象体现，使我们可以回答系统的问题并直观理解系统

2. 问题求解

在有限的预算和时间下，模型寻求合理的解决方案

OOSE: object-oriented software engineering

需求获取

需求分析

系统设计

对象设计

实现

测试

3. 知识获取

软件工程师收集数据，组织成信息，形成知识

非线性过程

4. 原理驱动

软件工程师需要了解作出决定的环境条件和做出这些决定的基本原理用来应对变化

SE 概念:

技术、方法和工具的集合，可以在有限的时间、预算以及变化出现的情况下实现高质量的软件系统

参与者和身份

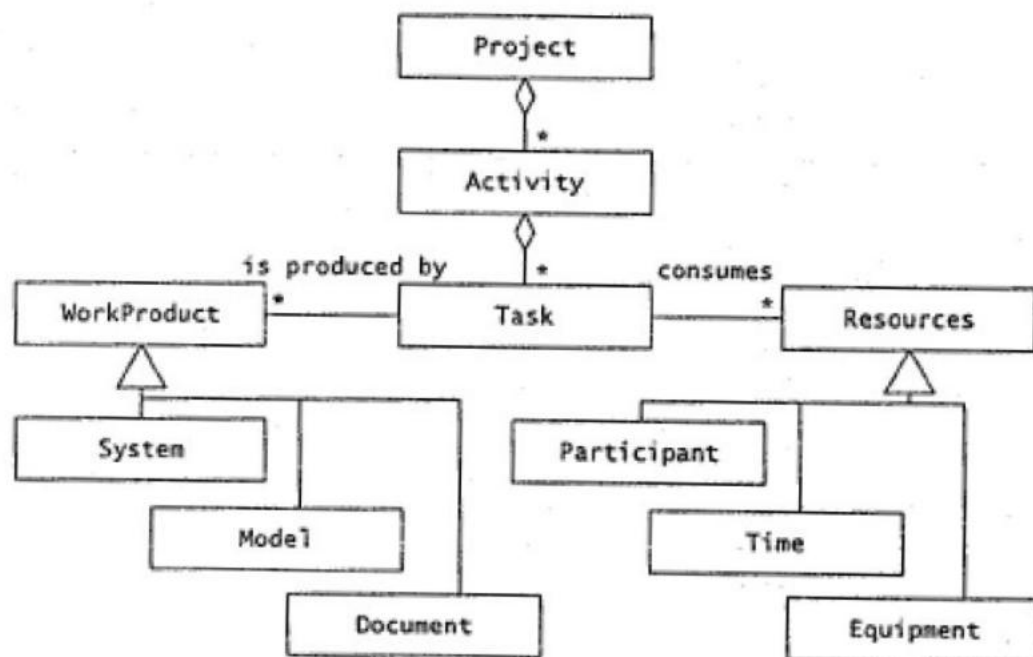
系统和模型

生产产品（内部产品+交付产品）

功能需求和非功能需求

符号，方法，方法论

UML Diagram for SE Concepts



方法论:

方法的集合，解决一类问题或说明如何以及何时每个方法需要被用到
本书用到的方法论:

1. 需求获取和分析
2. 系统设计和对象设计
3. 变化相关的活动
4. 配置管理

SE 不仅是有关开发，也关于管理

开发活动：需求获取，需求分析，系统设计，对象设计，实现测试
管理活动：交流，原理管理，软件配置管理，项目管理，软件生命周期

第二章

UML: Unified Modeling Language

面向对象软件建模中出现的标准

创始人：

OMT (James Rumbaugh)

OOSE (Ivar Jacobson)

Booch (Grady Booch)

用途广泛：

功能模型：用例图（用户角度）

对象模型：类图（对象、属性、联系、操作角度）

需求分析—>分析对象模型—>应用概念

系统设计—>系统设计模型—>系统接口描述

对象设计—>对象设计模型—>解决方案对象的详细描述

动态模型：交互图—>在一系列对象之间进行一系列消息交换来描述行为

状态机图—>针对某一个对象的状态转换

活动图—>针对控制 and 数据流描述行为

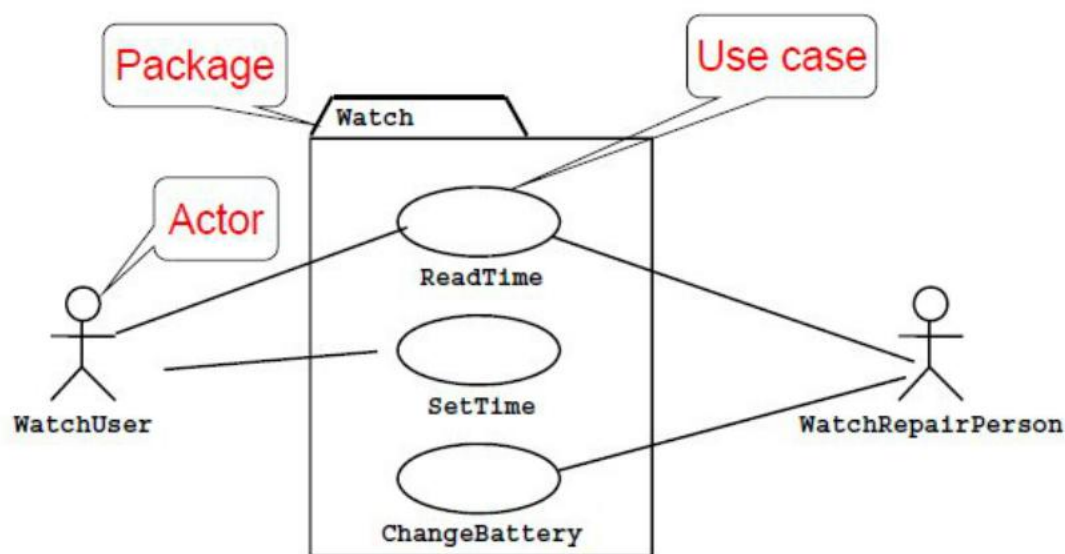
用例图：

描述系统功能，在需求获取和分析时使用，从外部角度来关注系统行为

用例：描述系统提供的功能，产生用户可见的结果

参与者：任何与系统交互的人（用户，另一个系统，系统的物理环境）

参与者在系统边界外，用例在系统边界内



（咱们一定要记住用例名是写在这个椭圆下面的啊 T T 学长考试的时候全写在里面了!）

类图：

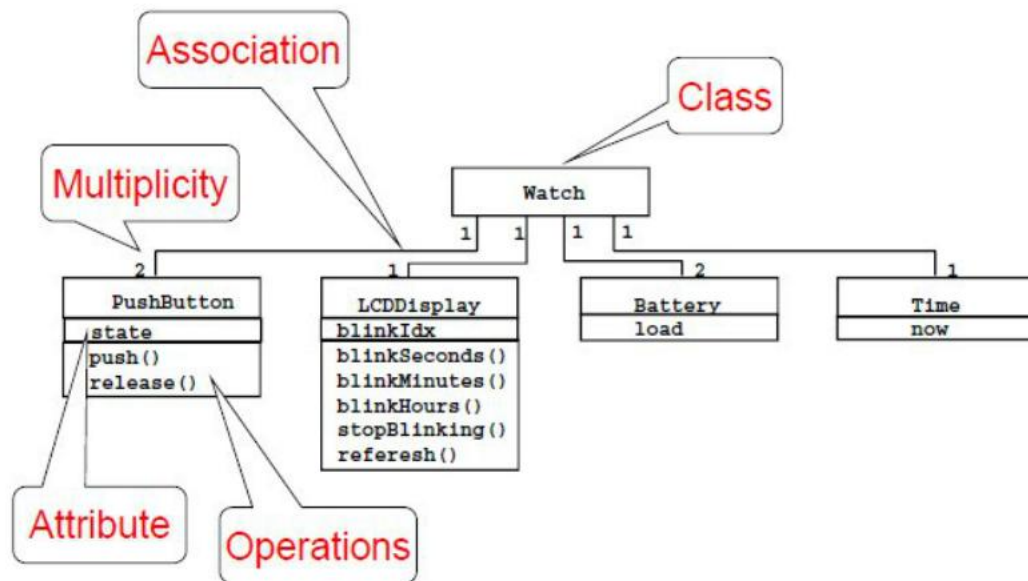
描述系统的结构

类：描述具有相同结构和行为的对象集的抽象

对象：在系统执行过程中被创建、修改和销毁的类的实体
有状态（包括属性值和与其他对象的联系）

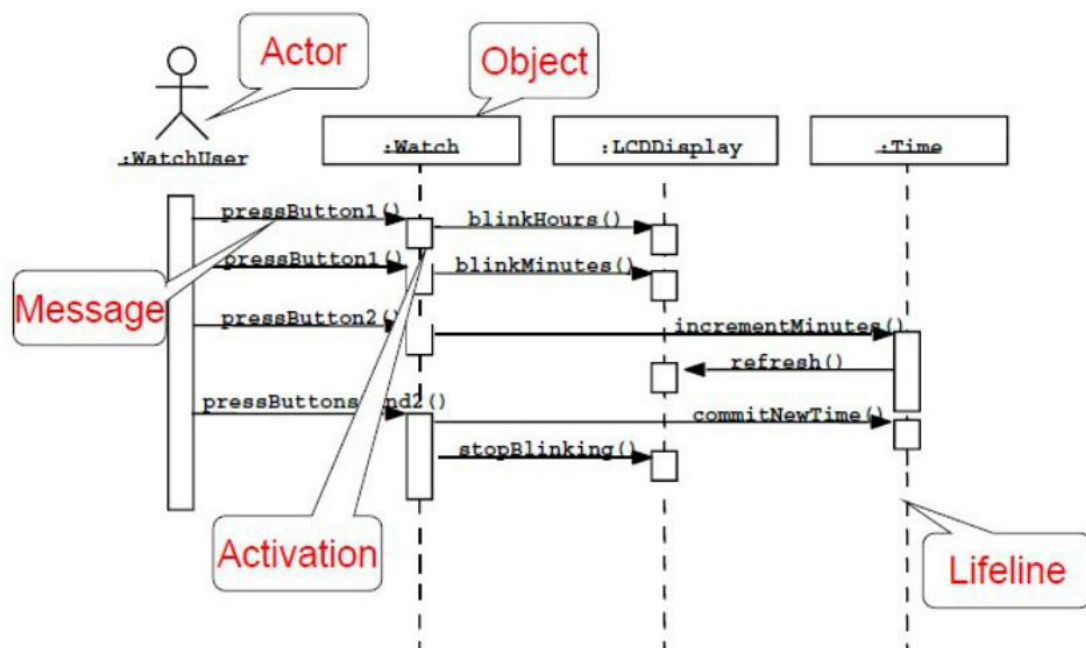
类图的成分：类，对象，属性，操作，联系

Class diagrams represent the structure of the system



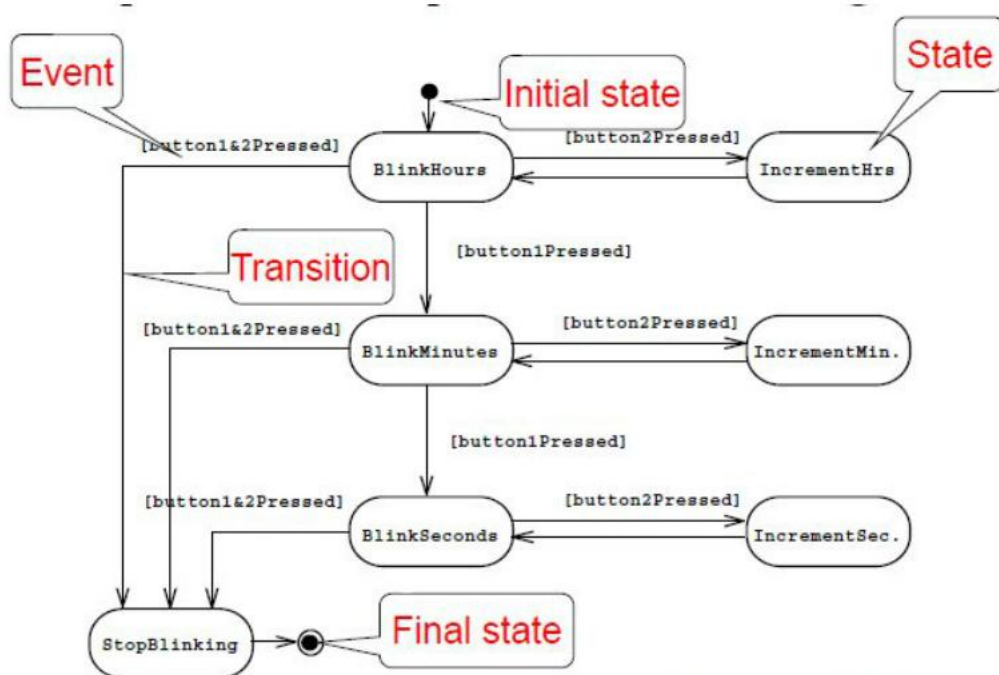
交互图：

在用例中涉及的对象（参与对象），表现的就是这些对象之间发生的交互



Sequence diagrams represent the behavior as interactions

状态机图：转换包括对象未来可以转向的状态和转变条件



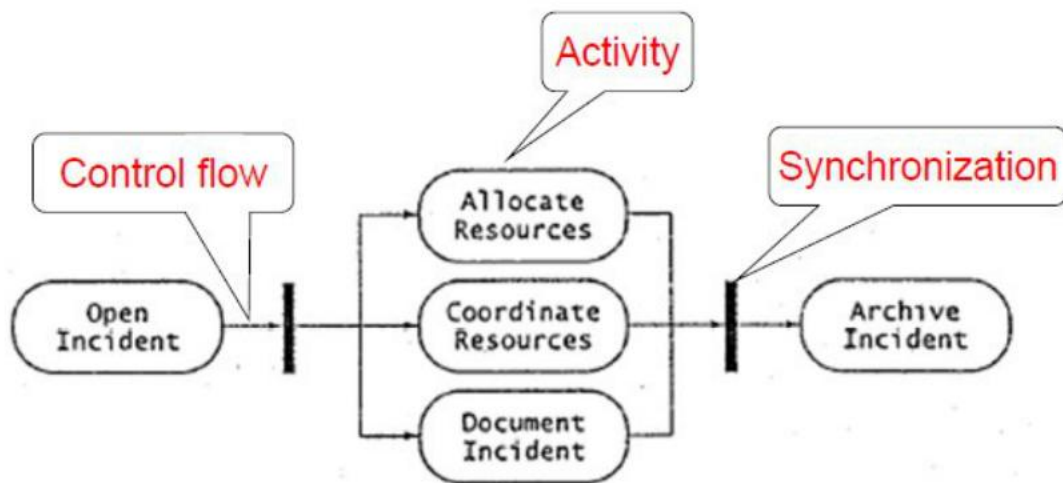
Represent behavior as states and transitions

活动图：

活动：代表一系列操作执行的建模元素

针对活动来描述系统行为

其他活动的结束、对象的可用和外部的活动都可以出发活动的执行
与流程图相似：控制流（操作发生的顺序），数据流（操作中对象的交互）



Activity diagrams represent the behavior of system with respect to activities

用例图：

《extend》代表异常或很少调用的用例，箭头指向被扩展的用例

《include》代表被分离出来的用例，箭头指向使用的用例

《inheritance》代表一个用例可以通过添加细节特化另一个更一般的用例，箭头指向那个一般的用例

用例描述：

1. 用例名称
2. 参与者
3. 事件流
4. 进入条件
5. 退出条件
6. 特殊需求

类图：

表现系统的结构

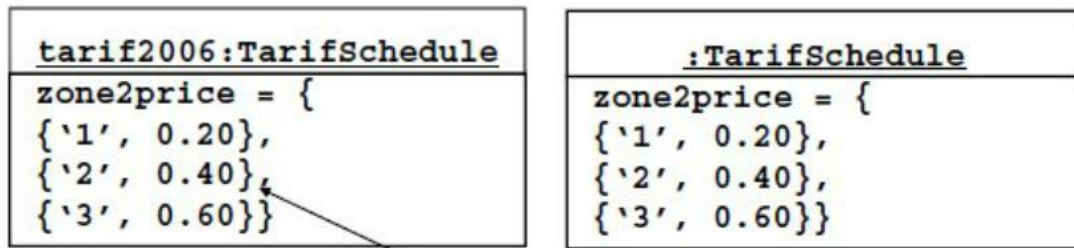
需求获取分析中对应用域概念建模

在系统设计中对于子系统建模

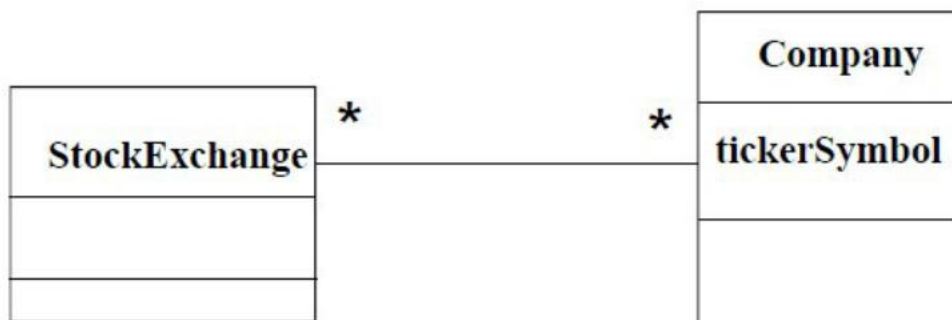
在对象设计中说明类的详细行为和属性

类：代表一种概念，封装属性和操作，每个属性有一个类型，每个操作有一个签名，只有类名是强制的信息

实例/对象：实例代表一种现象，实例名有下划线，名字可以只包括类名



Many-to-Many Associations



***** : zero to many

a: any integer including 0

a...b: a to b

a...*: a to many

关系也可以通过属性来刻画

关系类可以用自己的操作和属性，用虚线连在关系线上

聚集：

共享聚集：表示一种“属于”继承，空心菱形

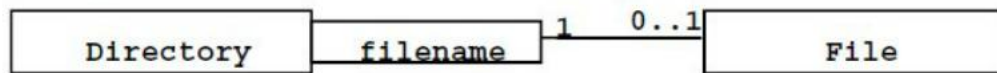
组合聚集：更强形式的聚集，生命周期必须一致，实心菱形

限制（qualification）：减少关系的复杂性

Without qualification



With qualification



继承：“是一种”的继承，通过分类简化分析模型，子类继承父类的操作和属性

包：可以用 UML 包机制来组织子系统

对象模型建立步骤：

1. 找到新的类
2. 定义名字、属性、方法
3. 找到类之间的关系
4. 标注一般的关系 (has\owns, etc.)
5. 决定关系之间的多重性
6. 重审关系
7. 找到分类（使用继承）
8. 简化、重组

顺序图：

在分析中，优化用例描述，找到更多的对象（参与对象）

在系统设计中，优化系统接口

消息→参与对象，消息是参与对象中的方法

水平虚线箭头代表数据流

在消息前有个*代表迭代重复发送消息，在消息前有个[布尔表达式]代表一种发送消息的条件

消息指向一个对象的激活（就是对象下面的长方形）则代表创建（creation），

在最后的激活上有个✖代表销毁，销毁可以代表一个对象有用的生命的结束

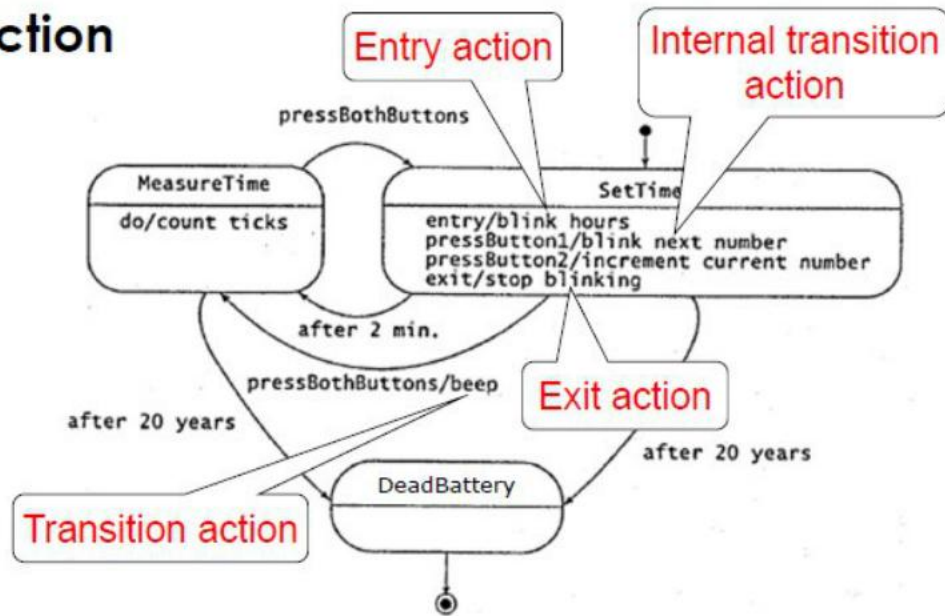
状态机图：

状态：满足一个对象（或系统）某种属性的一种情形

转换：被事件、条件或时间触发的状态转变

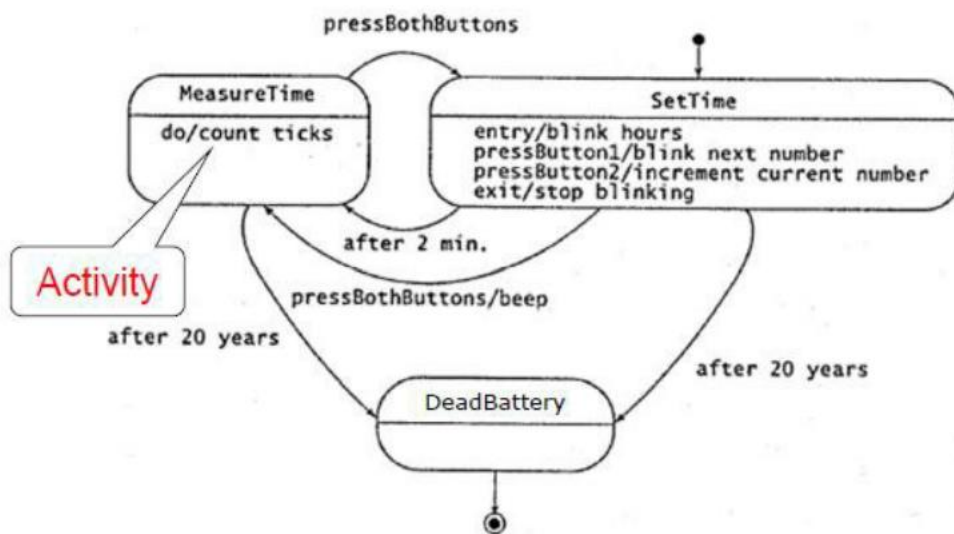
Action 动作：不可被打断的，有转换 action，内部转换 action，entry/action, exit/action

Action



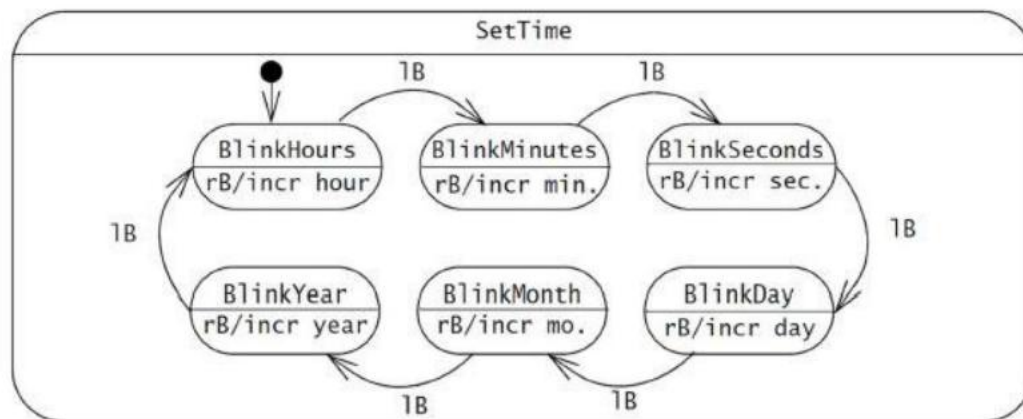
Activity 活动: 在状态内部的可能是长时间的活动, 当退出转换被调用的时候 activity 会被打断, 用 do 做标签

Activity



嵌套状态机: 内部转换来减少复杂度

Nested State Machines



活动图：

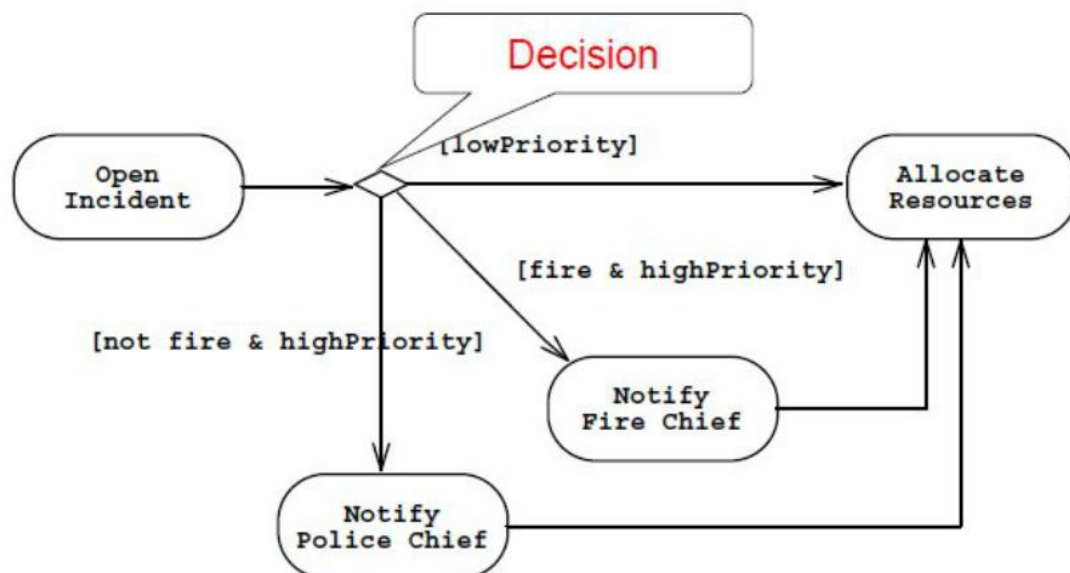
是状态图的特殊形式

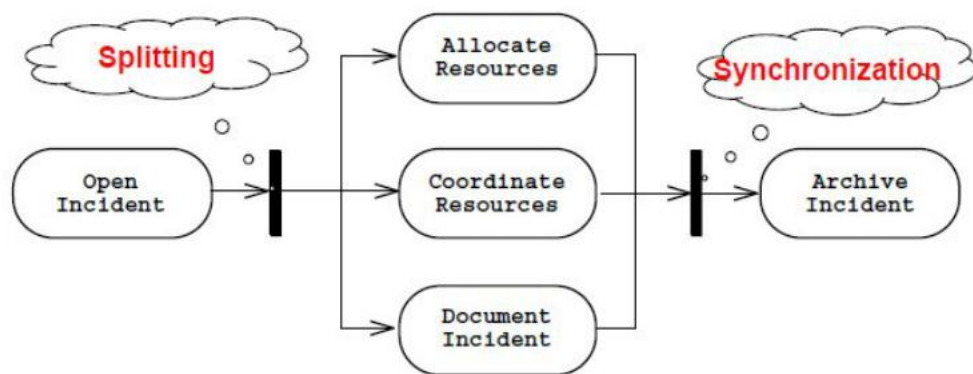
活动图中的状态是活动（“功能”）

活动结束后自动跳转，不需要驱动

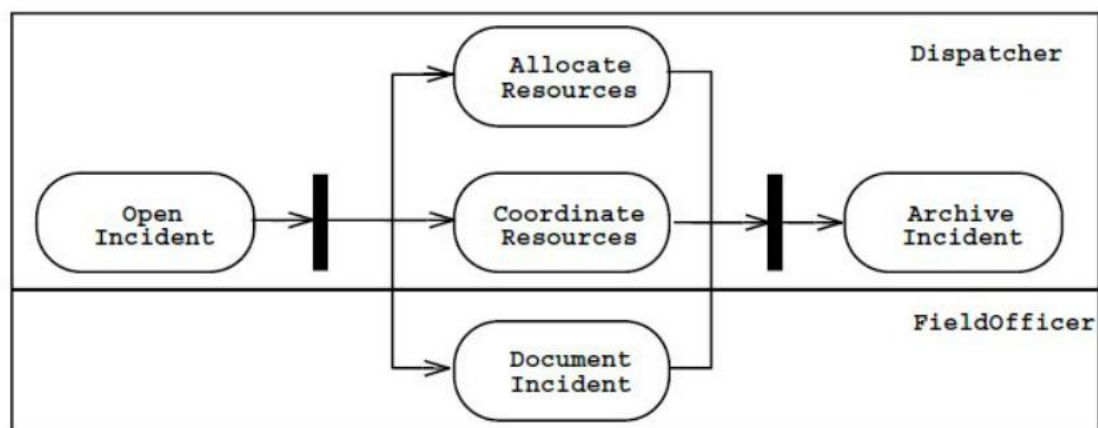
活动图对于描述系统的工作流程非常有用

活动允许对“决定”（一个菱形）和“并发性”（竖线，包括把控制流分成几个线程和同步多个活动）进行建模





活动还可以被分成若干个泳道，代表这些活动是被哪个对象或子系统实现的



第三章

角色：

定义了责任（要去做什么）

责任被指定到角色身上，角色被指定到人身上

任务：

描述了被管理人员追踪的最小数量级的工作，包括：

1. 角色
2. 工作产品
3. 开始时间
4. 计划工期
5. 需要资源

工作产品：

可见的任务产出，交付到用户手中的叫作交付产品

活动：

一系列相关的任务

可能会以不同的名字再次被包含到更高级的活动当中
允许关注分离
在活动中存在优先关系

计划内沟通：

问题陈述，客户评审，项目浏览，同行评审，现状浏览，集思广益，发布，事后浏览……

计划外沟通：

需求的澄清，需求的变化，问题求解

同步沟通机制：

面对面交谈，问题和正式会面，会议，同步群件

异步沟通机制：

电子邮件，新闻组，万维网，Lotus 公司的 Notes

（其他重点可以看书，还是多看看中文比较好，毕竟试卷是中文....）

第四章

软件生命周期：

需求获取，需求分析，系统设计，系统细节设计，实现，测试

需求说明用自然语言，分析模型用 UML

功能需求：描述系统和环境之间的交互，与实现无关

非功能需求：不与功能行为直接相关的方面

1. 可用性：必须是可测量的（网购需要的步数）
2. 鲁棒性（健壮性）：错误的输入，外界环境的改变…
3. 有效性：系统正常运行时间的比例（不要总是 down 机）
4. 性能
5. 可支持性

限制（伪需求 Pseudo requirements）：由客户或环境提出

需求确认：

1. 正确性：是否代表了客户的本意
2. 完整性：系统能应用的场景是否全面
3. 相容性：没有相互矛盾的需求
4. 明确性：需求只能有一种解释方式
5. 可实现性：需求能够被实现和交付
6. 可追踪性：每个系统行为都可以追踪到一系列的功能需求

需求获取的不同方式：

1. 绿地工程：需求从用户和客户处抽取
2. 再工程：新需求从现存系统中抽取
3. 界面工程：需求从技术使能者或新的市场需求抽取

场景：一种陈述性描述，所描述的是，在人们使用计算机系统和应用的时候他们所做的和所经历的。（这是根据老师 PPT 上的原句翻译的，个人认为书上 99 面的翻译是错的。

原句是：“A narrative description of what people do and experience as they try to make use of computer systems and applications.”）

As-is 场景：描述了当前情况，常用于再工程项目，通过观察用户并将他们的活动描述成场景，一次来理解当前系统

Visionary 空场景：描述了未来的系统，常用于绿地工程和再工程项目。开发者和用户两个方面。

关于用例的在前面已经有了，再稍作补充：

用例是场景的抽象提取

用例可以被多个对象使用

分析的关键：从用例出发，找到参与对象

第五章

对象建模的步骤：

1. 类定义
2. 寻找属性
3. 寻找方法
4. 寻找类之间的关系

实体对象：代表被系统追踪的持久性信息（应用域对象，也被称作“业务对象”）

边界对象：代表用户和系统之间的交互

控制对象：代表系统所执行的控制任务

不同的对象类型让我们更好的应对变化

对象类型起源于 Smalltalk (Alan Kay)：

Model, View, Controller (MVC)

Model <-> Entity Object

View <-> Boundary Object

Controller <-> Control Object

版型机制，模板机制 (stereotype mechanism)：

在 UML 中可以引入新的建模元素类型 《String》Name

还可以用图标 (icons) 和图形符号来定义模板

在用例中寻找参与对象：

1. 选中一个用例，观察事件流

2. 做语法分析 (Abbott' s Technique), 名词就是对象或类的候选项, 动词就是操作的候选项
3. 定义不同对象的类型

Mapping parts of speech to model components (Abbot's Technique)

<i>Example</i>	<i>Part of speech</i>	<i>UML model component</i>
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	Constraint, class, association

注意：这只是在用例中寻找对象，如果是整个对象定义过程，则是如下：

1. 从用户和应用域专家处获得一系列场景
2. 从场景中抽取出用例
3. 再进行上面“用例中寻找参与对象”的过程，同时还要进行的是组建 UML 类图

组建类图的过程：

1. 类定义（语法分析，领域专家）
2. 属性和操作的定义（有时在类之前就被发现）
3. 类关系的定义
4. 多重性的定义
5. 任务的定义
6. 继承的定义

动态图为对象模型发现和补充操作

顺序图：

布局：第一列——初始化用例的参与者

第二列——边界对象

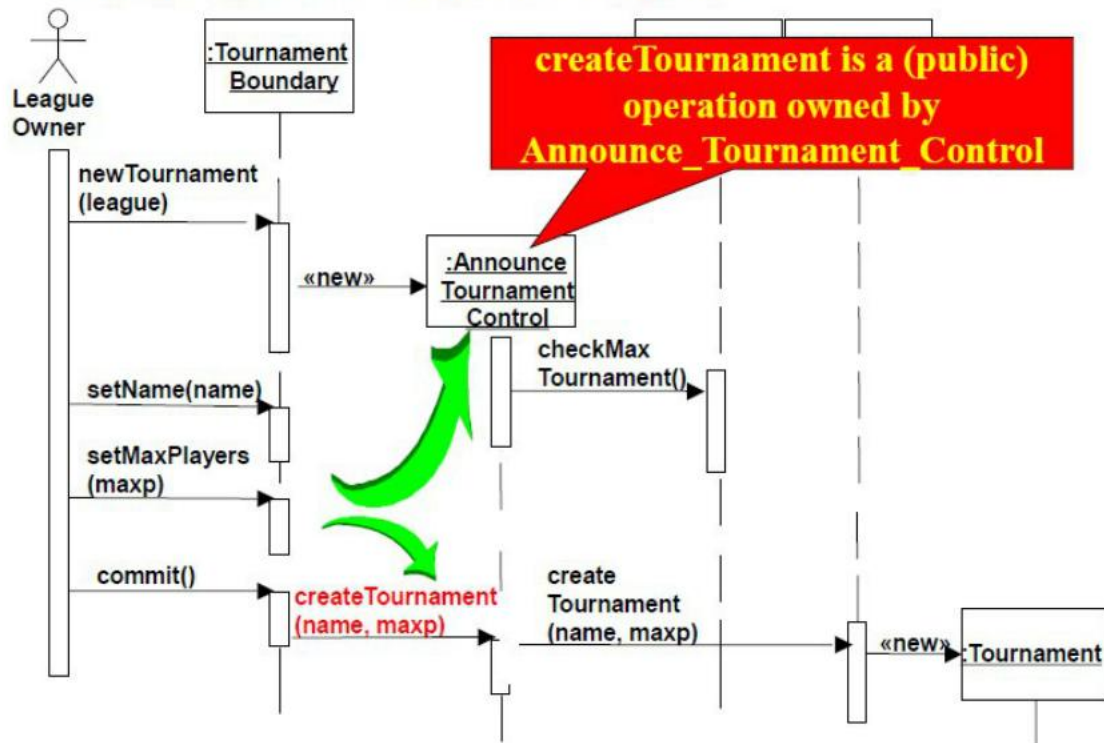
第三列：管理剩余用例的操作对象

创建：边界对象在用力初始化的时候被创建，控制对象由边界对象创建

访问：实体对象被控制对象和边界对象访问，实体对象决不能调用边界对象和控制对象

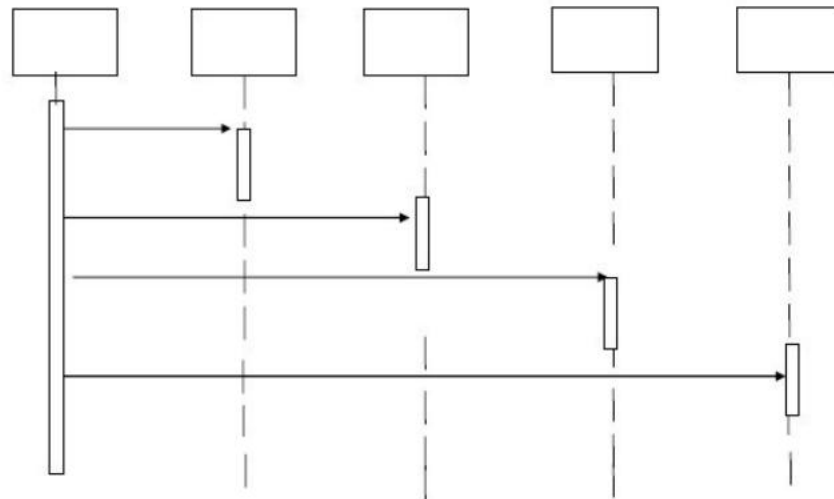
构建顺序图中，可能会有新出现的事件，这些事件可以被认为是接受这个事件的类的一个 public 权限的操作

Example from the Sequence Diagram



Fork Diagram

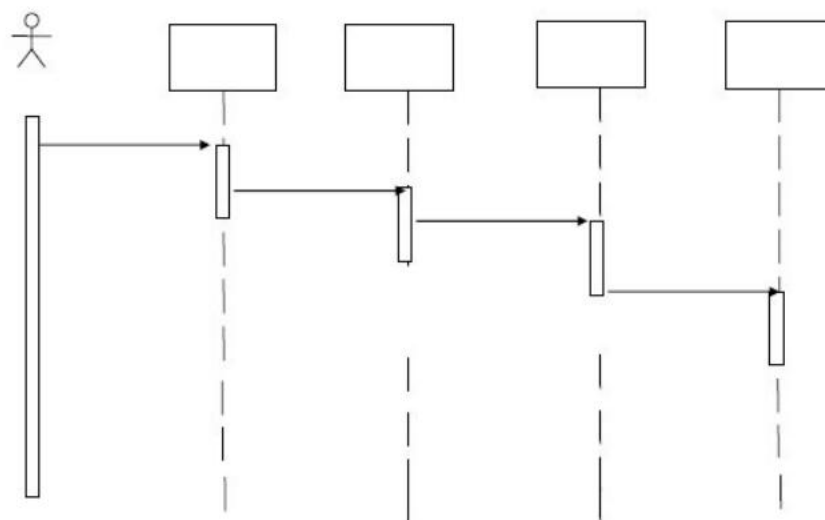
- ◆ Much of the dynamic behavior is placed in a single object, usually the control object. It knows all the other objects and often uses them for direct questions and commands.



集中控制式结构，操作可以交换顺序，有新需求时可以插入新操作

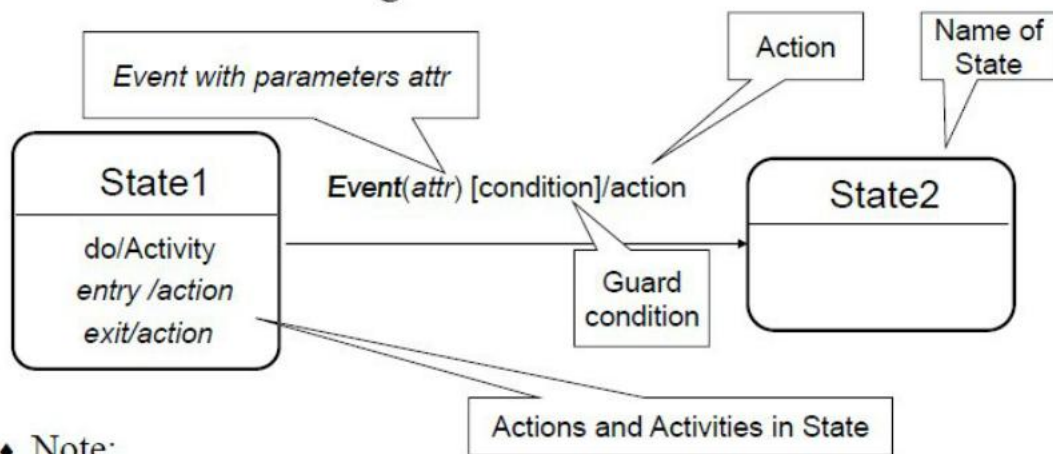
Stair Diagram

- ◆ The dynamic behavior is distributed. Each object delegates some responsibility to other objects. Each object knows only a few of the other objects and knows which objects can help with a specific behavior.



非集中式控制结构，操作之间有很强的联系和顺序性

UML 状态图符号 (Notation)



◆ Note:

事件是斜体，条件用[]括起，动作和活动之前都要加/
Notation 是根据 Harel 的工作基础上建立起来的

超状态:

其中有很多嵌套子状态

从其他状态进入超状态是进入的第一个子状态

从超状态的任意一个子状态都可以离开

两种类型的并发性:

系统并发性:

对象并发性:

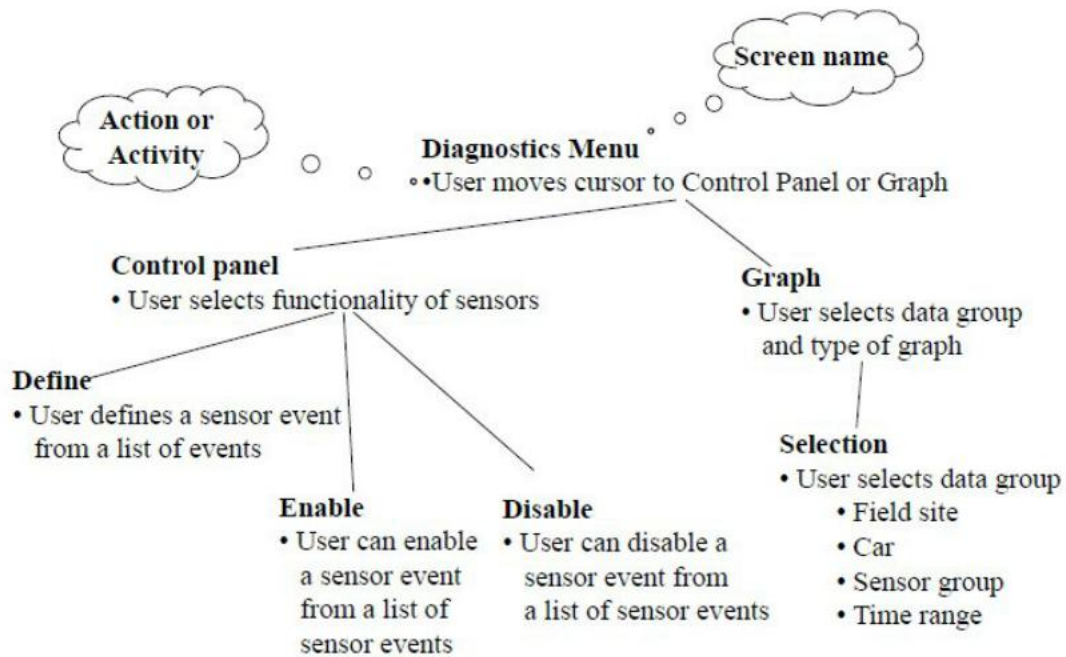
导航路线:

状态图可以被用作用户接口的设计

状态: 界面名称

活动/动作: 在界面名下面, 前面有个“•”, 一般只有退出动作会写出来

状态转换: 退出动作的结果 (点击按钮、选择菜单、光标移动)



Model verification 检验模型之间转换

model validation 把模型结合实际进行比较（更高一层），正确性、完整性、一致性、无歧义性、可实现性


对象模型占主要：系统有很多类，这些类的状态无关紧要，但是类之间的关系很复杂

动态模型占主要：模型有很多事件：输入、输出、异常、错误等

功能模型占主要：模型执行复杂的转换（计算包含许多步）

RAD (Requirement Analysis Document)

Requirements Analysis Document Template

1. Introduction
2. Current system
3. Proposed system
 - 3.1 Overview
 - 3.2 Functional requirements
 - 3.3 Nonfunctional requirements
 - 3.4 Constraints ("Pseudo requirements")
 -  3.5 System models
 - 3.5.1 Scenarios
 - 3.5.2 Use case model
 - 3.5.3 Object model
 - 3.5.3.1 Data dictionary
 - 3.5.3.2 Class diagrams
 - 3.5.4 Dynamic models
 - 3.5.5 User interface
4. Glossary

第六章

“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.”

– C. A. R. Hoare

设计目标是对不同利益关注者的权衡

子系统:

在UML中以包的形式出现

对象模型中的对象和类是子系统的“种子”

服务:

功能模型中的用例是服务的“种子”

服务在系统设计中被定义

子系统接口:

是服务的优化，需要被很好的设计并很小

子系统接口在对象设计中被定义

Application programmer' s interface (API) :
APIs 在实现中定义

层次:

一个子系统向另一个子系统提供服务

某个层次只依赖于它下层的的服务, 并且对它的上层一无所知

UML 中, Depend on 用实线, calls 用虚线

划分:

一个层次可以水平的被分为独立的子系统, 即划分

同一层的划分之间可以互相提供服务

划分是低耦合的子系统

P2P 关系

虚拟机:

上下层之间是一种“提供服务”的关系(关于子系统)

封闭架构(模糊): 上层只能向其下一层调用操作

可维护性, 灵活性

开放结构(透明): 每个虚拟机可以向任何一个下层的虚拟机调用操作
运行效率

一致性: 测量一个子系统中各个类之间的依赖程度(高)

耦合性: 测量各个子系统之间的依赖程度(低)

体系结构风格: 子系统分解的模式

客户端/服务器:

一个或多个服务器向称谓客户端的子系统实例提供服务

客户端知道服务器端的接口

服务器端不知道客户端的接口

常用于数据库

缺点: 不能满足 P2P 的要求

P2P:

ISO = International Standard Organization

OSI = Open System Interconnection

仓库(repository):

子系统访问和修改单一的数据结构

子系统之间相互独立, 子系统之间交互需要通过仓库完成

MVC

第七章

UML 构件图：编译时下子系统之间的设计依赖关系

UML 部署图：描述系统的执行时间配置

文件系统：数据只由一个人写由很多人读

数据库：数据被并发的人读和写

访问矩阵：每一行代表系统的参与者，每一列代表希望控制的类

访问表：参与者，类，操作

访问控制表：参与者，操作（针对于某个类）

能力：类，操作（针对于某个参与者）

第八章

对象设计：对需求分析进行补充并作出实现决定

应用对象：与系统相关的对象（应用域专家和终端用户定义）

求解对象：与应用域没有对应（开发人员定义）

继承：用新的操作或重新操作来扩展基类

1. 描述分类：需求分析中使用

2. 接口定义：对象设计中使用，定义所有对象的签名

实现继承（类继承）：父类的操作已经被实现

定义继承（子类型化）：只是继承被声明的操作，并不实现

类库：被动的，不影响控制流

框架：主动的，影响控制流

授权：抓取一个操作发送到另一个对象上

组合模式：模型动态聚集

适配器模式：与已经存在的系统关联

桥模式：与已经存在的和未来的系统关联，用于多重实现

facade 模式：与子系统关联

英文全称：

UML: unified modeling language（考到）

OMT: object modeling technology

OCL (Object Constraint Language)（考到）

OOAD (Object Oriented Analysis And Design)（考到）

FRIEND: First Responder Interactive Emergency
Navigational Database

ODD: The Object Design Document

CMM: Capability Maturity Model for Software

LOC: length of code (代码行)

COCOMO: constructive cost model

Application programmer' s interface (API)

ISO : International Standard Organization

OSI : Open System Interconnection (当时写下来觉得没什么用, 没想到居然真的考到这个了!)

OOSE: object-oriented software engineering

RAD : Requirement Analysis Document

“There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.”
– C. A. R. Hoare

OMT (James Rumbaugh)

OOSE (Ivar Jacobson)

Booch (Grady Booch)

复(yu)习到后面几章的时候离考试已经没几个消失了, 所以写下来的东西就很少了。。。

学长只能帮你们到这儿了, 学弟学妹们加油吧!

题目每年都差不多了, 尤其大题都是原题, 题目在人人网上都有分享的, 打印机和医院监控系统那两道网上也都有原题答案
然后二叉树那题我是这么写的不知道对不对

